Mascota Virtual

Manual Técnico

Versión: 0100

Fecha:19/04/2022

Manual técnico

Descripción: Detalla la información técnica de la elaboración del programa

Desarrollador: Cristian Alejandro Vásquez Escobar, 202131936

Manual técnico

Descripción: Detalla la información técnica de la elaboración del programa

Manual técnico

Descripción: Detalla la información técnica de la elaboración del programa

Requerimientos para la ejecución del juego

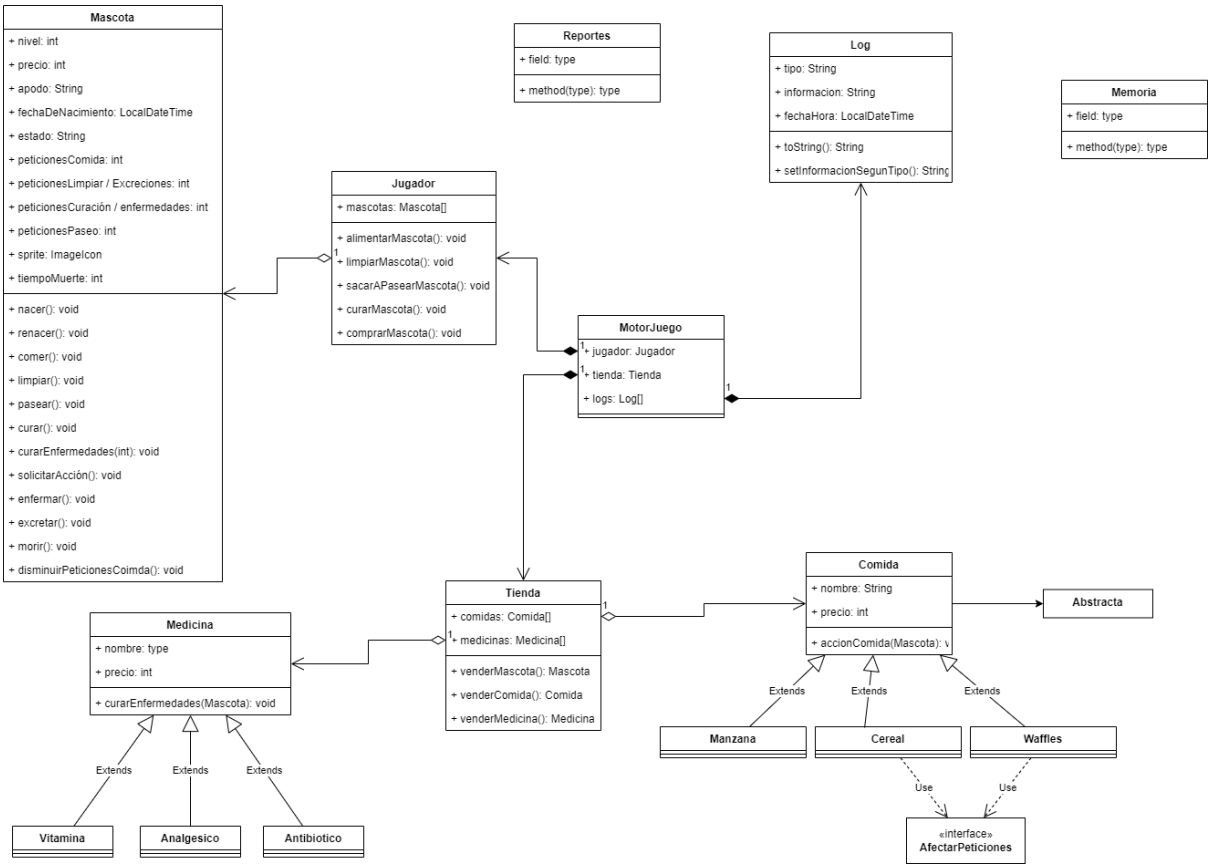
- Una computadora
- Se necesita tener instalada la máquina virtual de java. Esto se puede hacer desde su página y para cualquier sistema operativo: Descargar Java
- Alguna terminal para poder interpretar comandos, si se encuentra en windows con el cmd o el windows powershell será suficiente.

