Universidad Rey Juan Carlos I

Escuela Técnica Superior de Ingeniería Informática

Titulación: Ingeniería de computadores

Asignatura: Laboratorio de Dispositivos Móviles

PRÁCTICA 4:

GARBAGE MAN

Nombre:

Juan Manuel Real Domínguez (nº expediente: 162)

25/01/2022

ÍNDICE:

EXPLICACIÓN JUEGO REALIZADO

El juego consiste en que eres un basurero que tiene que recoger distintos tipos de basura que aparecen a la derecha de la pantalla, cada una con un valor distinto. Esa basura la recoge el basurero y las guarda en bolsas que se mostrarán como si fuera la cola del snake, implementado así para dar feedback al usuario de que ha cargado la basura. Luego esa basura se debe de tirar a un determinado contenedor. La gracia del juego consiste en que el basurero solo puede cargar hasta 21 bolsas al mismo tiempo, y se deberá de jugar con el valor de cada basura para evitar pasarse. Al mismo tiempo, cada contenedor tendrá un valor determinado de basuras que se puede tirar, y si tiras más bolsas de las que tienes, perderás. Tampoco puedes dejar que se acumule demasiada basura o perderás.

CAMBIOS REALIZADOS

El juego implementado ha tomado como base el proyecto "EjemploJuegoPiratas" dado en clases. Se ha modificado lo máximo posible para dejar únicamente el movimiento y el JollyRoger, ya que se ha pretendido crear un juego con una mecánica totalmente distinta a la del juego Snake.

Los cambios realizados se pueden resumir en los siguientes:

- Jugabilidad y objetivos
- Cambios en el rango de movimiento
- Nuevo personaje y animación de movimiento
- Nuevos botines
- Más elementos en la IU.

Hablamos de estos cambios más en profundidad en los siguientes apartados:

JOLLYROGER

Movimiento

Para la realización de este juego, se ha limitado el movimiento del personaje en el eje y, pudiendo andar únicamente entre los valores [2,12]. Esto se ha implementado así para evitar que el personaje sea tapado por el icono de pausa. Además se ha quitado la teletransportación entre laterales del personaje. Esto se puede ver en el siguiente fragmento de código perteneciente al método *avance().*

```
//LIMITES
if(barco.x < 0)
    barco.x = 0;
if(barco.x > 9)
    barco.x = 9;
if(barco.y < 2)
    barco.y = 2;
if(barco.y > 12)
    barco.y = 12;
```

Ilustración 1 Limites y movimiento del personaje

Tirar basura

Para eliminar las bolsas de basura de la cola del JollyRoger cuando esta se tira a los contenedores, se ha añadido el siguiente método para eliminar estos elementos del arrayList. Se le pasa como parámetro el número de bolsas que se van a eliminar.

```
public void tirarbasura(int it){
    for(int i=0;i<it;i++){
        partes.remove( index: partes.size()-1);
    }
}
```

Ilustración 2 Metodo tirarBasura

Constructor

Como inicialmente el basurero aparece sin basura, se ha eliminado del constructor las sentencias que añaden elementos a la cola del JollyRoger. Únicamente se añade un elemento, que será la cabeza.

```
public JollyRoger() {
    direccion = ARRIBA;
    partes.add(new Tripulacion( x: 5,  y: 6));
    /*
    partes.add(new Tripulacion(5, 6));
    partes.add(new Tripulacion(5, 7));
    partes.add(new Tripulacion(5, 8));
     */
}
```

Ilustración 3 Constructor JollyRogger

Mundo

En esta clase se han creado numerosas variables para el empeño del juego, pero para simplificar la explicación vamos a hablar de ellas únicamente en los métodos que se hace uso.

