UNC

# INGENIERÍA DE SOFTWARE

## Trabajo Práctico Nº 1


**GRUPO:**
Error 404

**INTEGRANTES:**
Rueda Horacio
Severini Montanari Alejo
Vega Cuevas Silvia Jimena
Wortley Agustina Daniela

**DOCENTES:**
Martin M. Miceli
Martin F. Bustos Menas


**2020**

FCEFyN

# Plan de Gestión de las Configuraciones

## Simulador de contagios


### Autores:

Rueda Horacio
Severini Montanari Alejo
Vega Cuevas Silvia Jimena
Wortley Agustina Daniela


### Versión:

1.0.2

**INDICE**

# 1. INTRODUCCIÓN

## 1.1 Propósito y alcance del plan

En este documento se pretende explicar la idea que llevaremos a cabo bajo el nombre “Simulador de contagios”. En el mismo se describen, además, todas las actividades de gestión de configuración  y cambios que serán realizadas durante todo el ciclo de vida del proyecto.

## 1.2 Herramientas de gestión de configuraciones

Herramienta de control de versiones:  GitHub: https://github.com/

Herramienta de integración continua:  CircleCI: https://circleci.com/

Herramienta de gestión de tareas:  Git Issues

# 2. GESTIÓN DE CAMBIOS Y CCB

## 2.1 Propósito

Esta sección del documento, se refiere al análisis de los costos y beneficios de los cambios propuestos, aprobando aquellos que lo ameritan e indagando cual o cuales de los componentes del sistema se modificarán.

Se entiende como cambio, todo lo que afecte la línea base del sistema. Este podría ser el reporte de un bug, en el que se describen sus síntomas, o una petición para agregar alguna funcionalidad al sistema.

Esta administración, será realizada por el CCB (Change Control Board)

## 2.2 Integrantes CCB

| Nombre | Rol | Contacto |
|---|---|---|
| Rueda Horacio | Diseñador Programador | horange88@gmail.com |
| Severini Montanari Alejo | Tester Programador | alejoseverini@gmail.com |
| Vega Cuevas Silvia Jimena | Arquitecto Programador | jimevega28@gmail.com |
| Wortley Agustina Daniela | Analista Programador | agus.wortley@gmail.com |

## 2.2 Detalles de cada rol

El trabajo será tal que todos los integrantes de la CCB aportaran en todas las áreas del proyecto, llamándose el rol de cada uno como “colaborador general”. Éste rol comprende a su vez los roles:

- Analista: entiende las necesidades del cliente, y asegura que la solución que está siendo desarrollada se ajusta a esas necesidades. Participa en la definición de requerimientos.
- Arquitecto: traduce los requisitos en una solución técnica. Piensa en el sistema antes de que éste se desarrolle y se ocupa de su seguimiento en tiempo de desarrollo.
- Diseñador: diseña y desarrolla la interfaz gráfica para el usuario, buscando la mejor interacción entre sistema y usuario.
- Programador: encargado de la programación propiamente dicha.
- Tester: construye los tests que se llevarán a cabo en el sistema.

### 2.3 Método de gestión de cambio y frecuencia de reuniones

Las reuniones entre los integrantes del grupo se realizarán principalmente una vez por semana para definir los lineamientos a seguir en el desarrollo del software.

En las reuniones, se analizaran los cambios propuestos y en base al análisis, se decidirá o no la aprobación de los mismos.

El análisis será completo: se discutirán tanto los beneficios y sus costos como las consecuencias (de no aprobar algún cambio). En algunos casos las reuniones tendrán solo fines informativos con el objetivo de que cada integrante comunique el trabajo que ha realizado y se encuentra realizando.

Esta administración, será realizada por el CCB (Change Control Board). Se deberá pactar un horario y lugar al cual puedan asistir todos.

## 3. IDENTIFICACIÓN DE LA CONFIGURACIÓN

Los planes generales, documentacion e items que aplican para realease serán mantenidos en la carpeta "*.\SimuladorDeContagios\release*"

Para cada release, una nueva carpeta será creada como .*SimuladorDeContagios\release\ReID* donde ID será reemplazado por la palabra ID seguido de un numero de versión a la fecha. Por ejemplo:

"SimuladorDeContagios\release\src\ReID01"

Las clases utilizadas para testeo se mantendrán bajo:

*SimuladorDeContagios\release*

Para el guardado de documentación y diagramas del proyecto se encontrará la carpeta:

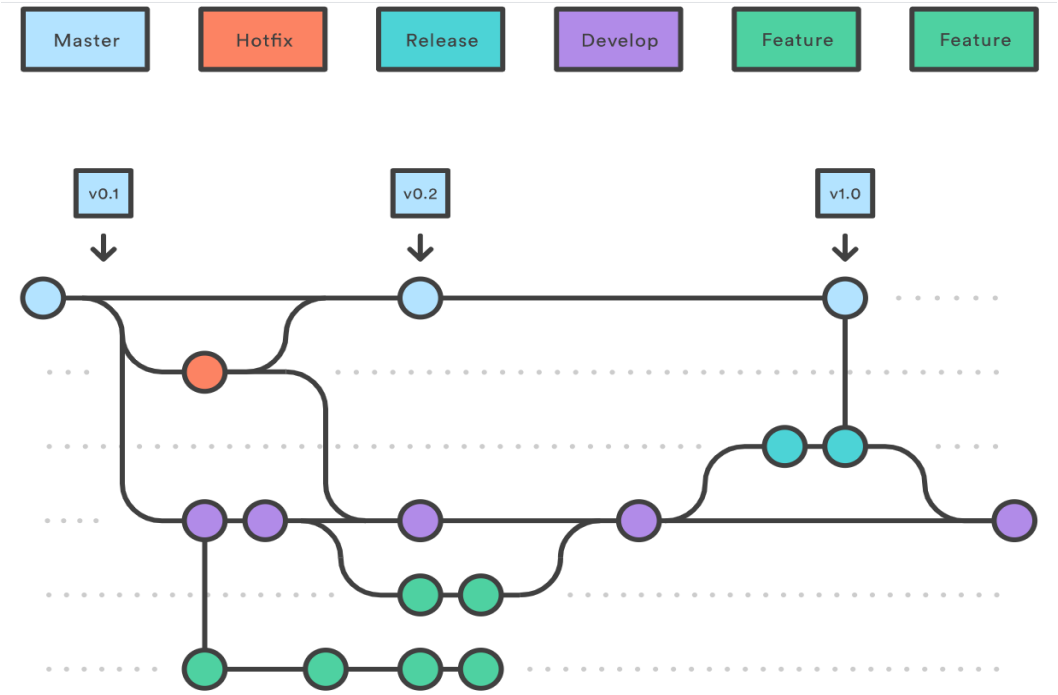
*SimuladorDeContagios\docs*

## 4. GESTIÓN DE LA CONFIGURACIÓN DEL CÓDIGO FUENTE

En esta sección se describen diferentes elementos de fuentes de gestión. Cubre aspectos relacionados con el branching, tagging y estrategia de merging, además de sus niveles de calidad esperados para todo el producto.

## 4.1 Definición de ramas

Nos basaremos en GitFlow Workflow, que es una metodología de trabajo basada en el división de las distintas etapas de producción de software en distintas ramas del repositorio. Como puede verse en la siguiente imagen:




**El esquema puede verse de la siguiente forma:**

- **master**: En la rama máster se encuentran las *releases* estables de nuestro software. Esta es la rama que un usuario típico se descargará para usar nuestro software, por lo que todo lo que hay en esta rama debería ser funcional. Sin embargo, puede que las últimas mejoras introducidas en el software no estén disponibles todavía en esta rama.
- **develop**: En esta rama surge de la última release de master. En ella se van integrando todas las nuevas características hasta la siguiente release.
- **feature-X**: Cada nueva mejora o característica que vayamos a introducir en nuestro software tendrá una rama que contendrá su desarrollo. Las ramas de feature salen de la rama develop y una vez completado el desarrollo de la mejora, se vuelven a integrar en develop.
- **release-X**: Las ramas de release se crean cuando se va a publicar la siguiente versión del software y surgen de la rama develop . En estas ramas, el desarrollo de nuevas características se congela, y se trabaja en arreglar bugs y generar documentación. Una vez listo para la publicación, se integra en master y se etiqueta con el número de versión correspondiente. Se integran también con develop, ya que su contenido ha podido cambiar debido a nuevas mejoras.

- **hotfix-X**: Si nuestro código contiene bugs críticos que es necesario parchear de manera inmediata, es posible crear una rama hotfix a partir de la publicación correspondiente en la rama master. Esta rama contendrá únicamente los cambios que haya que realizar para parchear el bug. Una vez arreglado, se integrará en master, con su etiqueta de versión correspondiente y en develop.

## 4.2 Definición de etiquetas

Normas de etiquetado y de nombramiento de archivos:
Para la numeración de las versiones se utilizará la notación de tres números separados por puntos y precedidos por la letra “v”. Ejemplo: v1.2.4

Su significado será:

- El tercer dígito (build) representa correcciones de bugs o errores encontrados. También se suelen incluir cambios no funcionales (correcciones ortográficas, agregado de comentarios, etc.).
- El segundo dígito (release) representa modificaciones funcionales, es decir se han añadido, eliminado o modificado funcionalidades al código.
- El primer dígito (versión propiamente dicha) representa cambios mayores en el diseño del código.

Los planes generales y documentación que apliquen para el release, serán nombrados de la siguiente forma:

- Plan de Gestion de las Configuraciones - Simulador de contagios
- Documento de Requerimientos - Simulador de Contagios
- Documento de Arquitectura y Diseño del Sistema
- Casos de Prueba y matriz de Trazabilidad

y estarán en el folder:

*“ proyecto/SimuladorDeContagios/PlanesDelProyecto “*

Por cada release una nueva carpeta será creada de la forma:

*“ Proyecto/ReleaseID “*

## 4.3 Repositorio del código fuente

Repositorio Git: https://github.com/horange88/SimuladorContagios

# 5. POLÍTICA DE FUSIÓN DE ARCHIVOS (MERGING)

Se usará la política “merge from tags”. Las tags se aplicarán cada vez que se haga un merge en las versiones fuente. De forma genérica, el formato es: *merge-from-<source>-<yyyymmdd>-<nn>*.

Los merges al Master en este caso podremos hacerlos los 4 integrantes del grupo, pero por lo general hay una persona encargada de chequear los cambios antes de realizar el merge.

# 6. FORMA DE ENTREGA DE LOS RELEASE

Los releases se entregarán de las siguiente manera:

El formato del programa a entregar es un archivo ejecutable .jar que no necesita de instalación

Además se anexara un archivo de texto plano README.TXT detallando los requerimientos necesarios para la ejecución del programa y otras instrucciones necesarias

Se subirá una nueva versión del programa al Drive que deberá ser descargada por el usuario, sin borrar la anterior, ya que en caso de producirse algún error no testeado con esa versión, el usuario podrá volver a la versión que le funcionaba correctamente.

