UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
CENTRO UNIVERSITARIO DE OCCIDENTE
FACULTAD DE INGENIERÍA
CARRERA: INGENIERÍA EN CIENCIAS Y SISTEMAS

PROYECTO 1

MANUAL TÉCNICO

Docente:	Ing. Oliver Ernesto Sierra Pac
Curso:	Introducción a la Programación y C.
Estudiante:	Rudy Alessandro Reyes Oxláj
Registro Académico:	202031213
Fecha:	18/04/2022

MANUAL TÉCNICO

Algoritmos de cada función o método:

Clase Mascota:

//DEFINIENDO ATRIBUTOS DE ESTA CLASE
private String nombre;
private double ataque;
private double vida;
private String [] tipo;
private int nivel;
private Habilidad habilidad;
private Comida comida;
private int tierMascota;

método **getNombre**(){
retornar nombre;
}

método **getAtaque**(){
retornar ataque;
}

método **setAtaque**(recibir ataque){
this.ataque = ataque;
}

método **setVida**(recibir vida){
this.vida = vida
}

método **getTipo**(){
retornar tipo
}

método **getNivel**(){
retornar nivel;
}

método **setExp**(recibir exp){
this.exp = exp
}

método **getTier**(){
retornar tier;
}

método **getHabilidad**(habilidad){
this.habilidad = habilidad;
}

método **setHabilidad**(){
retornar habilidad;
}

método **getEfectoComida**(){
retornar comida
}

método **setEfectoComida**(comida){
this.comida = comida;
}

Clase Jugador:

//ATRIBUTOS DE LA CLASE JUGADOR

private int vida;

private int oro;

private int victorias;

private string nombre;

private [] mascotas;

private int contadorMascotas;

método **setVida**(recibir vida){
this.vida = vida
}

método **getVida**(){
retornar vida;
}

método **getOro()**{
retornar oro;
}

método **setOro**(oro){
this.oro = oro;
}

método **reiniciarOro**(){
oro=10;
}

método **getVictorias**(){
retornar victorias;
}

método **setVictorias**(){
victorias++;
}

método **getNombre**(){
retornar nombre;
}

método **setMascota**(mascota){
si contadorMascotas<5
	entonces this.mascotas[contador] = mascota;
de lo contrario
	imprimir “ya no se puede agregar más mascotas”
}

método **elimiarMascota**(int i){
mascotas[i] = null;
}

método **getMascota**(i){
retornar mascotas[i]
}

método **getMascotas**(){
retornar mascotas[]
}

método **pelear**(Jugador enemigo){
int posicionMascota = Util.posicionMascota(mascotas);
int posicionEnemigo = Util.posicionMascota(mascotasEnemigas);

double ataque = mascota[posicionMascota].getAtaque;

enemigo.getMascota[posicionEnemigo].setVida(-ataque);

}

Clase Habilidad:

//ATRIBUTOS DE LA CLASE HABILIDAD
String nombre;
String nombreHabilidad

método activarHabilidad(){

}

método **getNombre**(){
retornar nombre;
}

método **getNombreHabilidad**(){
retornar nombreHabilidad;
}

Clase Comida:

//ATRIBUTOS DE LA CLASE COMIDA
private string nombre;
private boolean tipoEfecto;
private int tier;

método **getNombre**(){
retornar nombre;
}

método i**sTipoEfecto**(){
retornar tipoEfecto;
}

método **activarEfecto**(){
//activar efecto de las clases heredadas
}

Clase Campo:

//ATRIBUTOS DE LA CLASE CAMPO
private string nombreCampo;

método **activarCampo**(Mascota[] mascotas){
//activa el campo de las clases heredadas
}

