Super Auto Pets

Manual técnico

Versión: 0100

Fecha: 04/04/2022

Desarrollador: Cristian Alejandro Vásquez Escobar, 202131936

Manual de usuario

Descripción: Detalla la información para el uso del juego.

Manual de usuario

Descripción: Detalla la información para el uso del juego.

Requerimientos para la ejecución del juego

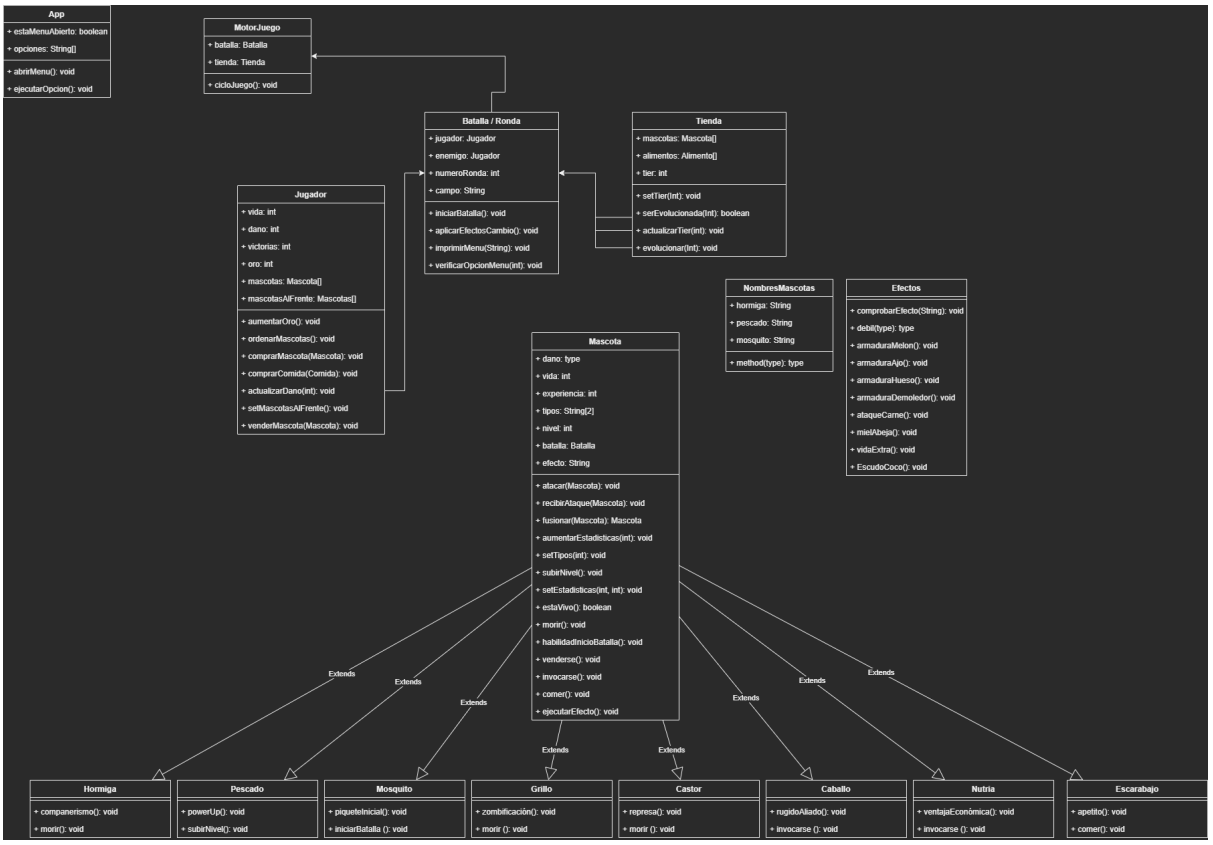
- Una computadora.
- Se necesita tener instalada la máquina virtual de java. Esto se puede hacer desde su página y para cualquier sistema operativo: Descargar Java.
- Alguna terminal para poder interpretar comandos, si se encuentra en windows con el cmd o el windows powershell será suficiente.

Manual de usuario

Descripción: Detalla la información para el uso del juego.