Para poder correr el programa, el usuario necesita tener instalado un sistema operativo Microsoft Windows, Mac OS X, GNU/Linux, Solaris y el entorno de ejecución de Java JRE (Java Runtime Environment) actualizado en el mismo.

# 7. HERRAMIENTA DE SEGUIMIENTO DE TAREAS

Github nos proporciona la herramienta Issues para hacer seguimiento de errores. Una vez creado el repositorio se tendrá acceso a ella. El mismo se podrá acceder a través del siguiente enlace:

https://github.com/horange88/SimuladorContagios/issues

## 8. HISTORIAL DE VERSIONES

| Versión | Comentario | Fecha |
|---|---|---|
| 1.0.0 | Versión inicial para primer entrega | 7 Mayo 2020 |
| 1.0.1 | División de documentos. Actualización de herramientas. Cambio de roles en CCB. Instrucciones de Instalación | 19 Junio 2020 |
| 1.0.2 | Cambio en herramienta de gestión de tareas | 23 Junio 2020 |

# Documento de Requerimientos

## Simulador de contagios

**Presentado a:**

Ing. Martín M. Miceli
Ing. Martin F. Bustos Menas

**Presentado por:**

Rueda Horacio
Severini Montanari Alejo
Vega Cuevas Silvia Jimena
Wortley Agustina Daniela

**Dia de presentacion:**

7 de Mayo, 2020

**ÍNDICE**

# 1. INTRODUCCIÓN

Este documento presenta las funciones y requerimientos del Simulador de Contagios. Así como también diagramas UML para graficar su uso, además de la matriz de trazabilidad entre requerimientos y casos de usos.

# 2. DESCRIPCIÓN GENERAL

El proyecto consiste en un simulador de contagios que permite visualizar y analizar la propagación de una enfermedad contagiosa entre la población de una cierta área. Dadas algunas condiciones iniciales (población, dimensiones del área, cantidad de enfermos, etc), la idea es que el simulador funcione de la siguiente manera:

• Genera N personas en una posición (x,y) aleatoria, dentro de una cierta área, donde alguna de ellas están enfermas.

• En cada paso, las personas se mueven aleatoriamente, y si una persona sana se encuentra cerca de una enferma, se contagia.

• Al cabo de un cierto tiempo, las personas enfermas mueren (con una cierta probabilidad) o de lo contrario se recuperan.

Se podra visualizar en tiempo real un mapa (grilla) con la posición de cada persona y su estado (sana, enferma, recuperada o fallecida) y gráficos que muestran la curva de contagios, recuperación y fallecimientos.

Se podra cambiar el comportamiento de las personas (disminuir o aumentar su movilidad, simulando una cuarentena) , la probabilidad de fallecimiento (edad avanzada, otras enfermedades, saturación del sistema sanitario, etc) y la forma en que se propaga la enfermedad

# 3. LIMITACIONES GENERALES

- Limitaciones de Software:

Se necesita tener instalado un sistema operativo Microsoft Windows, Mac OS X, GNU/Linux, Solaris y el entorno de ejecución de Java JRE (Java Runtime Environment)

- Limitaciones de Hardware

Cada operador necesitará una PC capaz de ejecutar alguno de los sistema operativo antes mencionados

# 4. SUPOSICIONES Y DEPENDENCIAS

El sistema asume que los usuarios tienen la habilidad adecuada para usar computadoras y software de computadora.

# 5. LISTA GENERAL DE REQUERIMIENTOS

## 5.1 Requerimientos funcionales

**REQ. 1: Selección de modos de ejecución**

Al iniciar el programa, se mostrará un menú interactivo donde se podrá elegir:

- Correr una nueva simulación
- Ver historial de simulaciones (últimas 20 simulaciones).
- Comparar dos simulaciones del historial.

**REQ. 2: Generación de resumen de simulación**

Al finalizar la simulación se deben generar gráficos y tablas con los datos obtenidos en la misma.

**REQ. 3: Visualización en tiempo real**

Durante la ejecución de la simulación, el sistema deberá mostrar en pantalla la siguiente información:

- Cantidad de personas sanas.
- Cantidad de personas enfermas.
- Cantidad de personas curadas..
- Cantidad de muertes.
- Ventana gráfica que muestre la posición, estado e interacción de las personas.

**REQ. 4: Ingreso de parámetros de simulación**

El usuario deberá poder ingresar los parámetros de la simulación antes de ejecutarla. También podrá modificarlos durante la ejecución de la misma

**REQ. 5: Validación de parámetros de simulación**

El sistema deberá imprimir en pantalla un mensaje de error en caso de que los datos ingresados por el usuario no sean válidos.

**REQ. 6: Ventana de ayuda**

El sistema deberá ofrecer al usuario una ventana de ayuda

**REQ. 7: Almacenamiento de resultados**

Una vez finalizada la simulación, el sistema deberá guardar automáticamente los resultados de la misma, junto con los parámetros de simulación.

## 5.2 Requerimientos no funcionales

**REQ. 8:** Una persona sin conocimientos básicos informáticos debe poder aprender a usar la herramienta sin entrenamiento en no más de 5 minutos.

**REQ. 9:** No deberá haber más de 8 botones interactivos en pantalla simultáneamente.

**REQ. 10:** La simulación no debería tardar más de 10 minutos en ejecutarse.

**REQ. 11:** Ante un fallo, el sistema debe reiniciarse en no más de 5 segundos.

**REQ. 12:** El tamaño de la aplicación no deberá ser mayor a 20 MB.

**REQ. 13:** El tiempo de aparición de la ventana de resumen no deberá ser mayor a 2 segundos.

**REQ. 14:** El programa deberá ser desarrollado en Java, por lo que será portable y 0% dependiente.

# 6. DETALLE DE REQUERIMIENTOS

## 6.1 REQ. 1: Selección de modos de ejecución

### 6.1.1 Descripción

Al iniciar el sistema el usuario deberá poder elegir entre ejecutar una nueva simulación, ver historial de las últimas 20 simulaciones ejecutadas y comparar los resultados de dos simulaciones ejecutadas previamente.

### 6.1.2 Pantalla

El sistema deberá mostrar en pantalla un menú mostrando las opciones:

- Ejecutar una nueva simulación con visualización en tiempo real
- Ejecutar una nueva simulación sin visualización en tiempo real
- Ver historial de simulaciones ejecutadas.
- Comparar dos simulaciones del historial.

### 6.1.3 Ingreso de datos

No corresponde

### 6.1.4 Procesamiento

No corresponde.

### 6.1.5 Salida

Si el usuario seleccionó una ejecutar una nueva simulación, y los datos ingresados son correctos, el sistema mostrará un mensaje indicando que se ha iniciado la simulación

En el caso en que el usuario haya seleccionado ver el historial de simulaciones ejecutadas, el sistema mostrará la lista de las últimas 20 simulaciones realizadas.

En el caso en que el usuario haya seleccionado comparar dos simulaciones anteriores, el sistema mostrará la lista de las últimas 20 simulaciones realizadas.

## 6.2 REQ. 2: Generación de resumen de simulación

### 6.2.1 Descripción

Al ejecutar una nueva simulación el sistema deberá generar un resumen con los resultado de la misma.

### 6.2.2 Pantalla

El sistema deberá mostrar en pantalla tablas y gráficos con los resultados estadísticos de la simulación realizada:

- Duración de simulación

- Pico de contagios
- Pico de fallecimientos
- Pico recuperaciones

#### 6.2.3 Ingreso de datos

No corresponde

#### 6.2.4 Procesamiento

No corresponde

#### 6.2.5 Salida

No corresponde

## 6.3 REQ. 3: Visualización en tiempo real

#### 6.3.1 Descripción

Durante la ejecución de una simulación con visualización en tiempo real, el sistema deberá mostrar en pantalla, en cada paso de simulación, información gráfica y numérica acerca del proceso de simulación.

#### 6.3.2 Pantalla

No corresponde

#### 6.3.3 Ingreso de datos

No corresponde

#### 6.3.4 Procesamiento

No corresponde

#### 6.3.5 Salida

El sistema deberá mostrar en pantalla una ventana con la siguiente información:

- Un plano indicando la posición y estado de cada una de las personas (sanas, enfermas, recuperadas, fallecidas)
- Un gráfico indicando la cantidad de personas sanas, enfermas, recuperadas y fallecidas,
- Parámetros de simulación
- Resultados estadísticos expresados en forma numérica

## 6.4 REQ. 4: Ingreso de parámetros de simulación

#### 6.4.1 Descripción

El usuario deberá ingresar los parámetros de la simulación antes de ejecutarla. También podrá modificar ciertos parámetros durante la ejecución de la misma.

### 6.4.2 Pantalla

Antes de iniciar una simulación el sistema mostrará en el centro de la pantalla una ventana con campos para el ingreso de los parámetros indicados en el apartado “Ingreso de datos”.

A lo largo de la simulación el sistema mostrará a un lado de la ventana gráfica los campos necesarios para la modificación de los parámetros de simulación en tiempo real.

### 6.4.3 Ingreso de datos

Al ejecutar una nueva simulación el sistema deberá solicitar al usuario el ingreso de los parámetros de simulación:

- Área: expresada en km^2.
- Población total y población inicial de infectados.
- Tasa de mortalidad.
- Tiempo de recuperación.
- Tiempo de incubación.
- Radio de contagio.
- Inmunidad.
- Movilidad.
- Tiempo de simulación.

### 6.4.4 Procesamiento

Los datos ingresados por el usuario serán validados según lo indicado en el REQ. 5

### 6.4.5 Salida

El sistema mostrará en pantalla un mensaje indicando si los datos ingresados por el usuario son válidos o no, según el REQ. 5.

## 6.5 REQ. 5: Validación de parámetros de simulación

### 6.5.1 Descripción

El sistema deberá validar que los datos ingresados por el usuario, verificando que se encuentren dentro de los rangos y tipos esperados por el sistema. En caso contrario deberá informar al usuario que alguno de los datos ingresados no es correcto.

### 6.5.2 Pantalla

No corresponde

### 6.5.3 Ingreso de datos

No corresponde

#### 6.5.4 Procesamiento

Los datos ingresados por el usuario deberán ser validados teniendo en cuenta los siguientes criterios:

- Que se encuentren dentro del rango definido por el sistema
- Que sean del tipo definido por el sistema
- Que los campos obligatorios no estén vacíos

#### 6.5.5 Salida

En caso que no se cumpla alguno de los criterios de validación, el sistema deberá informar al usuario que alguno de los datos ingresados no es correcto. Si los datos son correctos, el sistema continuará con la ejecución solicitada por el usuario.

