Pruebas

Introducción

Durante todo el proyecto se implementará un plan de pruebas unitarias, las cuales buscan verificar la correcta funcionalidad de las clases, y código del proyecto. Debido a la naturaleza del proyecto, muchas clases resultan casi imposible de testear mediante un plan de pruebas unitarias, por lo cual requieren pruebas de integración, y su complejidad, puede llegar a ser demasiado alta, por lo cual se procede a validar y comprobar el correcto funcionamiento del proyecto, mediante pruebas de usuarios. Entregándole una versión a jugable del proyecto a diferentes usuarios, no familiarizados con el proyecto, para que prueben todas las funcionalidades, y verifiquen su correcto funcionamiento.

Alcance

El plan de pruebas tiene dos componentes, el primero a nivel de código y el otro a través del testeo de funcionalidad, con los usuarios.

Pruebas por código: Para el fin de realizar estas pruebas se hace uso del framework de pruebas unitarias Junit.

En la clase constantesTest:

> **testRutas y testLados:**
>
> Se testean las rutas y constantes más importantes, con el fin de que sus valores sean correctos. Cambios o modificaciones de forma errónea a las variables dentro de esta clase, son las causantes de la mayor parte de errores durante el desarrollo.
>
> constantes.RUTA_MAPA, se espera que su valor sea: “/mapas/mapa1", en caso contrario la prueba no pasara.
>
> Constantes.RUTA_ICONO_RATON, se espera que su valor sea: "/imagenes/iconos/iconoCursor.png", en caso contrario la prueba no pasara.

Constantes.RUTA_PERSONAJE, se espera que su valor sea: "/imagenes/hojas_Personajes/Santana.png", en caso contrario la prueba no pasara.

Constantes.RUTA_ZOMBIE, se espera que su valor sea: /imagenes/hojas_de_enemigos/”, en caso contrario la prueba no pasara.

Constantes.RUTA_ICONO_VENTANA, se espera que su valor sea: "/imagenes/iconos/iconV.png", en caso contrario la prueba no pasara.

Constantes.RUTA_LOGOTIPO, se espera que su valor sea: "/imagenes/iconos/logo.png", en caso contrario la prueba no pasara.

Constantes.RUTA_OBJETOS, se espera que su valor sea: "/imagenes/hojas_de_objetos/1.png", en caso contrario la prueba no pasara.

Constantes.FUENTE_PIXEL, se espera que su valor sea "/fuentes/Crumbled-Pixels.ttf", en caso contrario la prueba no pasara.

Constantes.LADO_SPRITE, se espera que su valor sea: 32, en caso contrario la prueba no pasara.

Constantes.LADO_TILE, se espera que su valor sea: 32, en caso contrario la prueba no pasara.

En la clase: **CargadorRecursos:**

Se hace uso de la clase ComparadorRecursos, la cual posee los métodos necesarios, para determinar la similitud entre dos recursos multimedia, y decidir si son iguales o no, según cada tipo(imagen, sonido, texto).

testCargarImagenCompatibleOpacaMismaImagen:

Se hace dos llamados al método cargarImagenCompatibleOpaca() del cargador de recursos y se pasa como argumento la ruta a la misma imagen las dos veces, luego mediante el comparador de recursos se verifica que ambas imágenes sean 100% iguales, debido a que poseen la misma ruta, en caso contrario el test debe fallar

testCargarImagenCompatibleOpacaDiferenteImagen:

Se hace dos llamados al método cargarImagenCompatibleOpaca() del cargador de recursos y se pasa como argumento dos rutas de imágenes distintas, las dos veces, luego mediante el comparador de recursos se verifica que ambas imágenes no sean 100% iguales, ya que sus rutas son distintas, en caso de ser iguales. El test fallara.

testCargarImagenCompatibleTranslucidaDiferenteImagen:

Se hace dos llamados al método cargarImagenCompatibleTranslucida() del cargador de recursos y se pasa como argumento dos rutas de imágenes distintas, las dos veces, luego mediante el comparador de recursos se verifica que ambas imágenes no sean 100% iguales, ya que sus rutas son distintas, en caso de ser iguales. El test fallara.

testCargarImagenCompatibleOpacaMismaImagen:

Se hace dos llamados al método cargarImagenCompatibleOpaca() del cargador de recursos y se pasa como argumento la ruta a la misma imagen las dos veces, luego mediante el comparador de recursos se verifica que ambas imágenes sean 100% iguales, debido a que poseen la misma ruta, en caso contrario el test debe fallar

testLeerArchivoTexto:

se realiza un llamado al método leerArchivoTexto() del cargdor de recursos, y se pasa como argumento la ruta al archivo .json con la información del mapa del juego. Luego se verifica que el método en verdad regresa un objeto no vacio, en caso de hacerlo, el método deberá fallar.

testCargarFuente:

se realiza un llamado al método cargarFuente() del cargdor de recursos, y se pasa como argumento la ruta hacia un archivo de tipo fuente Luego se verifica que el método en verdad regresa un objeto no vacio, en caso de hacerlo, el método deberá fallar.

testCargarFuenteMismaFuente:

se realizan dos llamados al método cargarFuente() del cargdor de recursos, y se pasa como argumento la ruta hacia un archivo de tipo fuente Luego se verifica que el método retorne el mismo valor e n ambos cssos, debido a que es el mismo archivo de fuentes, en caso contrario, el test fallara.

testCargarSonido:

se realiza un llamado al método cargarSonido() del cargdor de recursos, y se pasa como argumento la ruta hacia un archivo de tipo .wav Luego se verifica que el método en verdad regresa un objeto no vacio, en caso de hacerlo, el método deberá fallar.

testCargarSonidoMismosSonidos:

se realizan dos llamados al método cargarSonido() del cargdor de recursos, y se pasa como argumento la ruta hacia un archivo de tipo fuente Luego se verifica que el método retorne el mismo valor en ambos cssos, debido a que es el mismo archivo de sonido, en caso contrario, el test fallara.

testCargarDiferentesSonidos:

Se hace dos llamados al método cargarSonido() del cargador de recursos y se pasa como argumento dos rutas de sonidos distintos, las dos veces, luego mediante el comparador de recursos se verifica que ambos sonidos no sean 100% iguales, ya que sus rutas son distintas, en caso de ser iguales. El test fallara.

En la clase HojaSprite:

Se verifica el correcto funcionamiento de la clase hojaSprite y Sprite.

testCargarHoja:

Se verifica que se pueda crear correctamente una nueva instancia de la clase HojaSprite, enviándole como argumento la ruta hacia un imagen de hoja de sprites válidas, el test debe comprobar que la instancia nueva, no sea nula, y que ambas imágenes sean distintas, lo que significa que el constructor genere un objeto distinto cada vez que se invoca. En caso contrario el test debe fallar

testCargarSprite:

Se verifica que se logre extraer un Sprite en especifico de una hoja dada y ya instanciada, esto se hace mediante el método getSprite de la clase HojaSprite. Se verifica que el método no devuelva un objeto nulo, y que dado dos hojas distintas de sprites, los sprites obtenidos sean distintos. En caso contrario el test debe fallar

testCargarImagenSprite:

Se verifica que se logre extraer la imagen de un Sprite en especifico de una Sprite dado y ya instanciado, esto se hace mediante el método getImagen de la clase clase Sprite. Se verifica que el método no devuelva un objeto nulo, y que dado Sprites distintos, las imágenes obtenidas sean distintas. En caso contrario el test debe fallar.