Elaboración del programa

Diagrama de clases UML



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

Herramientas

- Para la realización del programa “super auto pets” se utilizó el lenguaje de java en su versión 8
- El ambiente de desarrollo utilizado es el editor de texto 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.

Diseño

Para facilitar el desarrollo del programa se crearon un total de 76 clases.

Manual de usuario

Descripción: Detalla la información para el uso del juego.

Helpers

En el package helpers se encuentran un total de 6 clases, la mayoría de forma estática.

```java
public static int obtenerIndiceMascota(Mascota _mascota, Mascota[] _aliados) {
    int totalMascotas = totalMascotas(_aliados);
    for (int i = 0; i < totalMascotas; i++) {
        Mascota mascota = _aliados[i];
        if (mascota.equals(_mascota)) {
            return i;
        }
    }
    return -1;
}

public static Mascota obtenerMascotaAleatoria(Mascota[] mascotas, int totalMascotas) {
    int randomNumber = (int) Math.floor(Math.random() * (totalMascotas - 1 + 1) + 1);
    return mascotas[randomNumber - 1];
}

public static boolean mascotaExiste(Mascota _mascota, Mascota[] _mascotas) {

    int totalMascotas = totalMascotas(_mascotas);

    for (int i = 0; i < totalMascotas; i++) {
        Mascota mascota = _mascotas[i];
        if (mascota.equals(_mascota)) {
            return true;
        }
    }

    return false;
}
```

Helpers: La funcionalidad principal es realizar métodos con una lógica algo compleja o más independiente, para no abultar con código y hacer más comprensibles a las demás.
Comidas: Esta maneja los efectos de la comida y su implementación
Efectos: Aquí se manejan los efectos producto de las habilidades de las mascotas
Menu: En esta clase se manejan menús, serviciales al momento de imprimir opciones y elegir las opciones
NombresMascotas: Una clase estática para almacenar el nombre de todas las mascotas. con métodos serviciales para estas mismas.
NombresTipos: Clase estática que almacena el nombre de los tipos.

Jugador

Crea el jugador, con todos los atributos que le van a servir a lo largo de lo que se desarrolla la partida. Tiene sus métodos para poder modificar sus atributos, así como inicializarlos. también tiene métodos para el control de sus mascotas.

Manual de usuario

Descripción: Detalla la información para el uso del juego.

```
public class Jugador {

    public int vida = 10;
    public int ataque = 1;
    public int victorias = 0;
    public int oro = 10;
    public Mascota[] mascotas;
    public Mascota mascotaPeleadora;
    public int indice = 0;
    public int totalMascotas = 0;
    public Jugador oponente;
    public Mascota[] mascotasUsadas = new Mascota[200];
    public int totalMascotasUsadas = 0;
    public int totalDanoCausado = 0;
    public int totalDanoRecibido = 0;
    public int oroUsado = 0;

    public Jugador(Jugador _oponente) {
        oponente = _oponente;
    }

    public void setOponente(Jugador _oponente) {
        oponente = _oponente;
    }
```

Jugador Real

```
public JugadorReal(Jugador _oponente) {
    super(_oponente);
}

public boolean comprarMascota(String _nombre) {

    boolean hayEspacios = super.comprarMascota(_nombre);
    if (hayEspacios) {
        System.out.println("Has comprado una mascota de la clase " + _nombre);
        System.out.println("Ahora tienes un total de " + oro + " de oro");

    } else {
        System.out.println("ya no tienes espacio para más mascotas, intenta vender alguna
    }

    return true;
```

Esta clase extiende de **Jugador**, tiene métodos diferentes moldeados a la interacción del programa con el jugador.

IA

```java
public class IA extends Jugador {

    public IA(Jugador _oponente) {
        super(_oponente);
        mascotas = new Mascota[5];
    }

    public void comprarMascotas(int tier) {
        boolean hayEspacios = (totalMascotas < 5);
        while (oro ≥ 3 && hayEspacios) {
            hayEspacios = comprarMascotaAleatoria(tier);
        }
    }
```

En esta clase se crea una “inteligencia artificial que compra mascotas hasta que no tenga oro, siempre compra mascotas en relación al tier de la partida.

Mascota

Esta clase maneja a la mascota en su estado simple, con muchas variables necesarias para las batallas individuales, para las fusiones, reportes y demás cosas…
También tiene muchos métodos que se ejecutan según avanza la partida.

```
public class Mascota {

    private double ataque;
    private double vida;
    public int experiencia = 0;
    public String[] tipos;
    public int nivel = 1;
    public String[] efectos = new String[30];
    public int indiceEfectos = 0;
    public String[] comidas = new String[30];
    public int indiceComidas = 0;
    public String nombre;
    public int tier = 1;
    public Mascota[] aliados;
    protected Mascota[] enemigos;
    protected int danosEfectoArmaduraMelon = 0;
    public boolean estaDesmayado = false;
    public boolean pastelitoQuitado = false;
    public int totalDanoCausado = 0;
    public int totalDanoRecibido = 0;

    public Mascota(Mascota[] _aliados, Mascota[] _enemigos) {
        this.aliados = _aliados;
        this.enemigos = _enemigos;
    }
```

Además, la clase Mascota tiene muchas más clases hijas, las cuales se encargan de manejar cada uno de los animales, por ejemplo , la clase Caballo


```java
public class Caballo extends Mascota {

    public Caballo(Mascota[] _aliados, Mascota[] _enemigos) {
        super(_aliados, _enemigos);
        setEstadisticas(2, 1, new String[] {
                NombresTipos.mamifero, NombresTipos.domestico
        }, NombresMascotas.caballo);
    }

    @Override
    public void iniciarBatalla() {
        super.iniciarBatalla();
        rugidoAliado();
    }

    public void rugidoAliado() {
        int totalAliados = HelperClass.totalMascotas(aliados);

        for (int i = 0; i < totalAliados; i++) {
            Mascota aliado = aliados[i];
            aliado.setAtaque(aliado.getAtaque() + nivel);
        }
    }

}
```

En su método constructor se define la vida y ataque que cada caballo tendrá por defecto, los tipos y su nombre. Además también se le crea su habilidad, en este caso es rugidoAliado() y esta habilidad se ejecuta en cierto momento, momento determinado por el @override, en este caso, su habilidad se ejecuta apenas inicia la batalla.

```java
public class Mapache extends Mascota {
    public Mapache(Mascota[] _aliados, Mascota[] _enemigos) {
        super(_aliados, _enemigos);
        setEstadisticas(5, 4, new String[] {
                NombresTipos.solitario
        }, NombresMascotas.mapache);
    }

    @Override
    public void morir() {
        super.morir();
        repartirPoder();
    }

    public void repartirPoder() {
        int indiceMapache = HelperClass.obtenerIndiceMascota(this, aliados);
        if (indiceMapache < 4 && aliados[indiceMapache + 1] != null) {
            aliados[indiceMapache + 1].setAtaque(aliados[indiceMapache + 1].getAtaque() * nivel);
        }
    }
}
```

MotorJuego

En esta clase se define el motor del juego, el cuál va a manejar cómo funciona el juego, ejecutando sus métodos y llevando el control de cómo se desarrolla la partida

```java
public class MotorJuego {

    public String modo;
    public Jugador jugador;
    public Jugador oponente;
    public Tienda tienda;
    public Batalla batalla;
    public Menu menu = new Menu(new String[] {
            "1- Batallar",
            "2- Acciones entre batallas",
    });
    public boolean juegoTerminado = false;
    public FileWriter log;
    PrintWriter printWriter;

    public MotorJuego() {
        jugador = new JugadorReal(null);
        tienda = new Tienda();

    }

    public void iniciarJuego() {
        jugador.mascotas = new Mascota[5];
        try {
            log = new FileWriter("log-batallas.txt");
            printWriter = new PrintWriter(log);
            printWriter.printf("El juego ha iniciado en modo %s \n", modo);
        } catch (IOException e) {
            System.out.println("Ha ocurrido un error");
            e.printStackTrace();
```

Este motor ejecuta un ciclo del juego que se sigue ejecutando por medio de la interacción del jugador hasta que uno de los dos jugadores gane el juego.

Esta clase también tiene hijos, los cuales tienen métodos externos o métodos modificados dependiendo el modo de juego que se este jugando, por ejemplo:

MotorModoArena

Dentro de este se inicializa una ia, la cual va a competir contra el jugador como oponente.

```java
public class MotorModoArena extends MotorJuego {

    public MotorModoArena() {
        super();
        modo = "Arena";
        oponente = new IA(jugador);
        jugador.setOponente(oponente);
        batalla = new Batalla(jugador, oponente);

    }

    @Override
    public boolean empezarBatalla() {

        if (jugador.totalMascotas == 0) {
            System.out.println("No puedes batallar sin mascotas, ve a la tienda a comprar");
            return true;
        }

        printWriter.printf("Empieza el turno de batalla # %d \n", batalla.numeroRonda);

        jugador.agregarMascotasUsadas();
        oponente.agregarMascotasUsadas();
        inicializarIA();
        batalla.iniciarBatalla(printWriter);
        tienda.siguienteRonda();
        jugador.actualizarAtaque(tienda.ronda);
```

Tienda.

En esta clase se maneja la tienda y todas sus funcionalidades, como la compra de mascotas, de comida; venta de mascotas y fusión de mascotas.

```
public class Tienda {

    public String[] mascotas;
    public String[] comidas;
    public int tier = 1;
    int capacidad = 3;
    public int ronda = 1;

    public Tienda() {
        mascotas = new String[capacidad];
        actualizarComidas();
        rellenarMascotas();
    }

    public void siguienteRonda() {
        actualizarTier();
        actualizarComidas();
        evolucionar();
        rellenarMascotas();
    }

    public void actualizarTier() {
        ronda++;
        if (ronda == 2 || ronda == 4 || ronda == 6 || ronda == 8 || ronda == 10 || ronda == 1
            tier++;
        }
    }
```

Batalla

Esta clase se encarga de realizar todas las acciones que tengan que ver con la batalla.

```
public class Batalla {

    public int numeroRonda = 1;
    private Jugador jugador;
    private Jugador oponente;

    public Batalla(Jugador _jugador, Jugador _oponente) {
        jugador = _jugador;
        oponente = _oponente;
    }

    public void actualizarRonda() {
        numeroRonda++;
        if (numeroRonda == 4) {
            jugador.ataque = 2;
            oponente.ataque = 2;
        }
        if (numeroRonda == 7) {
            jugador.ataque = 3;
            oponente.ataque = 3;
        }
    }

    public void verificarTotalMascotas() {
        jugador.totalMascotas = HelperClass.totalMascotas(jugador.mascotas);
        oponente.totalMascotas = HelperClass.totalMascotas(oponente.mascotas);
    }
```

Por medio del método iniciarBatalla el programa realiza la batalla, dándole al jugador a elegir un campo. aplicando las habilidades de todos los animales que tengan habilidades al inicio de la batalla. definiendo las dos mascotas que van a pelear y haciendo peleas individuales hasta que uno de los dos jugadores se quede sin mascotas y el otro gane.

Manual de usuario

Descripción: Detalla la información para el uso del juego.

```java
public void iniciarBatalla(PrintWriter printWriter) {
    Campos.seleccionarCampo(jugador, oponente);
    aplicarHabilidadesInicioBatalla();

    jugador.setMascotaPeleadora();
    oponente.setMascotaPeleadora();

    while (jugador.totalMascotas != 0 && oponente.totalMascotas != 0) {
        batallaIndividual(printWriter);
        verificarTotalMascotas();
    }

    aplicarEfectosFinal();

    verificarMascotasMuertas(jugador);
    verificarMascotasMuertas(oponente);

    if (jugador.totalMascotas == 0 && oponente.totalMascotas == 0) {
        HelperClass.imprimirTextoGuiones("Hubo un empate, ambos jugador se quedan sin masc
    } else if (jugador.totalMascotas == 0) {
        ganarBatalla(oponente, "oponente");
    } else if (oponente.totalMascotas == 0) {
        ganarBatalla(jugador, "jugador");
    }

}
```