INTEGRANTES:

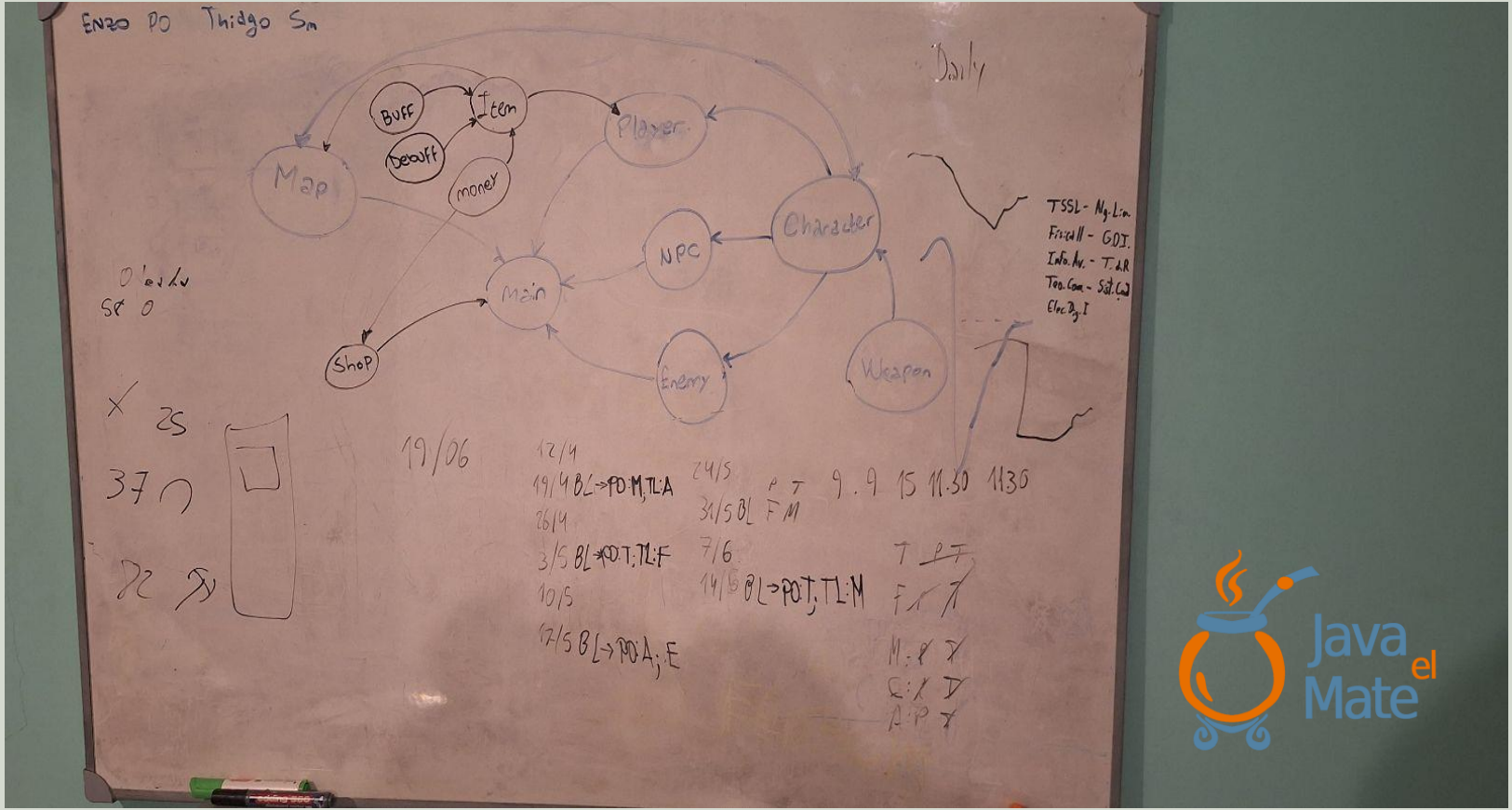
- Avila Facundo
- Biancucci Thiago
- Laura Enzo
- Rivera Alex
- Rivera Mariano