Objetivo

Se creará y ejecutará un set de pruebas unitarias en clases fundamentales, esto con el objetivo de comprobar su correcto funcionamiento y eliminar posibles errores. En aquellas las cuales no sea posible implementar pruebas unitarias, se comprobará su funcionalidad a través de pruebas de usuario, los cuales sumarán validez y confianza acerca del buen funcionamiento del producto.

Plan de prueba

Criterios de aceptación

- No se acepta documentación con errores ortográficos.
- No se acepta documentación con un grado de completitud menor al 70% con respecto a los issues del producto.
- No se aceptan las pruebas unitarias que no cumplan los objetivos esperados.
- No se aceptan las pruebas de integración en las que alguna de las pruebas unitarias falle o como resultado final, no se cumpla la funcionalidad establecida en el requerimiento.

- No se aceptan las pruebas de regresión las cuales luego de corregir/agregar una funcionalidad arrojen un resultado distinto.

Para cada uno de estos ítems, cabe destacar que se acepta el caso contrario al mismo, por ejemplo “Se acepta documentación sin errores ortográficos”.

Equipo de Pruebas

La asignación y definición de los roles con sus respectivas tareas y responsabilidades para los integrantes del equipo de pruebas será:

Nota: Debido al bajo personal del equipo, el cual solo cuenta con 5 integrantes, existirán varios roles que deberán ser cumplidos por una misma persona, en este caso el encargado de pruebas, Dylan Cantillo.

Test Lead: Dylan Cantillo es la persona responsable de la supervisión de las pruebas en el proyecto. También es responsable de los procesos utilizados para garantizar la calidad de la entrega.

- Coordinación del equipo de pruebas.
- Representación del equipo de pruebas en varias reuniones.
- La generación de diversos informes.
- La identificación de riesgos, análisis y mitigación.
- Presentar Informes sobre cambios al Test Manager sobre el progreso de la prueba.
- Gestión del repositorio.

Test Manager: Dylan Cantilllo.

- Escribe el plan de pruebas.
- Realiza el mantenimiento del plan de prueba.
- Monitorea el progreso de las prueba.
- Toma decisiones sobre la entrada y salida de una prueba.
- Toma decisiones sobre la suspensión / reanudación de las pruebas.
- Toma decisiones sobre cambios en la estrategia de prueba / plan de pruebas.

Business Analyst: Luis Felipe Evillla.

- Identificar escenarios adecuados para el test de aceptación de usuario.
- Crear casos de pruebas para los test de aceptación de usuario.
- Realizar pruebas funcionales.
- Trabajar con el cliente para definir los requerimientos del negocio.

Developers: Mariana Oquendo.

- Realizar pruebas unitarias.
- Ejecutar las pruebas unitarias.
- Revisiones de código
- Crea los casos y los datos de prueba del sistema.
- Administrar el tiempo a la programación del proyecto para que todas las tareas asignadas se completen a tiempo.

Tester: Dylan Cantillo.

- Ejecutar el plan de pruebas.
- Buscar, informar y seguir errores descubiertos durante las pruebas del sistema.
- Analizar los resultados.
- Realizar Control de la conformidad con:los estándares de codificación, Ayuda (online, manuales),Pruebas de Navegación, la experiencia del usuario y pruebas de usabilidad, pruebas de rendimiento, las pruebas de compatibilidad de plataforma, la prueba de escalabilidad, copia de seguridad y restauración, las pruebas de instalación.

Suspensión y reanudación

Cuando se lleve a cabo la ejecución de un set de pruebas y alguno de los elementos de prueba falle (no pase la prueba), se notificará a los responsables del desarrollo de dicho elemento y se suspenderá la prueba hasta tanto se solucionen los inconvenientes. Al momento de reanudar las pruebas suspendidas se deberá evaluar, según la complejidad del caso, la necesidad de ejecutar el set de pruebas nuevamente o reanudar desde el punto en el cual se había suspendido.

Entregables de prueba

Los distintos entregables de las pruebas son:

- Especificación del diseño de pruebas.
- Especificación de casos de prueba.
- Especificación del procedimiento de prueba.
- Resumen del informe de prueba.
- Los datos de entrada y salida de las distintas pruebas.
- Logs de errores.

Necesidades ambientales

Serán necesarios para un correcto aislamiento de todas las fases en las que puede encontrarse cada versión del sistema, contar con distintos entornos para la ejecución y pruebas del sistema

Desarrollo

Entorno altamente volátil. Los datos serán ingresados por los mismos desarrolladores de acuerdo a las funcionalidades que vayan desarrollando. Es el ambiente de pruebas que utilizarán los desarrolladores

Pruebas

Ambiente de Prueba:

- Un BackUp completo antes de la ejecución de cualquier caso de prueba.
- Al finalizar una jornada de pruebas se realiza un backup incremental.
- Al detectar errores de impacto “Crítico” en la aplicación, se realizará backup completo e inmediato de la base de datos y del sitio de la aplicación, asociándolo a un documento descriptivo del escenario y del caso de prueba que se estaba llevando a cabo. Esto con el fin de poder reproducir estos errores posteriormente sobre un ambiente controlado y facilite la detección de la causa y la corrección del mismo.
- Todas las pruebas se realizarán con un sistema operativo Windows 10.
- El sistema debe contar con Java jdk 8 o superior.
- EL sistema debe tener instalado una versión de la máquina de virtual de Java estable y actualizada.

Producción

Entorno de producción en el que se encontrarán los datos reales y ejecutables estables. Excede el entorno de pruebas.

Aspectos de ejecución de las pruebas

Aspectos de entrada y salida

Plan de Prueba, Aspectos de Entrada

Una vez desarrollado cada uno de los módulos del aplicativo se comenzará a realizar el set de pruebas. Set de pruebas completo y claro. Claridad en el procedimiento para el desarrollo de las pruebas. Tener un entorno de pruebas adecuado. Toda la documentación requerida para la realización de las pruebas debe estar disponible.

Plan de Prueba, Aspectos de Salida

El sistemas será sometido a 3 ciclos de pruebas: 1)Pruebas a cada módulo, 2) Pruebas luego de la integración entre módulos, y 3) Pruebas después de corregir las fallas del segundo ciclo, este último con el fin de asegurar que la corrección de errores no haya generados unos nuevos.

Definición del Entorno de Pruebas

Sistema operativo Windows o distribución Linux.	Windows 7 o superiores. Distros Linux: -Debian/Ubuntu/Kubuntu	
Java jdk	8.0 o superior	
Java Virtual Machine	1.6 o superior	
Netbeans IDE	8.0	

Conjunto de datos		Mínimamente se definirán un conjunto de datos que pongan a prueba los casos bordes.

Definir el Alcance y la Estrategia de Pruebas

En esta tarea se especifican y justifican de los niveles de pruebas a realizar, así como el marco general de planificación de cada nivel de prueba, según el siguiente esquema:

- Definición de los perfiles implicados en los distintos niveles de prueba.
- Planificación temporal.
- Criterios de verificación y aceptación de cada nivel de prueba.
- Definición, generación y mantenimiento de verificaciones y casos de prueba.
- Análisis y evaluación de los resultados de cada nivel de prueba.
- Productos a entregar como resultado de la ejecución de las pruebas.