## 6.6 REQ. 6: Ventana de ayuda

#### 6.6.1 Descripción

El sistema deberá ofrecer al usuario una ventana de ayuda acerca de las operaciones que se pueden realizar y de los rangos de valores esperados para cada campo.

#### 6.6.2 Pantalla

El sistema mostrará en pantalla una nueva ventana con la información de ayuda para el usuario

#### 6.6.3 Ingreso de datos

No corresponde.

#### 6.6.4 Procesamiento

No corresponde.

#### 6.6.5 Salida

No corresponde.

## 6.7 REQ. 7: Almacenamiento de resultados

#### 6.7.1 Descripción

Una vez finalizada la simulación, el sistema deberá guardar automáticamente los resultados de la misma, junto con los parámetros de simulación. Esta información se guardará en archivos de texto plano, indicando la fecha y hora de la simulación

#### 6.7.2 Pantalla

No corresponde.

#### 6.7.3 Ingreso de datos

No corresponde.

### 6.7.4 Procesamiento

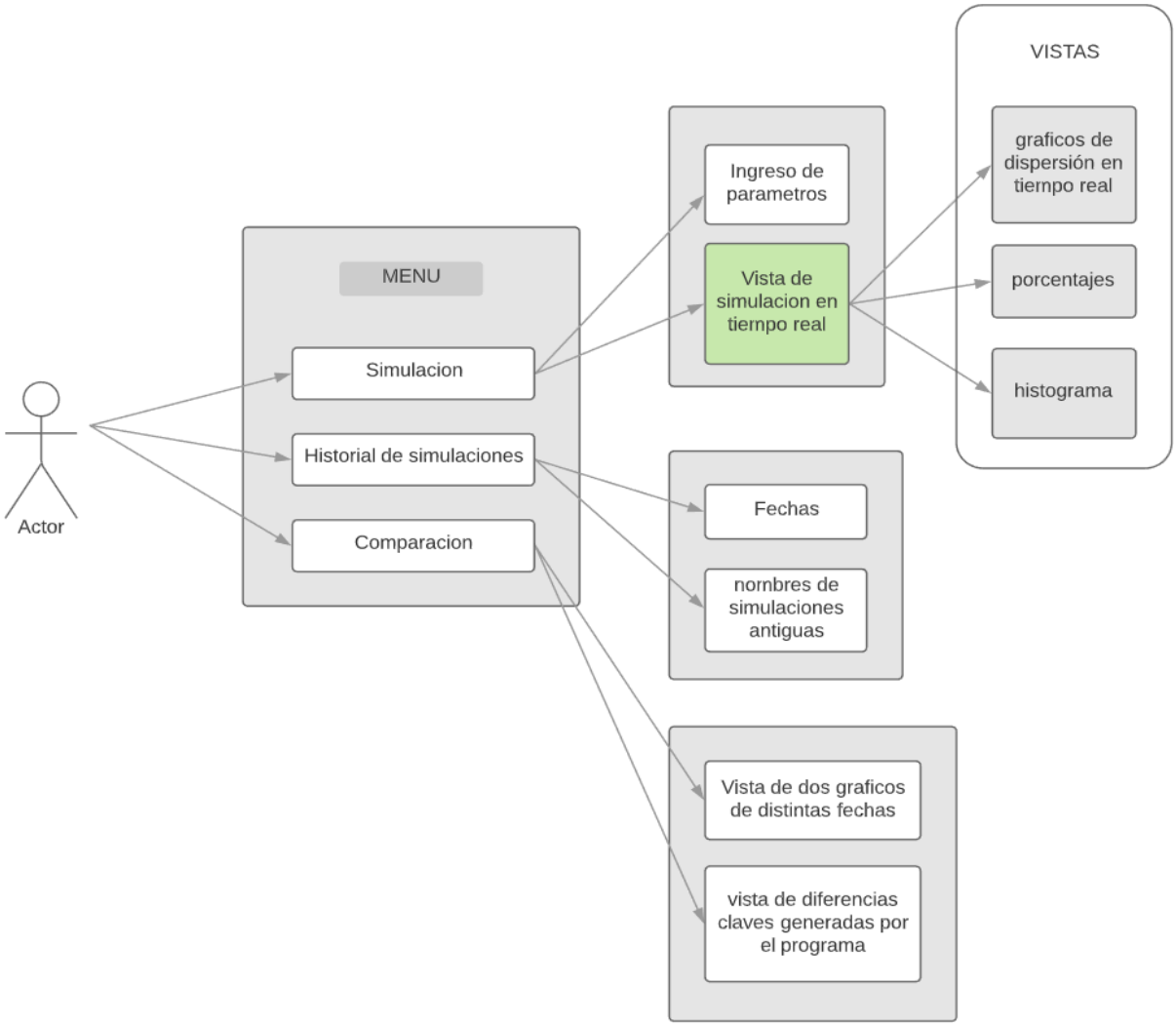
No corresponde.

### 6.7.5 Salida

No corresponde.

# 7. DIAGRAMAS

## 7.1 Caso de uso

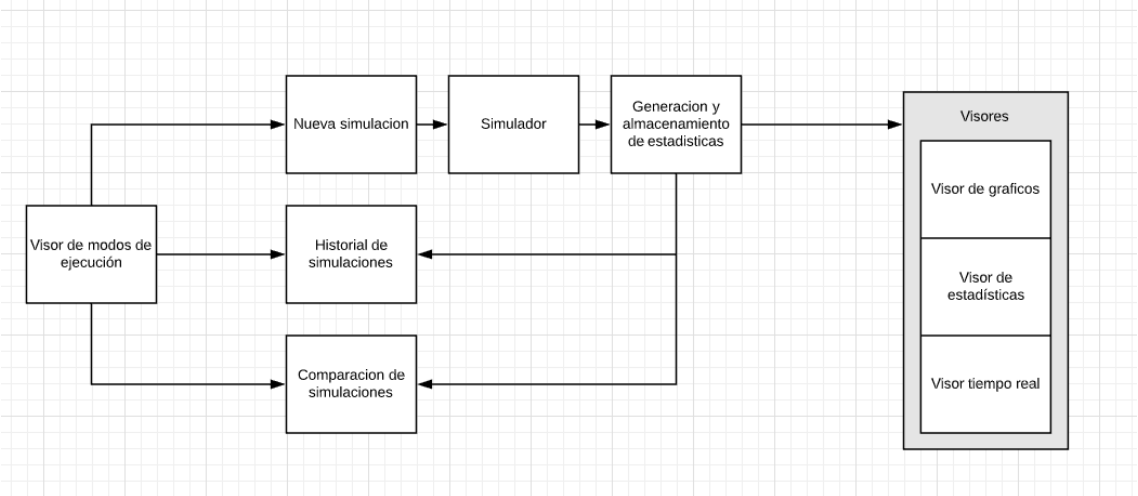


## 7.2 Diagrama de Actividades

En el siguiente diagrama se representa gráficamente cómo se llevará a cabo el proceso de crear una nueva simulación.

## 7.3 Diagrama de Arquitectura preliminar

Permite asociar requerimientos y casos de usos con los sistemas, subsistemas y módulos identificados.




## 8. HISTORIAL DE VERSIONES

| Versión | Comentario | Fecha |
|---|---|---|
| 1.0.0 | Versión inicial para primer entrega | 7 Mayo 2020 |
| 1.0.1 | Actualización requerimiento 9 y 14 | 19 Junio 2020 |

# Casos de prueba de Sistemas y Matriz de trazabilidad

## Simulador de contagios

# INDICE

# 1. INTRODUCCION

En el siguiente documento se detallan los casos de prueba del sistema tanto normales como casos de prueba alternativos contra los requerimientos funcionales y no funcionales.

Luego se seleccionarán un subconjunto de ellos para que actuen como Smoke Test.

Finalmente se detalla la matriz de trazabilidad representativa del proyecto.

# 2. CASOS DE PRUEBA

## 2.1 Smoke Test

### 2.1.1 Prueba REQ.1.1:

●Iniciar el programa

**Resultado esperado:** Aparecerá un Menú donde se podrá elegir “Correr una nueva simulación”, “Ver historial de simulaciones” o “Comparar dos simulaciones del historial”.

### 2.1.2 Prueba REQ.2:

●Iniciar el programa

●Seleccionar “Correr nueva simulación”

●Seleccionar “Ver en tiempo real”

●Ingresar todos los parámetros solicitados

**Resultado esperado:** Se correrá la correspondiente simulación y se mostrará por pantalla tablas y gráficos con los datos obtenidos de la misma.

## 2.2 Prueba de Sistema

### 2.2.1 Prueba REQ.3:

●Iniciar el programa

●Seleccionar “Correr nueva simulación”

●Seleccionar “Ver en tiempo real”

●Ingresar los parámetros solicitados

**Resultado esperado:** Se mostrará en tiempo real la simulación, pudiendo ver gráficamente la interacción de las personas sanas y contagiadas.

### 2.2.2 Prueba REQ.6:

●Iniciar el programa

●Seleccionar la ventana de ayuda

**Resultado esperado:** Se abrirá la ventana de ayuda y brindará información de utilidad al usuario.

### 2.2.3 Prueba REQ.7:

●Iniciar el programa

●Seleccionar “Correr nueva simulación”

●Seleccionar “Ver en tiempo real” o “Saltar a datos finales”

●Ingresar los parámetros solicitados

●Correr la simulación

●Finalizada la misma, saltar los gráficos y tablas y volver al Menú Inicial

●Seleccionar “Ver historial de simulaciones”

●Seleccionar la última simulación

**Resultado esperado:** Se mostrarán mediante gráficos y tablas los datos de la última simulación.

## 2.3 Pruebas Normales

### 2.3.1 Prueba REQ.1.2:

Verificar que es posible correr una nueva simulación en tiempo real. Ingresar parámetros válidos para la simulación y seleccionar vista de gráficos

**Resultado esperado:** Se generarán los correspondientes gráficos de dispersión en tiempo real correspondientes a los parámetros ingresados

### 2.3.2 Prueba REQ.1.3:

Verificar que es posible correr una nueva simulación en tiempo real. Ingresar parámetros válidos para la simulación y seleccionar vista de porcentajes

**Resultado esperado:** Se generará una tabla con los datos y porcentajes calculados respecto de los valores de parámetros ingresados

### 2.3.3 Prueba REQ.1.4:

Verificar que es posible correr una nueva simulación en tiempo real. Ingresar parámetros válidos para la simulación y seleccionar vista de histogramas

**Resultado esperado:** Se generará una representación gráfica de las variables en forma de barras, según los parámetros ingresados, formando el correspondiente histograma

## 2.4 Pruebas de sistemas Negativos (alternativos)

### 2.4.1 Prueba REQ.4:

- Iniciar el programa
- Seleccionar “Correr nueva simulación”
- Seleccionar “Ver en tiempo real” o “Saltar a datos finales”