ColocarContenedores

Esta función se llama únicamente en el constructor de esta clase. Su funcionamiento es inicializar los arrays correspondientes a contenedor y basura. Como solo habrá 4 contenedores, que se colocarán a la izquierda de la pantalla, este array será de ese tamaño. Y además, como estos serán estáticos durante toda la partida, estos serán inalterables en un futuro. Además se crean también la basura que aparecerá en cada posición Y disponible, pero se ocultará poniendo su posición X<0. Así sabremos cuales son las que se deberán de mostrar por pantalla.

```
private void colocarContenedores(){
    int y=MUNDO_ALTO-2;
    for(int i=0;i<MUNDO_ALTO;i++){
        if(i<4){
            contenedor[i]= new Botin( x: 0,y,  tipo: 5+i);
            y -=(int)MUNDO_ALTO/4;
        }

        botin[i] = new Botin( x: -10,i, tipo: 0);
    }
}
```

Ilustración 4 Colocar Contenedores

ColocarBotín

Para evitar complicaciones en el código se ha mantenido el nombre del método, pero coloca basura. El inicio es similar al implementado en un inicio, solo que como podremos tener numerosas basuras en pantalla, también se marcan como ocupadas las casillas en las que hay basura. La basura siempre va a aparecer a la derecha de la pantalla, por lo que su cordenada X siempre será 9. Además si la posición Y en la que se va a situar esta, está ocupada, busca la posición de encima, así hasta que encuentre una libre. Finalmente colocará una pieza de basura(botín según la variable) y la guardará en la posición del array correspondiente.

```
private void colocarBotin() {
    for (int x = 0; x < MUNDO_ANCHO; x++) {
        for (int y = 0; y < MUNDO_ALTO; y++) {
            campos[x][y] = false;
        }
    }

    Tripulacion parte = jollyroger.partes.get(0);
    campos[parte.x][parte.y] = true;

    for(int i = 0;i<MUNDO_ALTO;i++){
        if(i==0 || i==1)
            campos[9][i]= true;
        if(botin[i].x>0){
            campos[9][botin[i].y] = true;
        }
    }

    //Siempre van a aparecer a la derecha de la pantalla
    int botinX = 9;
    int botinY = random.nextInt(MUNDO_ALTO);
    while (true) {
        if (campos[botinX][botinY] == false)
            break;
            botinY += 1;
            if (botinY >= MUNDO_ALTO) {
                botinY = 2;
            }

    }
    botin[botinY] = new Botin(botinX, botinY, random.nextInt(maxBasura));
    botinPuesto++;
}
```

Ilustración 5 Colocar Botín

Update

Este método se puede decir que es el que más ha sido cambiado.

Para empezar, comprueba si el personaje ha colisionado con una basura o con un contenedor. En caso de que sea basura, suma el valor correspondiente a la cola del personaje, indicando así que carga con X bolsas de basura. Y en caso de que sea un contenedor, resta las bolsas correspondientes a la cola y luego suma eso a la puntuación.

```java
public void update(float deltaTime) {
    if (finalJuego)
        return;

    tiempoTick += deltaTime;

    while (tiempoTick > tick) {
        tiempoTick -= tick;
        tiempoColocar += tick;

        jollyroger.avance();
        if (jollyroger.comprobarChoque()) {
            finalJuego = true;
            return;
        }


        Tripulacion head = jollyroger.partes.get(0);
        for(int i=0;i<MUNDO_ALTO;i++){
            if (head.x == botin[i].x && head.y == botin[i].y) {
                cola += botin[i].valor;
                botin[i].x = -10;
                jollyroger.abordaje(botin[i].valor);
                botinPuesto--;
            }
            if(i<4){
                //Tiramos la basura en el contenedor correspondiente
                if(head.x == contenedor[i].x && head.y == contenedor[i].y){
                    puntuacion += contenedor[i].valor;
                    cola -= contenedor[i].valor;
                    if(cola>=0)
                        jollyroger.tirarbasura(contenedor[i].valor);
                }
            }


        }
```

Ilustración 6 Update, comprobar colisión

Posteriormente, tocan las comprobaciones para ver si se ha perdido, esto ocurrirá si la cola es menor a 0, mayor a 21 o se ha agotado el tiempo, indicando así que se nos ha acumulado la basura.