Registro de pruebas

Descripción

El registro de pruebas es un mecanismo empleado para tener un control detallado sobre las pruebas realizadas en el sistema almacenando datos importantes para: resolución de problemas, análisis de los procedimientos de evaluar el proyecto a medida que se va construyendo, por lo tanto se hace necesario llevar a cabo, en paralelo al proceso de desarrollo, un proceso de evaluación o comprobación de los distintos productos o modelos que se van generando.

Entradas del registro

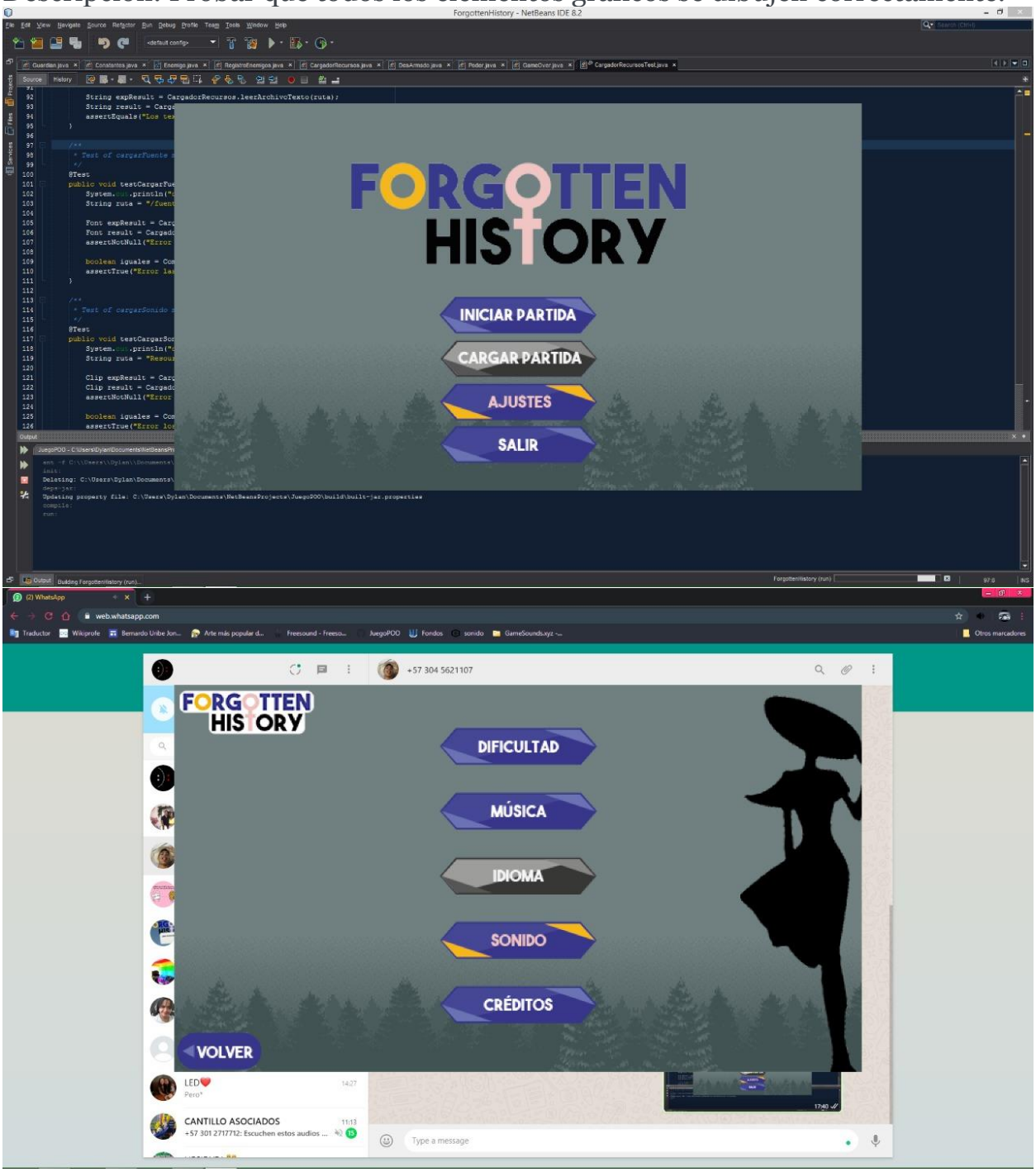
Ejecuciones: Se identificarán las ejecuciones realizadas sobre el sistema. Se deberá indicar el número de ejecución de la prueba, la fecha de la ejecución de la misma y algún identificador que identifique unívocamente la prueba.

