



IES Laguna de Jotznel

Trabajo de Fin de Grado

Pathway Story Game

Desarrollo de Aplicaciones Multiplataforma

Nombre del Alumno: Daniel Galeano Freijo

Nombre del Tutor del TFG: Carlos Ollero Sánchez

ÍNDICE:

1.	Resumen del proyecto	2
2.	Historia principal del juego y su desarrollo	2
2.1.	Esquema Historia	3
3.	Planteamiento de la programación y desarrollo	3
3.1.	Estructura programada de la historia (Estructura de Nodos)	4
3.2.	Estructura programada de las ventanas	6
4.	Diseño gráfico	14
5.	Preparación final de la aplicación	16
6.	Requisitos de sistema	17
7.	Conclusión	17
8.	Bibliografía	18
9.	Agradecimientos	18

1. Resumen del proyecto:

Este trabajo de fin de Grado se centra en la premisa de crear un juego para escritorio en el que la idea principal es desarrollar una historia que un usuario pueda seguir.

En este juego se avanzará mediante el uso de ventanas y el final del juego vendrá definido por las decisiones que tome la persona que esté jugando pudiendo descubrir más de una decena de finales.

Este juego ha sido programado en Java usando el IDE Eclipse y WindowBuilder como editor de ventanas siendo el uso de estas las que te dan acceso no solo al juego si no también a la información que deriva de este.

La meta final que fue propuesta al principio del trabajo de desarrollo era la de no solo tener un juego entretenido si no generar en el usuario la necesidad de interactuar con los datos del juego, es por esto que hemos creado distintas opciones como la de poder ver cuantos finales del total han sido descubiertos.

2. Historia principal del juego y su desarrollo:

La historia que define “Pathway Story” (siendo así como hemos nombrado a este juego) se centra en la premisa de un personaje que quiere huir del país ya que es un fugitivo de la justicia.

Al principio del juego se nos plantean las opciones de elegir un pasaporte falso con el que intentar escapar y cual es el transporte por el que queremos apostar a la hora de poder escabullirse de la policía.

Una vez tomadas estas decisiones se comenzará la travesía del protagonista pudiendo acabar en distintos finales entre los que están conseguir huir o ser atrapado por la justicia.

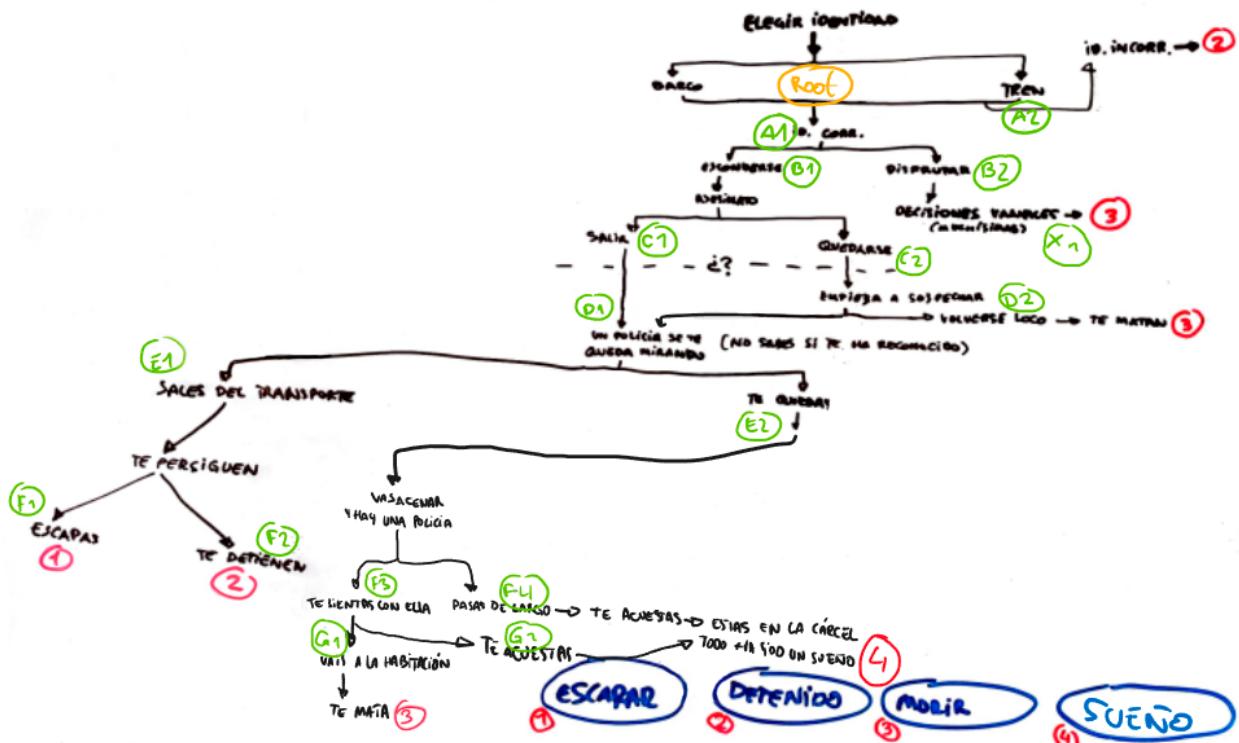
A la hora de desarrollar esta historia se plantearon varias opciones como la de ser un naufrago que tiene que intentar sobrevivir pero se optó finalmente por la idea planteada en el párrafo anterior.

Esta situación deja abierta la posibilidad de ampliar la historia y poder así mejorar aún más el producto final.

La historia ha sido planteada para que todos los caminos puedan ser ordenados en una estructura de árbol que mejorará, como se explicará en el siguiente punto, el proceso de programación del código.

En la siguiente hoja se adjunta una imagen del esquema final que ha seguido la historia del juego, así como todos sus finales.