Luego hay que ver si se nos ha acumulado la basura y llevar la cuenta del tiempo para limpiarla, para ello comprobamos que el botín puesto es menor al máximo – 3, ya que tenemos 2 posiciones donde no se pone botín y 1 extra para poder moverse con tranquilidad. Si es así, restamos el tiempo y vemos si es par esta variable (esto se debe a que el tick se llama cada medio segundo y nosotros queremos restar cada segundo), si es así, entonces restamos 1 el tiempo disponible. Si no, entonces es que acabamos de iniciar la cuenta y la establecemos a 30 segundos. Y finalmente, si no se cumple que la basura se nos ha acumulado, indicamos el tiempo negativo, para saber que no lo tenemos que tener en cuenta.

```
//SI la cola es menor que 0 significa que hemos tirado mas basura de la que debemos
//SI la cola es mayor que 21, es que hemos cogido más basura de la que podemos cargar
//Si el tiempo es 0, es que hemos acumulado demasiada basura.
//EN cualquiera de los casos perdemos
if (cola<0 ||cola >21 || tiempo==0) {
    finalJuego = true;
    return;
}

//Si tenemos el num maximo de basuras, llevamos la cuenta del tiempo
if(botinPuesto==MUNDO_ALTO-3){
    restarTiempo++;
    //Restamos cada segundo, porque el tick se cuenta cada 0,5 s, por lo tanto se suma 1 cada medio segundo.
    if(tiempo>0 && restarTiempo%2==0)
        tiempo--;
    //Si el tiempo es negativo, es que tenemos que empezar la cuenta.
    else if(tiempo<0)
        tiempo=30;

    if(restarTiempo==100) //Restauramos la variable a un valor 0 para asegurarnos de que no se produzca desbordamiento
        restarTiempo=0;
}
else{
    //Si no tenemos el num maximo de basuras, ponemos un valor negativo, para diferenciarlo de que lleve la cuenta.
    tiempo=-2;
}
```

Ilustración 7 Update, comprobar perder y tiempo

A continuación, como queremos una dificultad creciente, se han establecido umbrales para aumentar esta. Siendo a la puntuación 21 cuando aparecerá un tipo nuevo de basura, puntuación 30 cuando el tiempo que aparece la basura se reduce, 41 aparece un nuevo tipo de basura y finalmente 51 y 75 que se reduce el tiempo de aparición de la basura.

Finalmente colocamos la basura nueva si ha pasado el tiempo entre la última basura que fue colocada y no se nos ha acumulado la esta.

```
        //Aumentamos la dificultad dependiendo de la puntuacion
        if(puntuacion>21 && aumentodificultad1){
            maxBasura++;
            aumentodificultad1=false;
        }
        if(puntuacion>30 && aumentodificultad2){
            nRandom--;
            aumentodificultad2=false;
        }
        if(puntuacion>41 && aumentodificultad3){
            maxBasura++;
            aumentodificultad3=false;
        }
        if(puntuacion>51 && aumentodificultad4){
            nRandom--;
            aumentodificultad4=false;
        }
        if(puntuacion>75 && aumentodificultad5){
            nRandom--;
            aumentodificultad5=false;
        }


        //Tiempo para colocar botín
        if(tiempoColocar>=nRandom && botinPuesto<MUNDO_ALTO-3){
            tiempoColocar=0;
            colocarBotin();
        }


    }
}
```

Ilustración 8 Update, aumentar dificultad y colocar botín

Pantalla Juego

En esta clase se ha modificado lo justo como para adaptar la interfaz a las nuevas implementaciones tales como la puntuación, el tamaño de la cola o las distintas basuras o contenedores que aparecen simultáneamente en la pantalla. Además de añadir una animación de movimiento de andar al personaje principal.