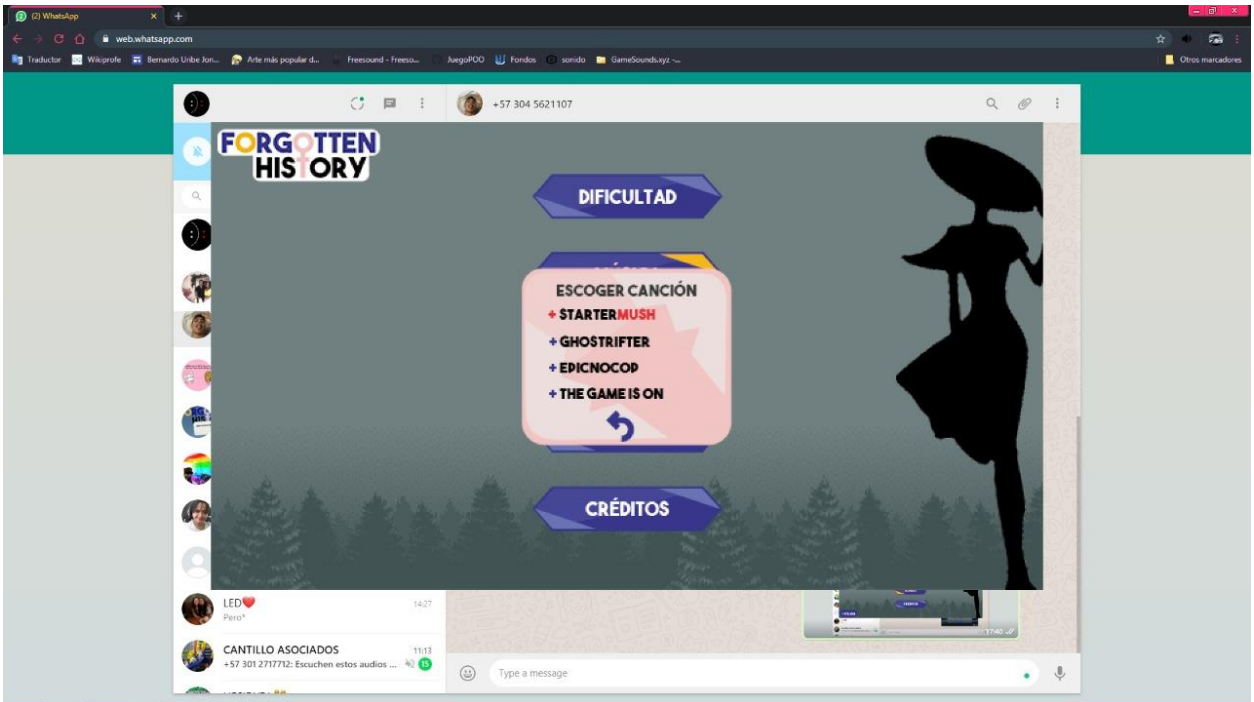
Fecha de prueba: 6/06/2020 10:00 a.m

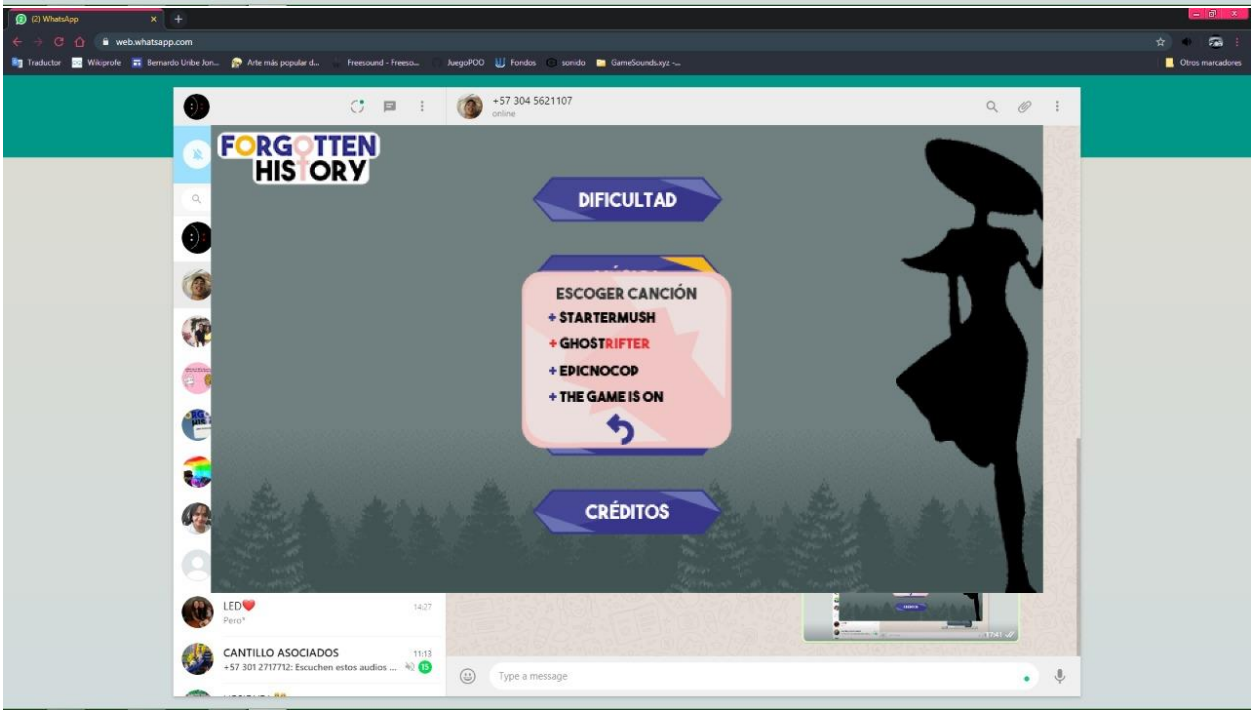
Descripción: Probar la correcta integración de todos los elementos entre si, y comprobando que no existan errores o bugs.

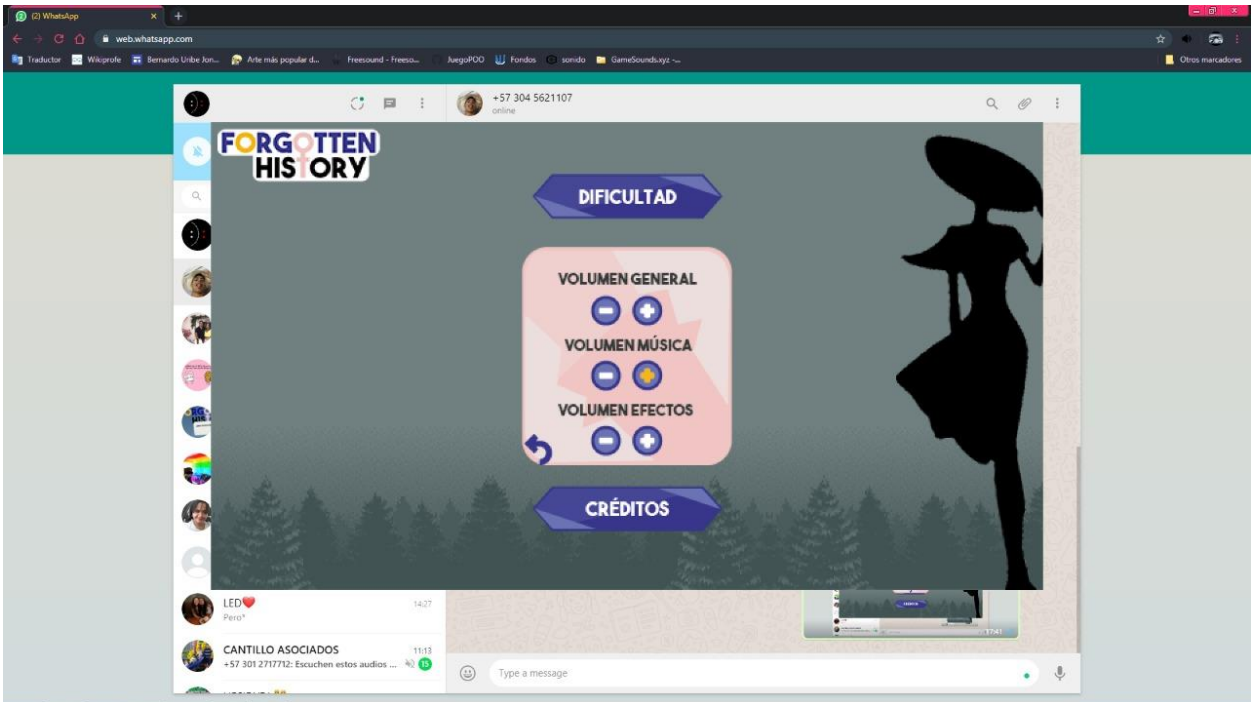
Fecha de prueba: 5/06/2020 9:30 p.m

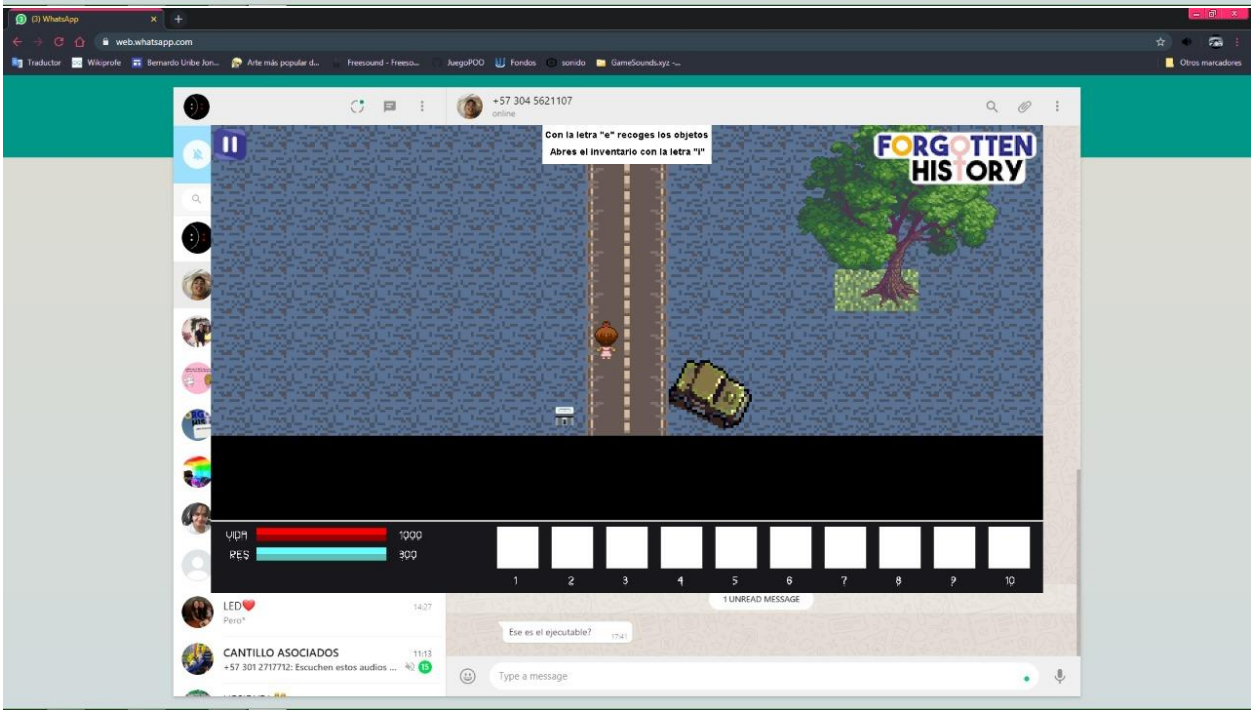
Descripción: Probar que todos los elementos gráficos se dibujen correctamente.

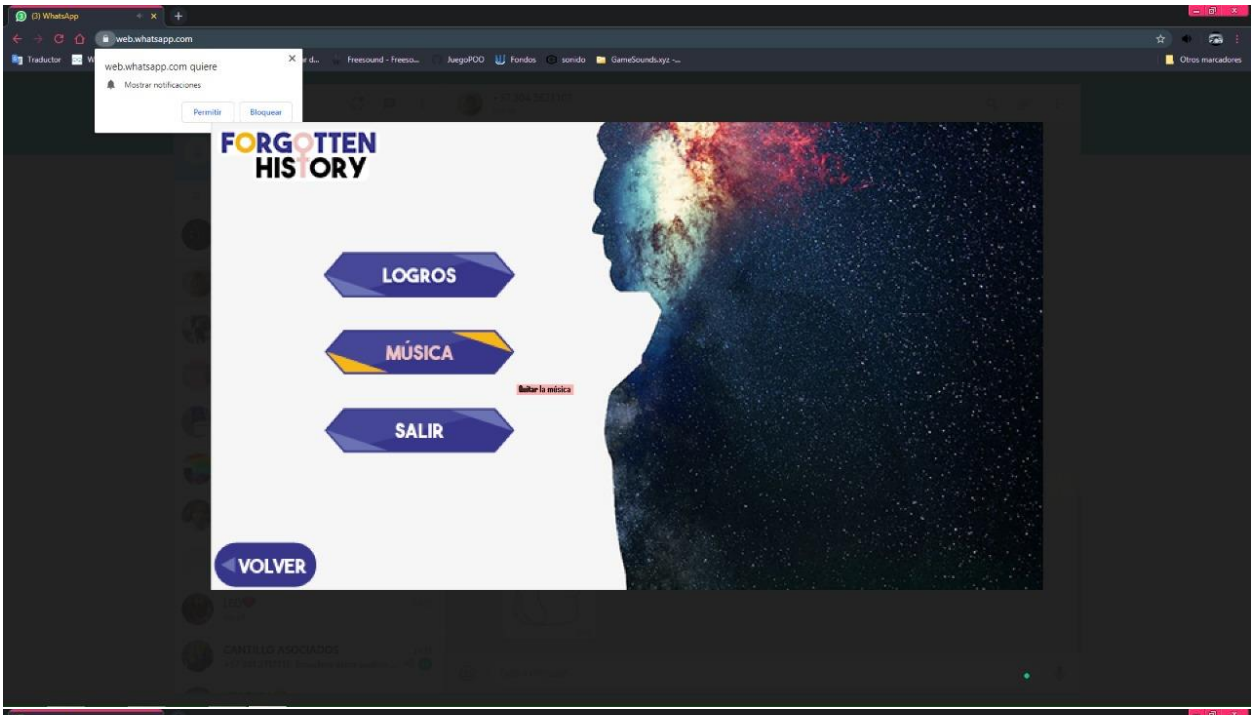


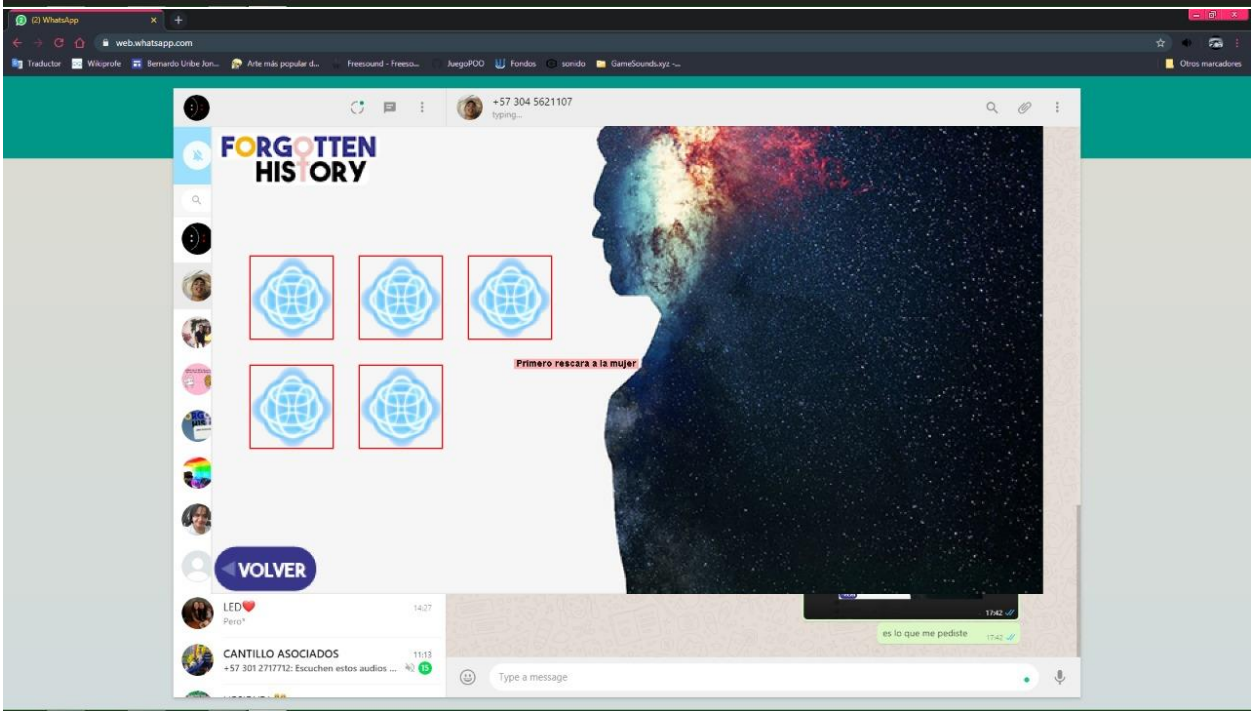

FORGOTTEN HISTORY
DIFICULTAD
ESCOGER CANCIÓN
+ STARTERMUSH
+ GHOSTRIFTER
+ EPICNOCOP
+ THE GAME IS ON
CRÉDITOS

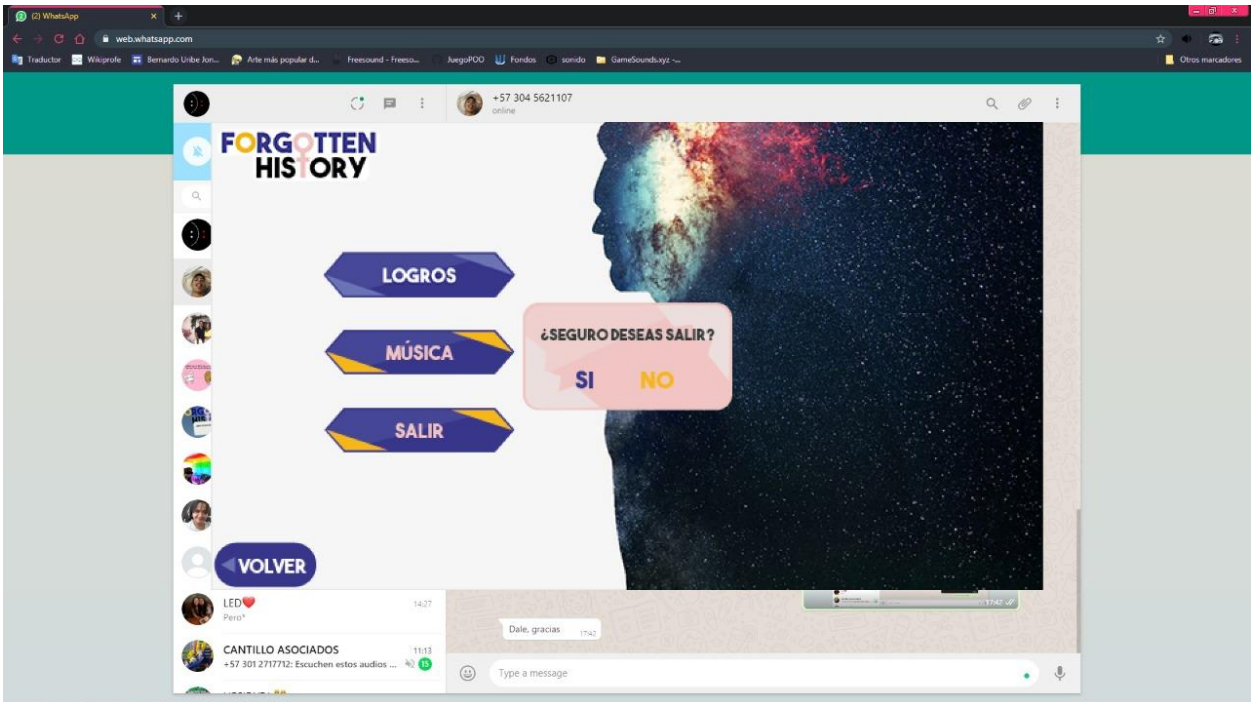

FORGOTTEN HISTORY
DIFICULTAD
ESCOGER CANCIÓN
+ STARTERMUSH
+ GHOSTRIFTER
+ EPICNOCOP
+ THE GAME IS ON
CRÉDITOS

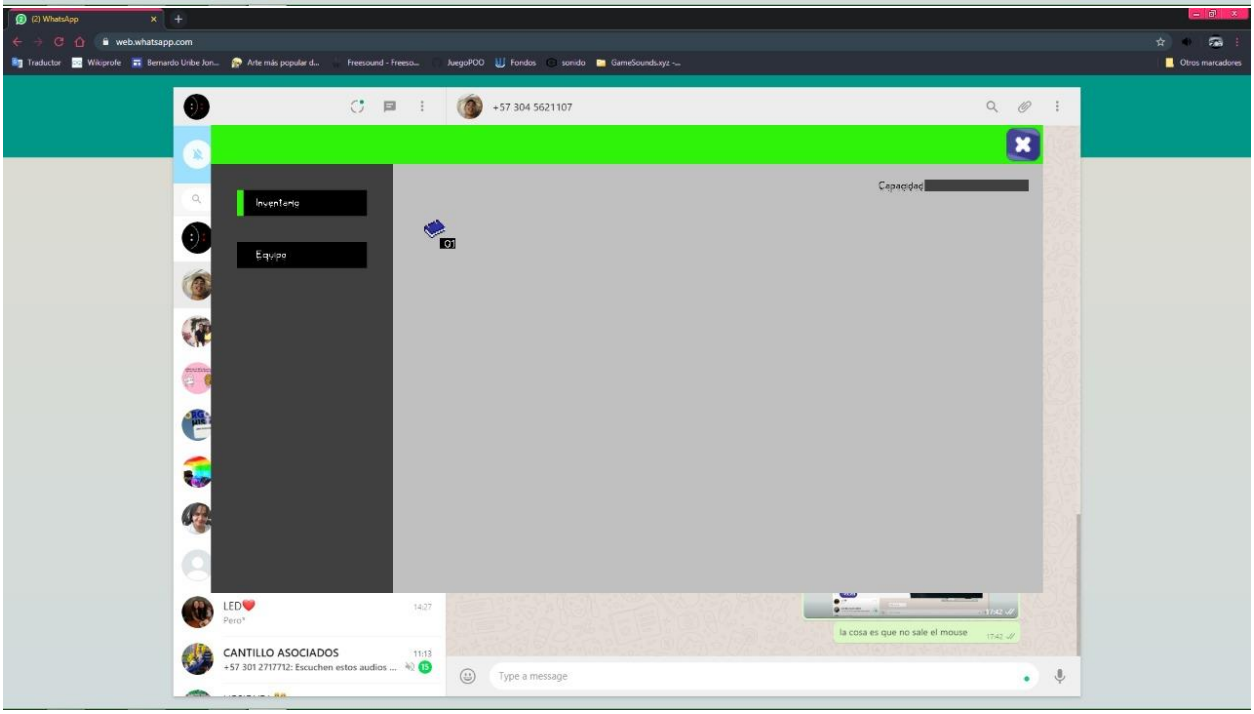

FORGOTTEN HISTORY
DIFICULTAD
VOLUMEN GENERAL
VOLUMEN MÚSICA
VOLUMEN EFECTOS
CRÉDITOS

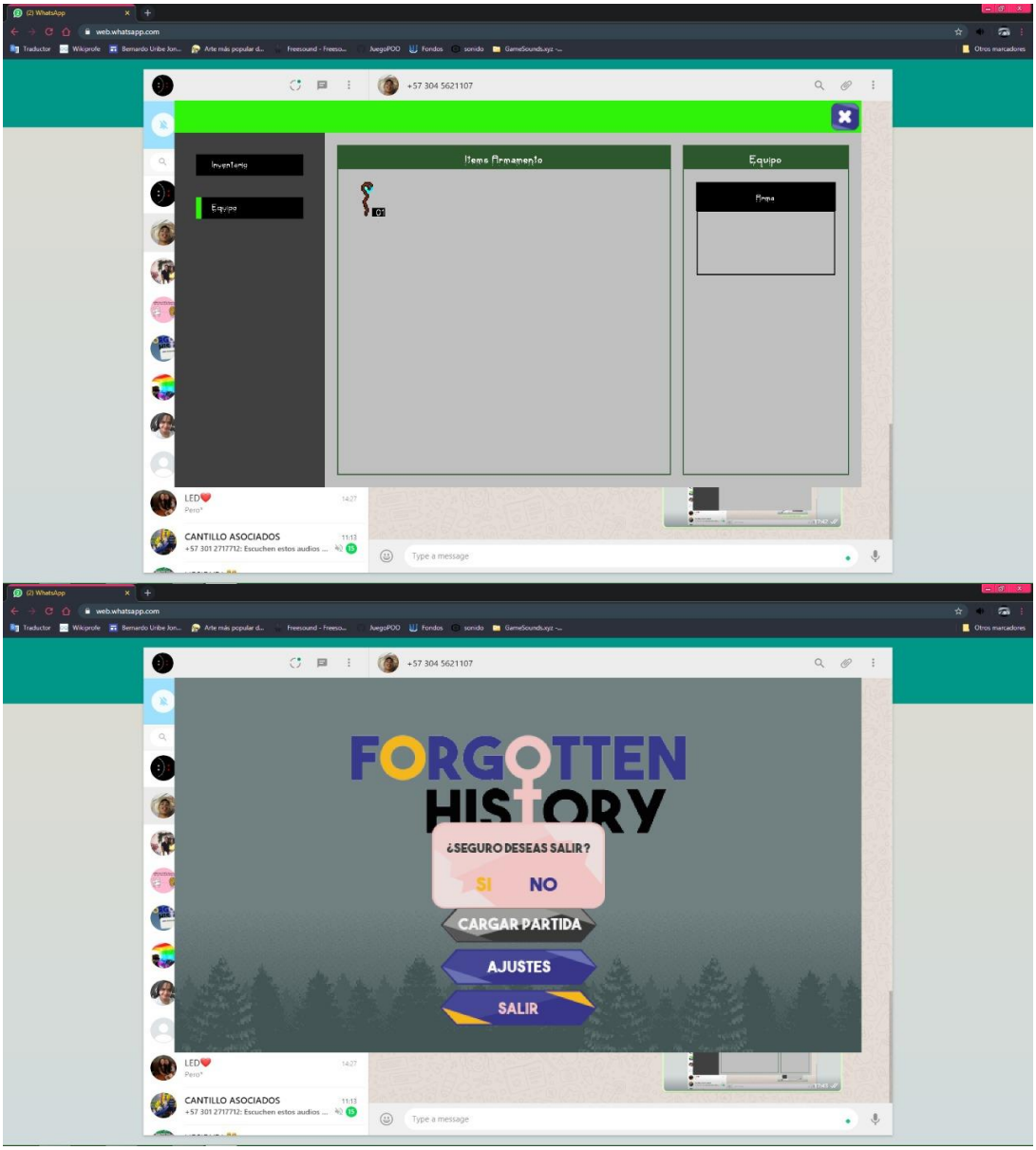

Con la letra "e" recoges los objetos