método **imprimirCampo**(){
mostrar el campo en pantalla
}

método **getNombreCampo**(){
retornar nombreCampo;
}

Clase ModoVersus/Creativo/Arena:

```
//LAS 3 CLASES COMPARTEN LA MISMA ESTRUCTURA
//METODOS DE CLASE
private Jugador jugador1;
private Jugador jugador2;
private Mascota[] mascotasTienda1;
private Mascota[] mascotasTienda2;
private CompraMascotas compra1;
private CompraMascotas compra2;
private int ronda=1;
private int tier=1;

private Mascota[] copiaMascotaJugador1;
private Mascota[] copiaMascotaJugador2;
private Campo campoJugador1;
private Campo campoJugador2;

método batalla(){

Hacer{
mostrar ronda en pantalla;
llamar al método iniciarBatalla();

Si(victoriasJugador1>=10 o victoriasJugador>=10){
        batallaTerminada = true;

        Si( victoriasJugador1>=10)
        imprimir el jugador 1 ha ganado
        De lo contrario
        imprimir el jugador 2 ha ganado
}

}Mientras(batallaTerminada=false)

}

método iniciarBatalla(){
llamar al método menuEntreBatallas(jugador1)
llamar al método menuEntreBatallas(jugador2)

Si (jugador1 tiene >=1 mascotas y jugador2 tiene >=1 mascotas){
        llamar al método seleccionarCampo(jugador1)
        llamar al método seleccionarCampo(jugador2)

        boolean peleaTerminada= true;
        int pelea =1;
```

```
 hacer{
   imprimir ********** PELEA " + pelea + " **********
imprimir (campoJugador1.imprimirCampo(jugador1.getNombre())
imprimir campoJugador2.imprimirCampo(jugador2.getNombre()));

imprimir la mascotas de los jugadores
jugador1.pelear(jugador2);
jugador2.pelear(jugador1);

   //AQUI EMPIEZAN LAS HABILIDADES EN BATALLAS
ActivarHabilidades.habilidadesEnBatallas();
ActivarHabilidades.habilidadesEnBatallas(jugador2.getMascotas(),jugador1.getMascotas(),jugador2.ge
tNombre());

ActivarHabilidades.habilidadesAlMorir(jugador1.getMascotas(),jugador2.getMascotas(),
jugador1.getNombre());

ActivarHabilidades.habilidadesAlMorir(jugador2.getMascotas(),jugador1.getMascotas(),
jugador2.getNombre());

   //COMPROBAMOS SI QUEDAN MASCOTAS

Si(jugador1 tiene 0 mascotas o jugador2 tiene 0 mascotas){
 peleaTerminada = false;
ronda ++;
aumentarTier();
jugador1.reiniciarOro();
jugador2.reiniciarOro();

Si(jugador1 >=1 mascotas){
 //GANO EL JUGADOR 1
imprimir que el jugador 1 ha ganado esta ronda
jugador1.setVictorias();

}de lo contrario Si (jugador 2 tiene mascotas vivas){

imprimir que el jugador 2 ha ganado la ronda
 jugador2.setVictorias();
}de lo contrario{
imprimir que es un empate
}

}Si hay animales vivos{
pelea++;
}
}mientras (peleaTerminada = true);
}
}
```

método **menuEntreBatallas**(){

int opcion = 0;

comprasJugador.llenarMascotasTienda(ronda,tier,mascotasTienda);

CompraComida.llenarComida(comidasTienda,tier);

jugador.reiniciarMascotas(copiaMascotas);

Hacer{
Imprimir ********** MENÚ ENTRE BATALLAS **********"
Imprimir 1. Comprar Mascota 2. Comprar Comida 3. Ordenar Mascotas"
4. Fusionar Mascotas 5. Vender Mascotas 6. Empezar Batalla");
opcion = Util.solicitarNumero("Digite una opcion: ",1,6);

Casos (opcion) {
caso 1:
llamar a comprasJugador.ComprarMascotas(mascotasTienda,jugador, comidasTienda);
llamar a copiarMascotas(jugador,copiaMascotas);
salir
caso 2:
llamar a CompraComida();
salir;
caso 3:
llamar a OrdenarVenderMascotas.ordenarMascotas(jugador);
llamar a copiarMascotas(jugador,copiaMascotas);
salir;
caso 4:
llamar a OrdenarVenderMascotas.fusionarMascota(jugador,mascotasTienda);
llamar a copiarMascotas(jugador,copiaMascotas);
salir;
caso 5:
llamar a OrdenarVenderMascotas.venderMascotas(jugador);
llamar a copiarMascotas(jugador,copiaMascotas);
salir
}
}
}Mientras(opcion!=6);
}
}

Clase CompraMascotas:

método **mostrarMascotasDeLaTienda**(){
llamar al método Util.mostrarMascotas(jugador);
imprimir ********** BIENVENIDO A LA TIENDA DE MASCOTAS **********"

```
        Si (Util.cantidadMascotas(mascotasTienda) >= 0) {
            Si (jugador.getOro() >= 3) {

                imprimir "ESTAS SON LAS MASCOTAS DISPONIBLES
                hacer (int i = 0; i < mascotasTienda.length; i++) {
                    Si (mascotasTienda[i] != null) {
                        imprimir las mascotas de la tienda
                    }
                }

                pedir la mascota a comprar

                Si(opcionCompra!=(Util.cantidadMascotas(mascotasTienda)+1)) {
                    Si (Util.cantidadMascotas(jugador.getMascotas()) < 4) {
                        dar la mascota comprada al jugador

                        Activamos las habilidades de las mascotas que se compran en la tienda

                        jugador.setOro(-3);
                        eliminamos la mascota que se compró de la tienda
                    } de lo contrario {
                        imprimir "Ya no tiene espacio para mas mascotas"
                    }
                }de lo contrario {
                    imprimir "Vuelva Pronto!!!
                }
            } de lo contrario {
                imprimir "No tiene suficiente oro para seguir comprando mascotas”
            }
        } de lo contrario {
            imprimir "Ya no se pueden comprar más mascotas en la tienda0";
        }
}
```

método **llenarMascotasDeLaTienda**(){

```
int animalesDisponibles = 0;
        Si(ronda <= 3){
            animalesDisponibles = 3;
        }de lo contrario si (ronda <=6){
            animalesDisponibles = 4;
        }de lo contrario{
            animalesDisponibles = 5;
        }

        desde int i=0 hasta i <animalesDisponibles; i++){
            int mascotaAleatoria = Util.generarRandom(0,desbloqueoTier(tier));
```

```
        mascotasTienda[i] = new Mascota(mascotas.getMascota(mascotaAleatoria));
    }
}

método desbloqueoTier(){
int desbloqueo=0;

    Casos (tier){
        caso 1:
            desbloqueo=7;
            salir;
        caso 2:
            desbloqueo= 15;
            salir;
        caso 3:
            desbloqueo = 26;
            salir;
        caso 4:
            desbloqueo= 34;
            salir;
        caso 5:
            desbloqueo= 42;
            salir;
        caso 6:
            desbloqueo= 51;
            salir;
        caso 7:
            desbloqueo= 53;
            salir;
    }


    retornar desbloqueo;

}
```

Clase OrdenarVenderMascotas:

```
método ordenarMascotas(){

Imprimir "********** ORDENAR MASCOTAS **********"
    mostrar las mascotas del jugador
    int cantidadMascotas = Util.cantidadMascotas(jugador.getMascotas());
    Si (cantidadMascotas > 0) {
        pedir la mascota que desea mover el jugador
        pedir la posicion a donde desea mover la mascota

        Si (mascotaSeleccionada != mascotaMovida) {
```

```
            Si (mascotaSeleccionada > mascotaMovida) {
                Mascota copiaMascota = jugador.getMascota(mascotaSeleccionada);//Aqui creamos una
copia de la mascota
                Mascota[] mascotas = jugador.getMascotas();
                Desde (nt i = mascotaSeleccionada – 1 hasta i >= mascotaMovida; i--) {
                   mascotas[i + 1] = mascotas[i];
                }
                mascotas[mascotaMovida] = copiaMascota;
            } De lo contrario {
                Mascota copiaMascota = jugador.getMascota(mascotaSeleccionada);//Aqui creamos una
copia de la mascota
                Mascota[] mascotas = jugador.getMascotas();
                Desde (int i = mascotaSeleccionada;  hasta i < mascotaMovida; i++) {
                   mascotas[i] = mascotas[i + 1];
                }
                mascotas[mascotaMovida] = copiaMascota;
            }

            Mostramos las mascotas del jugador
         } De lo contrario  {
            Mostramos las mascotas del jugador
         }

      } De lo contrario {
         Imprimir "No hay mascotas para ordenar"
      }

método fusionarMascotas{
Imprimir "********** FUSIONAR MASCOTAS **********"
      Imprimir ("1. Fusionar desde el propio mazo\t\t 2. Fusionar desde la tienda"
       pedimos la opcion
       Si(opcion==1) {
          mostramos las mascotas del jugador
          int cantidadMascotas = Util.cantidadMascotas(jugador.getMascotas());

          if (cantidadMascotas > 0) {
             pedimos la mascota 1 al jugador
              pedimos la mascota 2 al jugador

             Si  (mascota1 != mascota2) {
                Si { mascota 1 es igual a mascota 2){
                   realizarFusion(jugador, mascota1, mascota2);
                } de lo contrario {
                   Imprimir Las mascotas no son del mismo tipo"
                }
             } De lo contrario {
                imprimir "No se puede fusionar"
             }
          } De lo contrario {
```

```
            Imprimir “No hay mascotas para fusionar";
        }
    }Si(opcion == 2){
        fusionamos las mascotas desde la tienda
    }

}

método venderMascotas(){
imprimir "********** VENDER MASCOTAS **********";
    mostramos las mascotas del jugador
    pedimos la cantidad de mascotas
    Si (cantidadMascotas>=0){
        pedimos la mascota a vender

        Si(mascota1 != (cantidadMascotas+1)) {
            imprimimos "Esta mascota ha sido vendida por " + coste + " de oro";
            jugador.setOro(coste);
            activamos las habilidades de las mascotas que se venden

            eliminamos la mascota vendida del jugador

            jugador.setContadorMascotas(-1);

        }de lo contrario{
            Imprimir "Vuelva pronto ";
        }

    }De lo contrario{
        imprimir "No hay mascotas para vender"
    }
}
```

Clase comprarComida:

```
método mostrarComidaTienda{
imprimir ********** BIENVENIDO A LA TIENDA DE ALIMENTOS **********");
Si(cantidad de alimentos >=0 y las mascotas del jugador son>=0){
        imprimir ESTAS SON LOS ALIMENTOS DISPONIBLES
        Si(jugador.getOro()>=3){
            mostrar los alimentos de la tienda


            solicitar la comida a comprar

           Si opcionCompra!= (Util.cantidadAlimentos(comidasTienda)+1)){
```

```
            mostramos las mascotas del jugador

            pedimos la mascota a darle el alimento



             Si(comidasTienda[opcionCompra].isTipoEfecto()){
                darComidaEfecto(comidasTienda,opcionCompra,jugador,opcionMascota);
             }de lo contraio{
                de damos la comida a la mascota
                comidasTienda[opcionCompra]=null;
             }
          }

       }de lo contrario{
          imprimir "No tiene oro suficiente para comprar alimentos;
       }
    }de lo contrario{

       imprimir Util.cantidadAlimentos(comidasTienda)<0? "Ya no hay más alimentos en la tienda\n":
"No tiene mascotas para alimentar\n");
    }

método llenarComida(){
 creamos un objeto comidas

    desde (int i=0 hasta i<comidasTienda.length; i++){
       generamos una comida aleatoria
       comidasTienda[i] = comidas.getComida(comidaAleatoria);

    }
}

método desbloqueTier(){
int desbloqueo=0;

    Casos (tier){
       caso 1:
          desbloqueo=2;
          salir;
       caso 2:
          desbloqueo= 5;
          salir;
       caso 3:
          desbloqueo = 9;
          salir;
       caso 4:
          desbloqueo= 12;
          salir;
```

```
            caso 5:
               desbloqueo= 14;
               salir;
            caso 6:
               desbloqueo= 16;
               salir;
            caso 7:
               desbloqueo= 17;
               salir;
         }


         retornar desbloqueo;

}
```

Clase ArchivoInformacionMascotas:

```
método crearArchivo(){
archivoMascotas = new File(nombreArchivo);

         archivoMascotas.delete();

         archivoMascotas.createNewFile;
}

método escribirMascotas(){

 tratar {
            FileWriter escribir= new FileWriter(archivoMascotas,true);

            desde(int i=Util.cantidadMascotas(mascotas) hasta i>= 0; i--) {
               escribir.write("animal,"+mascotas[i].getNombre() + "\r\n");
            }
            escribir.close();

         } atrapar
         }
}

método escribirComida(){

 tratar {
            FileWriter escribir= new FileWriter(archivoMascotas,true);

            desde(int i=Util.cantidadMascotas(mascotas) hasta i>= 0; i--) {
              Si(mascotas[i].getEfectoComida()!=null) {
                  escribir.write(“comida”, nombreComida);
```

```
            }
        }
        escribir.close();

    } atrapar
    }
}

método escribirCampo(){
 tratar {
        FileWriter escribir = new FileWriter(archivoMascotas, true);
        escribir.write("campo," + campo);

        escribir.close();

    } atrapar
}

método leerArchivosMascota(){

String [] fila;
    int i=0;
    String cadena;

    tratar {
        FileReader lector = new FileReader(archivoALeer);
        BufferedReader lectura = new BufferedReader(lector);

        cadena = lectura.readLine();

        hacer mientras(cadena!=null){
            fila = cadena.split(",");
            llenarMatrizDatos(infoMascotas,fila,i);
            cadena = lectura.readLine();
            i++;
        }

    } atrapar
}
```