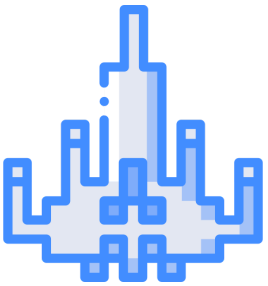
TP Final - Programación I

KINTAL GONZALO: 46021345
kintalgonzalo40@gmail.com

MIRANDA BRUNO: 43088570
brunomiranda@outlook.com.ar

RAMIREZ LUCAS: 42617176
lucas_809@live.com



LOST GALAXIAN

A continuación presentamos el juego lost galaxian en el cual consiste de una nave llamada Astro-MegaShip que va derribando enemigos llamados Destructores Estelares y esquivando asteroides. Los enemigos disparan iones que atraviesan los asteroides tratando de destruir a la Astro-MegaShip. El juego consiste en eliminar a todos los Destructores de la pantalla utilizando los disparos de la nave o llegar a 200 puntos, la Astro-MegaShip tiene 3 vidas, que las va perdiendo cada vez es impactada por un ion enemigo, pero cuidado, si un asteroide toca la nave, la destruye, independientemente de las vidas que tenga en ese momento.

Para esto implementamos las clases:
En representación del jugador se creó la clase Nave, la cual se encargará de eliminar a los Destructores y esquivar los obstáculos
►***Nave:*** En esta clase se crearon las variables instancia:
x, y: Serán las variables encargadas de darle una posición a la nave.
Escala: Gracias al método dibujarImagen() del entorno, le damos un tamaño a la nave.
Velocidad: Indica la velocidad inicial de la Astro-MegaShip
Diámetro: Modelando con círculos le damos el diámetro exacto para las colisiones.
Ángulo: Sirve para dibujar en el ángulo que se espera (En este caso, 0, ya que debe estar mirando hacia arriba.
Imagen: Se importa una imagen gracias a la herramienta Image.

Métodos:

· **dibujar(Entorno e)** → Muestra por pantalla la nave
· **moverDerecha(Entorno e)** → Aumenta la x en 2 cuando se presiona “d” o flecha a la derecha.
· **moverIzquierda()** → Aumenta la x en 2 cuando se presiona “a” o flecha a la izquierda.
· **disparar()** → Retorna un proyectil de la clase Proyectil en la parte superior de la nave.
· **subirVelocidad()** → Aumenta en 0.3 la velocidad cuando la nave recibe el item de velocidad.

►***Vidas:*** Para la representación de un ítem de vida que irá cayendo cada cierto tiempo del juego, para sumar vidas hasta un máximo de 3 e indicar en la parte superior derecha la cantidad de vidas que se posee.
En esta clase se crearon las variables instancia:

x, y: Serán las variables encargadas de darle una posición desde donde baje (únicamente varía la posición en **x**)
Escala: Gracias al método dibujarImagen() del entorno, le damos un tamaño al ícono de Vida.
Diámetro: Modelando con círculos le damos el diámetro exacto para las colisiones.
Ángulo: Sirve para dibujar en el ángulo que se espera (En este caso, 0, ya que debe estar mirando hacia arriba.
Imagen: Se importa una imagen gracias a la herramienta Image.

Métodos:

·**dibujar(Entorno e)** → Muestra por pantalla la imagen de la vida
·**bajar()**→ Va sumando valores en **y** para que la misma descienda
·**tocaNave(nave)**→ Es una booleano que devuelve true si la vida toca la nave
·**agregarVida()**→ Agrega una vida al arreglo
·**tocaAbajo()**→ Un booleano que devuelve true si toca la para inferior de la pantalla

►***Rayo:*** Para la representación de un ítem de rayo que irá cayendo cada cierto tiempo del juego, para aumentar la velocidad de la Astro-MegaShip.
En esta clase se crearon las variables instancia:
x, y: Serán las variables encargadas de darle una posición desde donde baje (únicamente varía la posición en **x**)
Escala: Gracias al método dibujarImagen() del entorno, le damos un tamaño al ítem de Rayo.
Diámetro: Modelando con círculos le damos el diámetro exacto para las colisiones.
Ángulo: Sirve para dibujar en el ángulo que se espera (En este caso, 0, ya que debe estar mirando hacia arriba).
Imagen: Se importa una imagen gracias a la herramienta Image.

