Hernández Sierra Andrea
Rangel Rada Jonathan Alejandro

Reporte final del proyecto

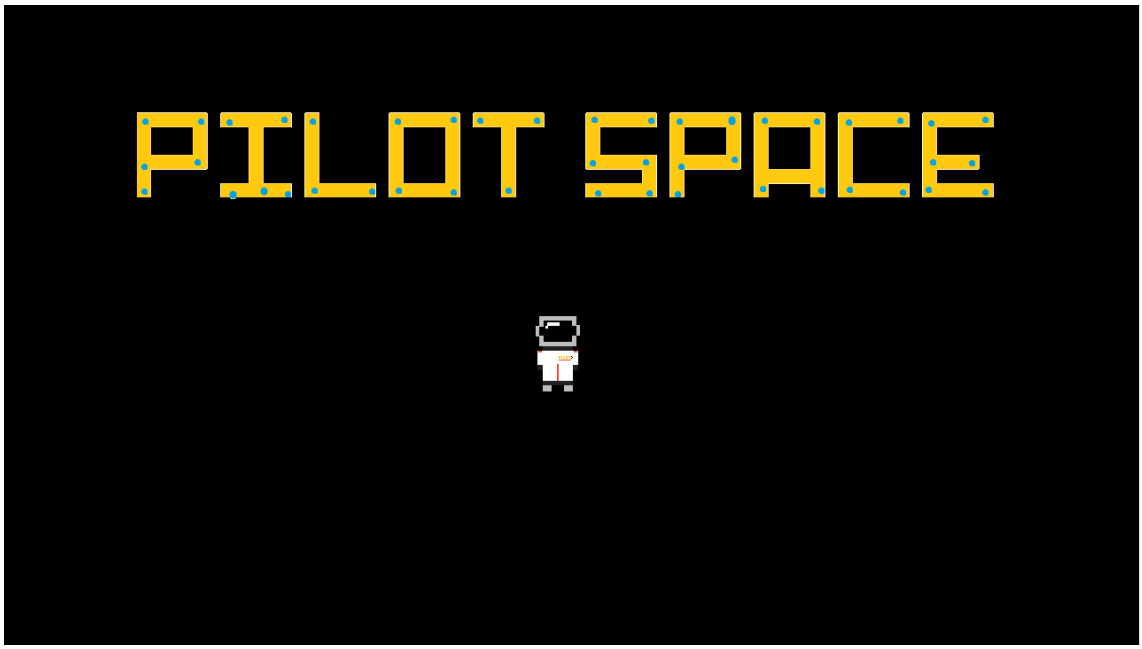
Introducción

Al cursar la materia de “Programación orientada a objetos” nos ha dado a conocer las características principales de lo que es el paradigma orientado a objetos y con esto el poder realizar un proyecto desarrollando un videojuego aplicando los conocimientos obtenidos durante el curso.

Discusión y análisis

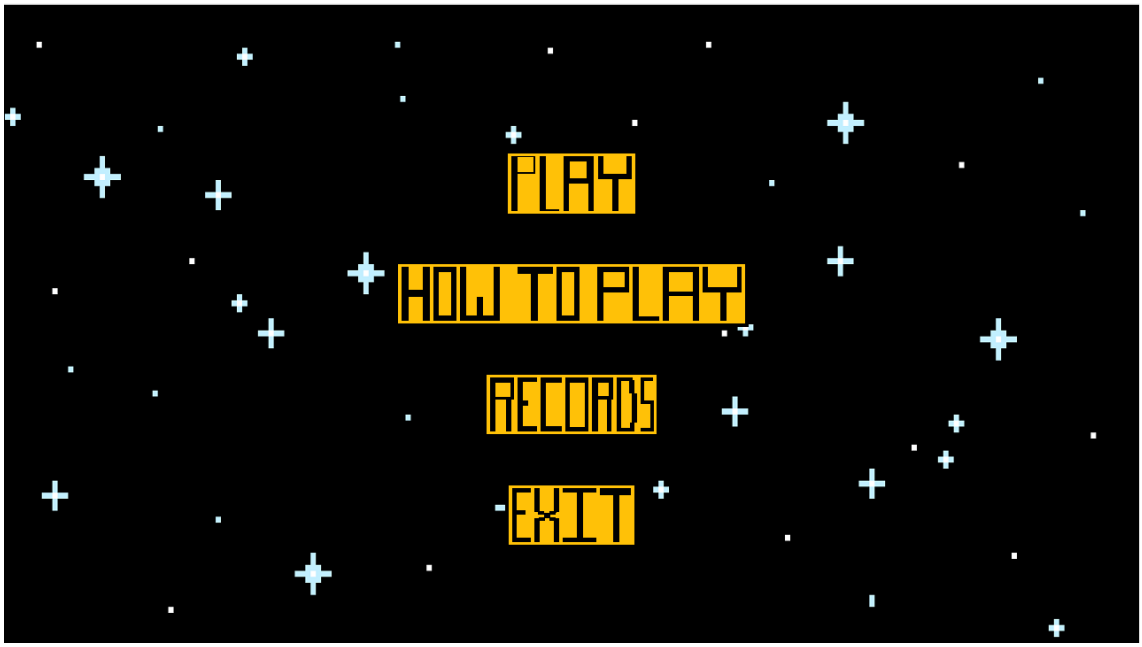
El videojuego desarrollado tiene como nombre “PilotSpace” y trata de una nave que se encuentra en el espacio y repentinamente se encuentra con naves alienígenas; una vez que se han derrotado a 3 naves aparecerá una nave enemiga más grande y con mayor vida. El objetivo es durar el mayor tiempo posible.

Una vez iniciado el juego se encuentra con esa pantalla, donde deberá dar clic para pasar a la siguiente pantalla:



Una vez dado clic nos mostrará una pantalla de inicio, donde se encontrarán cuatro botones y es ahí donde aplicamos una característica de la programación orientada a objetos: la creación e implementación de interfaces.

Estos son los botones mostrados en la pantalla:



Y esta es la interfaz y ejemplos de los métodos:

```
import greenfoot.*;

public interface ButtonAction
{
    void onClick(Actor actor);
}
```

```
public class PlayAction implements ButtonAction
{

    public void onClick(Actor actor)
    {
            Button.soundOfbotton();
            Charge charge = new Charge();
            actor.getWorld().addObject(charge,1244,678);
            charge.setRotations();
            MainTitle.stopMusic();
            Greenfoot.setWorld(new Space());
    }
}
```

```
public class HowToPlayAction implements ButtonAction
{
    public void onClick(Actor actor)
    {
        Greenfoot.setWorld(new Controls());
        Button.soundOfbotton();
    }
}
```

Se creó la interfaz “ButtonAction” en donde se encuentra el método “onClic” y es utilizado porque cada botón necesita una acción, pero es más que obvio que depende el botón es la función que va a realizar. Es donde se aplica otra característica de este paradigma, que sería el polimorfismo.

Cuando damos clic en el botón “Records” nos mostrará un listado de los mejores tiempos, aquí es donde utilizamos un “ArrayList” es utilizado para que una vez que se lee el archivo donde se guardan los records se guarda el nombre del jugador y el tiempo hecho.

Aquí es el resultado gráfico:

Y aquí el código que lee y muestra la lista:

```
public void printRecords()
{
    int i = 0;
    ArrayList<Ship> fileOfRecords = new ArrayList<Ship>();
    fileOfRecords = Space.openFile();
    Collections.sort(fileOfRecords, Ship.ShipTimeComparator);
    for(Ship fileOfRecord: fileOfRecords)
    {
        if(i < 7)
        {
            showText(fileOfRecord.getNickName(), (getWidth()/4) + 50, (getHeight()/2) + aumento);
            showText("" + fileOfRecord.getTimeInGame() + " seconds", (getWidth()/2) + 200 , (getHeight()/2) + aumento);
            aumento += 50;
        }
        i ++;
    }
}
```

Otras de las características que se utilizaron en el proyecto fueron los atributos y métodos estáticos, estos son utilizados porque son características de la clase, no de los objetos. Nos ayudan para poder llamar los métodos desde cualquier parte del programa. Estos fueron utilizados para saber el tiempo de juego y modificar atributos de las clases, así como también para guardar el juego.

Aquí ejemplos:

```
public static void save(Ship shipPlayer)
{
    File recordsFile = new File("records.txt");
    try(FileWriter writer=new FileWriter(recordsFile,true)){
        writer.append("" + shipPlayer.getTimeInGame()).append(",").append(""+shipPlayer.getNickName()).append(System.lineSeparator());
    }
    catch(IOException e){
    }
}
```

```
public static ArrayList openFile()
{

    ArrayList<Ship> fileTexts = new ArrayList<Ship>();
    try{
        List<String> lines= Files.readAllLines(Paths.get("records.txt"));
        for(String line:lines){
            Ship shipPlayers = new Ship();
            String []values=line.split(",");
            shipPlayers.setTimeInSpace(Integer.parseInt(values[0]));
            shipPlayers.setNickName(values[1]);
            fileTexts.add(shipPlayers);
        }
    } catch (IOException e) {


    }
    return fileTexts;
}
```

Aquí también se observa que utilizamos las excepciones, así como también el método try catch.

Algo que más fue utilizado en este proyecto que representa el paradigma orientado a objetos es la herencia, que es la capacidad de tener una clase padre y poder dar o “heredar” los métodos de la clase padre.

Aquí un ejemplo:

```
public class Fire extends Actor
{
    public void removeIfFireShipOutOfTheWorld()
    {
        if(isAtEdge())
            getWorld().removeObject(this);
    }
}
```

Clase padre

```
public class FireShipA extends Fire
{
    public void act()
    {
        move(5);
        removeIfFireShipOutOfTheWorld();
    }
}
```

Clase hijo

Conclusión

Haber realizado este proyecto nos ayudó a poder entender a la perfección las características principales de la programación orientada a objetos que son:

1- Polimorfismo
2- Herencia
3- Abstracción
4- Encapsulamiento

Cada una de estas características fueron utilizadas al desarrollar este proyecto.

Con este proyecto nos damos cuenta de que existen maneras muy distintas de programar, y para poder elegir la forma de hacerlo debe ser siempre preguntándonos qué queremos hacer y qué paradigma se ajusta más a nuestras necesidades.