Mostrar Puntuación, cola y tiempo

Para ello se han modificado dos métodos, primero el método *updateRunning*, donde se recogen y actualizan las variables y los Strings correspondientes a la puntuación, tamaño de la cola y el tiempo.

```
if(antiguaPuntuacion != mundo.puntuacion) {
    antiguaPuntuacion = mundo.puntuacion;
    puntuacion = "" + antiguaPuntuacion;
    if(Configuraciones.sonidoHabilitado)
        Assets.contenedorSon.play( volume: 1);
}

//Cargamos el tamaño de la cola en la IU
if(antiguaCola != mundo.cola){
    if(Configuraciones.sonidoHabilitado && antiguaCola<mundo.cola )
        Assets.basuraSon.play( volume: 1);
    antiguaCola = mundo.cola;
    cola = ""+antiguaCola;

}

//En caso de que haya una cuenta atrás, cargamos el tiempo en la IU
if(mundo.tiempo>0){
    if(mundo.tiempo%2==0 && antiguoTiempo>mundo.tiempo ){
        if(Configuraciones.sonidoHabilitado){
            Assets.reloj1.play( volume: 1);
        }
    }
    else if(mundo.tiempo%2==1 && antiguoTiempo>mundo.tiempo){
        if(Configuraciones.sonidoHabilitado){
            Assets.reloj2.play( volume: 1);
        }
    }
    antiguoTiempo = mundo.tiempo;
    tiempo = ""+mundo.tiempo;

}
else if(mundo.tiempo<0){
    tiempo="";
    antiguoTiempo=-3;
}
```

Ilustración 9 Cargar puntuación, tamaño cola y tiempo

Y posteriormente se han mostrado por pantalla en el método *present*:

```
//IMPRIMIMOS LA PUNTUACION QUE LLEVAMOS
drawText(g, puntuacion, x: ((g.getWidth() - puntuacion.length()*20)/4)+30 , y: g.getHeight() - 42);
//Imprimimos el tamaño de la cola que tenemos
drawText(g, cola, x: ((g.getWidth()*2 - cola.length()*20)/4)+30 , y: g.getHeight() - 42);
//Si hay cuenta atrás la mostamos
drawText(g, tiempo, ((g.getWidth() - tiempo.length()*20)/2) , y: 20 );
```

Ilustración 10 Imprimir puntuación, tamaño cola y tiempo

Dibujar basura y contenedores

Esto se hace al comienzo del método *drawWorld(), de la siguiente manera:*

```
//DIbujamos las basuras que correspondan y los contenedores
for(int i=0;i<Mundo.MUNDO_ALTO;i++){
    //DIBUJAMOS LAS BASURAS
    Pixmap stainPixmap = null;
    Pixmap contePixmap= null;

    if(botin[i].x>0){
        if(botin[i].tipo== Botin.BASURA0)
            stainPixmap = Assets.basura0;
        if(botin[i].tipo == Botin.BASURA1)
            stainPixmap = Assets.basura1;
        if(botin[i].tipo == Botin.BASURA2)
            stainPixmap = Assets.basura2;
        if(botin[i].tipo == Botin.BASURA3)
            stainPixmap = Assets.basura3;
        if(botin[i].tipo == Botin.BASURA4)
            stainPixmap = Assets.basura4;

        int xb = botin[i].x * 32;
        int yb = botin[i].y * 32;
        g.drawPixmap(stainPixmap, xb, yb);
    }

    //DIBUJAMOS CONTENEDORES
    if(i<4){
        if(contenedor[i].tipo == Botin.CONTENEDOR1)
            contePixmap = Assets.contenedor1;
        if(contenedor[i].tipo == Botin.CONTENEDOR2)
            contePixmap = Assets.contenedor2;
        if(contenedor[i].tipo == Botin.CONTENEDOR3)
            contePixmap = Assets.contenedor3;
        if(contenedor[i].tipo == Botin.CONTENEDOR4)
            contePixmap = Assets.contenedor4;

        int xc = contenedor[i].x * 32;
        int yc = contenedor[i].y * 32;

        g.drawPixmap(contePixmap, xc, yc);
    }

}
```