Métodos:

·**dibujar(Entorno e)** → Muestra por pantalla la imagen de la vida
·**bajar()**→ Va sumando valores en **y** para que la misma descienda
·**tocaNave(nave)**→ Es una booleano que devuelve true si la vida toca la nave
·**agregarRayo()**→ Agrega un rayo al arreglo
·**tocaAbajo()**→ Un booleano que devuelve true si toca la para inferior de la pantalla

►***Proyectil:*** Esta clase representará los iones disparados tanto de aliados, como de los destructores, donde cada uno buscará eliminar a su enemigo. En esta clase se crearon las variables instancia:
x, y: Estas variables indican dónde irán aparecer el disparo. En el caso del jugador aparecerá sobre el cañón de la nave, y en el del Destructor, sobre el cañón pero mirando hacía abajo.
Escala: Gracias al método dibujarImagen() del entorno, le damos un tamaño a la nave.
Diámetro: Modelando con círculos le damos el diámetro exacto para las colisiones.
Ángulo: Sirve para dibujar en el ángulo que se espera.
Imagen: Se importa una imagen gracias a la herramienta Image dependiendo que disparo se deba dibujar, enemigo o aliado.

Métodos:

· **dibujar(Entorno e)** → Muestra por pantalla el proyectil aliado.
· **dibujarProyectilEnemigo(Entorno e)** →Muestra por pantalla el proyectil enemigo
· **moverArriba()** → Este método se usa para la la nave a la hora de disparar para que el proyectil vaya subiendo
· **moverAbajo()** →Este método se usa para la la nave enemiga a la hora de disparar para que el proyectil vaya bajando.
· **agregarProyectil(Proyectil[] arr, Proyectil p)**→ Agrega proyectiles dependiendo el arreglo puede ser para aliados o enemigos.
· **llegoAlBorde(Entorno e)**→ Booleano para saber si llego al borde de arriba
· **llegoAlBordeAbajo(Entorno e)**→ Booleano para saber si llego al borde de abajo
· **tocaObstaculo(Obstacullo obs)**→ Booleano para saber si un proyectil aliado toca un obstáculo
· **tocaEnemigo(Enemigo ene)**→ Booleano para saber un proyectil aliado toca una nave enemiga
· **tocaLaNave(Nave nave)**→ Booleano para saber si un proyectil enemigo toca la nave

►***Obstáculo:*** Esta clase representará como su nombre dice los obstáculos, en este caso, asteroides que caerán y que si impactan contra la nave, la destruirán.

En esta clase se crearon las variables instancia:
x, y: Serán las variables encargadas de darle una posición desde donde baje (únicamente varía la posición en **x**) y bajarán hasta **y = -10** para evitar la superposición con otros obstáculos o enemigos.
Escala: Gracias al método dibujarImagen() del entorno, le damos un tamaño a la nave.
Diámetro: Modelando con círculos le damos el diámetro exacto para las colisiones.
Ángulo: Indicará para qué lado se moverá el asteroide.
Velocidad: Indicará la velocidad del asteroide.
Imagen: Se importa una imagen gracias a la herramienta Image.

Métodos:

· **dibujar(Entorno e)** → Muestra por pantalla los obstáculos.
· **mover()** → Le da el movimiento a los obstáculo para que bajen
· **tocaLaNave(Nave nave)** → Un booleano para saber si un obstáculo toca la nave
· **rebotar(Entorno e)** → Le cambia el ángulo para que el obstáculo rebote
· **tocaBorde(Entorno e)** → Un booleano para saber si llego a los bordes del costado
· **tocaObstaculo(Obstaculos obs)** → Booleano para saber si toca otro obstáculo
· **tocaEnemigo(Enemigos e)** → Booleano para saber si un obstáculo toca un enemigo
· **llegoAlBordeAbajo(Entorno e)** → Booleano para saber si llego al borde de abajo
· **agregarObstaculo(Obstaculos[] obs)** → Método para agregar obstáculos al arreglo

►***Enemigo:*** Esta clase representará a los Destructores Estelares, quienes buscarán disparar con sus iones al Astro-MegaShip o colisionar con él para destruirlo.
En esta clase se crearon las variables de instancia
x, y: Serán las variables encargadas de darle una posición desde donde baje (únicamente varía la posición en **x**) y bajarán hasta **y = -10** para evitar la superposición con otros obstáculos o enemigos.
Escala: Gracias al método dibujarImagen() del entorno, le damos un tamaño a la nave.

Diámetro: Modelando con círculos le damos el diámetro exacto para las colisiones.
Ángulo: Indicará para qué lado se moverá el Destructor.
Velocidad: Indicará la velocidad del Destructor.
Imagen: Se importa una imagen gracias a la herramienta Image

Métodos:

· **dibujar(Entorno e)** → muestra por pantalla los enemigos
· **mover()** → le da el movimiento a los enemigos
· **llegoAlBorde(Entorno e)** → Booleano para saber si llego a los costados de la pantalla
· **rebotar(Entorno e)** → Método para cambiar el ángulo del enemigo
· **tocaEnemigo(Enemigos ene)** →Booleano para saber si toca a otro enemigo
· **tocaLaNave(Nave nave)**→ Booleano para saber si un enemigo toca la nave
· **tocaObstaculo(Obstaculos obs)**→ Booleano para saber si un enemigo toca un obstáculo
· **llegoAlBordeAbajo(Entorno e)**→ Booleano para saber si llego al borde de abajo
· **agregarEnemigo(Enemigos[] ene)**→ Agrega un enemigo al arreglo.

►***Fondo:*** Clase implementada únicamente para darle movimiento al fondo.

Implementación del código fuente:

```
// ------------------------------------------------------- N A V E ------------------------------------------------------

// Antes de dibujar la nave, me aseguro que no sea null
if (this.nave != null) {
    nave.dibujar(entorno);
}

// Movimiento de la nave
if (entorno.estaPresionada('a') && this.nave != null || entorno.estaPresionada(entorno.TECLA_IZQUIERDA) && this.nave != null) {
    nave.moverIzquierda();
}
if (entorno.estaPresionada('d') && this.nave != null || entorno.estaPresionada(entorno.TECLA_DERECHA) && this.nave != null) {
    nave.moverDerecha(entorno);
}

// Agrega los disparos de la nave al arreglo
if (entorno.sePresiono(this.entorno.TECLA_ESPACIO) && this.nave != null) {
    Proyectil.agregarProyectil(proyectiles, nave.disparar());
}
```

```
//------------------------------------------------- O B S T A C U L O S ----------------------------

// Dibujar los obstaculos
for (int i=0; i<obstaculos.length; i++) {
    if(obstaculos[i] != null) {
        obstaculos[i].dibujar(entorno);
    }
}

// Movimiento de los obstaculos
for (int i = 0; i < this.obstaculos.length; i++) {
    if (obstaculos[i] != null && obstaculos[i].getY() <= -10 ) {
        obstaculos[i].bajar();
    }
    if (obstaculos[i] != null && obstaculos[i].getY() >= -10) {
        obstaculos[i].mover();
    }
    if(obstaculos[i]!= null && obstaculos[i].tocaBorde(entorno)) {
        obstaculos[i].rebotar(entorno);
    }
}

// Si un obstaculo toca la nave le quita vida
for (int i = 0; i < this.obstaculos.length; i++) {
    if (obstaculos[i] != null && obstaculos[i].tocaLaNave(nave)) {
        vidas = vidas -1;
    }
}


// si un obstaculo desaparece se vuelve null
for (int i = 0; i < this.obstaculos.length; i++) {
    if (this.obstaculos[i] != null && this.obstaculos[i].llegoAlBordeAbajo(entorno))
        this.obstaculos[i] = null;
    }

// colision de obstaculo con otro obstaculo
for (int i = 0; i < this.obstaculos.length; i++) {
    for(int j = i+1; j< this.obstaculos.length; j++) {
        if (obstaculos[i] != null && obstaculos [j]!= null && obstaculos[i].tocaObstaculo(obstaculos[j])) {
            obstaculos[i].rebotar(entorno);
            obstaculos[j].rebotar(entorno);
        }else {
            Obstaculo.agregarObstaculo(obstaculos);
        }
    }
}
```

```
//------------------------------------------------- E N E M I G O S ---------------------------------

// Dibujar los enemigos
for (int i = 0; i < this.enemigos.length; i++) {
    if (this.enemigos[i] != null)
        this.enemigos[i].dibujar(entorno);
}


// Movimiento de los enemigos
for (int i = 0; i < this.enemigos.length; i++){
    if (enemigos[i] != null && enemigos[i].getY() <= -10 ) {
        enemigos[i].bajar();
    }
    if (enemigos[i] != null && enemigos[i].getY() >= -10 ) {
        enemigos[i].mover();
    }
    if (enemigos[i] != null && enemigos[i].llegoAlBorde(entorno)) {
        enemigos[i].rebotar(entorno);
    }
}

//colision de nave enemigo con nave aliada quita vida
for (int i = 0; i < this.enemigos.length; i++) {
    if (enemigos[i] != null && nave != null && enemigos[i].tocaLaNave(nave)) {
        vidas = vidas -1;
    }
}




// colision de un enemigo con otro
for (int i = 0; i < this.enemigos.length; i++) {
    for(int j = i+1; j< this.enemigos.length; j++) {
        if (enemigos[i] != null && enemigos [j]!= null && enemigos[i].tocaEnemigo(enemigos[j])) {
            enemigos[i].rebotar(entorno);
            enemigos[j].rebotar(entorno);
        }else {
            Enemigo.agregarEnemigo(enemigos);
        }
    }
}

// colision de enemigo con obstaculo
for (int i = 0; i < this.enemigos.length; i++) {
    for(int j = 0; j< this.obstaculos.length; j++) {
        if (enemigos[i] != null && obstaculos [j]!= null && enemigos[i].tocaObstaculo(obstaculos[j])) {
            enemigos[i].rebotar(entorno);
            obstaculos[j].rebotar(entorno);
        }
    }
}

// si un enemigo desaparece se vuelve null
for (int i = 0; i < this.enemigos.length; i++) {
    if (this.enemigos[i] != null && this.enemigos[i].llegoAlBordeAbajo(entorno))
        this.enemigos[i] = null;
}
```

```
//---------------------------------------------------- P R O Y E C T I L E S - A L I A D O S----------------------

// Crea los disparos de la nave los dibuja y los mueve
for (int i = 0; i < proyectiles.length; i++) {
    if (proyectiles[i] != null) {
        proyectiles[i].moverArriba();
        proyectiles[i].dibujar(entorno);
    }
}

// Eliminar proyectil si llega al borde  de arriba
for (int i = 0; i < this.proyectiles.length; i++) {
    if ((this.proyectiles[i] != null) && (this.proyectiles[i].llegoAlBorde(entorno)))
        this.proyectiles[i] = null;
}


// Si un proyectil toca un obstaculo se vuelve null
for (int i = 0; i < this.proyectiles.length; i++) {
    for (int j = 0; j < this.obstaculos.length; j++) {
        if(proyectiles[i] != null && obstaculos[j] != null && proyectiles[i].tocaObstaculo(obstaculos[j])) {