Manual técnico

Descripción: Detalla la información técnica de la elaboración del programa

Elaboración del programa

Diagramas de clases UML



En el siguiente link se puede observar de mejor manera: click aquí

Herramientas

- Para la realización del programa “Mascota virtual” se utilizó el lenguaje de programación java en su versión 17
- El ambiente de desarrollo utilizado para el frontend es Apache Netbeans 12.6
- El ambiente de desarrollo utilizado para el backend es visual studio code en su versión 1.64.2
- Fue realizado desde una laptop hp 15-db0 con windows 10.
- Para una mejor organización se utilizó el sistema de versiones git con github.
- Para el desarrollo se utilizó maven.
- Para la elaboración del archivo .jar se utilizó maven.

Clases

Para la realización del programa se crearon un total de 41 clases.

HiloPeticion

```java
public class HiloPeticion extends Thread {

    private int peticiones;
    private int peticionesMaximas;
    private int tiempoMilisegundos;
    private String tipo;
    private Mascota mascota;

    public HiloPeticion(int peticionesMaximas, int segundos, String tipo, Mascota mascota) {
        this.peticiones = 0;
        this.peticionesMaximas = peticionesMaximas;
        this.tiempoMilisegundos = segundos * 1000;
        this.mascota = mascota;
        this.tipo = tipo;
    }

    @Override
    public void run() {
        while (peticionesMaximas > peticiones) {

            try {
                sleep(tiempoMilisegundos);
            } catch (InterruptedException ex) {
                System.out.println("Error en el sleep " + ex);
            }
            JOptionPane.showMessageDialog( parentComponent: null, "Tu mascota " + mascota.getApodo() + " realizó una
                    title: "Petición",
                    JOptionPane.INFORMATION_MESSAGE);
            peticiones++;
            actualizarPeticiones();
        }
        limitePeticionesAlcanzado();

    }
```

En esta clase se manejan hilos que actúan cuando la mascota pide ser alimentada o paseada.

Manual técnico

Descripción: Detalla la información técnica de la elaboración del programa

Jugador

```java
public class Jugador {

    public Mascota[] mascotas;
    public int totalMascotas;
    public Medicina[] medicinas;
    public int totalMedicinas;
    public Comida[] comidas;
    public int totalComidas;
    public int dinero;

    public Jugador() {
        this.mascotas = new Mascota[10];
        this.totalMascotas = 0;
        this.medicinas = new Medicina[100];
        this.totalMedicinas = 0;
        this.comidas = new Comida[100];
        this.totalComidas = 0;
        dinero = 250;
```

```
elpers\Get.java
```

```java
    public void revivirMascota(Mascota mascota) {
        int precioRevivir = mascota.getNivel() * 5 + 10;
        if (dinero > precioRevivir) {
            mascota.nacer();
            dinero -= precioRevivir;
            JOptionPane.showMessageDialog( parentComponent: null,
                    "Has revivido a tu mascota! pasas a tener " + dinero + " de dinero.", title: "Mascota revivida
                    JOptionPane.INFORMATION_MESSAGE);
        } else {
            JOptionPane.showMessageDialog( parentComponent: null,
                    "No tienes el dinero suficiente para realizar el pago, necesitas " + precioRevivir,
                    title: "Dinero insuficiente",
                    JOptionPane.WARNING_MESSAGE);
        }

    }
```

Aquí se tenían todos los atributos del jugador, sus mascotas, las medicinas compradas, las mascotas, dinero, etc. También tiene métodos que pueden realizar la compra de una mascota, comida y medicina; así como el de revivir una de sus mascotas.

Log

```java
public class Log {

    private String tipo;
    private String informacion;
    private LocalDateTime fechaHora;
    private int nivel;

    public Log(String tipo, Mascota mascota) {
        this.tipo = tipo;
        fechaHora = LocalDateTime.now();
        setInformacionSegunTipo(tipo, mascota);
    }

    public Log(String tipo, Mascota mascota, int nivel) {
        this.tipo = tipo;
        fechaHora = LocalDateTime.now();
        this.nivel = nivel;
        setInformacionSegunTipo(tipo, mascota);
    }
```

Maneja todos los logs que se generan a la hora que hay una acción generada por el jugador. sobreescribe el método toString() para devolver un mensaje con los datos de cada log.

Mascota

```
public class Mascota {

    private int nivel;
    public int precio;
    private String apodo;
    private String estado;
    private ImageIcon sprite;
    private String nombre;
    public HiloPeticion hiloPeticionComida;
    public int segundosPeticionComida;
    private String peticionesComidaString;
    public HiloPeticion hiloPeticionPaseo;
    public int segundosPeticionPaseo;
    private String peticionesPaseoString;
    public MascotaFrame frame;
    private String peticionesLimpiarString;
    private int peticionesLimpiar;
    private int contadorComidasParaExcretar;
    private String peticionesCurarString;
    private int peticionesCurar;
    private Log[] logs;
    private int tiempoMuerte;
    private int cantidadBatallasSiguienteNivel;
    private int batallasRealizadas;
    private Jugador jugador;
```

Esta clase maneja a cada mascota, con atributos que son útiles a través de toda la aplicación. Con métodos para realizar acciones como subir de nivel, nacer, ir de paseo, etc.

Manual técnico

Descripción: Detalla la información técnica de la elaboración del programa

```java
public void nacer() {
    this.nivel = 0;
    setCantidadBatallasSiguienteNivel();
    this.hiloPeticionComida = new HiloPeticion( peticionesMaximas: 5, segundosPeticionComida, PeticionesTypes.
    hiloPeticionComida.start();
    hiloPeticionComida.actualizarPeticiones();
    this.hiloPeticionPaseo = new HiloPeticion( peticionesMaximas: 4, segundosPeticionPaseo, PeticionesTypes.pa
    hiloPeticionPaseo.start();
    hiloPeticionPaseo.actualizarPeticiones();
    setPeticiones(PeticionesTypes.limpiar, peticionesLimpiar);
    setPeticiones(PeticionesTypes.curar, peticionesCurar);
    this.estado = EstadosTypes.vivo;
    logs[HelperClass.getTotalLogs(logs)] = new Log(LogTypes.nacimiento, this);
}

public void pasear(boolean gano) {
    if (peticionesCurar <= 0) {
        hiloPeticionPaseo.reiniciarPeticiones();
        hiloPeticionPaseo.actualizarPeticiones();
        logs[HelperClass.getTotalLogs(logs)] = new Log(LogTypes.paseo, this);
        if (gano)
            batallar();
    }
}

public void subirNivel() {
    nivel++;
    setCantidadBatallasSiguienteNivel();
    batallasRealizadas = 0;
    JOptionPane.showMessageDialog( parentComponent: null, "Tu mascota " + apodo + " ha subido a nivel " + nive
            JOptionPane.INFORMATION_MESSAGE);
    logs[HelperClass.getTotalLogs(logs)] = new Log(LogTypes.nivel, this, nivel);
    if (nivel == 5) {
        JOptionPane.showMessageDialog( parentComponent: null,
                message: "Como tu mascota subió al nivel 5, morirá después de cierto tiempo, podrás revivirl
                title: "Subió nivel",
                JOptionPane.WARNING_MESSAGE);
        HiloMuerte hiloMuerte = new HiloMuerte(tiempoMuerte, this);
        hiloMuerte.start();
```


```java
public void batallar() {
    batallasRealizadas++;
    int randomNumber = (int) Math.floor(Math.random() * (15 - 1 + 1) + 1);
    int dineroGanado = 10 + 20 * nivel + randomNumber;
    jugador.agregarDinero(dineroGanado);
    if (batallasRealizadas == cantidadBatallasSiguienteNivel) {
        subirNivel();
    }
}

public void ganarBatallar(Jugador jugador) {
    int randomNumber = (int) Math.floor(Math.random() * (15 - 1 + 1) + 1);
    int dineroAgregado = 10 + (20 * nivel) + randomNumber;
    jugador.agregarDinero(dineroAgregado);
}

public void morir() {
    JOptionPane.showMessageDialog(frame,
            "Lamentablemente tu pokemon " + apodo + " de la clase " + nombre + " ha muerto :(", title: "Pokemc
            JOptionPane.WARNING_MESSAGE);
    this.estado = EstadosTypes.muerto;
    hiloPeticionComida.reiniciarPeticiones();
    hiloPeticionComida.actualizarPeticiones();
    hiloPeticionPaseo.reiniciarPeticiones();
    hiloPeticionPaseo.actualizarPeticiones();
    peticionesCurar = 0;
    peticionesLimpiar = 0;
    setPeticiones(PeticionesTypes.curar, peticionesCurar);
    setPeticiones(PeticionesTypes.limpiar, peticionesLimpiar);
    logs[HelperClass.getTotalLogs(logs)] = new Log(LogTypes.muerte, this);
}

public void comer(Comida comida) {
    comida.accionComida(this);
    hiloPeticionComida.actualizarPeticiones();
    excretar();
    logs[HelperClass.getTotalLogs(logs)] = new Log(LogTypes.comida, this);
}
```