2.1. Esquema Historia:



3. Planteamiento de la programación y desarrollo:

La programación de esta aplicación se ha diseñado para las siguientes funciones:

- Mostrar y diseñar las distintas ventanas que la forman.
- Desarrollar la estructura de la historia del juego.
- Dar las funcionalidades que permiten avanzar en el juego.
- Creación de datos sobre las partidas.
- Gestión y almacenamiento de estos datos.
- Permitir el acceso y lectura de estos datos.
- Permitir la eliminación de la información no necesaria.
- Control de errores y depuración del código para una mejor eficiencia y evitar errores.

Una vez esto fué planteado se comenzó con la estructuración del programa pero para poder definir esto correctamente fue necesario elegir la estructura con la que se construirá la historia ya que todo giraría en torno a esto y es aquí donde llegamos al punto más relevante de este TFG. El eje sobre el que se sostiene toda la aplicación:

3.1. Estructura programada de la historia (Estructura de Nodos):

Como hemos mencionado anteriormente la historia sobre la que se basa el juego tiene una estructura de árbol, una estructura basada en todas las decisiones que te llevan a seguir unos caminos (ramas) que te llevan a los finales de la “aventura”.

Esto puede llevarte a la idea de programar la historia con una consecución de bucles “if/else” que vayan cambiando las ventanas del juego según las decisiones que se vayan tomando pero, si se piensa fríamente, esto es una idea pésima que solo hará que terminemos con un código extremadamente extenso y en el que, además, controlar errores sería una verdadera pesadilla.

Es por esto que se planteó la siguiente idea: Crear una estructura de datos propia y personalizada que pueda cubrir todas las necesidades de la aplicación.

Y así fué como se llegó a la estructura de Nodos que explicaré a continuación.

Aprovechando la POO que nos brinda Java se crearon las clases “Nodo” y “StoryTree”.

La clase Nodo, encargada de representar cada punto de toma de decisiones de nuestro juego almacenará:

- Su ID con el que podremos identificar el punto de la historia del que se trata.
- Un Nodo “Padre” que nos indicará desde donde se ha llegado a ese punto.
El primer nodo de todos, que indicará que se comienza a recorrer la estructura de árbol será el nodo “Root” que se diferenciará del resto porque no tiene un nodo Padre.
- Un Nodo “Hijo Izquierda” que nos indicará la siguiente rama hacia la izquierda en nuestro árbol.
- Un Nodo “Hijo Derecha” que al igual que el Hijo Izquierdo nos indicará la siguiente rama pero esta vez hacia la derecha.
- Un Array con las opciones que se mostrarán en esa ventana y que corresponderán a la llegada a cada Nodo hijo.
- Una cadena con la opción de transporte elegida al principio de la historia.
- Y finalmente el texto y la URL de la imagen que aparecerá en nuestra ventana de juego.

La clase StoryTree es sobre la que se da forma al árbol, el esquema de la aventura.

En ella se instanciarán y almacenarán todos los datos de cada uno de los objetos Nodo que forman la historia.

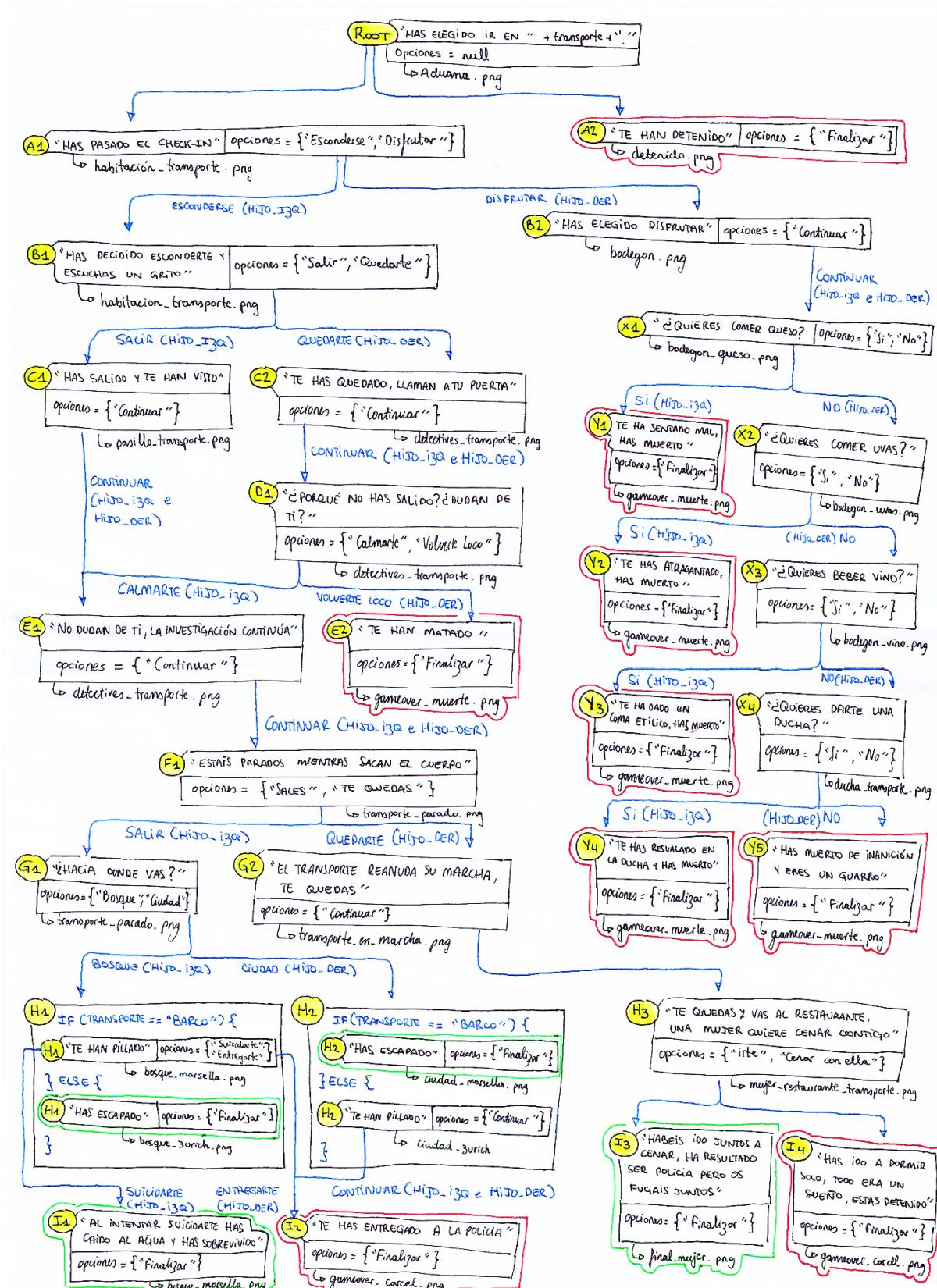
De esta manera se irán creando uno a uno cada Nodo con su Padre y sus Hijos y se irán creando las conexiones que nos llevarán a montar todo el diagrama completo.