            proyectiles[i] = null;
        }
    }
}

// Si un proyectil toca un enemigo lo vuelve null al enemigo y al proyectil
for (int i = 0; i < this.proyectiles.length; i++) {
    for (int j = 0; j < this.enemigos.length; j++) {
        if(enemigos[j] != null && proyectiles[i] != null && proyectiles[i].tocaEnemigo(enemigos[j])) {
            proyectiles[i] = null;
            enemigos[j] = null;
            puntaje = puntaje + 10;
            eliminados = eliminados + 1;
        }
    }
}



  //---------------------------------------------------- P R O Y E C T I L E S - E N E M I G O S ----------------

  // cuando el contador es 100 y el y es positico el enemigo dispara
  if (contador == 100) {
        for (int i = 0; i < enemigos.length; i++) {
          if (enemigos[i] != null && enemigos[i].getY() >= 0) {
            Proyectil.agregarProyectil(proyectilesEne, enemigos[i].disparar());
            contador = 0;
          }
        }
  }

  // Crea los disparos de los enemigos los dibuja y los mueve
  for (int i = 0; i < proyectilesEne.length; i++) {
      if (proyectilesEne[i] != null) {
          proyectilesEne[i].moverAbajo();
          proyectilesEne[i].dibujarProyectilEnemigo(entorno);
      }
  }
```

```
// cuando el proyectil enemigo toca el borde de abajo se vuelve null
for (int i = 0; i < this.proyectilesEne.length; i++) {
    if (this.proyectilesEne[i] != null && this.proyectilesEne[i].llegoAlBordeAbajo(entorno)) {
        this.proyectilesEne[i] = null;
    }
}

// si un proyectil enemigo toca la nave le quita vida y si vidas es igual a 0 se vuelve null
for (int i = 0; i < this.proyectilesEne.length; i++) {
    if (proyectilesEne[i] != null && nave != null  && proyectilesEne[i].tocaLaNave(nave)) {
        vidas = vidas -1;
        proyectilesEne[i] = null;

    }
}



 //------------------------------------ ITEMS ----------------------------------------

 //--------VIDAS-----------------------
 // Agrega vidas al arreglo
 if (contadorVida == 600) {
     Vida.agregarVida(vida, 70 + Math.random()*700, 0);
     contadorVida = 0;
 }
 // Dibuja las vidas y les da movimiento
 for (int i = 0; i<vida.length; i++) {
     if(vida[i]!= null) {
     vida[i].dibujar(entorno);
     vida[i].bajar();
     } // suma vida si toca la nave y la vuelve null
     if(vida[i]!= null && vida[i].tocaNave(nave) && vidas <3) {
         vida[i]= null;
         vidas = vidas + 1;
         }
     }
 // Si llega al borde de abajo lo hace null
 for (int i = 0; i<vida.length; i++) {
     if(vida[i] != null && vida[i].tocoAbajo(entorno)) {
         vida[i] = null;
     }
 }
```

```
// Dibujar las vidas en pantalla
if( vidas == 1) {
    entorno.dibujarImagen(nav, 700, 30, 0, 0.05);
}
if( vidas == 2) {
    entorno.dibujarImagen(nav, 700, 30, 0, 0.05);
    entorno.dibujarImagen(nav, 740, 30, 0, 0.05);
}
if( vidas == 3) {
    entorno.dibujarImagen(nav, 700, 30, 0, 0.05);
    entorno.dibujarImagen(nav, 740, 30, 0, 0.05);
    entorno.dibujarImagen(nav, 780, 30, 0, 0.05);
}

// ------------- RAYO ---------------------

if (contadorRayo == 1000) {
    Rayo.agregarRayo(rayo, 70 + Math.random()*700, 0);
    contadorRayo = 0;
}

// Dibuja los rayos y les da movimiento
for (int i = 0; i<rayo.length; i++) {
    if(rayo[i]!= null) {
        rayo[i].dibujar(entorno);
        rayo[i].bajar();
    } // Suma velocidad si toca la nave y lo vuelve null
    if(rayo[i]!= null && rayo[i].tocaNave(nave)) {
        rayo[i]= null;
        nave.subirVelocidad();
    }
}

// Si llega al borde de abajo lo hace null
for (int i = 0; i<vida.length; i++) {
    if(vida[i] != null && vida[i].tocoAbajo(entorno)) {
        vida[i] = null;
    }
}
```

```
//---------------------------- F I N - D E L - J U E G O ----------------------------------------------

if(nave != null && vidas <= 0) {
    nave = null;
}

}

// Mensaje al terminar el juego si se ganó o se perdió
if ( enemigos[0] != null && enemigos[1] != null && enemigos[2] != null && enemigos[3] != null &&
        enemigos[0].getY() < 0 && enemigos[1].getY() < 0 && enemigos[2].getY() < 0 && enemigos[3].getY() < 0) {
    gano = true;
    entorno.cambiarFont("IMPACT", 70, Color.CYAN);
    entorno.escribirTexto("YOU WIN", 280, 300);
    entorno.cambiarFont("impact", 30, Color.LIGHT_GRAY);
    entorno.escribirTexto("SCORE : " + puntaje, 350, 350);
    entorno.escribirTexto("ELIMINADOS : " + eliminados, 320, 400);
}
if(puntaje >= 200) {
    entorno.cambiarFont("IMPACT", 70, Color.red);
    entorno.escribirTexto("YOU WIN", 280, 300);
    entorno.cambiarFont("impact", 30, Color.LIGHT_GRAY);
    entorno.escribirTexto("SCORE : " + puntaje, 350, 350);
    entorno.escribirTexto("ELIMINADOS : " + eliminados, 320, 400);
}

if(vidas <= 0) {
    entorno.cambiarFont("IMPACT", 70, Color.red);
    entorno.escribirTexto("GAME OVER", 250, 300);
    entorno.cambiarFont("impact", 30, Color.LIGHT_GRAY);
    entorno.escribirTexto("SCORE : " + puntaje, 350, 350);
    entorno.escribirTexto("ELIMINADOS : " + eliminados, 320, 400);
}
```

Métodos complejos de la clase Nave:

```
public Proyectil disparar() {
    return new Proyectil (x, y-diametro/2);
}
```

Métodos complejos de la clase Obstaculo:

```
public boolean tocaObstaculo (Obstaculo obs) {
    if (Math.sqrt(Math.pow((x-obs.getX()),2) + Math.pow((y-obs.getY()),2)) <= diametro/2 + obs.getDiametro()/2) {
        return true;
    }
    return false;
}


public static void agregarObstaculo(Obstaculo[] obs) {
    for(int i = 0; i < obs.length; i++) {
        if(obs[i] == null) {
            obs[i] = new Obstaculo( 150*i + 100 , -20*i  -20);

            return;
        }
    }
}
```

En los métodos tocaLaNave y tocaEnemigo se utilizó la misma fórmula que se usó en el método tocaObstaculo.

Métodos complejos de la clase Enemigo:

```
  public Proyectil disparar() {
      return new Proyectil (x, y+diametro/2);
  }

  public boolean tocaEnemigo(Enemigo ene) {
      if (Math.sqrt(Math.pow((x - ene.getX()),2) + Math.pow(( y - ene.getY()),2)) <= diametro / 2 + ene.getDiametro() / 2) {
          return true;
      }
      return false;
  }

  public static void agregarEnemigo(Enemigo[] ene) {
      for(int i = 0; i < ene.length; i++) {
          if(ene[i] == null) {
              ene[i] = new Enemigo(   150*i + 175 ,  -50*i - 100);

              return;
          }
      }
  }
```

En los métodos tocaLaNave y tocaObstaculo se utilizó la misma fórmula que se usó en el método tocaEnemigo.

Métodos complejos de la clase Proyectil:

```
// Agrega proyectil al arreglo
public static void agregarProyectil(Proyectil[] arr, Proyectil p) {
    for(int i = 0; i < arr.length; i++) {
        if(arr[i] == null) {
            arr[i] = p;

            return;
        }
    }
}


//colision del proyectil con el obstaculo
    public boolean tocaObstaculo (Obstaculo obs) {
        if (Math.sqrt(Math.pow((x-obs.getX()),2) + Math.pow((y-obs.getY()),2)) <= diametro/2 + obs.getDiametro()/2) {
            return true;
        }
        return false;
    }
```

Métodos complejos de la clase Rayo:

```
public boolean tocaNave(Nave nave) {
    if (Math.sqrt(Math.pow((x - nave.getX()),2) + Math.pow((y - nave.getY()),2)) < diametro/2 + nave.getDiametro()/2) {
        return true;
}

    return false;
}

public static void agregarRayo(Rayo[] r, double x, double y) {
    for(int i = 0; i < r.length; i++) {
        if(r[i] == null) {
            r[i] = new Rayo (x, y);

            return;
        }
    }
}
```

En la clase Vida se utilizaron los mismos métodos. Lo único que cambia es que el método para agregar se llama “agregarVida” y se le pasa como parámetro el arreglo de las vidas.

Problemas que fueron surgiendo en el transcurso del TP y sus respectivas soluciones:

Problemas al querer disparar proyectiles, al estar constantemente en el método tick() siempre se llamaba a nuestro método "llegoAlBorde()" dándonos error de index out of bounds, solucionandolo haciendo un condicional para que cuando sea Null, no corra esa parte del código.
Revisa si el proyectil detecta el borde de arriba y que sea un espacio Null con el if.
BUG: cuando un proyectil toca el borde, se eliminan proyectiles desde la nave

```
for (int i = 0; i < proyectiles.length; i++) {
    if(proyectiles[i] != null) {
    if(proyectiles[i].llegoAlBorde(entorno)) {
        Esto es para ver cuando se vuelve true al tocar

System.out.println(proyectiles[i].llegoAlBorde(entorno));
        Proyectil.eliminarProyectil(proyectiles, nave);
    }
    }
```

El problema del proyectil se pudo solucionar ya que en el código hacía dos preguntas por separado entonces cuando llegaba a la segunda condicion al estar modificada por la primera tiraba error, se soluciono haciendo un if con dos condiciones al mismo tiempo.
Cómo lo solucionamos:

```
    for(int i=0; i<this.proyectiles.length;i++) {
        if((this.proyectiles[i]!=null) &&
(this.proyectiles[i].llegoAlBorde(entorno)))
        this.proyectiles[i]=null;
    }
```

Problema con los proyectiles cuando chocan un obstáculo para que se vuelvan null, se borran antes de tiempo o sobre todo un mismo x.

```
    public void colisionaObstaculo () {
        for(int i=0; i<this.proyectiles.length;i++) {
            for(int J=0; J<this.obstaculo.length;J++)
        if ((this.proyectiles[i]!=null) &&
            (this.proyectiles[i].getY() + this.proyectiles[i].getAlto() / 2 <=
this.obstaculo[J].getY() + this.obstaculo[J].getDiametro()/2) &&
```

```
                    (this.proyectiles[i].getY() + this.proyectiles[i].getAlto() / 2 >=
this.obstaculo[J].getX() + this.obstaculo[J].getDiametro()/2))
                    this.proyectiles[i]= null;
                }
        }
```

se soluciono cuando tomamos los valores de x ya que por solor trabajar con y daba un error ahora se elimina bien el proyectil cuando toca un obstáculo y se puede volver a disparar sin problemas (linea 134-143)

como se soluciono

```
public void colisionaObstaculo () {
                for(int i=0; i<this.proyectiles.length;i++) {
                    for(int J=0; J<this.obstaculo.length;J++)
                if ((this.proyectiles[i]!=null) &&
                    (this.proyectiles[i].getX() - this.proyectiles[i].getAncho() / 2
> this.obstaculo[J].getX() - this.obstaculo[J].getDiametro()/2) &&
                    (this.proyectiles[i].getX() + this.proyectiles[i].getAncho() / 2
< this.obstaculo[J].getX() + this.obstaculo[J].getDiametro()/2) &&
                    (this.proyectiles[i].getY() + this.proyectiles[i].getAlto() / 2 <
this.obstaculo[J].getY() + this.obstaculo[J].getDiametro()/2))
                    this.proyectiles[i]= null;
                }
        }
```

Problema cuando los obstáculos colisionaban con la nave: Cuando un obstáculo tocaba la nave, la nave se volvía null pero me tiraba un error de tipo; Cannot invoke "juego.Nave.getX()" because "nave" is null. Esto sucedía porque en el método tocaLaNave (en la clase Obstáculos), una vez que la nave se volvía null, ya no se podía acceder a los atributos de la nave mediante los getters.
El método estaba de la siguiente forma:

```
public boolean tocaLaNave(Nave nave) {
                if (this.x < nave.getX() + nave.getAncho() && this.x +
this.diametro > nave.getX() &&
                    this.y < nave.getY() + nave.getAlto() && this.diametro + this.y >
nave.getY()) {
                    return true;
                }
                return false;
```

```
}
```

Cómo se solucionó:
Para solucionar este problema, lo único que tuvimos que hacer es antes de entrar en el condicional, preguntar si nave != null. Haciendo esto ya se solucionaba el problema, ya que cuando la nave se volvía null, no entraba al condicional a preguntar por los getters de la clase Nave
El código quedó de la siguiente manera:

```
public boolean tocaLaNave(Nave nave) {
		if (nave != null ) {
			if (this.x < nave.getX() + nave.getAncho() && this.x + 
this.diametro > nave.getX() &&
					this.y < nave.getY() + nave.getAlto() && 
this.diametro + this.y > nave.getY()) {
						return true;
				}
		}
		return false;
	}
```

Otro problema que tuvimos dentro del TP fue volver a crear los obstáculos y los enemigos. Para solucionar esto creamos métodos en los cuales recorrimos el arreglo y a cada obstáculo le dimos valores en x con una diferencia de 100 + 150 * la posición del obstáculo en el arreglo, por ejemplo: 100 + 150*2. Y para los enemigos hicimos lo mismo pero con diferentes valores de x para que se creen entre medio de cada obstáculo, por ejemplo:150*2 + 175

El último problema que tuvimos fue que cuando se creaban ya no se hacía uno dentro del otro, pero si al darle movimiento podía pasar que cuando uno este bajando justo se esté creando un obstáculo o un enemigo, entonces para solucionar esto creamos el método ‘bajar’ que hacía que tanto los obstáculos como los enemigos bajen recto hasta la posición y-10 y una vez que llegaban ahí recién se empezaban a mover con un ángulo de 45º.

Conclusión final:

En resumen, el desarrollo de este juego fue una experiencia gratificante para nosotros. A lo largo del proceso, aplicamos y consolidamos nuestros conocimientos en programación, resolviendo desafíos técnicos y superando obstáculos.

Durante la implementación del juego, pudimos poner en práctica conceptos fundamentales de la programación orientada a objetos, como la modularización del código, la creación de clases y la interacción entre objetos. Además, adquirimos experiencia en el manejo de gráficos y la captura de eventos para crear una experiencia de juego fluida y agradable a la vista.

Este proyecto nos enseñó la importancia del trabajo en grupo y la colaboración. A través de reuniones, asignación de tareas y comunicación constante, logramos maximizar nuestra productividad.

En definitiva, este trabajo práctico nos permitió consolidar nuestros conocimientos de programación, aplicarlos en un proyecto concreto y adquirir habilidades para el desarrollo de juegos. Estamos seguros de que esta experiencia nos ayudará para futuros desafíos en el ámbito de la programación y el desarrollo de videojuegos.