Ilustración 11 DrawWorld, pintar basuras y contenedores

Animación basurero

Como hemos dicho anteriormente, se ha añadido una animación de caminar al basurero. Para ello se ha creado una variable real donde se suma fracciones (0.25 en este caso) para establecer el tiempo entre animaciones, de misma manera luego se hace el módulo entre 6 para saber a qué frame le toca, ya que solo hay 6 imágenes distintas. Posteriormente, dependiendo de la posición en la que se encuentre, se selecciona la imagen correspondiente y se carga.

```
//Dibujamos nuestro basurero con la animación de caminar.
animacion = (animacion+0.25)%6;
int i = (int) animacion;
Pixmap headPixmap = null;
if(jollyroger.direccion == JollyRoger.ARRIBA)
    headPixmap = arriba[i];
if(jollyroger.direccion == JollyRoger.IZQUIERDA)
    headPixmap = izq[i];
if(jollyroger.direccion == JollyRoger.ABAJO)
    headPixmap = abajo[i];
if(jollyroger.direccion == JollyRoger.DERECHA)
    headPixmap = der[i];
int x = head.x * 32 + 16;
int y = head.y * 32 + 16;
g.drawPixmap(headPixmap,  x: x - headPixmap.getWidth() / 2,  y: y - headPixmap.getHeight() / 2);
```

Ilustración 12 Draw World, animación basurero

BOTÍN

Aclarar que el nombre del objeto se ha mantenido, pero en el juego nos referimos a él como basura o contenedores. Como hemos visto hay distintos tipos de “botínes”, y lo hemos indicado creando nuevas variables globales, indicando su tipo y ya sea basura o contenedor. Y se ha implementado una función auxiliar donde se asigna un valor dependiendo del tipo de botín que sea, y esta se guarda en un atributo también añadido.

```java
public static int BASURA0=0;
public static int BASURA1=1;
public static int BASURA2=2;
public static int BASURA3=3;
public static int BASURA4=4;
public static int CONTENEDOR1=5;
public static int CONTENEDOR2=6;
public static int CONTENEDOR3=7;
public static int CONTENEDOR4=8;

public int x, y;
public int tipo;
public int valor;

public Botin(int x, int y, int tipo) {
    this.x = x;
    this.y = y;
    this.tipo = tipo;
    valor = valor(tipo);
}
```

Ilustración 13 Botín, tipos y constructor

```java
private int valor(int tipo){
    switch (tipo){
        case 0:
            return 2;
        case 1:
            return 5;
        case 2:
            return 7;
        case 3:
            return 11;
        case 4:
            return 15;
        case 5:
            return 2;
        case 6:
            return 5;
        case 7:
            return 11;
        default:
            return 17;
    }
}
```

Ilustración 14 Botín, consultar valor

Assets

Es obvio que al utilizar nuevas imágenes y sonidos se han implementado nuevos Assets en la misma clase, y luego se han tenido que añadir o modificar sus cargas en la clase *LoadingScreen.*

DIFICULTADES ENCONTRADAS Y SU SOLUCIÓN

Por suerte, debido a que en la práctica 3 ya se había hecho un trabajo similar, no se han encontrado demasiadas dificultades. Aunque si es verdad, que en este caso al querer llegar más lejos a la hora de modificar e implementar cosas nuevas, todos estos cambios han pasado factura al tener demasiados cambios y parámetros en cuenta, pero por suerte su solución ha sido sencilla ya que la consola mostraba exactamente donde se producía el error y se solucionaba de manera sencilla, ya que normalmente era un array fuera de rango o un fallo del reloj con los ticks.

CONCLUSIONES

Me ha parecido una práctica interesante para reforzar todos los conocimientos adquiridos durante la asignatura. Además al ser esta práctica individual se ha podido profundizar en conceptos o implementaciones que quizás no se trabajó en prácticas anteriores debido a que le correspondían a mi compañero. Finalmente, estoy contento con la práctica que entrego y con la labor desempeñada en la asignatura.

BIBLIOGRAFÍA

Sonidos: http://www.sonidosmp3gratis.com/

Herramienta animación protagonista:
https://store.steampowered.com/app/292230/Game_Character_Hub/

Otros assets: https://www.freepng.es/

IMÁGENES: