Diseño y Programación Orientados a Objetos

1er. Semestre 2024

Tarea 2: Simulación de una mascota virtual en Java con interfaz gráfica

Lea detenidamente la tarea. **Si algo no lo entiende, consulte. Si es preciso, se incorporarán aclaraciones al final**. Esta interacción se asemeja a la interacción entre desarrolladores y clientes cuando algo no está del todo especificado.

1. Objetivos de la tarea

- Modelar objetos reales como objetos de software.
- Ejercitar la creación y extensión de clases dadas para satisfacer nuevos requerimientos.
- Reconocer clases y relaciones entre ellas en códigos fuentes Java.
- Ejercitar la compilación y ejecución de programas JavaFX desde una consola de comandos.
- Ejercitar la configuración de un ambiente de trabajo para desarrollar aplicaciones JavaFX. Se pide trabajar con un IDE el cual puede ser VSCode o IntelliJ.
- Manejar proyectos vía Git.
- Ejercitar la preparación y entrega de resultados de software (creación de makefiles, readme, documentación).
- Familiarización con desarrollos "iterativos" e "incrementales".
- Desarrollar software bajo la arquitectura MVC (Modelo-Vista-Controlador)

2. Descripción General

El propósito de esta tarea simular una **mascota virtual** mediante una interfaz gráfica, utilizando la biblioteca JavaFX. Se podrá desarrollar ya sea con las bibliotecas de JavaFX, con el uso de Scene Builder, o una combinación de ambas si así lo desean.

Este proyecto representa una extensión de la Tarea 1, profundizando en las interacciones y visualizaciones del comportamiento de la mascota.

Se debe modelar no solo la mascota, sino también los diversos elementos con los que esta puede interactuar. Este modelado incluye tanto la implementación de la lógica de funcionamiento como el desarrollo de los componentes gráficos.

Pueden ver un ejemplo de funcionamiento de mascota virtual en el siguiente enlace: Juego Pou

Tanto en Aula como en el repositorio Github de la asignatura se encontrará un código de ayuda, además de las imágenes para utilizar en esta tarea. No es obligatorio hacer uso de estos elementos.

2.1 Mascota virtual

El modelo para una mascota virtual en esta tarea contiene las siguientes características e indicadores:

- **Nombre** de la mascota

- **Edad**, la cual se mide en segundos (para evitar inconsistencias con el mundo real, considere que esta es una mascota de corta esperanza de vida).
- **Salud**, la cual toma valores entre 0 y 100 puntos
- **Energía**, la cual toma valores entre 0 y 100 puntos
- **Felicidad**, la cual toma valores entre 0 y 100 puntos

Algunos indicadores de la mascota pueden verse afectados entre sí:

- Si `edad` <= 5 y `salud`<=10 puntos, la `felicidad` bajará 20 puntos por cada 0.5 segundos.
- Si 5 < `edad` <= 10 y `salud`<=50, la `felicidad` bajará 20 puntos y la `energía` bajará 10 puntos por cada 0.5 segundos.
- Si `edad`>10 y `salud`<=50, la `felicidad` bajará 30 puntos y la `energía` bajará 20 puntos por cada 0.5 segundos.

En base a los indicadores de edad, salud, energía y felicidad, es posible determinar el **estado** en el que se encuentra la mascota. Estos estados están definidos como un dato tipo `enum` en los códigos de ayuda. A continuación se presentan la lista de estados posibles según orden de prioridad (de menor a mayor), así como sus condiciones específicas:

1. **Neutro**: Estado por omisión de la mascota.
2. **Feliz** : La felicidad de la mascota es mayor o igual a 60 puntos.
3. **Triste**: La felicidad de la mascota es menor o igual a 20 puntos.
4. **Hambriento**:
    - Si la salud es menor o igual a 20 puntos para una edad menor o igual a 5.
    - Si la salud es menor o igual a 50 puntos para una edad entre 5 y 10
5. **Enojado**: La edad es mayor a 5, mientras la salud y energía son menores o iguales a 30 puntos
6. **Cansado**: La energía de la mascota es menor o igual a 15 puntos.
7. **Muerto**: La salud o la energía de la mascota ha alcanzado el valor mínimo de 0, o la edad es mayor o igual a 15.

En caso de verificarse las condiciones para más de un estado a la vez, se privilegia el estado de mayor prioridad.

La mascota, al interactuar con alguna instancia de un ítem, puede modificar sus indicadores y, en consecuencia, su estado.

También pueden modificarse los indicadores y estado de la mascota cuando se duerme. Al dormir, la mascota aumenta su energía en 20 puntos cada 0.5 segundos, mientras que la salud y felicidad aumentan 2 puntos cada 0.5 segundos.

Tanto los indicadores como el estado de la mascota resultante se visualizan en cada instante tal como se ve en el siguiente ejemplo.

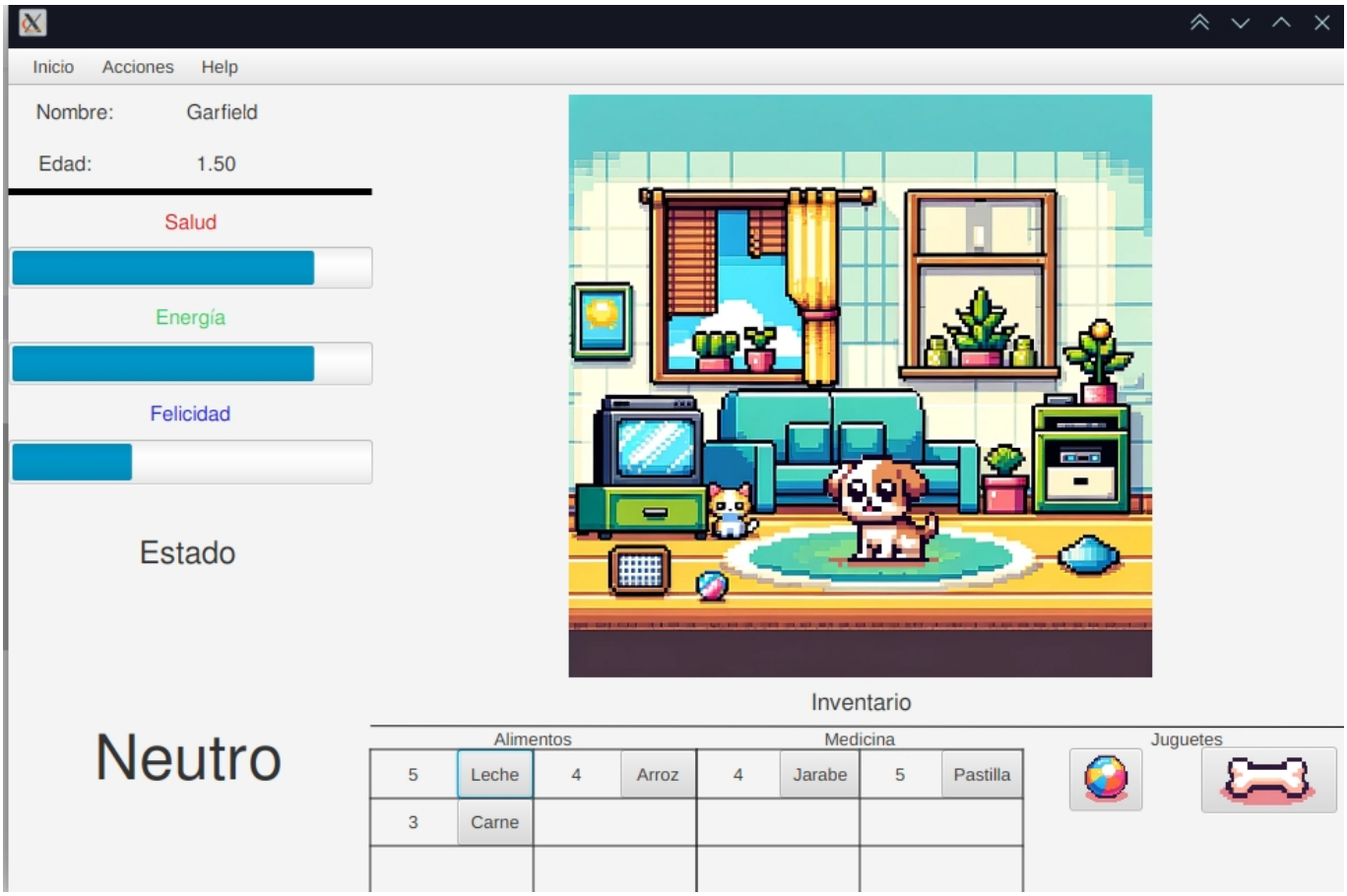


Figura 1: Ejemplo de visualización de la interfaz gráfica. Tanto la imagen de fondo como la imagen de la mascota fueron generadas por el modelo de AI DALL-E, de la empresa OpenAI.

Para este desarrollo, ud. utilizará la arquitectura **MVC** (Modelo-Vista-Controlador). En esta arquitectura, cada parte de su código estará enfocada en alguna funcionalidad del programa. Las clases deberán interactuar entre side manera similar al siguiente diagrama:

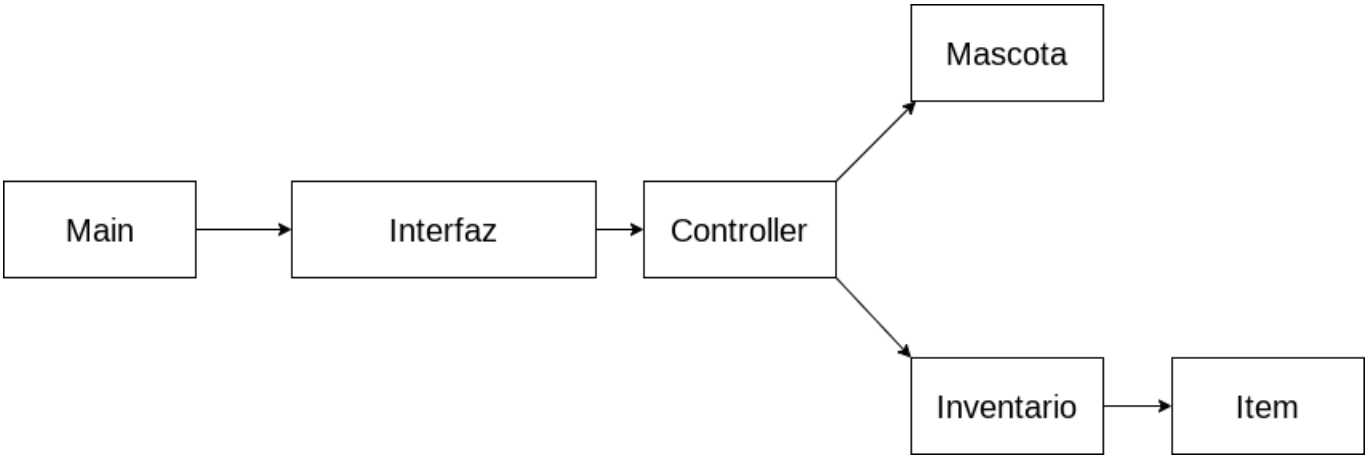


Figura 2: Su desarrollo debería ser similar a este. Sin embargo, pueden haber modificaciones. Ej. La `Mascota` y cada `Item` podrían tener asignada una clase interfaz directamente asociada a estas. Las flechas en este diagrama indican interacción entre las clases.

2.2 Ítems

Los ítems corresponden a diferentes elementos con los que puede interactuar la mascota virtual. Estos se pueden agrupar en las siguientes categorías/clases:

- **Alimento**, la cual, al ser utilizada en la mascota, le aumenta 20 puntos de energía y salud.

- **Medicina**, la cual, al ser utilizada en la mascota, le aumenta 40 puntos de salud.
- **Juguete**, el cual, al ser utilizado en la mascota, le aumenta 30 puntos de felicidad.

Cada instancia de un ítem podrá ser usado en la mascota una única vez, **a excepción de los juguetes**. Los juguetes pueden ser utilizados tantas veces como se desee sin que estos desaparezcan.

Adicionalmente, cada una de estas tres clases corresponde a sub clases de la clase base `Item`. Esta contiene todos los elementos comunes de los ítems. Esto incluye los siguientes atributos:

- `id`, el cual se utilizará para poder acceder a cada ítem.
- `cantidad`, correspondiente a la cantidad de elementos existente para dicho elemento. Si la cantidad tiene valor `-1`, se considera el objeto ilimitado (válido para los juguetes).
- `nombre`, correspondiente al nombre de dicho ítem.

2.3 Inventario

El inventario contiene todas las instancias de ítems disponibles. Este permitirá la adición o eliminación de ítems según sea necesario, garantizando una actualización constante de los elementos almacenados.

