PLAN DE PRUEBAS

JÁBEGA
TEAM

ÍNDICE

Integrantes

Nombre: Iván Díaz García, Correo: ivandiazuni@gmail.com
Nombre: Ángel Campos Salido, Correo: angelcamsal06@uma.es
Nombre: Adrian Fernandez Vera, Correo: adrianfeve@gmail.com
Nombre: Alberto Sánchez Aparicio, Correo:albertoalual03@gmail.com
Nombre: Youcef Abi Ruiz, Correo: youcefatbi@uma.es
Nombre: Manuel Ruiz Campos, Correo: manuelrc@uma.es
Nombre: Antonio Jesús Díaz Plaza, Correo:antoniojesusdiazplaza@gmail.com
Nombre: Ángel Tobaruela Baños, Correo: angeltoba@uma.es

GitHub del grupo: https://github.com/AngelUma/Jabega-Team

Trello:https://trello.com/invite/b/HS2u7KpC/ATTI56f83780148363e6d7423866ae20a00b3D028E4F/ingenieria-del-software-jabega-team

INTRODUCCIÓN - Sección 1

En este documento de plan de pruebas, se redactarán las comprobaciones realizadas en la programación para la realización del juego, desde el minuto inicial del proyecto.

PLAN DE PRUEBAS

Cambio de programa de código:

pasando de ser de Visual Studio Code a Eclipse, ya que el Visual Studio Code generaba unos problemas de colisiones y en el eclipse se simplifican estos problemas surgidos.

Cambio de código movimiento XY:

Tras intentar crear un código para que nuestro barco se moviese oblicuamente, no hemos podido conseguir que el código sea lo suficientemente lógico y estable para mantener el movimiento oblicuo, con lo que hemos decidido hacerlo lineal, en función a los ejes X e Y.

Cambios de logotipos y diseños:

El programa nos dejaba añadir ciertos pngs/jpgs, los cuales han tenido que ser reajustados dimensionalmente, ya que algunos no podían caber en la matriz donde se colocaba esa entidad.

Cambios en matrices de obstáculos:

Inicialmente, pensamos en crear una matriz de obstáculos que fuera de 1x1 píxeles, siendo así un punto en nuestro mapa, pero luego decidimos que sería mejor implementar en el código de matrices de obstáculos, una submatriz la cual pudiese ver los obstáculos como más de un píxeles, teniendo dimensiones de X x Y pixeles.

Pruebas de mockito

Para realizar las pruebas con mocks con más profundidad, necesitamos una versión más detallada y optimizada de nuestro código de juego. Para esto, supongamos que tenemos una clase Obstacle que representa los obstáculos en el juego. La clase tiene el método checkCollision() que verifica que ha ocurrido una colisión entre este obstáculo y un objeto del juego. Este método devuelve tru si hay colisión.

```
public class Obstacle {
    private int x;
    private int y;
    private int width;
    private int height;

    public Obstacle(int x, int y, int width, int height) {
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
    }

    public boolean checkCollision(Obstacle other) {
        // Verificar si hay una colisión entre este obstáculo y otro
        return (this.x < other.x + other.width &&
                this.x + this.width > other.x &&
                this.y < other.y + other.height &&
                this.y + this.height > other.y);
    }
}
```