ENZITO'S MISSADVENTURES

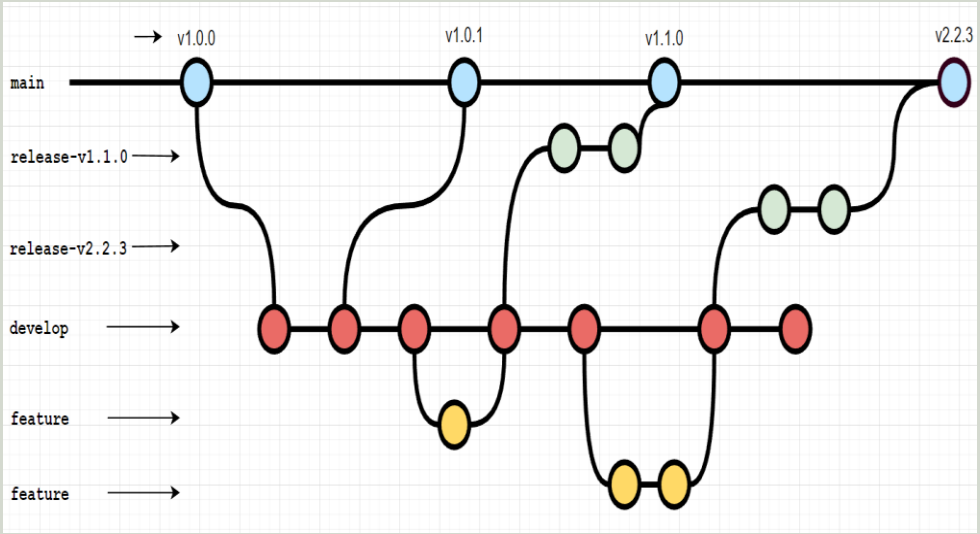
Control de versiones

Utilizamos la plataforma de GitHub para trabajar en equipo y tener el control de cada uno de nuestros cambios y poder trabajar de manera agil con gitflow.

ORGANIZACIÓN DE SPRINTS



GitFlow: Esquema de ramas

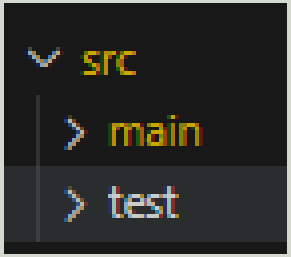


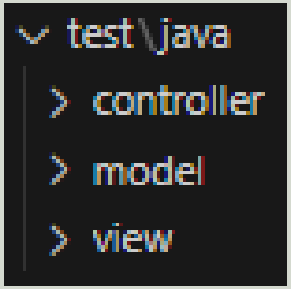
Trabajamos sobre:

- Main
- Develop
- Features
- Tests

Java el Mate

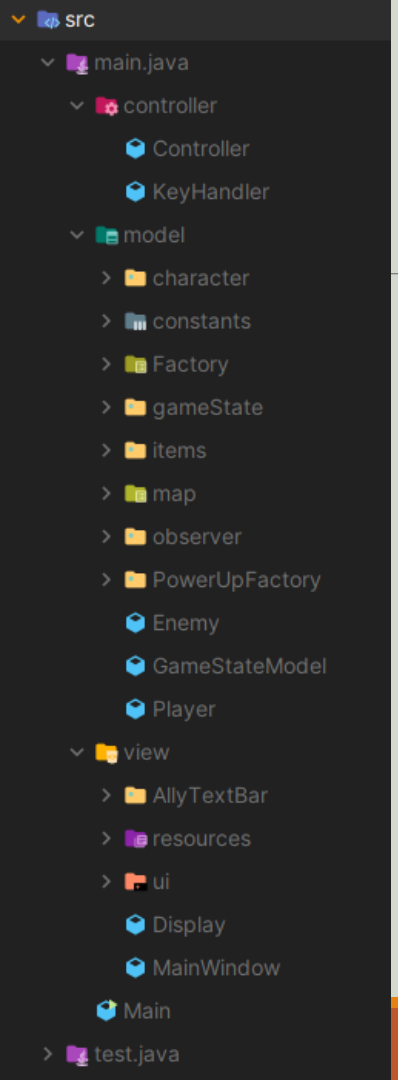
Carpetas y directorios del proyecto





Las clases fueron distribuidas en diferentes carpetas, de manera organizar y contar con una simple comprension del codigo y los patrones implementados y la arq. MVC

Java el Mate



Carpetas y directorios del proyecto

Las clases fueron distribuidas en diferentes carpetas, de manera organizar y contar con una simple comprension del codigo y los patrones de diseño y MVC implementados.

Java el Mate

REQUERIMIENTOS FUNCIONALES Y NO FUNCIONALES

Requerimientos No Funcionales

- Rendimiento mínimo de 60 FPS estables.
- Tiempo de carga mínimo (<10 seg) tanto al inicio como al cargar una partida.
- Código modular para permitir la extensión futura (armas, enemigos, mapas).
- Interfaz intuitiva, con curva de aprendizaje menor a 5 minutos. Que contenga iconos y colores llamativos para que el usuario comprenda y se oriente en el software.
- Capacidad de manejar al menos 500 enemigos simultáneos sin caídas críticas de rendimiento. (Mantener los FPS por encima de 50)
- Escalabilidad con capacidad de implementación de nuevas actualizaciones que agreguen armas, características, personajes, enemigos, etc.

Requerimientos Funcionales

- El sistema debe permitir guardar el dinero ganado en partidas.
- Controlar el movimiento del jugador mediante teclas.
- Detectar colisiones entre personajes y objetos.
- Implementar un sistema de puntuación con la puntuación actual de cada partida y la puntuación récord.
- Diseñar un menú principal con opciones.
- Gestionar niveles y progreso del jugador.

Java el Mate

Observer

- Actualizacion de los estados y el display.

```java
public interface Observer {
    void update();
}
```

```java
public interface Subject {
    void addObserver(Observer o);
    void removeObserver(Observer o);
    void notifyObservers();
}
```

```java
public class Player extends Character implements
Subject, Observer {
```

```java
@Override
public void update() {
    this.currentHp = player.getHp();
}
```

```java
@Override
public void update() {
    this.currentXP = player.getXp();
}
```

```java
public class Enemy extends NPC implements Subject {
```

```java
public void die() {
    this.alive = false;
    notifyObservers();
}

@Override
public void notifyObservers() {
    if (!alive) {
        for (Observer o : observers) {
            o.update();
        }
    }
}
```

```java
@Override
public void addObserver(Observer o) {
    if(!observers.contains(o)) observers.add(o);
}

@Override
public void removeObserver(Observer o) {
    observers.remove(o);
}
```

Patrones de Diseño

State

- Cambio entre los distintos estados y menus durante la ejecución.

Singleton

- Para asegurar una única instancia del GameMap, ScreenSettings y Player.

```java
private static ScreenSettings ScreenSINGLETON;

private ScreenSettings() {
    this.device = getDevice();
}

public static ScreenSettings getInstance() {
    if (ScreenSINGLETON == null) {
        synchronized(ScreenSettings.class) {
            if (ScreenSINGLETON == null) {
                ScreenSINGLETON = new ScreenSettings();
            }
        }
    }
    return ScreenSINGLETON;
}
```

Patrones de Diseño

Factory

- Generacion de enemigos y generacion de powerUps

```java
public interface Power {

    public void take(Player player);
```

```java
public class PowerUpFactory implements Runnable{
    private List<Power> generatedPowerUps;
```

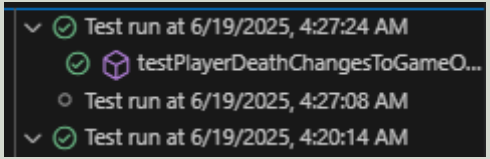
```java
if(active){
    Power powerUp;
    int[] position = generateValidPosition();
    switch (aliado.getName()) {
        case "Sergio":
            powerUp = new Choripan(position[0], position[1]);
            break;
```

```java
public class Choripan extends PowerUp implements Power {
```

Test Unitarios

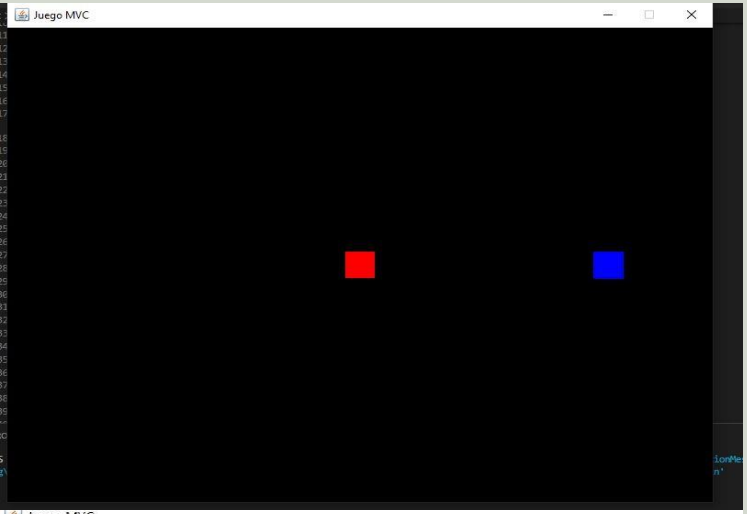
```java
public class EnemyTest {
    private Enemy enemy;
    private Player player;

    @BeforeEach
    public void setUp() {
        // Usa rutas válidas o mocks para las imágenes
        enemy = new Enemy(name:"TestEnemy", movSpeed:2, posX:100, posY:100, hp:50, bas
                su1:"src/main/java/view/resources/naranjita/naranjita_up1.png",
                su2:"src/main/java/view/resources/naranjita/naranjita_up1.png",
                sd1:"src/main/java/view/resources/naranjita/naranjita_up1.png",
                sd2:"src/main/java/view/resources/naranjita/naranjita_up1.png",
                sl1:"src/main/java/view/resources/naranjita/naranjita_up1.png",
                sl2:"src/main/java/view/resources/naranjita/naranjita_up1.png",
                sr1:"src/main/java/view/resources/naranjita/naranjita_up1.png",
                sr2:"src/main/java/view/resources/naranjita/naranjita_up1.png");
        player = Player.getInstance();
    }

    @Test
    public void testTakeDamage() {
        int initialHp = enemy.getHp();
        enemy.takeDamage(dmg:10);
        assertEquals(initialHp - 10, enemy.getHp());
    }

    @Test
    public void testDie() {
        enemy.takeDamage(dmg:1000);
        assertFalse(enemy.getIsAlive());
    }

    @Test
    public void testAttackPlayer() {
        int initialHp = player.getHp();
        enemy.attack(player);
```

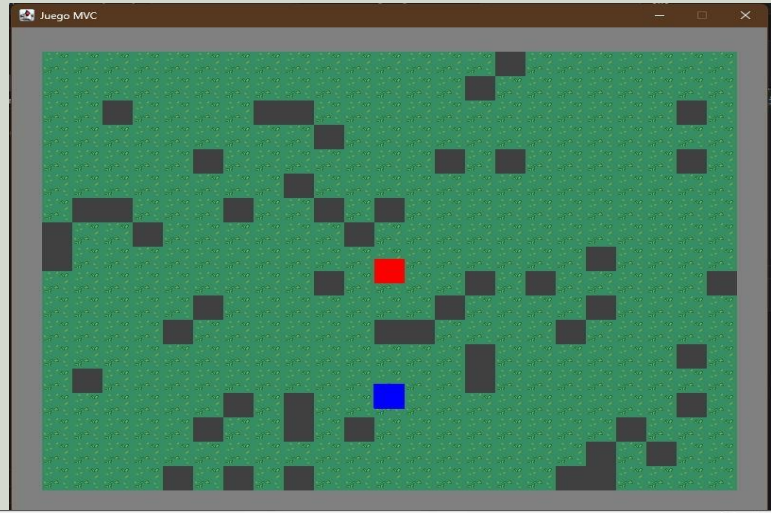
Se realizaron tests unitarios de las clases mas importantes para verificar que todo funcione y detectar y corregir errores que no contemplamos

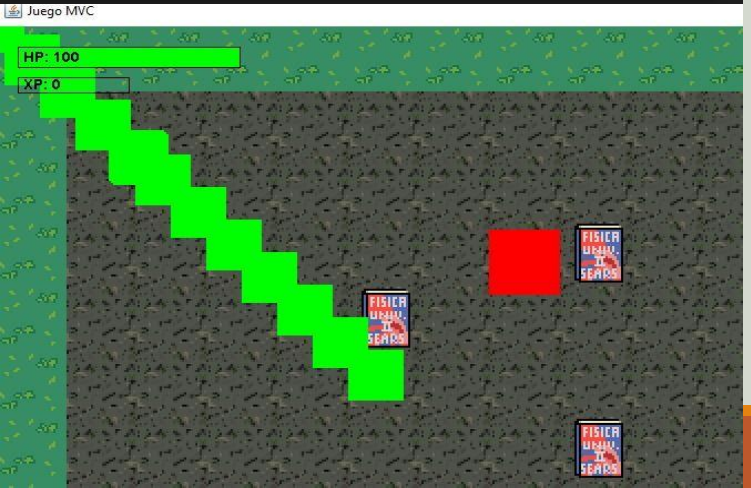


Java el Mate

AVANCE SECUENCIAL DEL PROYECTO

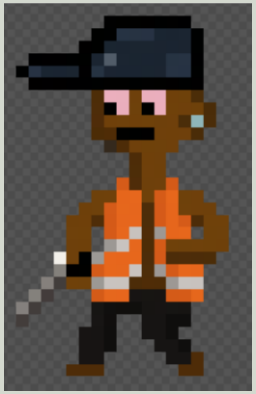






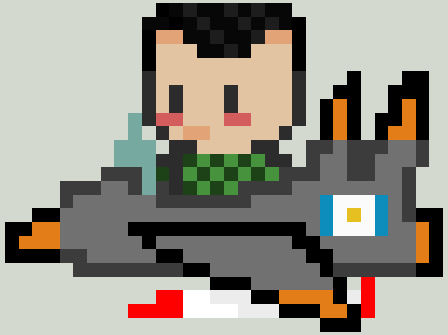
Enemigos

- Se crearon diferentes tipos de enemigos con estadisticas independientes
- Su generacion es mediante fabricas que implmentan el Factory Method



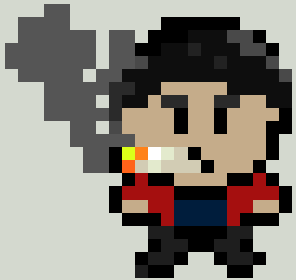
Aliados

- Se crearon diferentes tipos de aliados/maestros que nos dan distintos PowerUps
- De vida, daño, rapidez alta y baja.



Sergio

Danilo



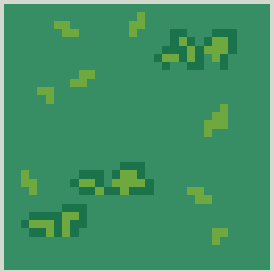
Nikitoo

JuanMa

Tiles

- Diseñamos a mano cada tile para el mapa
- Contamos con obstaculos para el jugador, region de spawn y zonas jugables.

Las cuales se ubican en la pantalla mediante una matriz de datos.