2.4 Configuración inicial

Al igual que la tarea 1, el programa contará con un archivo de configuración `config.csv` (referencia a archivos .csv), cuyos elementos se separan por `;`. Este archivo contiene tanto el nombre de la mascota, así como los ítems que tendrá a disposición en su inventario siguiendo el siguiente formato:

```
Garfield
1;Juguete;Pelota;images/toys/ball.png
2;Juguete;Hueso;images/toys/bone.png
3;Alimento;Leche;5
3;Alimento;Arroz;4
4;Alimento;Carne;3
5;Medicina;Jarabe;4
6;Medicina;Pastilla;4
```

Notar como en el caso de los juguetes, el último elemento representa *la ubicación de la imagen* de dicho juguete.

3. Etapas del Desarrollo

3.1 Etapa 1: Creación Interfaz gráfica

Recree la interfaz gráfica básica del programa. Tome como base la siguiente imagen (no es necesario implementar funcionalidades en esta etapa, solo la interfaz):

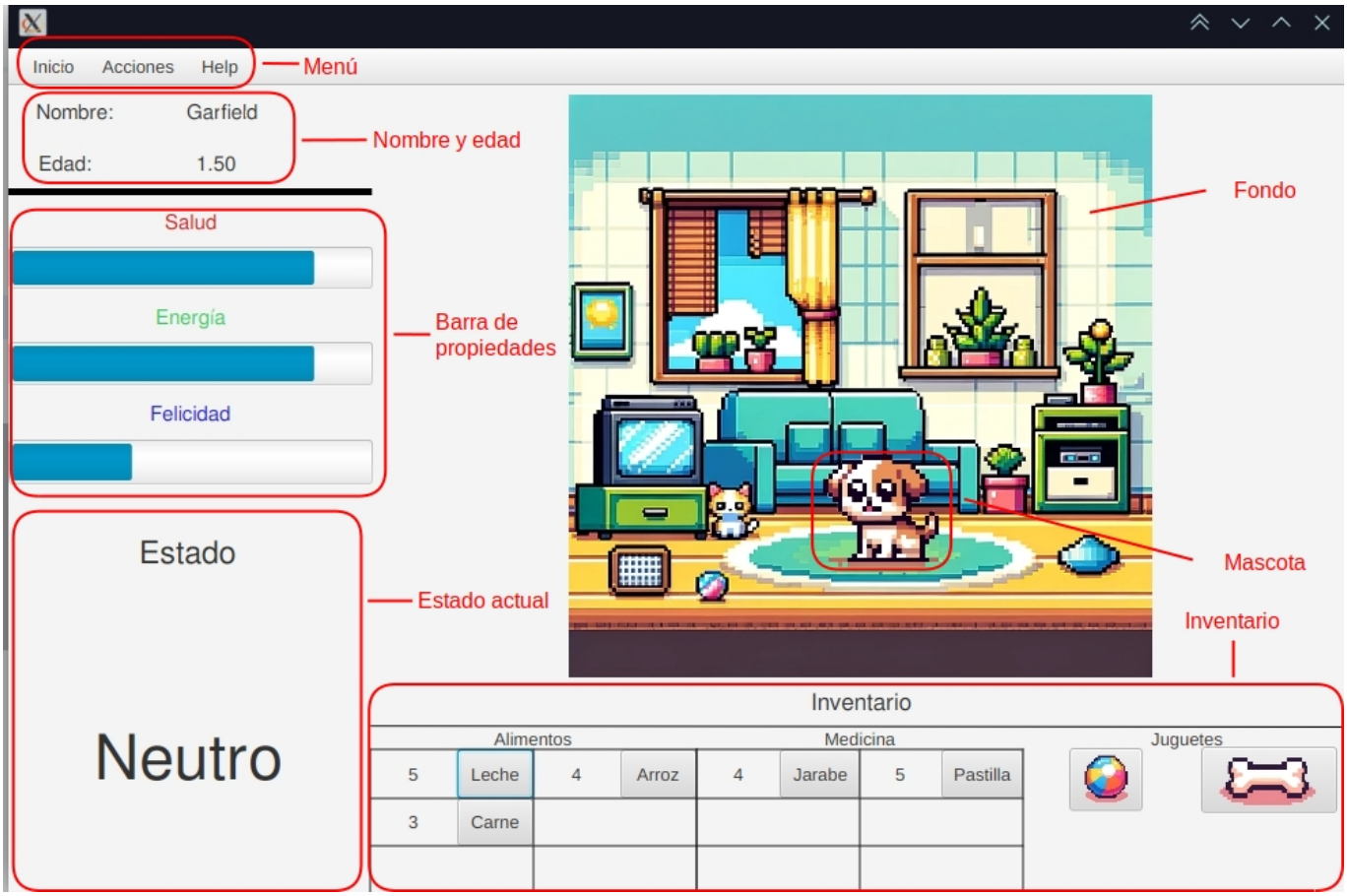


Figura 3: Datos adicionales sobre la interfaz gráfica.

Puede utilizar valores manuales (hardcodeados) para mostrar los elementos. Para las imagenes, puede utilizar aquellas disponibles en el código de ayuda en Aula/Github. Sin embargo, ¡sea libre de usar sus propias imágenes para el fondo, mascota y juguetes si así lo desea!

Hints:

- Las barras de propiedades pueden ser generadas con la clase `ProgressBar`.
- Cada ítem de inventario está compuesto por la cantidad disponible de dicho elemento (etiqueta/`Label`) y de un botón (`Button`), a excepción de los juguetes, que solo cuentan con un botón con la imagen del juguete.

3.2 Creación de la clase `Mascota` y su conexión con la interfaz

En esta etapa se desea probar las interacciones entre la `Mascota` y la interfaz. Para la creación de la clase `Mascota`, defina los atributos privados que representen el `nombre`, `edad`, `salud`, `energía`, `felicidad` y `estado` de la mascota. Además, implemente un método que actualice los indicadores y el estado de la mascota según las reglas definidas en la sección 2.1.

Los valores iniciales para los indicadores de `Mascota` en esta etapa serán los siguientes

- `edad = 0`
- `salud = 10`
- `energia = 10`
- `felicidad = 10`
- `estado = "Cansado"`

En este punto, cada botón presente en el inventario generará modificaciones tanto en los indicadores de la mascota como en el estado de la misma.

También se debe crear e implementar las clases `Inventario` e `Item`. Cada objeto `Item` deberá conectarse a su respectivo elemento en la interfaz gráfica. En este punto **no es necesario implementar el paso del tiempo** (se considera una edad fija de **cero** para esta etapa).

Hint: Se recomienda utilizar Propiedades y Vínculos (Property y Binding) para la conexión entre los indicadores de la `Mascota` y los elementos de la interfaz gráfica.

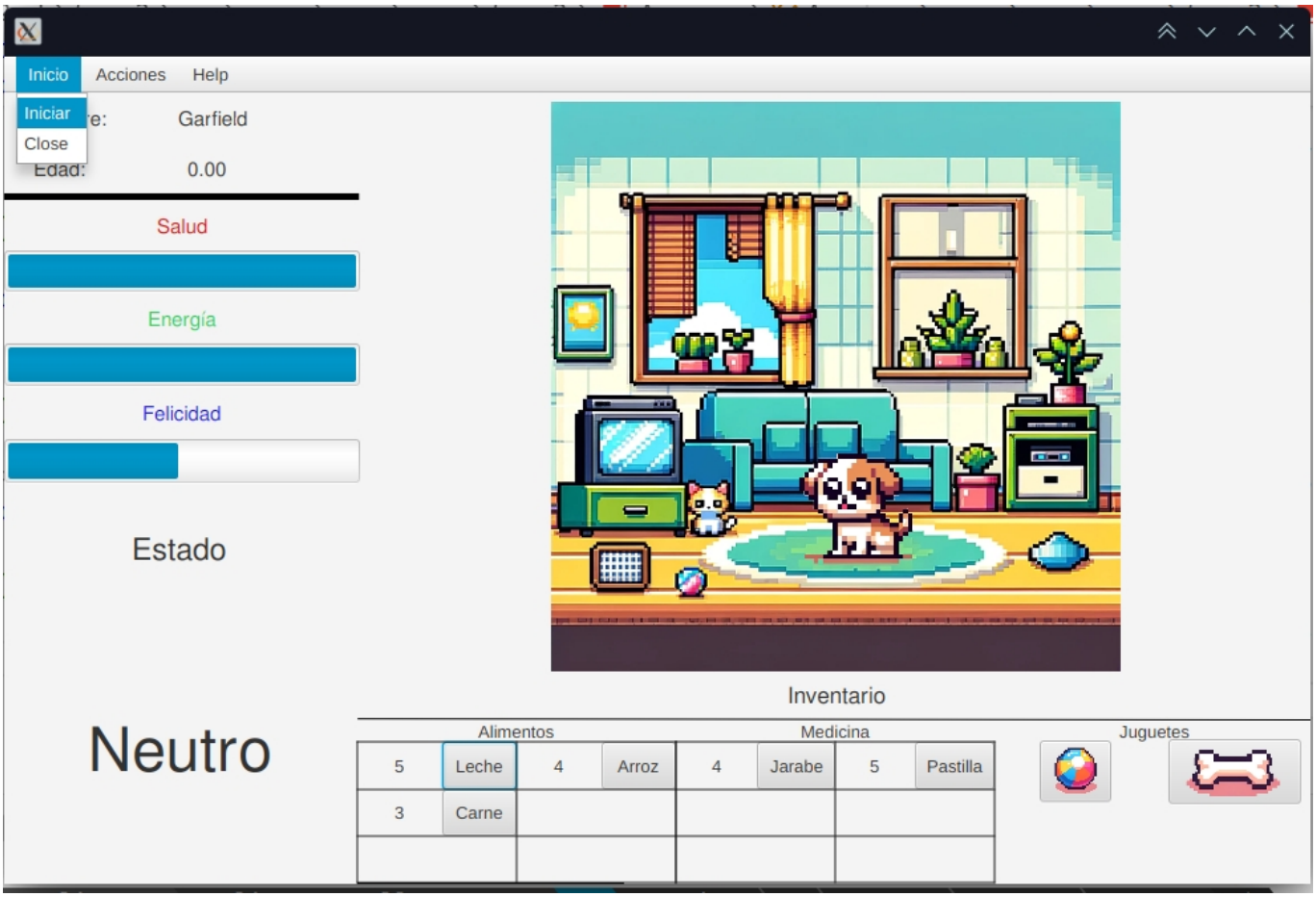
3.3 Creación de la clase Controlador

La clase `Controlador` será una clase intermedia que generará las conexiones entre la `Mascota` y los diferentes `Item` definidos en la etapa anterior.

Además, la clase `Controlador` será la encargada de implementar el paso del tiempo. Esto se realizará con un objeto `Timeline`, donde cada `KeyFrame` procesará los efectos del paso del tiempo en la mascota. Es decir:

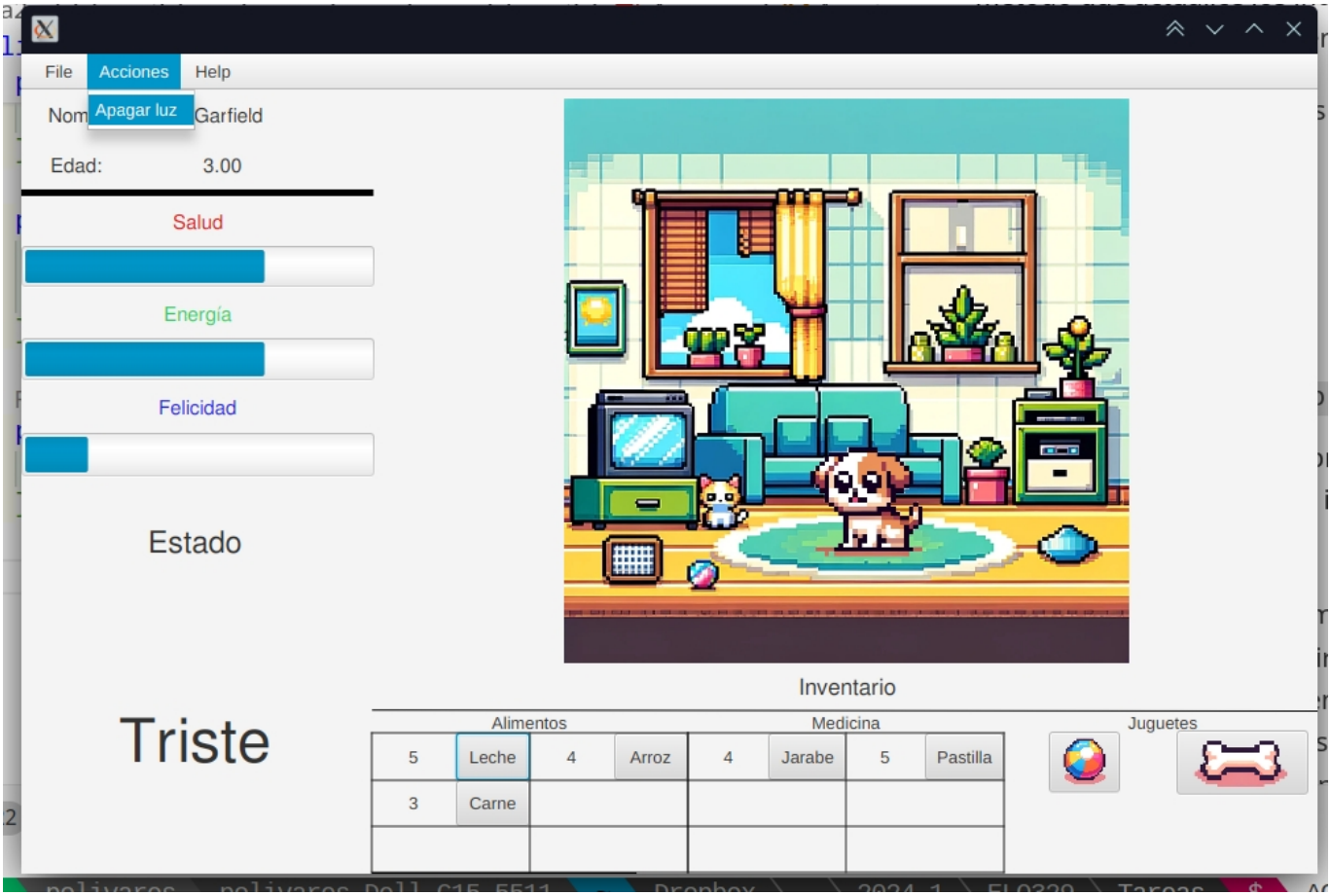
- Un aumento en la edad de la mascota, la cual aumenta en **0.5 segundos**.
- La modificación de la salud, la energía y la felicidad de la mascota, las cuales disminuyan en **5 puntos cada una** por cada 0.5 segundos.

Los `Keyframe` se ejecutarán cada **0.5 segundos**, y el `Timeline` solo comenzará cuando se indique desde la interfaz gráfica, tal como se muestra en la siguiente imagen:

