REVENITE CONTRA FATUM

"Risorto Contro il Fato"

Progetto di Giovanni Cifariello e Lorenzo Bellotta

Main

Il main si occupa di creare la scheda su cui andrà visualizzato l'intero gioco, di tipo JFrame.

```
public class Main {

    public static JFrame window;

    public static void main(String[] args) {

        window = new JFrame();
        window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        window.setResizable(false);
        window.setTitle("REVENITE CONTRA FATUM (DEMO)");


        GamePanel gamePanel = new GamePanel();
        window.add(gamePanel);

        gamePanel.config.loadConfig();
        if(gamePanel.fullScreenOn == true){
            window.setUndecorated(true);
        }

        window.pack();

        window.setLocationRelativeTo(null);
        window.setVisible(true);

        gamePanel.setupGame();
        gamePanel.startGameThread();
    }
}
```

GamePanel

Il GamePanel è il controller che gestisce il corretto funzionamento del gioco.

Esso genera le istanze di tutte le classi necessarie al corretto funzionamento del gioco e crea il game loop.

Il game loop consiste nel generare ogni secondo 60 volte il thread del gioco (numero scelto da noi con la variabile FPS).

Update permette a tutti le entità di muoversi, il Draw disegna tutto ciò che è visibile (ui, mappa, entità).

I vari stati di gioco (ad esempio menù, pausa, gioco, inventario eccetera) sono gestiti con dei valori final int che cambiano ogni volta il game state (e di conseguenza ciò che è visibile nello schermo).

Inoltre, aggiunge musica ed effetti sonori.

```
public void update() throws IOException{

    debug++;

    if(potionBlueDebug > 3600){
        player.attack= 2;
    }
    if(potionGreenDebug > 3600){
        player.speed= 4;
    }

    if(gameState == playState){
        player.update();
        potionBlueDebug++;
        potionGreenDebug++;
        waterCounter++;
        for(int i = 0; i < monster.length; i++){
            if(monster[i] != null){
                if(monster[i].isDied == false && monster[i].dying == false){
                    monster[i].update();
                }
                if(monster[i].isDied == true){
                    monster[i].checkDrop();
                    monster[i] = null;
                }
            }
        }
        for(int i = 0; i < projectile.length; i++){
            if(projectile[i] != null){
                if(projectile[i].isDied == false){
                    projectile[i].update();
                }
                if(projectile[i].isDied == true){
                    projectile[i] = null;
                }
            }
        }
        for(int i = 0; i < particleList.size(); i++){
            if(particleList.get(i) != null){
                if(particleList.get(i).isDied == false){
                    particleList.get(i).update();
                }
                if(particleList.get(i).isDied == true){
                    particleList.remove(i);
                }
            }
        }
    }
    if(gameState == pauseState){

    }
}
```

```
public void drawToTempScreen(){

    if(gameState == titleState){
        ui.draw(g2);
    }
    else{
        tileM.draw(g2);

        for(int i = 0; i < obj.length; i++){
            if(obj[i] != null){
                entityList.add(obj[i]);
            }
        }

        for(int i = 0; i < monster.length; i++){
            if(monster[i] != null){
                entityList.add(monster[i]);
            }
        }

        for(int i = 0; i < projectile.length; i++){
            if(projectile[i] != null){
                entityList.add(projectile[i]);
            }
        }
        for(int i = 0; i < particleList.size(); i++){
            if(particleList.get(i) != null){
                entityList.add(particleList.get(i));
            }
        }
        entityList.add(player);

        Collections.sort(entityList, new Comparator<Entity>() {
            @Override
            public int compare(Entity e1, Entity e2) {
                int result = Integer.compare(e1.worldY, e2.worldY);
                return result;
            }
        });

        for(int i = 0; i < entityList.size(); i++){
            entityList.get(i).draw(g2);
        }
        entityList.clear();

        eManager.draw(g2);

        map.drawMiniMap(g2);

        ui.draw(g2);
    }
}
public void drawToScreen(){

    Graphics g = getGraphics();
    g.drawImage(tempScreen, 0, 0, screenWidth2, screenHeight2, null);
    g.dispose();
}
```

AssetSetter

Questa classe permette di generare facilmente qualsiasi entità (oggetto o mostro) in modo molto semplice: si crea un nuovo oggetto/mostro e si settano le sue coordinate X e Y.

Nel GamePanel vengono richiamate le funzioni che li creano (setObject() e setMonster()) così da inserirli nel gioco.

Alcuni oggetti però necessitano dichiarazioni di altre loro caratteristiche:

- il cartello dell'indice da cui parte il suo testo;
- la chest del suo loot (che è un Arraylist)

```
gp.obj[i]= new OBJ_Sign(gp);
gp.obj[i].worldX = 14 * gp.tileSize;
gp.obj[i].worldY = 10 * gp.tileSize;
gp.obj[i].dialogueIndex = 0;
i++;

gp.obj[i]= new OBJ_Chest(gp);
gp.obj[i].worldX = 15 * gp.tileSize;
gp.obj[i].worldY = 11 * gp.tileSize;
loot.add(new OBJ_Key(gp));
loot.add(new OBJ_RedPotion(gp));
gp.obj[i].fillChest(loot);
loot.clear();
i++;
```

CollisionChecker

Ogni entità o tile ha una solidArea (spazio che occupa) e una variabile booleana collision (se è true vuol dire che ha le collisioni attive, altrimenti si può camminare su esso).

Ci sono diversi metodi che gestiscono le collisioni:

- **CheckTile**: verifica ogni tile, se la collisione è attiva blocca il giocatore, altrimenti può proseguire normalmente;
- **CheckObject**: verifica se l'oggetto ha le collisioni attive come il metodo precedente, ma ritorna un numero intero che permette quindi di interagire con esso (comodo per il player che usa gli oggetti in modo differente);
- **CheckEntity**: stesso funzionamento di CheckObject ma con i mostri, ritorna l'indice del mostro così che il player possa attaccarlo;
- **CheckPlayer**: usato dai mostri per colpirti.

```
public void checkTile(Entity entity){

    int entityLeftWorldX = entity.worldX + entity.solidArea.x;
    int entityRightWorldX = entity.worldX + entity.solidArea.x + entity.solidArea.width;
    int entityTopWorldY = entity.worldY + entity.solidArea.y;
    int entityBottomWorldY =  entity.worldY + entity.solidArea.y + entity.solidArea.height;

    int entityLeftCol = entityLeftWorldX/gp.tileSize;
    int entityRightCol = entityRightWorldX/gp.tileSize;
    int entityTopRow = entityTopWorldY/gp.tileSize;
    int entityBottomRow = entityBottomWorldY/gp.tileSize;

    int tileNum1, tileNum2;

    switch(entity.direction){
        case "up":
            entityTopRow = (entityTopWorldY - entity.speed)/gp.tileSize;
            tileNum1 = gp.tileM.mapTileNum[entityLeftCol][entityTopRow];
            tileNum2 = gp.tileM.mapTileNum[entityRightCol][entityTopRow];
            if(gp.tileM.tile[tileNum1].collision == true || gp.tileM.tile[tileNum2].collision == true){
                entity.collisionOn = true;
            }
            break;
        case "down":
            entityBottomRow = (entityBottomWorldY + entity.speed)/gp.tileSize;
            tileNum1 = gp.tileM.mapTileNum[entityLeftCol][entityBottomRow];
            tileNum2 = gp.tileM.mapTileNum[entityRightCol][entityBottomRow];
            if(gp.tileM.tile[tileNum1].collision == true || gp.tileM.tile[tileNum2].collision == true){
                entity.collisionOn = true;
            }
            break;
```

```
        case "left":
            entityLeftCol = (entityLeftWorldX - entity.speed)/gp.tileSize;
            tileNum1 = gp.tileM.mapTileNum[entityLeftCol][entityTopRow];
            tileNum2 = gp.tileM.mapTileNum[entityLeftCol][entityBottomRow];
            if(gp.tileM.tile[tileNum1].collision == true || gp.tileM.tile[tileNum2].collision == true){
                entity.collisionOn = true;
            }
            break;
        case "right":
            entityRightCol = (entityRightWorldX + entity.speed)/gp.tileSize;
            tileNum1 = gp.tileM.mapTileNum[entityRightCol][entityTopRow];
            tileNum2 = gp.tileM.mapTileNum[entityRightCol][entityBottomRow];
            if(gp.tileM.tile[tileNum1].collision == true || gp.tileM.tile[tileNum2].collision == true){
                entity.collisionOn = true;
            }
            break;
    }
}
```

TileManager

Permette di generare correttamente la mappa.

Dato un file di testo contenente numeri, associa ad ogni numero un tile differente e poi lo genera grande 48*48 pixel.

```
public void loadMap(String filePath){
    try{
        InputStream is = getClass().getResourceAsStream(filePath);
        BufferedReader br = new BufferedReader(new InputStreamReader(is));

        int col = 0;
        int row = 0;

        while (col < gp.maxWorldCol && row < gp.maxWorldRow){
            String line = br.readLine();
            while(col < gp.maxWorldCol){
                String numbers[] = line.split(",");
                int num = Integer.parseInt(numbers[col]);

                mapTileNum[col][row] = num;
                col++;

            }
            if(col == gp.maxWorldCol){
                col = 0;
                row++;
            }
        }
        br.close();

    }catch(Exception e){
        e.printStackTrace();
    }
}
```

Pathfinder

Il pathfinder consente al mostro, non appena viene colpito o il player si avvicina a lui, di inseguire il giocatore per attaccarlo.