Abres el inventario con la letra "i"
FORGOTTEN HISTORY
VIDA
RES
1000
300
1 UNREAD MESSAGE
Ese es el ejecutable?


web.whatsapp.com quiere
Mostrar notificaciones
Permitir
Bloquear
FORGOTTEN HISTORY
LOGROS
MÚSICA
Saltar la música
SALIR
VOLVER


web.whatsapp.com
+57 304 5621107
typing...
FORGOTTEN HISTORY
Primero rescara a la mujer
VOLVER
LED
Pero*
14:27
CANTILLO ASOCIADOS
11:13
+57 301 2717712: Escuchen estos audios ...
es lo que me pediste
Type a message


web.whatsapp.com
+57 304 5621107
online
FORGOTTEN HISTORY
LOGROS
MÚSICA
SALIR
¿SEGURO DESEAS SALIR?
SI
NO
VOLVER
LED
Pero*
14:27
CANTILLO ASOCIADOS
11:13
+57 301 2717712: Escuchen estos audios ...
Dale, gracias
17:42
Type a message


web.whatsapp.com
+57 304 5621107
Inventario
Equipo
Capacidad
LED
Pero*
14:27
CANTILLO ASOCIADOS
11:13
+57 301 2717712: Escuchen estos audios ...
la cosa es que no sale el mouse
17:42
Type a message

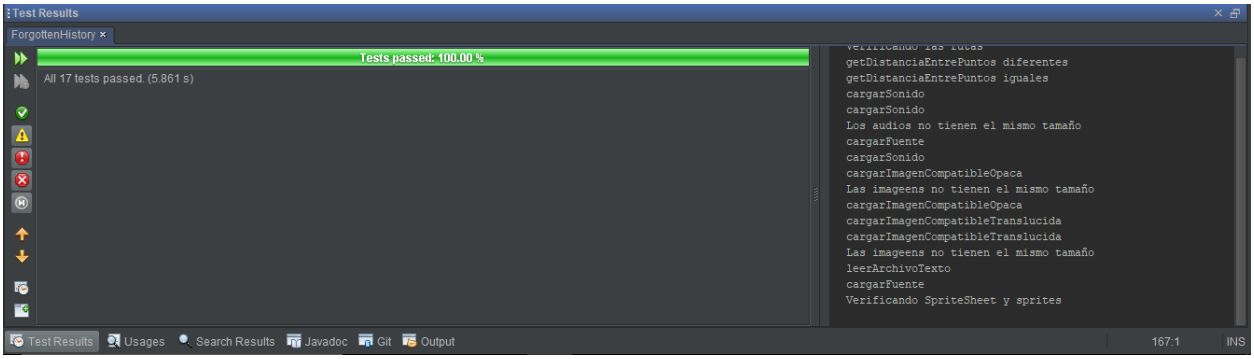


Vistas probadas: 12

Vistas exitosas: 12

Vistas erróneas: 0

Fecha de ejecución: 5/06/2020 9:00 pm

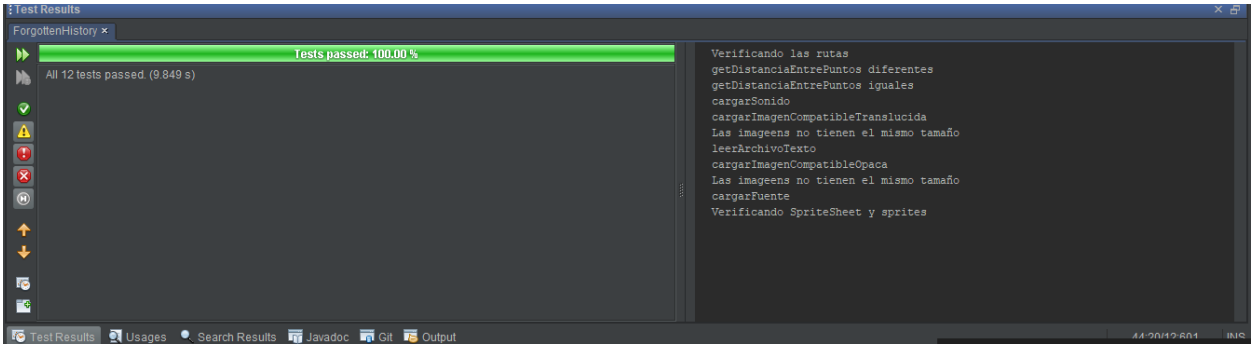


17 Pruebas realizadas.

17 pruebas exitosas

0 pruebas fallidas.

Fecha de ejecución: 3/06/2020 8:30 p.m



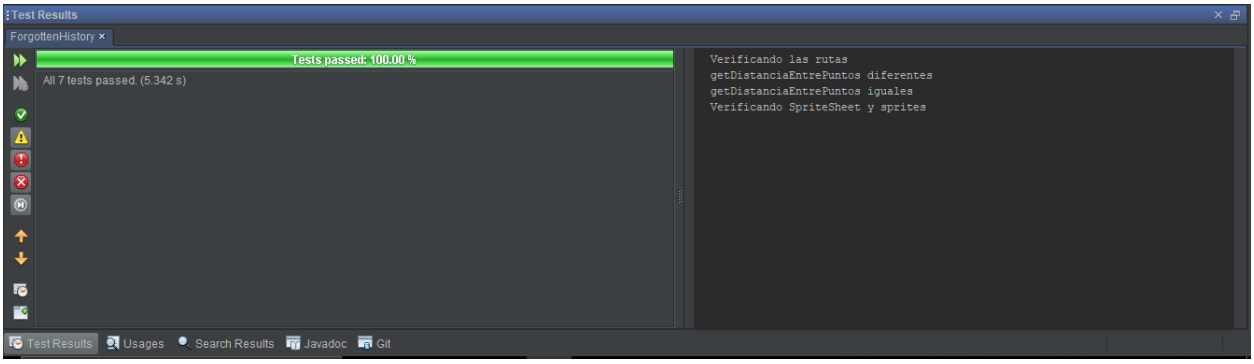
Resultado del procedimiento:

12 pruebas ejecutadas.

12 pruebas exitosas.

0 pruebas fallidas.

Fecha de ejecución: 10/05/2020 12:30 p.m



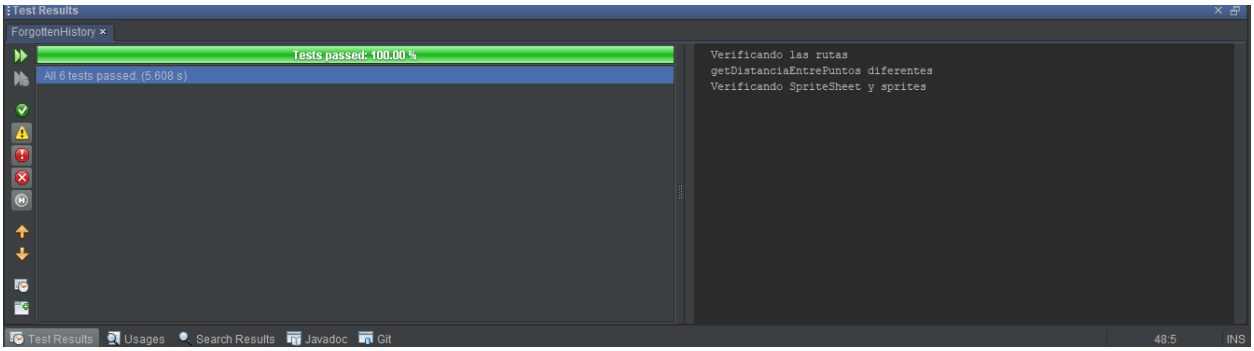
Resultados del procedimiento:

7 pruebas ejecutadas.

7 pruebas exitosas.

0 pruebas fallidas.

Fecha de ejecución: 30/04/2020 8:00 a.m

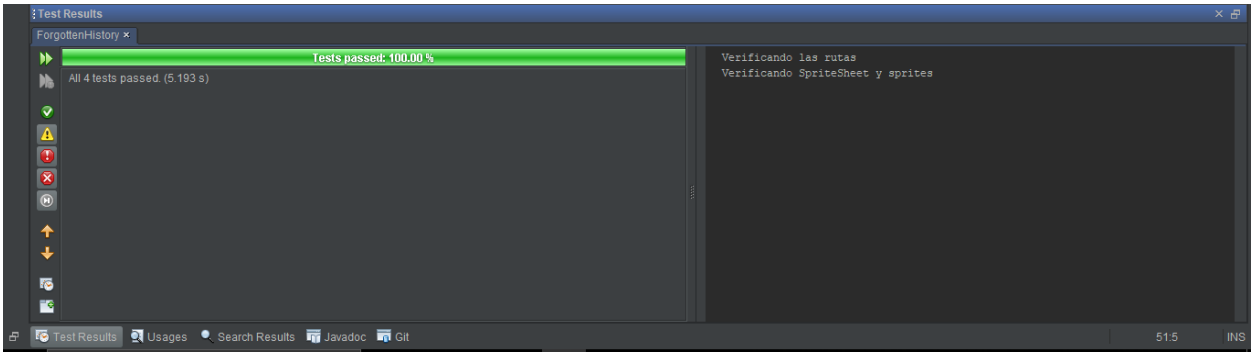


6 pruebas ejecutadas.

6 pruebas exitosas.

0 pruebas fallidas.

Fecha de ejecución: 30/04/2020 8:00 a.m



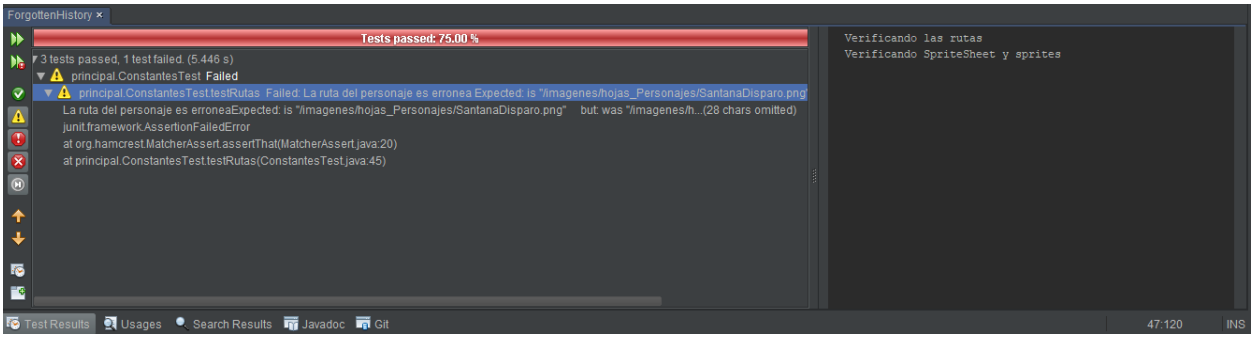
4 pruebas ejecutadas.

4 pruebas exitosas.

0 prueba fallidas.

Eventos anómalos Se adjuntará además lo producido por lo detallado en la siguiente sección.

Fecha de ejecución: 30/04/2020 8:00 a.m



4 pruebas ejecutadas.

3 pruebas exitosas.

1 prueba fallidas.

Razón del fallo: desarrolladores editaron mal el archivo con las rutas hacia los elementos multimedia.

Herramientas

Las herramientas que se sugieren para la realización del proceso de prueba son:

- Junit

Junit

JUnit es un conjunto de clases (framework) creado por por Erich Gamma y Kent Beck, que permite realizar la ejecución de clases Java de manera controlada, para poder evaluar si el funcionamiento de cada uno de los métodos de la clase se comporta como se espera. Es decir, en función de algún valor de entrada se evalúa el valor de retorno esperado; si la clase cumple con la especificación, entonces JUnit devolverá que el método de la clase pasó exitosamente la prueba; en caso de que el valor esperado sea diferente al que regresó el método durante la ejecución, JUnit devolverá un fallo en el método correspondiente.