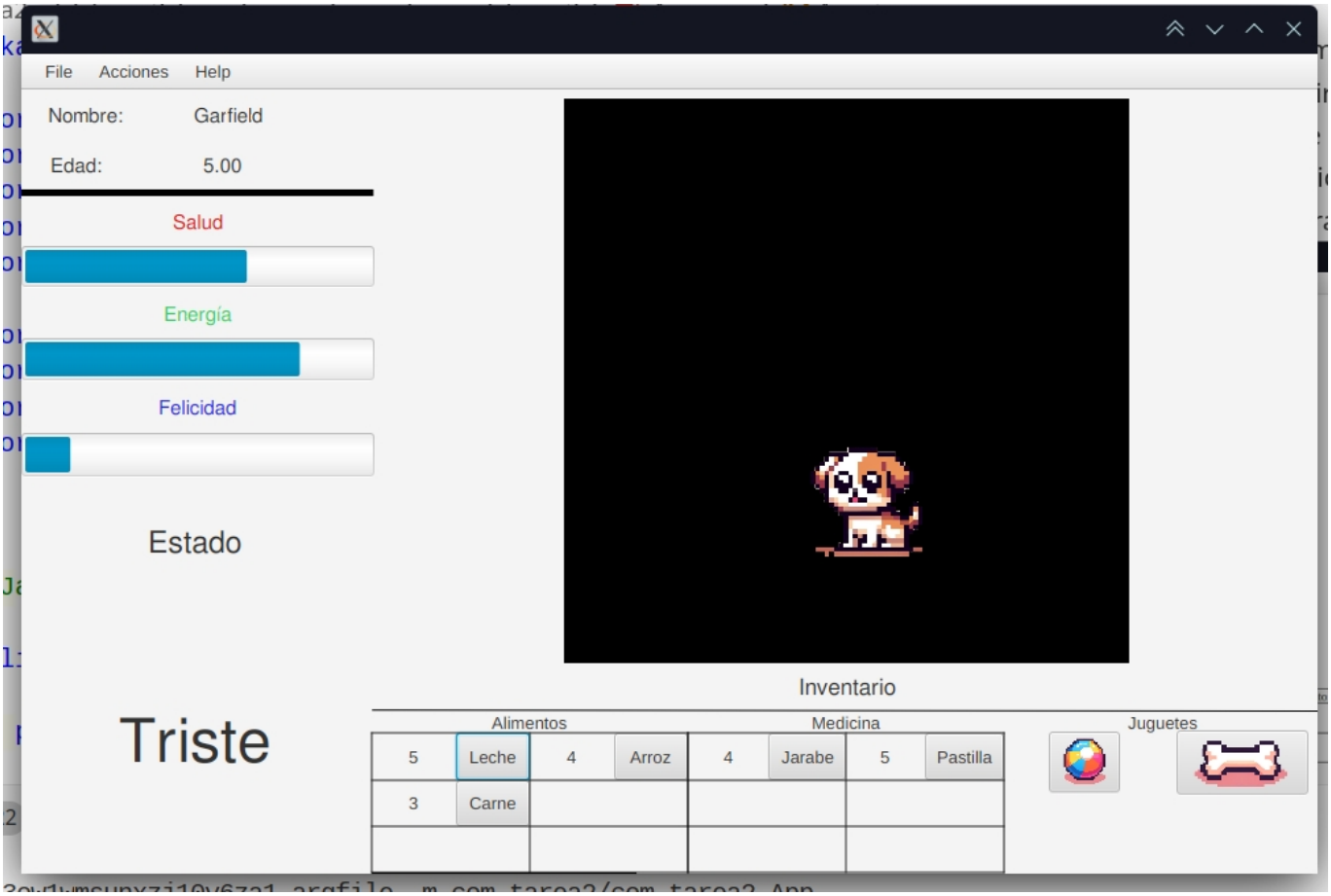


También en este punto se implementará en la clase `Mascota` el método `sleep` que permitirá a la `Mascota` dormir. Desde la interfaz, la funcionalidad de dormir se activará al momento de ingresar a una opción de

acciones de la barra de menú, como se indica en la siguiente imagen:



Tal como se indica en la sección 2.1, al dormir, la mascota aumenta su energía en 20 puntos cada 0.5 segundos, mientras que la salud y felicidad aumentan 2 puntos cada 0.5 segundos. El apagar la luz produce que el fondo de la `Mascota` cambie a uno totalmente negro tal como se muestra en la imagen a continuación:



.

3.4 Lectura de archivo `config.csv` e implementación de elementos finales del menú

En esta etapa ud. deberá leer desde un archivo `config.csv` tanto el nombre de la mascota, como los diferentes items que componen el inventario, según las indicaciones entregadas en la sección 2.4.

Adicionalmente, en esta etapa se implementará el menú con todas sus funcionalidades definidas según los siguientes `MenuItem`:

- **Inicio**: Iniciar, Reiniciar, Close/Salir.
- **Acciones**: Apagar luz, Prender luz.
- **Help**: About/Acerca de.

3.5 Extra crédito: Agregar animación a la `Mascota`

Agregue movimientos horizontales a la `Mascota` que emulen su caminar bajo las siguientes consideraciónes:

- La `Mascota` camina rápidamente por el hogar cuando su estado sea **Feliz**.
- La `Mascota` camina a velocidad media por el hogar cuando su estado sea **Neutro**.
- La `Mascota` no se mueve cuando está **Muerto**.
- La `Mascota` se mueve lentamente en cualquier otro estado.

Esta parte es voluntaria, su desarrollo otorga 10 puntos adicionales (la nota final se satura en 100). Si desarrolla esta parte, indíquelo en su documentación.

- **Hint**: Puede utilizar tanto `Transition` como `Timeline` para el movimiento de la `Mascota`

4. Elementos a considerar en su documentación

Entregue todo lo indicado en "Normas de Entrega de Tareas" (entrega por github/gitlab **obligatorio en esta tarea**).

Deberá entregar un proyecto IntelliJ o VSCode que incluya su desarrollo

En su archivo de documentación (`pdf` o `html`) incorpore el diagrama de clases final de la aplicación.

Para la entrega, deberá subir a Aula un enlace a su repositorio Github/Gitlab, dando acceso a los ayudantes para su revisión.