En la siguiente página podremos ver claramente tanto el id de cada Nodo como hacia donde apuntan o de donde vienen, la información de las opciones y a dónde lleva cada una y finalmente el texto y la imagen que se mostrará en la ventana.

La clase StoryTree almacenará:

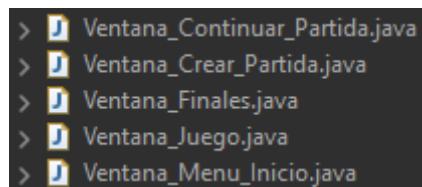
- El Nodo padre.
- El pasaporte y el transporte elegido al principio del juego.

De esta manera podremos almacenar datos que no se usan en el momento y que afectarán más adelante en la historia o varias veces. Podremos así también diferenciar por ejemplo unas imágenes de otras pudiendo por ejemplo utilizar la variable transporte para diferenciar una habitación en un tren de una habitación en un barco.



3.2. Estructura programada de las ventanas:

Una vez definida la estructura del StoryTree con cada uno de los Nodos, el proceso de desarrollo se pudo centrar en el siguiente paso que sería centrarse en la interfaz gráfica. Hasta el momento la capa Front-end se había diseñado con las siguiente estructura:



Tendremos una ventana principal llamada “Ventana_Menu_Inicial” desde la que se controlará el resto de ventanas.

En ella tendremos una imagen de portada y los botones “Nueva Partida”, “Continuar historia”, “Ver Finales” y “Salir” (esta última haciendo un simple dispose();).



Siguiendo el orden dispuesto en los botones de nuestra ventana inicial:

- Con el botón “Nueva Partida” tendremos acceso a la clase “Ventana_Crear_Partida” con la que podremos crear partidas nominales
- Con el botón “Continuar Partida” accederemos a la clase “Ventana_Continuar_partida” con la que podremos seguir una de las partidas ya creadas ejecutando distintas historias. Tanto el concepto de partida como el de historia serán explicados más adelante.
- Con el botón “Finales Descubiertos” podremos tener acceso a la clase “Ventana_Finales” en la que se podrá mostrar información de todos los finales descubiertos de la aventura que un usuario ha podido descubrir en una partida.
- Finalmente, al juego como tal se tendrá acceso mediante la clase “Ventana_juego” en la que se podrán ver imágenes, texto y las opciones de la historia, así como recorrer la aventura con la interacción con estos elementos.

Ahora pasaremos a explicar cada clase de cada ventana en detalle, pero para poder comprender exactamente la funcionalidad de estas, primero debemos explicar un par de conceptos que determinan la forma en la que un usuario interactúa con la aplicación más allá del juego en sí.

Estos conceptos son el de “Historia”, todo el conjunto de decisiones tomadas hasta llegar a un solo final, y el de “Partida”, una sesión nominal en la que se almacenan objetos Historia que están relacionados directamente al nombre de esa partida. Es decir cada historia es cada vez que se juega y cada partida es el conjunto de veces jugadas bajo un nombre. Las clases que definen estos conceptos y sus métodos principales están explicados en detalle al final de este punto en la página 14.

Ahora sí, pasaremos a explicar en detalle cada una de las ventanas que conforman la aplicación a parte de la ventana principal.

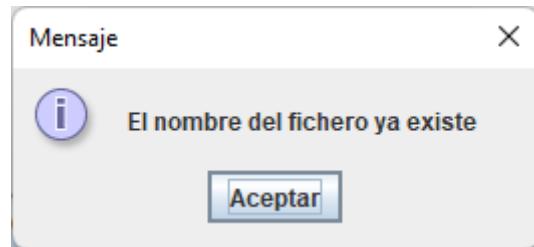
- La primera es la clase “Ventana_Crear_Partida” que lanza el siguiente JFrame.



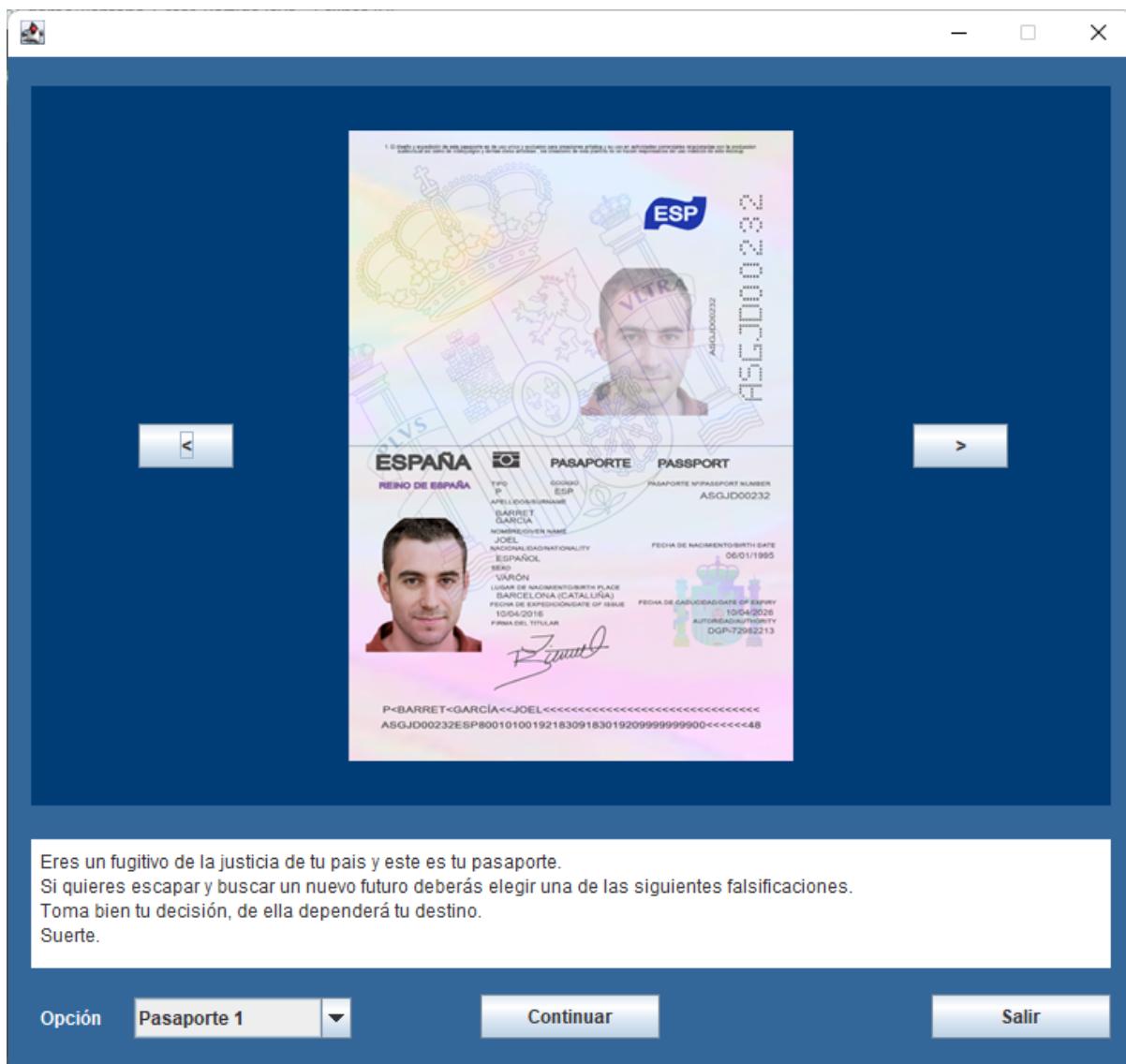
En el que mediante el uso del botón Crear Partida podremos, valga la redundancia crear una nueva partida con el nombre que le demos.

Una vez se pulse el botón Crear Partida se comprobará mediante el uso de la clase `File` y el método `.list()` si existe un fichero con el nombre obtenido del `TextField`.

Si existe, te lanzará el siguiente `showMessageDialog` que te avisará de que el fichero ya existe y no puede duplicarse.

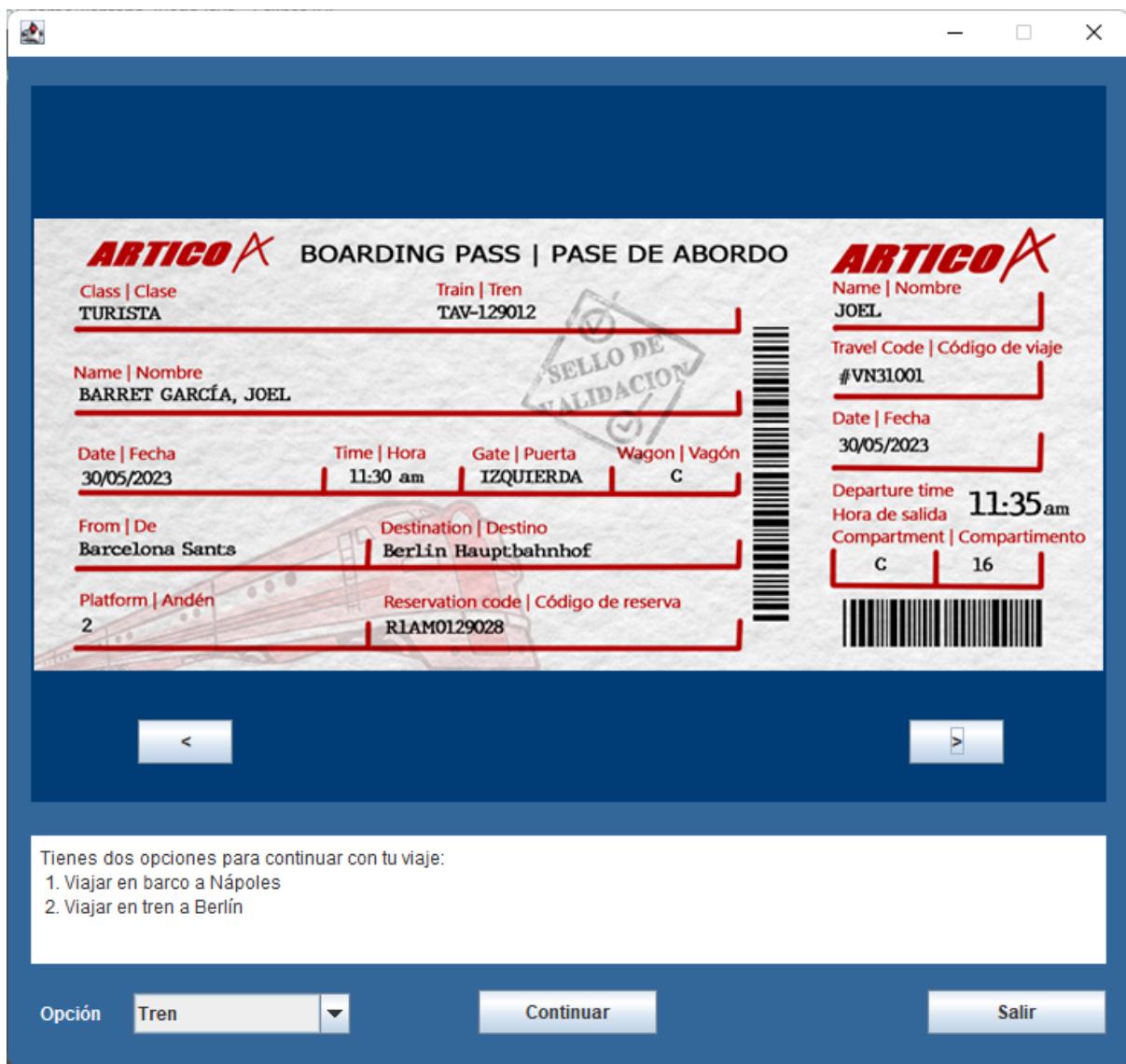


Por otro lado si el fichero no existe en el directorio que le hemos definido mediante la instancia de la clase `File()`, se lanzará el `JFrame` del juego que se corresponde con la clase "Ventana_juego".



En esta ventana como hemos mencionado anteriormente tenemos la primera imagen del juego (Correspondiente a la selección del pasaporte en la historia), las distintas opciones que apuntan hacia los hijos (obtenidas del nodo y representadas en un comboBox) y el botón continuar que te llevará al siguiente nodo y por tanto a la siguiente imagen, con sus correspondientes opciones, etc..

P.ej.:

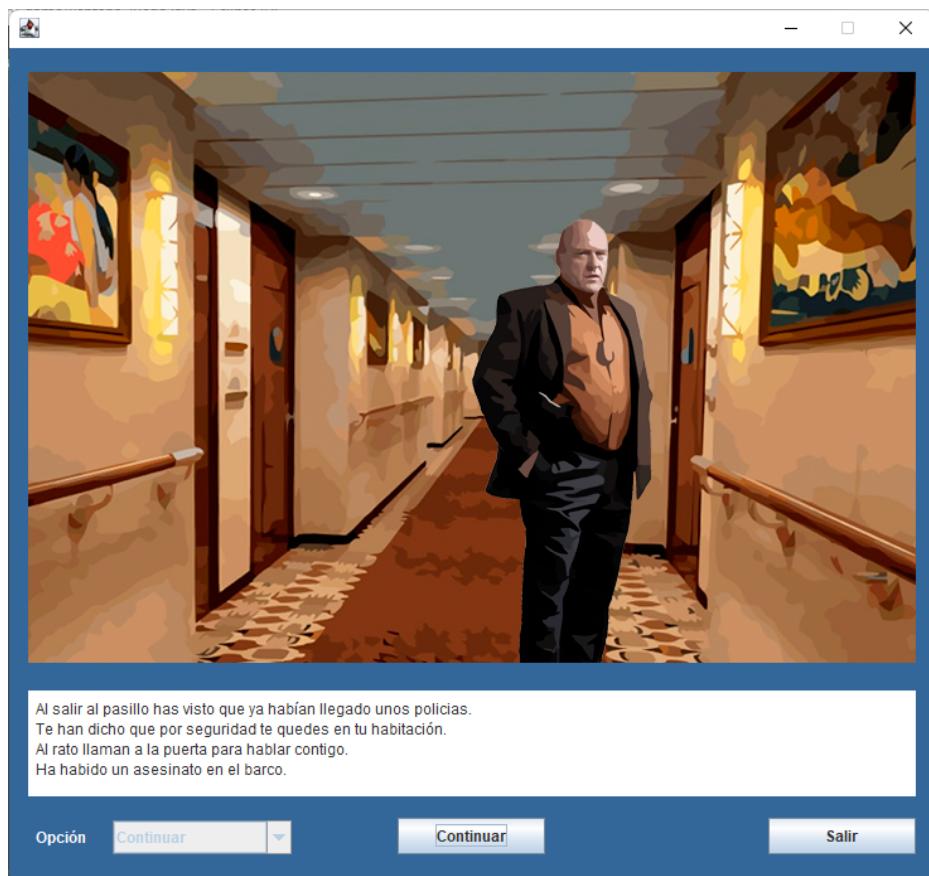


Si miramos el código de esta ventana veremos que el botón continuar tiene distintos estados, y esto es porque las primeras 2 ventanas no pertenecen como tal al StoryTree, si no que sirven simplemente para almacenar los datos del pasaporte y el transporte elegido.

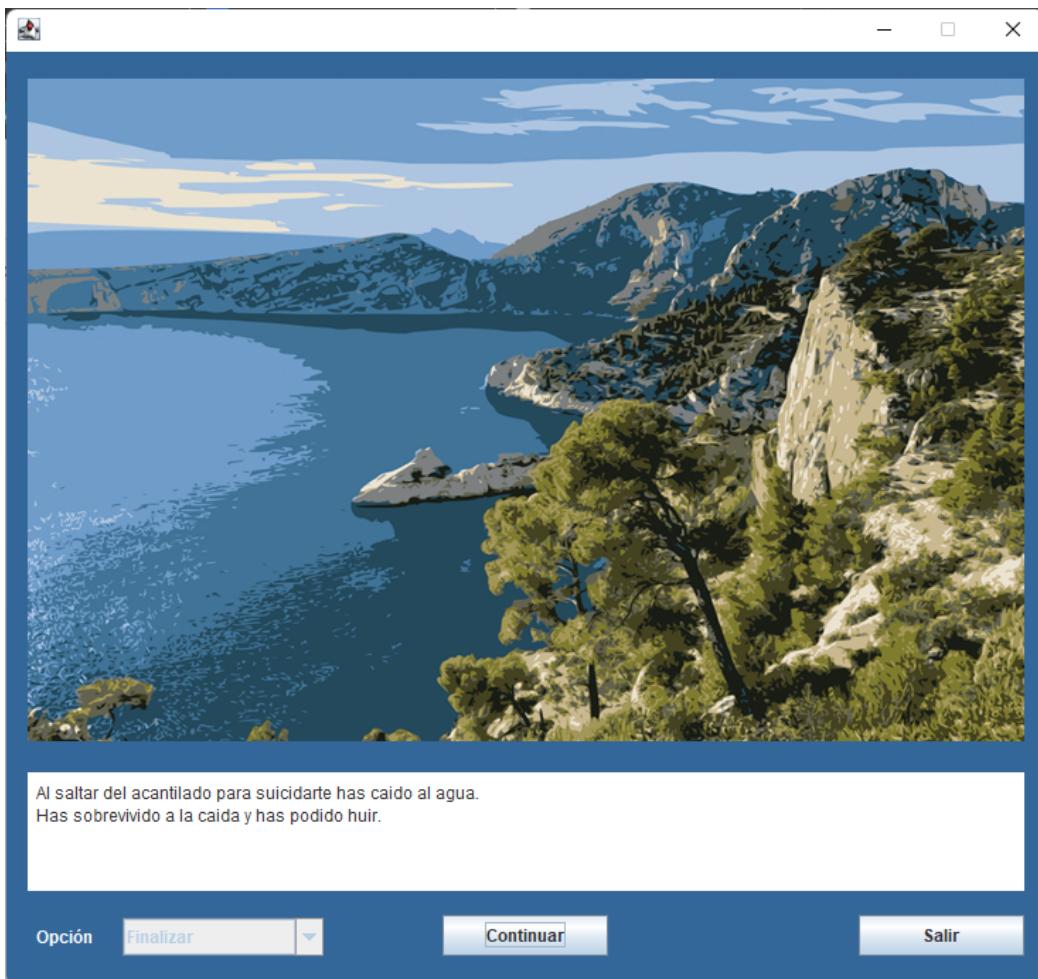
En la ventana juego podremos ver que cuando se cargan las opciones (Que recordemos vienen determinadas por el nodo) podemos tener 2, qué es lo más habitual, o solo 1 cuando no hay que tomar ninguna decisión o hemos llegado al final donde el comboBox se desactiva y se nos muestra únicamente la opción “Continuar” o “Finalizar” (caso que utilizaremos para saber que hemos llegado a un final y que por tanto deben almacenarse todos los nodos en la historia y la historia en la partida)

Cuando pulsamos el botón continuar y la opción del comboBox seleccionada es la de “Finalizar” se creará, como acabamos de decir, una instancia de la partida con el id de la partida que venimos utilizando.

Se utilizará un *ObjectInputStream* para recoger toda la información previa de la partida (si es que existe, en este caso no porque la partida es nueva y el fichero está vacío) y una vez almacenada en una variable, se le añadirá la información serializada de la historia actual y se volverá a escribir el fichero usando esta vez un *ObjectOutputStream*.

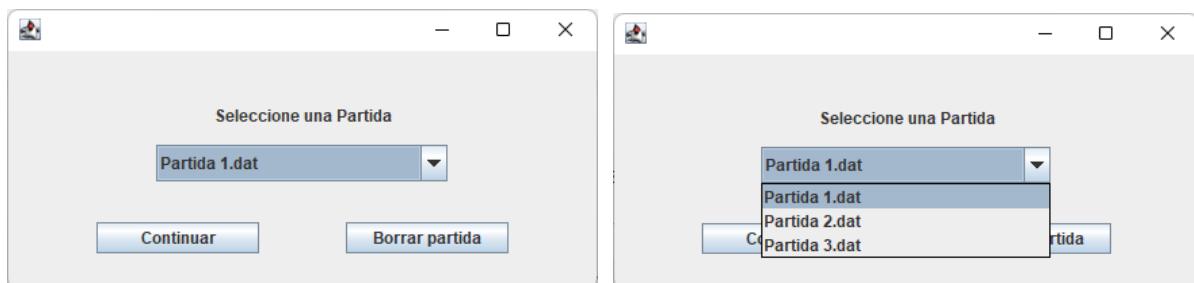


Caso con el comboBox desactivado y la opción Continuar.



Caso con el comboBox desactivado y la opción Finalizar.

La siguiente ventana de la que hablaremos es la correspondiente a la clase “Ventana_Continuar_Partida” que se ejecutará al pulsar el botón Continuar Partida en nuestra ventana principal.

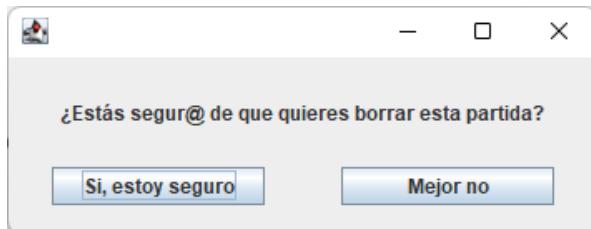


En esta ventana podremos utilizar el comboBox para seleccionar una partida existente y poder seguir realizando historias con ella. Este comboBox también utiliza la clase *File* y el método *.list()* para obtener las partidas almacenadas en el directorio de las partidas.

Una vez pulsado el botón continuar el funcionamiento es el mismo que cuando creamos una partida nueva. Esta vez sí que se reescribirá la información del fichero para añadir la info de la nueva historia.

Esta ventana también implementa la opción de borrar la partida seleccionada.

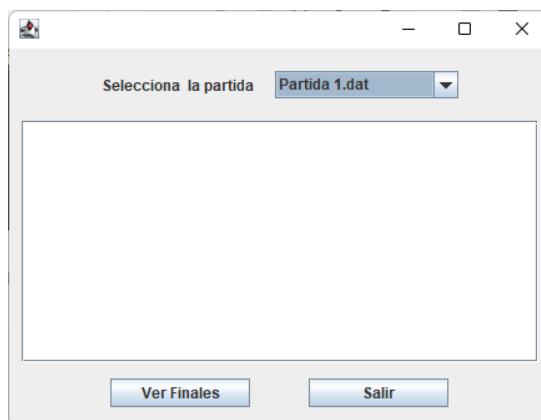
Este botón lanzará la siguiente ventana de confirmación (Cuya clase, “pop_up” está definida al final del mismo fichero).

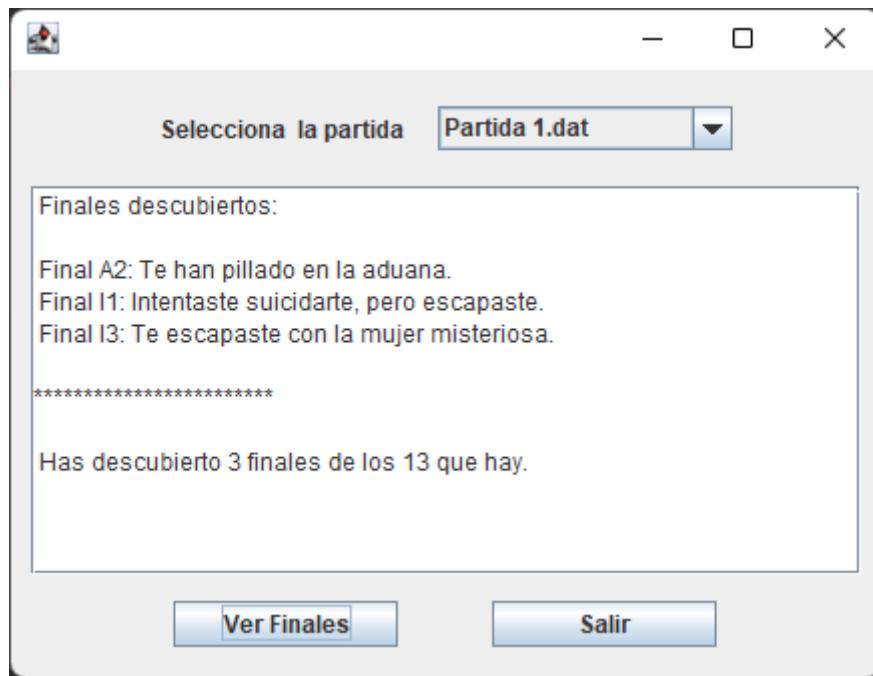


Si la opción elegida es la de “Si” se ejecutará el siguiente método y se borrará la partida.

```
1 package pathway_story_game;
2
3 import java.io.File;
4
5 public class Eliminar_partida {
6
7     public static void borrar_partida(String ruta) {
8         File archivo = new File(ruta);
9         archivo.delete();
10    }
11 }
12
```

Y por último, la ventana final es la correspondiente al botón “Finales Descubiertos” que mostrará la ventana de la clase “Ventana_Finales”. En la que si pulsamos el botón “Ver Finales” se analizarán los datos de una de las partidas detectadas por el comboBox de la misma manera que en la ventana “Continuar Historia”





Para extraer la información del fichero se utilizará la clase *ObjectOutputStream* que devolverá los objetos almacenados en este (objetos partida e historia). Utilizando el método *getHistorias()* de la clase Partida sacaremos todas Historias de este y con el método *getNodosVistados()* de la clase Historia sacaremos un arraylist de nodos visitados en el que el último objeto de esta lista siempre será un final.

```

1 package pathway_story_game;
2
3 import java.io.Serializable;
4 import java.util.ArrayList;
5
6 public class Partida implements Serializable {
7
8     private static final long serialVersionUID = 93255891115285188L;
9
10    private String id_partida;
11    private ArrayList<Historia> historias;
12
13    public Partida(String id_partida) {
14        this.id_partida = id_partida;
15        this.historias = new ArrayList<Historia>();
16    }
17
18    public String getId_partida() {
19        return id_partida;
20    }
21
22    public ArrayList<Historia> getHistorias() {
23        return historias;
24    }
25
26
27    public void addHistoria(Historia historia) {
28        historias.add(historia);
29    }
30
31 }
32
33

```

Clase partida

```

1 package pathway_story_game;
2
3 import java.io.Serializable;
4 import java.util.ArrayList;
5
6 public class Historia implements Serializable {
7
8     private static final long serialVersionUID = -106724157836309693L;
9     private ArrayList<Nodo> nodos_visitados;
10
11    public Historia() {
12        nodos_visitados = new ArrayList<Nodo>();
13    }
14
15    public ArrayList<Nodo> getNodos_visitados() {
16        return nodos_visitados;
17    }
18
19
20    public void addNodoVisitado(Nodo nodo) {
21        nodos_visitados.add(nodo);
22    }
23
24
25
26
27}
28

```

Clase historia

4. Diseño gráfico

En cuanto al diseño gráfico de esta aplicación se utilizaron como herramientas principales la página web www.vectorizer.io que proporciona un efecto artístico con paleta de colores reducida a las imágenes que le indiques y photoshop, utilizar para crear composiciones con varias imágenes así como para mejorar brillo, tono y contraste (entre otros) de cada imagen para darle un estilo uniforme a la aplicación.

Aquí tenemos dos ejemplos de una imagen vectorizada y otra después del proceso de vectorización.



Otro de los puntos importantes en cuanto a la implementación gráfica es la empleada para obtener las caras de las distintas personas representadas en los pasaportes del principio de nuestro juego



Cada una de estas caras ha sido generada por la inteligencia artificial en la página web www.thispersondoesnotexist.com.

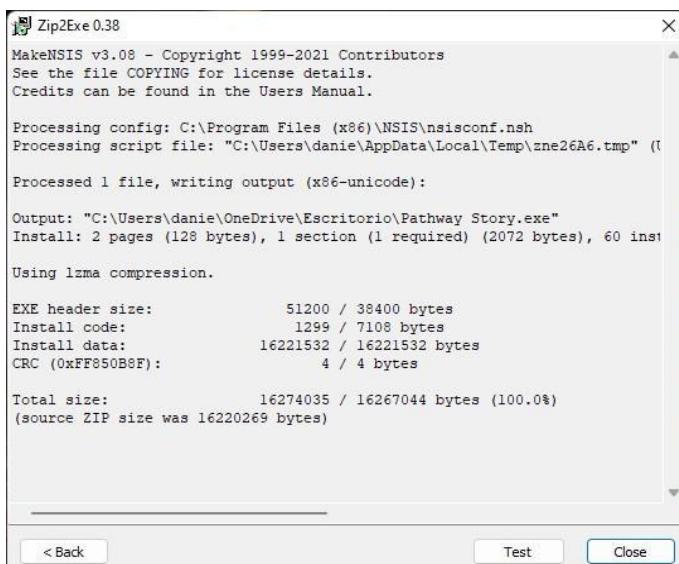
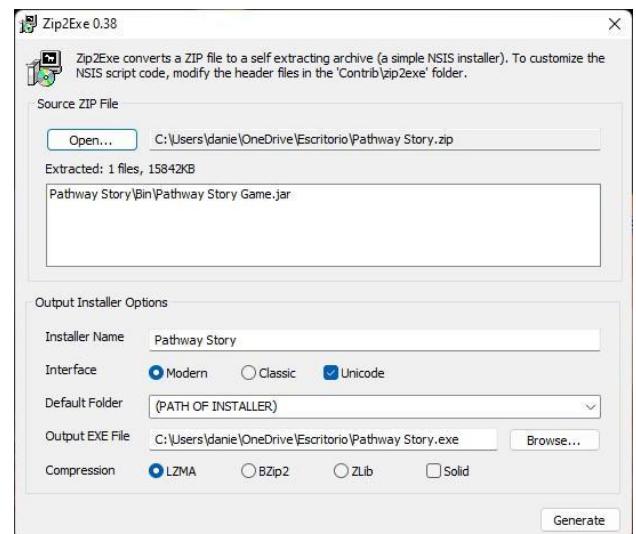
De esta manera hemos podido escoger los personajes que mejor se han adaptado a la historia que hemos creado.

Cada una de estas caras ha pasado también por photoshop para eliminarles el fondo y darle algunos retoques estéticos.



5. Preparación final de la aplicación:

Para dejar la aplicación 100% cerrada decidí crear un archivo .exe de instalación usando la Utilidad NSIS y siguiendo los siguientes pasos:



Archivo instalador con extensión .exe

El archivo instalador “Pathway Story.exe” te permitirá crear el directorio con el archivo jar ejecutable de la aplicación.



6. Requisitos de sistema:

Los requisitos para poder correr Pathway Story en tu pc son mínimos ya que es un gestor de ventanas simple que requiere de una capacidad de computación insignificante.

Lo que sí es importante, y requisito esencial, es tener instalada la versión 1.8 de Java en tu pc así como el correspondiente JDK.

A la hora de poder utilizar el instalador de nuestro juego (que creará el directorio donde se almacenará) tendremos que utilizar una máquina de Windows ya que este ejecutable tiene la extensión “.exe” .

7. Conclusión:

Se trata de un juego diseñado con una capacidad de expansión inmensa y con capacidad para implementar muchas nuevas funcionalidades.

Ha resultado ser un proceso extenso y con mucho ímpetu en las distintas fases de diseño.

El código ha quedado bastante claro y con posibilidad de que un equipo pueda trabajar sobre él sin problema.

La estructura final del proyecto de Java es la siguiente:

- src
 - carpeta de assets
 - imagenes.png
 - pathway_story_game
 - Eliminar_partida.java
 - Historia.java
 - Partida.java
 - StoryTree.java
 - Ventana_Continuar_Partida.java
 - Ventana_Crear_Partida.java
 - Ventana_Finales.java
 - Ventana_Juego
 - Ventana_Menu_inicio.java

La estructura final del programa es la siguiente:

- Carpeta principal
 - Carpeta bin
 - Pathway Story Game.jar
 - Carpeta partidas
 - archivos.dat

8. Bibliografía:

La bibliografía de este proyecto no será muy extensa debido a la relativa poca información que ha sido necesaria para poder llevarlo a cabo. Aún así ahí van mis fuentes de información principales:

- Documentación oficial de JAVA:
 - <https://docs.oracle.com/javase/7/docs/api/>
- Información sobre las estructuras de datos de Árbol:
 - <https://www.delftstack.com/es/howto/java/java-tree/>
 - <https://repositorio.upct.es/bitstream/handle/10317/2745/pfc4277.pdf?sequence=1>
- Generador de caras con IA:
 - <https://thispersondoesnotexist.com/>

9. Agradecimientos:

Agradecimientos a Guillermo López García y Diego Martín Siguero que me llevaron a dar el primer paso a la hora de elegir la estructura de datos de árbol que se ha implementado en el juego de mi aplicación así como a Jorge Monzón Moya y Jorge Torres Cañadas que probaron la historia múltiples veces y su opinión me ayudó a mejorar la historia.