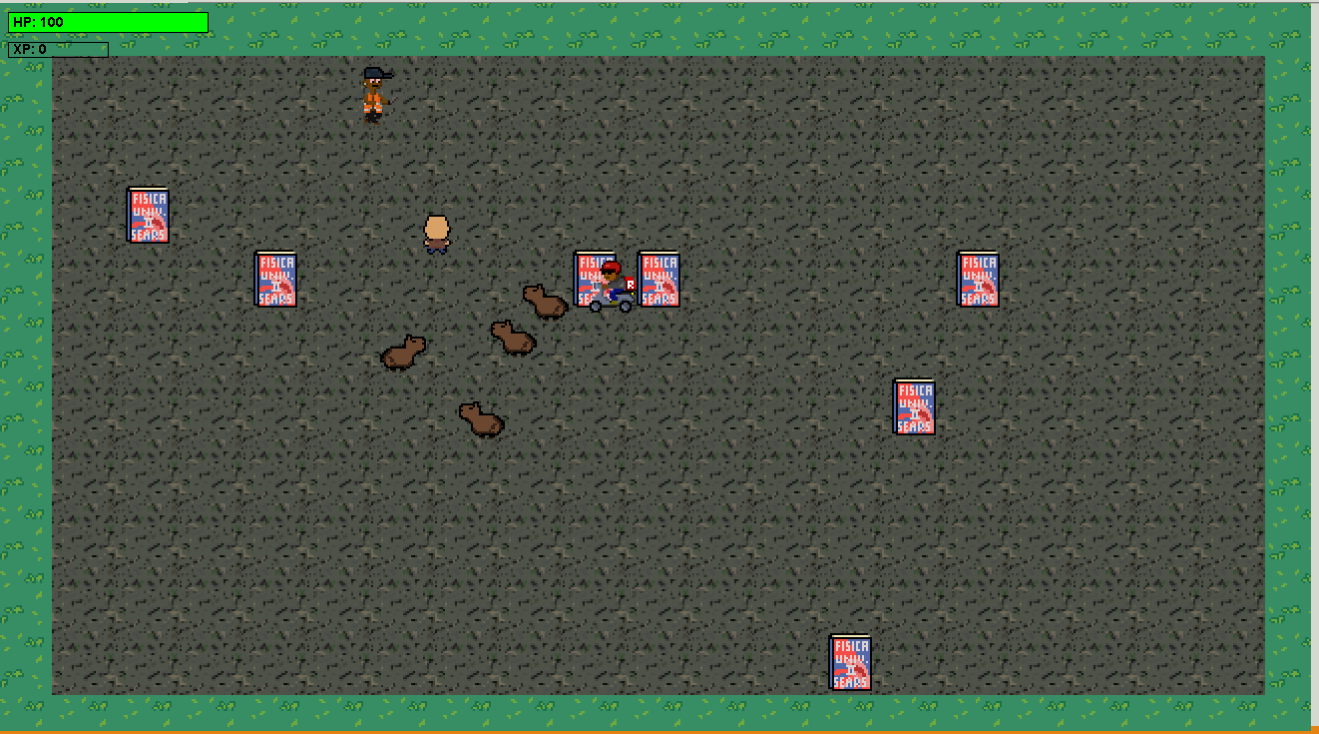


Spawn

Playable

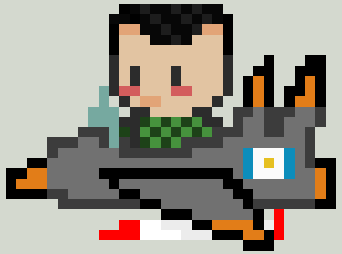
Obstacles

Jugabilidad y mapa



Se crea un mapa acorde a la pantalla preferida del Usuario.
A partir de ahi solo desplazate y sobrevive

Documentación del Proyecto

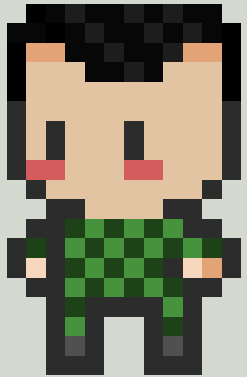


GitHub

https://github.com/lmarian0/IngenieriaDeSoftware.git

Informe

https://docs.google.com/document/d/1ym68LMc6FKvGFIPIiQARCFly1DH8z_45B9VtjKaF6Fk/edit?tab=t.0#heading=h.og1p1zqinviv



Jira

Sprint 5 - Java El Mate Game - Tablero de scrum - Jira