**Resultado esperado:** Se solicitará el ingreso de los parámetros para la simulación.

### 2.4.2 Prueba REQ.5:

- Iniciar el programa
- Seleccionar ”Correr nueva simulación”
- Seleccionar “Ver en tiempo real” o “Saltar a datos finales”
- Ingresar un área negativa

**Resultado esperado:** Mensaje de error y solicitación de reingreso del parámetro “Área

# 3. MATRIZ DE TRAZABILIDAD

## 3.1 Requerimientos - Casos de uso

| Identificación Requerimiento | Descripción del Requerimiento | Caso de uso |
|---|---|---|
| **REQ. 1** | Selección de modos de ejecución | Simulación,<br>Historial de simulaciones,<br>Comparación de simulaciones |
| **REQ. 2** | Generación de resumen de simulación | Simulación |
| **REQ. 3** | Visualización en tiempo real | Simulación |
| **REQ. 4** | Ingreso de parámetros de simulación | Simulación |
| **REQ. 5** | Validación de parámetros de simulación | Simulación |
| **REQ. 6** | Ventana de ayuda | Simulación,<br>Historial de simulaciones,<br>Comparación de simulaciones |
| **REQ. 7** | Almacenamiento de resultados | Simulación |

## 3.2 Requerimientos - Casos de prueba

| Identificación Requerimiento | Descripción del Requerimiento | Tipo de Caso de prueba | Caso de prueba |
|---|---|---|---|
| **REQ. 1** | Selección de modos de ejecución | Smoke Test | Prueba Req 1 |
| | | Prueba normal | Prueba Req 1.2<br>Prueba Req 1.3<br>Prueba Req 1.4 |
| **REQ. 2** | Generación de resumen de simulación | Smoke Test | Prueba Req 2 |
| **REQ. 3** | Visualización en tiempo real | Prueba de sistema | Prueba Req 3 |
| **REQ. 4** | Ingreso de parámetros de simulación | Prueba de sistema negativo | Prueba Req 4 |
| **REQ. 5** | Validación de parámetros de simulación | Prueba de sistema negativo | Prueba Req 5 |
| **REQ. 6** | Ventana de ayuda | Prueba de sistema | Prueba Req 6 |
| **REQ. 7** | Almacenamiento de resultados | Prueba de sistema | Prueba Req 7 |

.

## 3.3 Casos de prueba - Requerimientos

| Tipo de Caso de prueba | Caso de prueba | Identificación Requerimiento |
|---|---|---|
| **Smoke Test** | Prueba Req 1 | REQ. 1 |
| | Prueba Req 2 | REQ. 2 |
| **Prueba normal** | Prueba Req 1.2<br>Prueba Req 1.3<br>Prueba Req 1.4 | REQ. 1 |
| **Prueba de sistema** | Prueba Req 3 | REQ. 3 |
| | Prueba Req 4 | REQ. 4 |
| | Prueba Req 5 | REQ. 5 |
| | Prueba Req 6 | REQ. 6 |
| | Prueba Req 7 | REQ. 7 |

## 4. HISTORIAL DE VERSIONES

| Versión | Comentario | Fecha |
|---|---|---|
| 1.0.0 | Versión inicial para primer entrega | 7 Mayo 2020 |

# Documento de Arquitectura

## Simulador de contagios

### Presentado a:

Ing. Martín M. Miceli
Ing. Martin F. Bustos Menas

### Presentado por:

Rueda Horacio
Severini Montanari Alejo
Vega Cuevas Silvia Jimena
Wortley Agustina Daniela

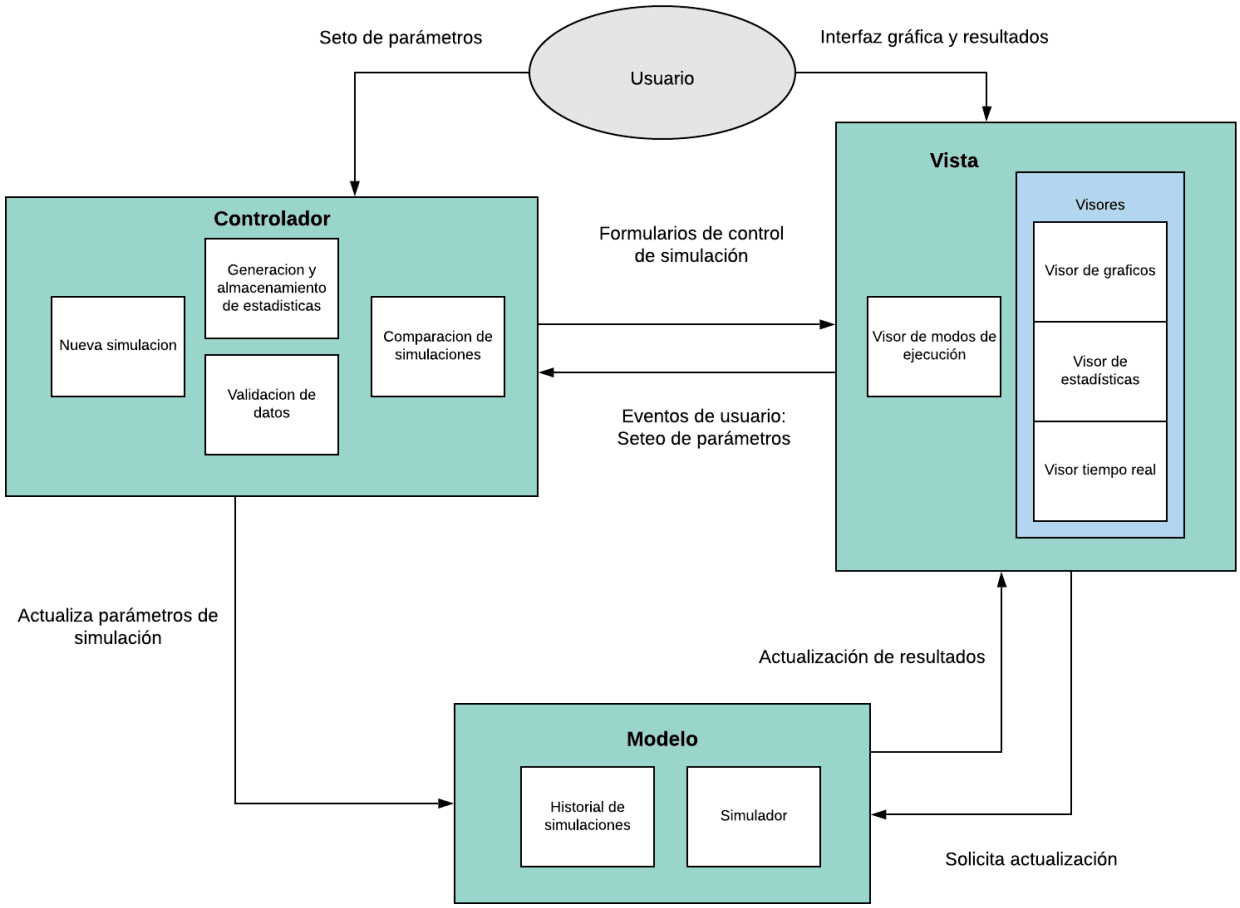
### Dia de presentacion:

19 de Junio, 2020

# PATRÓN DE ARQUITECTURA

Para esta aplicación, se implementó el patrón de arquitectura **Model View Controller** ya que se utiliza cuando existen múltiples formas de ver e interactuar con los datos y nos ayuda a desarrollar la aplicación gracias a su capacidad de modificar por medio de un controlador el modelo, y este muestra los cambios al usuario a través de las vistas. Y al ser más intuitivo y visible, es más fácil de aprender a usar; los fallos no son tan recurrentes gracias a la disposición de las clases en cada apartado (Modelo, Vistas y Controladores).

**Diagrama de arquitectura con patrón MVC**



**MODELO:**

Se encarga de los datos, consultando la base de datos. Actualizaciones, consultas, búsquedas, etc

**CONTROLADOR:**

Se encarga de recibir las órdenes del usuario y de solicitar los datos al modelo y de comunicarles a la vista.

**VISTAS:**

Son la representación visual de los datos, todo lo que tenga que ver con la interfaz gráfica. Menú desplegables, ventanas emergentes para dados, etc. Se emplea el método “4+1”.

**Detalle de las vistas:**

- Vista de modos de ejecución: se pueden visualizar 3 opciones que nos brinda el programa

- Vista de gráficos: en esta se podrá visualizar un gráfico de tipo histograma que nos brinda información sobre la cantidad de personas contagiadas, sanas, recuperadas y muertas.

- Vista de estadísticas: aquí se verá información estadística basada en los parámetros que ingrese el usuario.

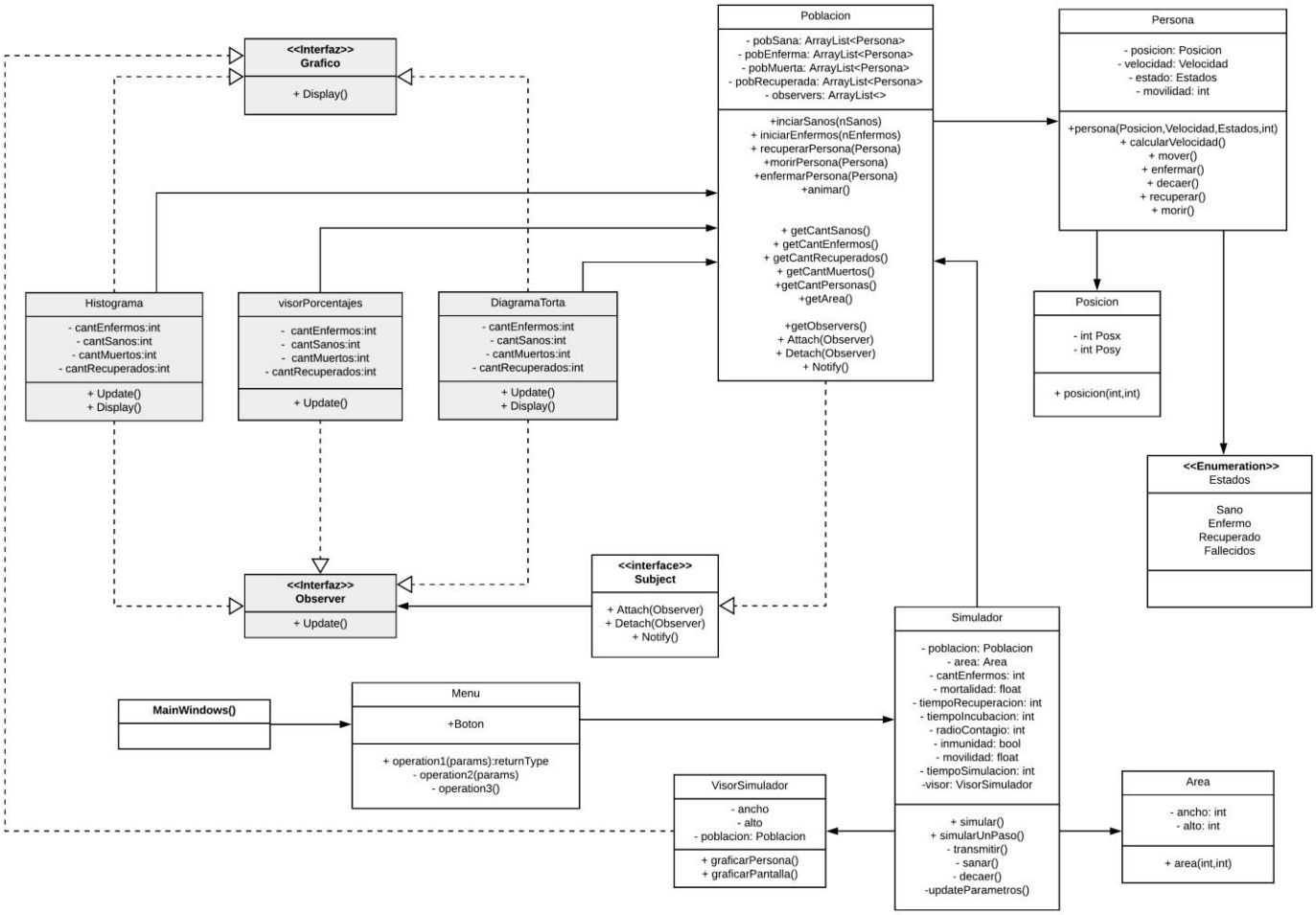
- Vista en tiempo real: aquí se verá el movimiento de las personas, representando gráficamente la velocidad de movimiento, identificando las personas contagiadas, enfermas, etc. En un área determinada .

Como se dijo previamente, se emplea el método “4+1” para el despliegue de información de cada una de las vistas:

## **VISTA LÓGICA**:

Muestra la funcionalidad para los usuarios finales.

Proponemos un diagrama de clases (simplificado) el cual describe la estructura del sistema con las clases del mismo sus atributos operaciones y relaciones entre los objetos.



## VISTA DE PROCESO:

Formado por procesos que interactúan en tiempo de ejecución.

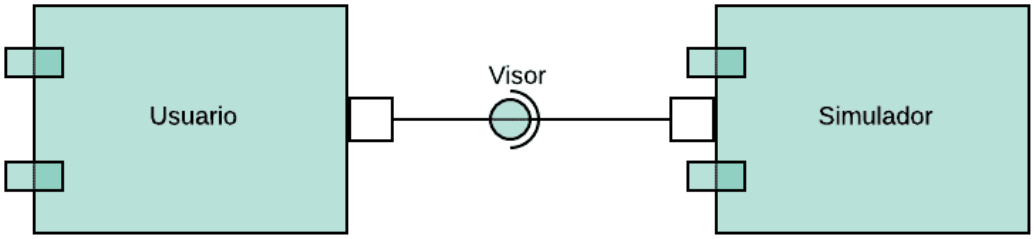
Emplearemos el diagrama de actividades el cual representa el algoritmo o proceso del software

## **VISTA DE DESARROLLO**:

Se desglosa el software para organizarnos y posteriormente desarrollarse el SW, al permitirnos tener una vista de alto nivel del sistema.

Se optó por realizar un diagrama de componentes el cual muestra la división del software en componentes y las dependencias entre estos.

**Diagrama de componentes**

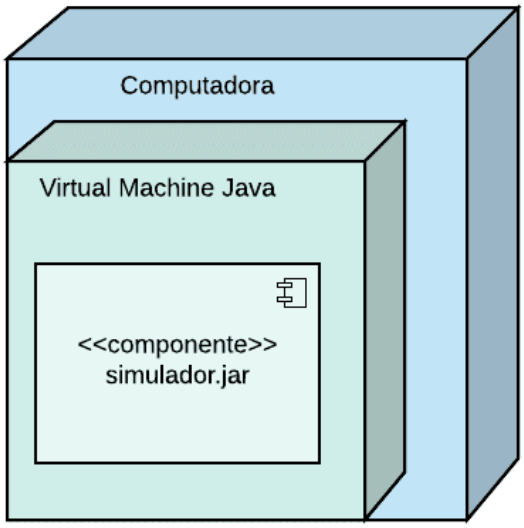



## **VISTA FÍSICA**:

Está relaciona la topología de componentes de software en la capa física, así como las conexiones físicas entre estos componente

Utilizamos el siguiente diagrama UML de despliegue modela la disposición física de software en nodos

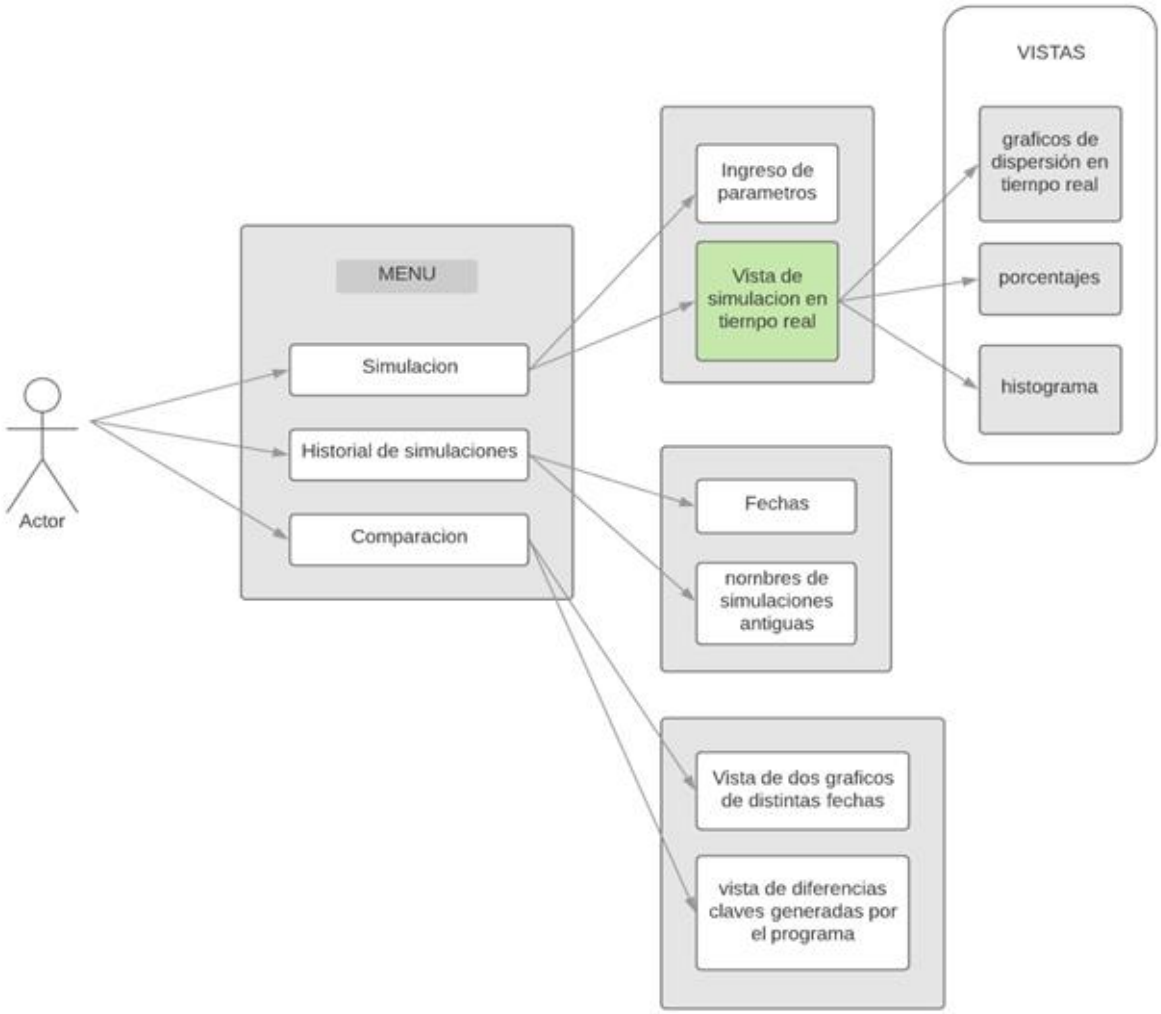
**Diagrama de despliegue**



## VISTA DE ESCENARIOS:

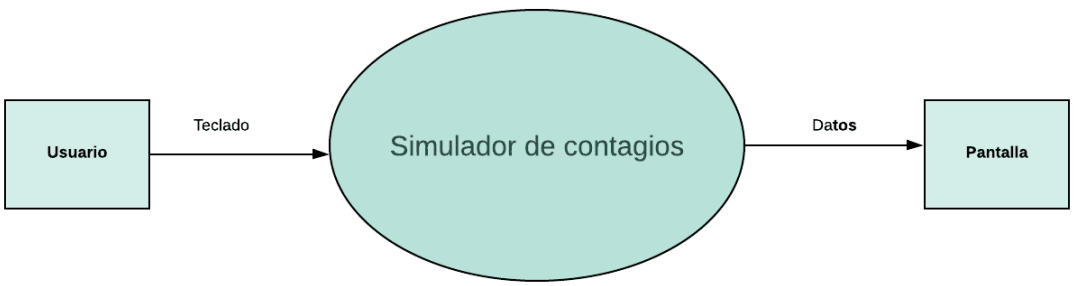
Permite relacionar las 4 vistas superiores.

Para este caso empleamos nuevamente un diagrama de casos de uso:

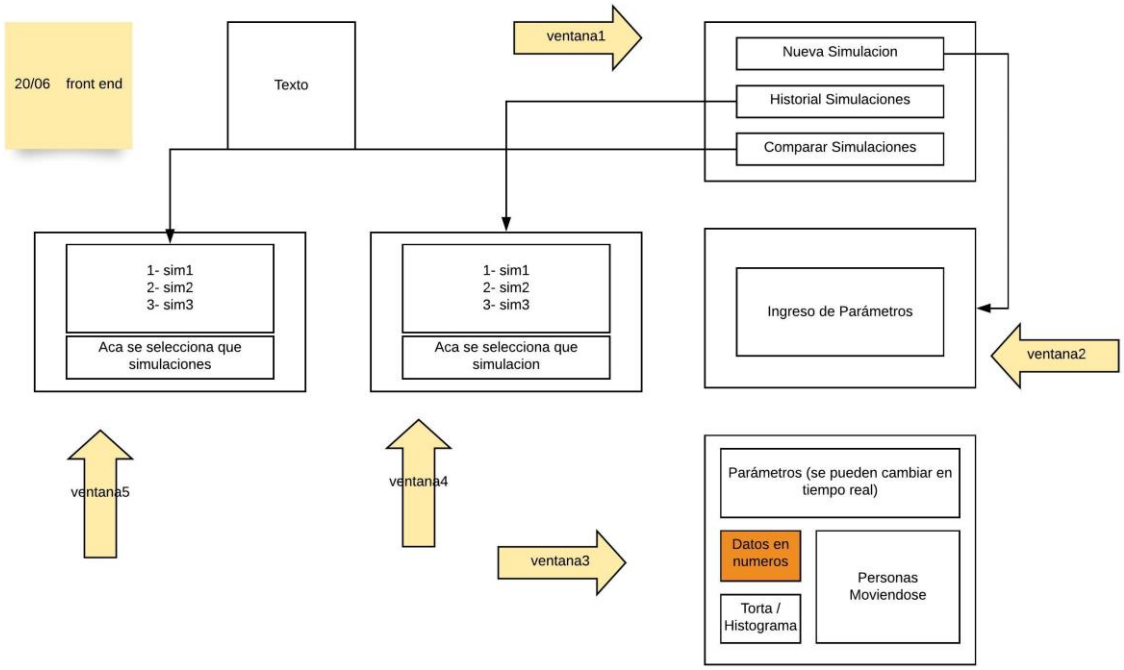



Se agrega un diagrama de contexto para poder definir los límites entre el sistema y su ambiente mostrando las entidades que interactúan con el.

**Diagrama de contexto**



# VISTA DE INTERFAZ GRAFICA



## CASOS DE PRUEBA DE INTEGRACION

Definimos pruebas de integración para poder verificar la correcta interacción entre la mayor cantidad de componentes del sistema.

1) a- El usuario inicia el programa y selecciona la opción “Simulación”.

   b- Ingresa los parámetros deseados y selecciona la visualización en tiempo real.

   c- Verifica que en pantalla se ve la simulación en tiempo real, sus estadísticas y su respectivo histograma.

2) a- El usuario inicia el programa y selecciona la opción “Simulación”

   b- Ingresa los parámetros deseados y selecciona la visualización en tiempo real.

   c- Una vez terminada la simulación, de vuelta en el Menú principal, selecciona la opción “Historial de Simulaciones”.

   d- Selecciona alguna de las simulaciones que hay en el historial.

   e- Verifica que se visualiza en pantalla los gráficos y estadísticas de la simulación ya corrida.

3) a- El usuario inicia el programa y selecciona la opción “Comparación”

   b- Selecciona las dos simulaciones a comparar.

   c- Verifica que se visualiza en pantalla los datos de ambas simulaciones a manera de comparación.

## HISTORIAL DE VERSIONES

| Versión | Comentario | Fecha |
|---|---|---|
| 1.0.0 | Version inicial para segunda entrega | 19 Junio 2020 |
| 1.0.2 | Actualizacion diagrama de clases | 23 Junio 2020 |

# Documento de Diseño

## Simulador de contagios

### Presentado a:

Ing. Martín M. Miceli
Ing. Martin F. Bustos Menas

### Presentado por:

Rueda Horacio
Severini Montanari Alejo
Vega Cuevas Silvia Jimena
Wortley Agustina Daniela

### Dia de presentacion:

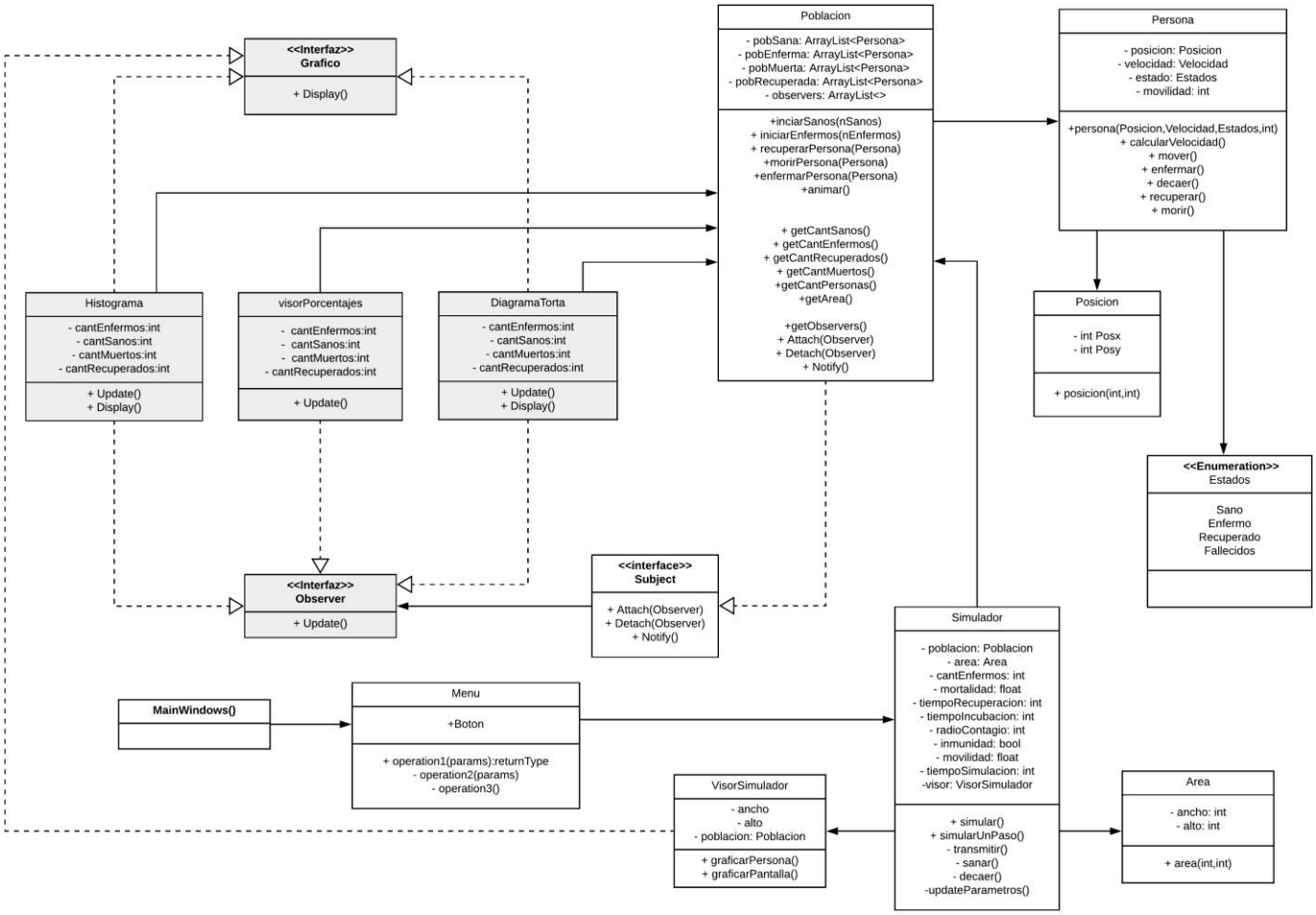
19 de Junio, 2020

# INTRODUCCION

En este documento se especificarán los diseños llevados a cabo para la realización del software, apoyándonos en distintos diagramas que nos serán de utilidad para ampliar el entendimiento de cómo llevar a cabo cada parte del SW.

En el diagrama de clases (simplificado) podemos ver la estructura del sistema de simulación con las clases del mismo sus atributos operaciones y relaciones entre los objetos.

Al final del documento puede observarse el [1]diagrama de clases completo

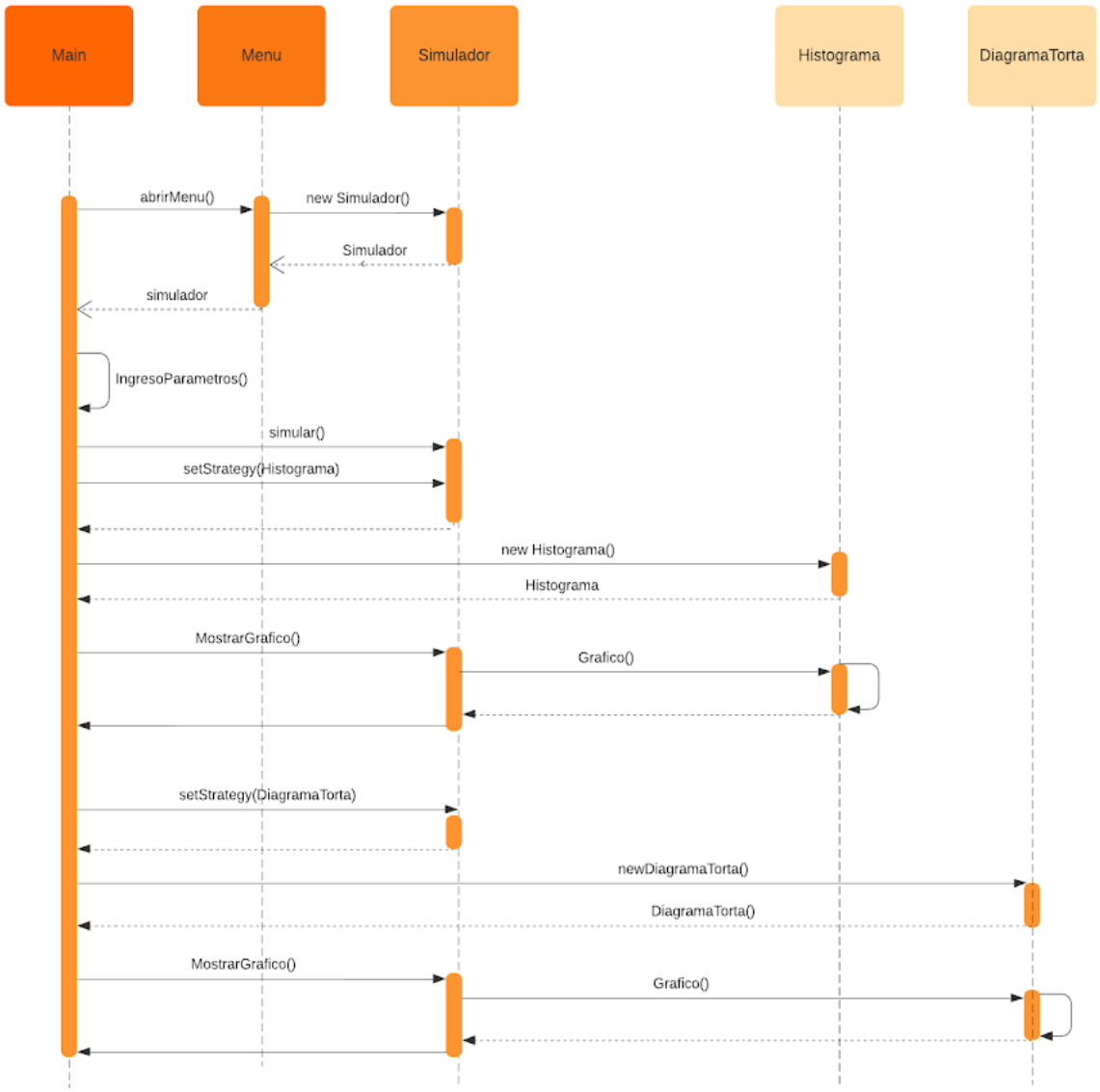


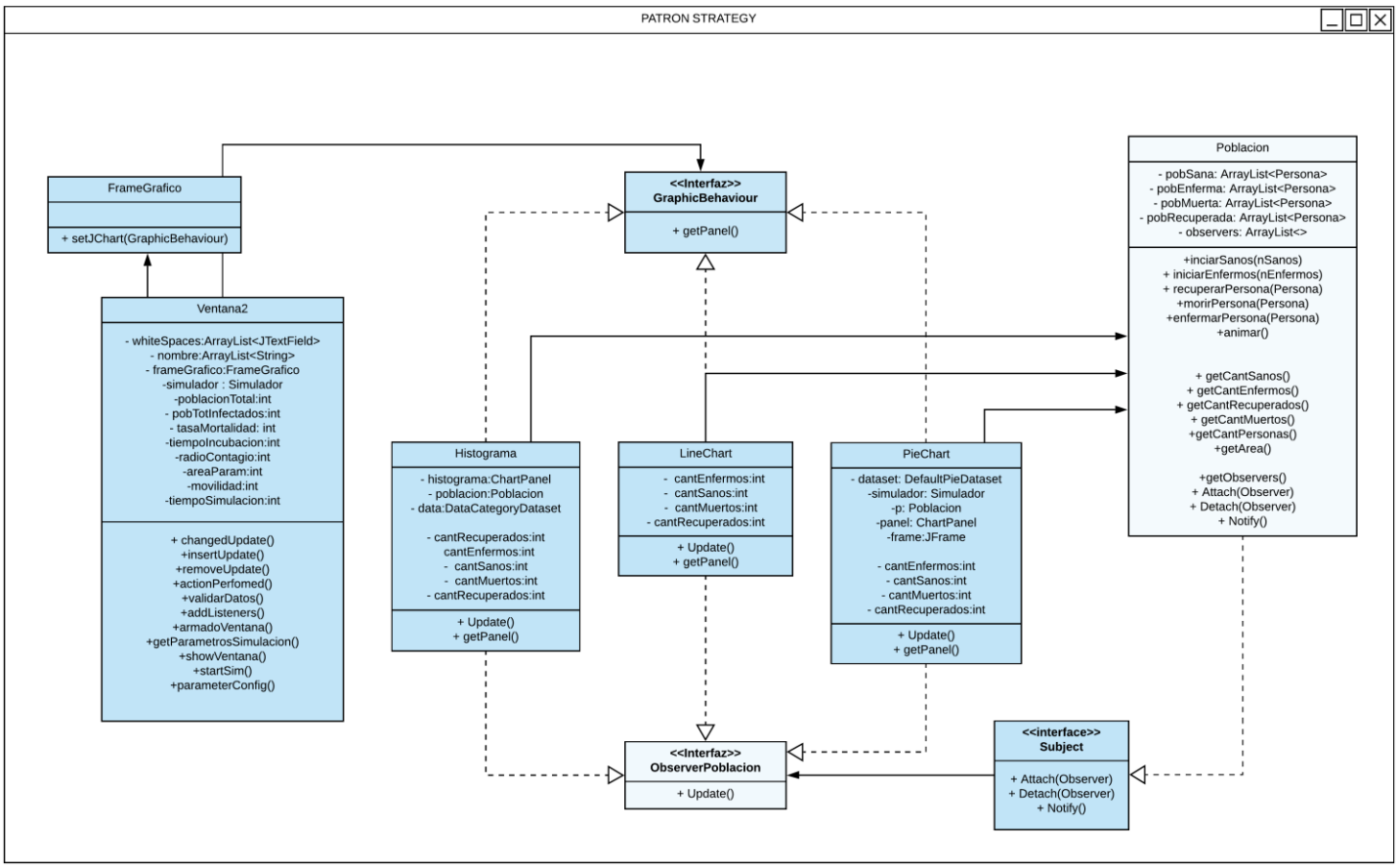
# PATRONES DE DISEÑO

**Diagramas de secuencia explicando la utilización de patrones de diseño**

## 1- Strategy

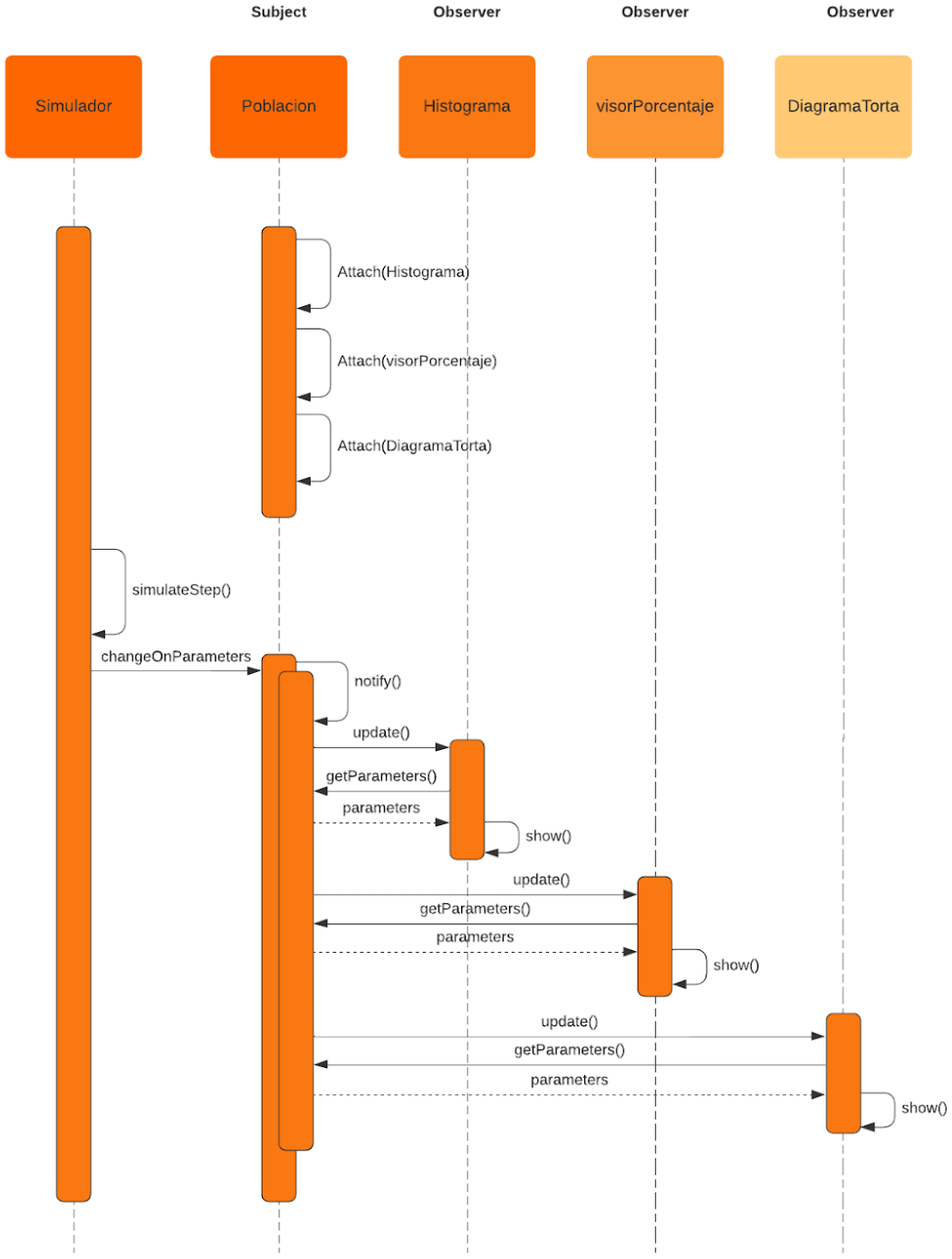
Mediante el patrón de Strategy evitamos que el código de los algoritmos se encuentren dentro de los sujetos, de esta manera no repetimos código, y además, podemos cambiar de forma dinámica el algoritmo que usa cada sujeto. En otras palabras, a través del patrón strategy el usuario puede modificar en tiempo real el modelo de gráfico a utilizar, ya sea: histograma, diagrama de torta y diagrama de lineas.

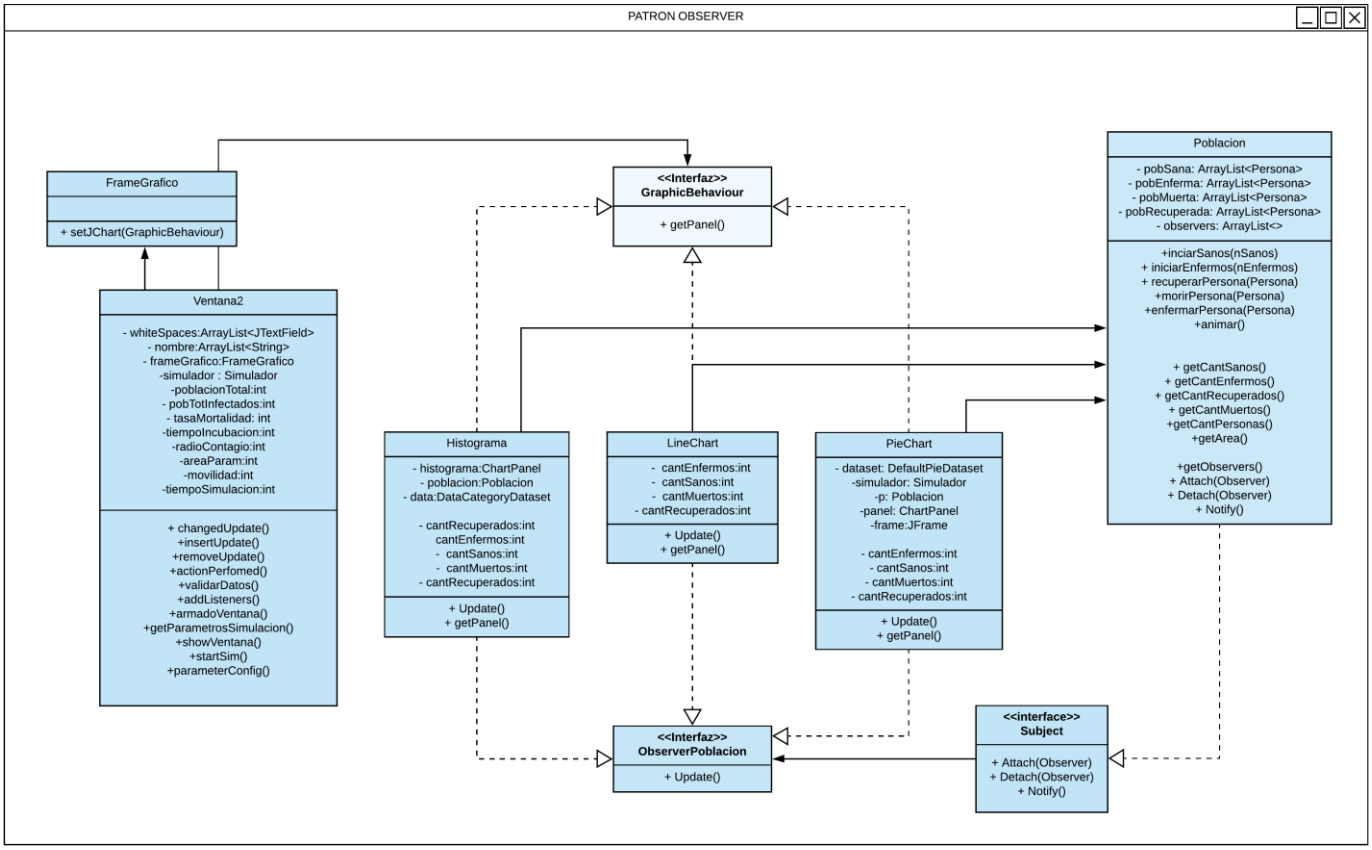



PATRON STRATEGY
FrameGrafico
+ setJChart(GraphicBehaviour)
<<Interfaz>>
GraphicBehaviour
+ getPanel()
Poblacion
- pobSana: ArrayList<Persona>
- pobEnferma: ArrayList<Persona>
- pobMuerta: ArrayList<Persona>
- pobRecuperada: ArrayList<Persona>
- observers: ArrayList<>
+inciarSanos(nSanos)
+ iniciarEnfermos(nEnfermos)
+ recuperarPersona(Persona)
+morirPersona(Persona)
+enfermarPersona(Persona)
+animar()
+ getCantSanos()
+ getCantEnfermos()
+ getCantRecuperados()
+ getCantMuertos()
+getCantPersonas()
+getArea()
+getObservers()
+ Attach(Observer)
+ Detach(Observer)
+ Notify()
Ventana2
- whiteSpaces:ArrayList<JTextField>
- nombre:ArrayList<String>
- frameGrafico:FrameGrafico
-simulador : Simulador
-poblacionTotal:int
- pobTotInfectados:int
- tasaMortalidad: int
-tiempoIncubacion:int
-radioContagio:int
-areaParam:int
-movilidad:int
-tiempoSimulacion:int
+ changedUpdate()
+insertUpdate()
+removeUpdate()
+actionPerfomed()
+validarDatos()
+addListeners()
+armadoVentana()
+getParametrosSimulacion()
+showVentana()
+startSim()
+parameterConfig()
Histograma
- histograma:ChartPanel
- poblacion:Poblacion
- data:DataCategoryDataset
- cantRecuperados:int
cantEnfermos:int
- cantSanos:int
- cantMuertos:int
- cantRecuperados:int
+ Update()
+ getPanel()
LineChart
- cantEnfermos:int
- cantSanos:int
- cantMuertos:int
- cantRecuperados:int
+ Update()
+ getPanel()
PieChart
- dataset: DefaultPieDataset
-simulador: Simulador
-p: Poblacion
-panel: ChartPanel
-frame:JFrame
- cantEnfermos:int
- cantSanos:int
- cantMuertos:int
- cantRecuperados:int
+ Update()
+ getPanel()
<<Interfaz>>
ObserverPoblacion
+ Update()
<<interface>>
Subject
+ Attach(Observer)
+ Detach(Observer)
+ Notify()

## 2- Observer

Este patrón se utiliza para que los visores implementados en el programa (“Histograma”, “VisorPorcentaje”,”DiagramaTorta”, “Grafico de Lineas”) se actualicen simultáneamente, cada vez que se actualiza el sujeto observado “Población”. De esta manera disminuimos la dependencia entre el sujeto y el observador, generando menos acoplamiento en el sistema.




PATRON OBSERVER
FrameGrafico
+ setJChart(GraphicBehaviour)
<<Interfaz>>
GraphicBehaviour
+ getPanel()
Poblacion
- pobSana: ArrayList<Persona>
- pobEnferma: ArrayList<Persona>
- pobMuerta: ArrayList<Persona>
- pobRecuperada: ArrayList<Persona>
- observers: ArrayList<>
+inciarSanos(nSanos)
+ iniciarEnfermos(nEnfermos)
+ recuperarPersona(Persona)
+morirPersona(Persona)
+enfermarPersona(Persona)
+animar()
+ getCantSanos()
+ getCantEnfermos()
+ getCantRecuperados()
+ getCantMuertos()
+getCantPersonas()
+getArea()
+getObservers()
+ Attach(Observer)
+ Detach(Observer)
+ Notify()
Ventana2
- whiteSpaces:ArrayList<JTextField>
- nombre:ArrayList<String>
- frameGrafico:FrameGrafico
-simulador : Simulador
-poblacionTotal:int
- pobTotInfectados:int
- tasaMortalidad: int
-tiempoIncubacion:int
-radioContagio:int
-areaParam:int
-movilidad:int
-tiempoSimulacion:int
+ changedUpdate()
+insertUpdate()
+removeUpdate()
+actionPerfomed()
+validarDatos()
+addListeners()
+armadoVentana()
+getParametrosSimulacion()
+showVentana()
+startSim()
+parameterConfig()
Histograma
- histograma:ChartPanel
- poblacion:Poblacion
- data:DataCategoryDataset
- cantRecuperados:int
cantEnfermos:int
- cantSanos:int
- cantMuertos:int
- cantRecuperados:int
+ Update()
+ getPanel()
LineChart
- cantEnfermos:int
- cantSanos:int
- cantMuertos:int
- cantRecuperados:int
+ Update()
+ getPanel()
PieChart
- dataset: DefaultPieDataset
-simulador: Simulador
-p: Poblacion
-panel: ChartPanel
-frame:JFrame
- cantEnfermos:int
- cantSanos:int
- cantMuertos:int
- cantRecuperados:int
+ Update()
+ getPanel()
<<Interfaz>>
ObserverPoblacion
+ Update()
<<interface>>
Subject
+ Attach(Observer)
+ Detach(Observer)
+ Notify()

# MATRIZ DE TRAZABILIDAD

## Clases y métodos contra unit test

| clase | UT |
|---|---|
| **Persona** | U1 al U11 |
| **Simulador** | U15 a U19 |
| **Poblacion** | U20 a U26 |
| **Posicion** | U12 a U14 |

# UNIT TEST

Se han generado Unit Tests para las clases del software utilizando eclipse IDE con la herramienta JUnit. Deben de implementarse para asegurar el correcto funcionamiento del programa. Los tests correrán automáticamente al subir código al repositorio, por medio del sistema de integración continua (CircleCI, ver CMP).

| ID | Nombre | Descripción | PassFail |
|---|---|---|---|
| U1 | testVelocidadCalculaaDebeSerCero | Testea que la velocidad en “x” y en “y” se inicializan en 0. | pass |
| U2 | testVelocidadCalculadaDebeCumplirLimites | Testea que la velocidad de la persona no sea mayor a la movilidad seteada. | pass |
| U3 | testMover | Testea que la persona se mueva en ambos ejes. | pass |
| U4 | testNoExcederLimitesArea | Testea movimientos fuera del área establecida. | pass |
| U5 | testEnfermar | Testea que el estado sea efectivamente “enfermo”. | fail |
| U6 | testEvolucionarEnfermedad | Testea que la enfermedad evolucione en una unidad (que falte menos para su recuperacion). | pass |
| U7 | testEvolucionarEnfermedadNoDeberiaSerNegativo | Testea que el tiempo faltante para la recuperación no sea negativo. | pass |
| U8 | testFinEnfermedad | Testea que el el fin de la enfermedad sea efectivo. | fail |

| | | | |
|---|---|---|---|
| U9 | testRecuperar | Testea que la persona cambie a estado recuperado | pass |
| U10 | testMorir | Testea que la persona muera. | pass |
| U11 | testDebeMorir | Se establece una probabilidad de 100% de muerte y comprueba que asi sea, tambien se prueba con 0% de muertes y debe morir en ninguna como resultado. | pass |
| U12 | testPosicionArea | Testea que la persona no se pase del área establecida. | pass |
| U13 | testMover | Testea que la distancia concuerde con la velocidad a la que se movieron desde el origen las personas sea la adecuada. | pass |
| U14 | testMoverNoDebeExcederLimites | Testea que al moverse la persona no exceda el límite del área | pass |
| U15 | testTodosDeberianMorir | Asigna una mortalidad de 100% y testea que todas las personas enfermas mueran | pass |
| U16 | testTodosDeberianRecuperarse | Testea que con una mortalidad del 0% toda la población se recupera. | fail |
| U17 | testTransmitirATodos | Testea que si se les asigna a las personas un radio de contagio mayor al área, entonces toda la población se contagia. | pass |
| U18 | testTransmitirANadie | Testea que si dos personas se encuentran a una distancia mayor que sus radios de contagio y no poseen movilidad, entonces no se contagian. | pass |
| U19 | testEstanCerca | Testea que dos personas se encuentren a una distancia menor que sus radios de contagio. | pass |
| U20 | testIniciarSanas | Testea la inicialización de la porción de la población sana. | pass |
| U21 | testIniciarEnfermos | Testea la inicialización de la porción de la población sana. | pass |
| U22 | testEnfermarPersona | Testea que las personas pasen del estado “Sano” a “Enfermo”. | pass |
| U23 | testMorirPersona | Testea que las personas pasen del | pass |

| | | | |
|---|---|---|---|
| | | estado “Enfermo” a “Muerto”. | |
| U24 | testRecuperarPersona | Testea que las personas pasen del estado “Enfermo” a “Recuperado”. | pass |
| U25 | testSetMovilidad | Testea que la movilidad se asigne correctamente. | pass |
| U26 | testSetMortalidad | Testea que la mortalidad se asigne correctamente. | pass |

## HISTORIAL DE VERSIONES

| Versión | Comentario | Fecha |
|---|---|---|
| 1.0.0 | Versión inicial para segunda entrega | 19 Junio 2020 |
| 1.0.2 | Actualización diagrama de clases<br>Incorporacion de diagramas de clases en Strategy y Observer | 23 Junio 2020 |

# [1] Diagrama de clases completo