MotorJuego

```java
package com.alejandrove.cuidarpokemon.backend;

public class MotorJuego {

    public Jugador jugador;
    public Tienda tienda;
    public Log[] logs;

    public MotorJuego() {
        jugador = new Jugador();
        tienda = new Tienda();
        logs = new Log[250];
    }

}
```

Esta clase tiene el estado del juego en general, del jugador, de una tienda y todos los logs.

Tienda

```java
public class Tienda {

    Comida[] comidas;
    Medicina[] medicinas;

    public Tienda() {
        llenarComidas();
        llenarMedicinas();
    }

    public void llenarComidas() {
        comidas = new Comida[] {
                new Manzana(),
                new Cereal(),
                new Waffle(),
        };
    }

    public void llenarMedicinas() {
        medicinas = new Medicina[] {
                new Vitamina(),
                new Analgesico(),
                new Antibiotico(),
        };
    }
```

Una clase para poder manejar el acceso del jugador a la compra de artículos como comida y medicina, o mascotas. Con métodos para vender estos mismos.

```java
public void venderPokemon(int idPokemon, String[] data, String apodo, Jugador jugador, Log[] logs) {

    Image image = null;
    try {
        URL url = new URL(data[1]);
        image = ImageIO.read(url);
    } catch (IOException e) {
    }

    ImageIcon sprite = new ImageIcon(image);

    jugador.comprarMascota(new Mascota(apodo, sprite, data[0], logs, jugador));
}

public void venderComida(Jugador jugador, Comida comida) {
    jugador.comprarComida(comida);
}

public void venderMedicina(Jugador jugador, Medicina medicina) {
    jugador.comprarMedicina(medicina);
}
```

Helpers

```java
public class HelperClass {

    public static boolean verificarPrecio(Jugador jugador, int precio, JFrame jframe) {
        if (precio > jugador.getDinero()) {
            JOptionPane.showMessageDialog(
                    jframe,
                    "Tienes " + jugador.getDinero() + " monedas y para realizar tu compra necesitas " + precio,
                    title: "Dinero insuficiente",
                    JOptionPane.ERROR_MESSAGE);
            return false;
        }
        return true;
    }

    public static void compraExitosa(JFrame jframe, Jugador jugador) {
        JOptionPane.showMessageDialog(
                jframe,
                "Tu compra se realizó con éxito, pasas a tener " + jugador.getDinero() + " monedas",
                title: "Compra realizada",
                JOptionPane.INFORMATION_MESSAGE);
    }

    public static void scaleImage(ImageIcon icon, JLabel label) {
        Image img = icon.getImage();
        Image imgScale = img.getScaledInstance(label.getWidth(), label.getHeight(), Image.SCALE_SMOOTH);
        ImageIcon scaledIcon = new ImageIcon(imgScale);
        label.setIcon(scaledIcon);
    }

    public static void abrirPokemonFrame(JFrame parent, MotorJuego juego, Mascota mascota) {
```

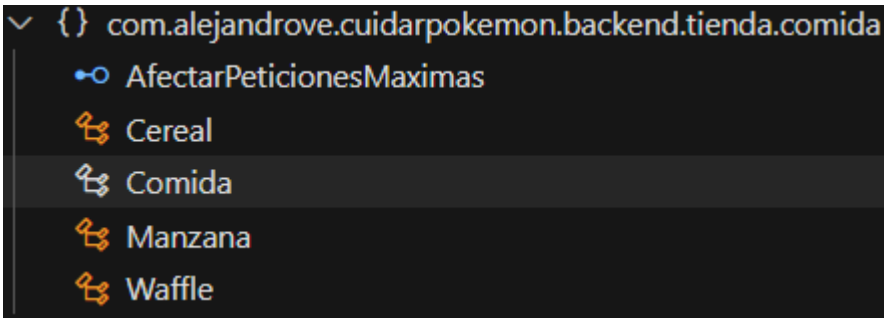
En esta clase se manejan métodos estáticos no propios de otras clases y que se pueden reutilizar en diferentes clases, además de mejorar la comprensión de otras clases evitando un abultamiento de código.

También hay un paquete de helpers donde hay clases como EstadosTypes y PeticionesTypes, que son útiles cuando se quieren comparar strings y así evitar confusiones cuando estas se escriben.

Comida

```java
public abstract class Comida {

    protected String nombre;
    protected int precio;

    public abstract void accionComida(Mascota mascota);

    /* GETTERS */
    public String getNombre() {
        return nombre;
    }

    public int getPrecio() {
        return precio;
    }
    /*  */

}
```

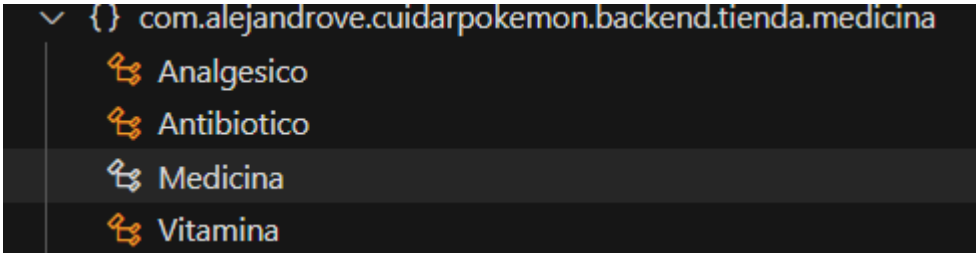
Una clase padre de cada una de las comidas.



Medicina

```java
public abstract class Medicina {

    public String nombre;
    public int precio;

    public abstract void curarEnfermedades(Mascota mascota);

    /* GETTERS */
    public String getNombre() {
        return nombre;
    }
    /*  */
}
```

Clase padre de todas las medicinas.



Juego de memoria

Memoria

```java
public class Memoria {

    public int parejas;
    public ImageIcon[] sprites;
    public int[] ids;
    public int totalIds;
    public Jugador jugador;
    public Jugador oponente;
    public int totalCardsLevantadas;
    public Card[] cardsLevantadas;
    public JFrame frame;
    public boolean tiempoDeEsperaActivo;
    public Jugador jugadorActivo;
    private int totalParejasEncontradas;

    public Memoria(int parejas, JFrame frame) {
        this.parejas = parejas;
        this.totalIds = 0;
        this.ids = new int[parejas];
        this.sprites = new ImageIcon[parejas];
        this.frame = frame;
        this.cardsLevantadas = new Card[2];
        this.tiempoDeEsperaActivo = false;
        this.totalParejasEncontradas = 0;
    }

    public void getImagenes() {
        for (int i = 0; i < parejas; i++) {
            int randomNumber = (int) Math.floor(Math.random() * (151 - 1 + 1) + 1);
            while (existeId(randomNumber)) {
                randomNumber = (int) Math.floor(Math.random() * (151 - 1 + 1) + 1);
            }
            String[] data = Get.nameAndUrl(randomNumber);
            Image image = null;
            try {
                URL url = new URL(data[1]);
                image = ImageIO.read(url);
            } catch (IOException e) {
                System.out.println(e);
            }
```

Aquí se genera el mini juego de memoria, con métodos como el de obtener las imágenes aleatorias, voltear la tarjeta,comprobar ganador, etc

Manual técnico

Descripción: Detalla la información técnica de la elaboración del programa

```
public void voltearCard(Card card) {
    if (!card.equals(cardsLevantadas[0]) && !tiempoDeEsperaActivo && !card.getCardEncontrad()) {
        card.voltearCard();
        cardsLevantadas[totalCardsLevantadas] = card;
        totalCardsLevantadas++;
        if (totalCardsLevantadas == 2) {
            tiempoDeEsperaActivo = true;
            Timer timer = new Timer();
            timer.schedule(new TimerTask() {
                @Override
                public void run() {
                    if (cardsLevantadas[0].getId() == card.getId()) {
                        JOptionPane.showMessageDialog(frame,
                                "Se ha encontrado una pareja, " + jugadorActivo.getNombre() + " mantiene el
                                title: "Pareja encontrada", JOptionPane.INFORMATION_MESSAGE);
                        totalParejasEncontradas++;
                        jugadorActivo.parejaEncontrada();
                        jugadorActivo.actualizarPunteo();
                        card.encontrada();
                        cardsLevantadas[0].encontrada();
                        comprobarGanador();
                    } else {
                        card.regresarCard();
                        cardsLevantadas[0].regresarCard();
                        JOptionPane.showMessageDialog(frame,
                                "Esta no es una pareja, termina el turno de " + jugadorActivo.getNombre(),
                                title: "Pareja no encontrada", JOptionPane.INFORMATION_MESSAGE);
                        cardsLevantadas = new Card[2];
                        cambiarJugadorActivo();
                    }
                    totalCardsLevantadas = 0;
                    tiempoDeEsperaActivo = false;
                }
            }, delay: 700);
        }
    }
}
```

.

```
public void comprobarGanador() {

    if (totalParejasEncontradas == parejas) {

        if (jugador.getParejasVolteadas() == oponente.getParejasVolteadas()) {
            JOptionPane.showMessageDialog(frame, message: "El juego ha terminado en empate!", title: "Juego te
                    JOptionPane.INFORMATION_MESSAGE);
            ((PasearFrame) ((MemoriaFrame) frame).parent).mascotaPaseada(gano: false);
        } else if (jugador.getParejasVolteadas() > oponente.getParejasVolteadas()) {
            JOptionPane.showMessageDialog(frame, message: "Tu pokemon ha ganado!", title: "Juego terminado",
                    JOptionPane.INFORMATION_MESSAGE);
            ((PasearFrame) ((MemoriaFrame) frame).parent).mascotaPaseada(gano: true);
        } else if (jugador.getParejasVolteadas() < oponente.getParejasVolteadas()) {
            JOptionPane.showMessageDialog(frame, message: "El pokemon rival ha ganado!", title: "Juego termina
                    JOptionPane.INFORMATION_MESSAGE);
            ((PasearFrame) ((MemoriaFrame) frame).parent).mascotaPaseada(gano: false);
        }
        frame.setVisible(b: false);
        frame.dispose();
        ((MemoriaFrame) frame).parent.setVisible(b: true);

    }
```

Jugador

```java
public class Jugador {
    private int parejasVolteadas;
    private JLabel label;
    private String nombre;

    public Jugador(JLabel label, String nombre) {
        this.parejasVolteadas = 0;
        this.label = label;
        this.nombre = nombre;
        actualizarPunteo();
    }

    public String getNombre() {
        return nombre;
    }

    public int getParejasVolteadas() {
        return parejasVolteadas;
    }

    public void parejaEncontrada() {
        parejasVolteadas++;
    }

    public void actualizarPunteo() {
        label.setText(nombre + ": " + parejasVolteadas);
    }
}
```

Clase para manejar cada uno de los jugadores del minijuego.

Card

```java
public class Card extends JLabel {

    private int id;
    private ImageIcon pokemonSprite;
    private Icon pokebola;
    private boolean cardEncontrada;

    public Card() {
        super();
    }

    public void setCardAttributes(int id, ImageIcon pokemonSprite) {
        this.id = id;
        this.pokemonSprite = pokemonSprite;
        this.pokebola = getIcon();
        this.cardEncontrada = false;
    }

    public void voltearCard() {
        HelperClass.scaleImage(pokemonSprite, this);
    }

    public void regresarCard() {
        setIcon(pokebola);
    }

    public void encontrada() {
        cardEncontrada = true;
    }

    public int getId() {
        return id;
    }

    public boolean getCardEncontrad() {
        return cardEncontrada;
    }
```

Clase hija de JLabel que nos facilita la implementación de las cards en el frontend, haciendo mucho más fácil los efectos visuales.