Viene utilizzato l'algoritmo di A* che, con l'utilizzo di nodi, calcola il percorso più veloce che poi il mostro andrà a seguire.

Salvataggio

Le classi DataStorage e SaveLoad consentono il salvataggio dei dati principali (tutto ciò che è inerente al player, ai mostri e agli oggetti).

SaveLoad si occupa di caricare e salvare dati in un file save.dat.

Nel save vengono inizializzate tutte le variabili di DataStorage e inserite nel file save.dat.

Nel Load viene letto il file save.dat e vengono riassegnati correttamente i valori inseriti al suo interno.

```
public void save(){

    try {
        ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(new File("src/res/save/save.dat")));

        DataStorage ds = new DataStorage();

        ds.maxLife = gp.player.maxLife;
        ds.life = gp.player.life;
        ds.attack = gp.player.defaultAttack;
        ds.key = gp.player.Key;
        ds.redPotion = gp.player.RedPotion;
        ds.bluePotion = gp.player.BluePotion;
        ds.greenPotion = gp.player.GreenPotion;
        ds.worldX = gp.player.worldX;
        ds.worldY = gp.player.worldY;
        ds.direction = gp.player.direction;
        ds.lighting = gp.player.lighting;

        for(int i = 0; i < gp.player.inventory.size(); i++){
            ds.itemNames.add(gp.player.inventory.get(i).name);
        }
```

```
            ds.mapObjectNames = new String[gp.obj.length];
            ds.mapObjectWorldX = new int[gp.obj.length];
            ds.mapObjectWorldY = new int[gp.obj.length];
            ds.mapObjectLootNames = new String[gp.obj.length][20];

            for(int i = 0; i < gp.obj.length; i++){

                if(gp.obj[i] == null){
                    ds.mapObjectNames[i] = "NA";
                }
                else{
                    ds.mapObjectNames[i] = gp.obj[i].name;
                    ds.mapObjectWorldX[i] = gp.obj[i].worldX;
                    ds.mapObjectWorldY[i] = gp.obj[i].worldY;
                    for(int j = 0; j < 20; j++){
                        if(j >= gp.obj[i].chestInventory.size()){
                            ds.mapObjectLootNames[i][j] = "NA";
                        }
                        else{
                            ds.mapObjectLootNames[i][j] = gp.obj[i].chestInventory.get(j).name;
                        }
                    }
                }
            }

            oos.writeObject(ds);

        } catch (IOException e) {
            System.out.println("Save exception!");
        }
    }
```

```java
public void load(){

    try{
        ObjectInputStream ois = new ObjectInputStream(new FileInputStream(new File("src/res/save/save.dat")));

        DataStorage ds = (DataStorage)ois.readObject();

        gp.player.maxLife = ds.maxLife;
        gp.player.life = ds.life;
        gp.player.attack = ds.attack;
        gp.player.Key = ds.key;
        gp.player.RedPotion = ds.redPotion;
        gp.player.BluePotion = ds.bluePotion;
        gp.player.GreenPotion = ds.greenPotion;
        gp.player.worldX = ds.worldX;
        gp.player.worldY = ds.worldY;
        gp.player.direction = ds.direction;
        gp.player.lighting = ds.lighting;

        gp.player.inventory.clear();
        for(int i = 0; i < ds.itemNames.size(); i++){
            gp.player.inventory.add(getObject(ds.itemNames.get(i)));
        }
```

```
        for(int i = 0; i < gp.obj.length; i++){

            if(ds.mapObjectNames[i].equals("NA")){
                gp.obj[i] = null;
            }
            else{
                gp.obj[i] = getObject(ds.mapObjectNames[i]);
                gp.obj[i].worldX = ds.mapObjectWorldX[i];
                gp.obj[i].worldY = ds.mapObjectWorldY[i];
                if (ds.mapObjectNames[i].equals("Chest")){
                    for(int j = 0; j < 20; j++){
                        if(ds.mapObjectLootNames[i][j].equals("NA") == false){
                            gp.obj[i].chestInventory.add(getObject(ds.mapObjectLootNames[i][j]));
                        }
                        else{
                            break;
                        }
                    }
                }
            }
        }

    } catch (IOException e){
        System.out.println("Load exception!");
    } catch (ClassNotFoundException ex) {
        System.out.print("Not found!");
    }
```

EventHandler

Questa classe implementa una serie di eventi che si verificano quando vengono calpestati determinati tile specifici.

Questi eventi si verificano solo se il personaggio si trova in una certa direzione (se si scrive però any vanno bene tutte).

Gli eventi implementati sono il checkpoint automatico, l'aggiunta o la rimozione del buio e incremento della vita (dato dall'acqua).

```
/*if(hit(x, y, dir) == true){
    evento(col, row);
}*/

if(hit(32,29,"any") == true){
    lightingOn(32,29);
}
if(hit(33,29,"any") == true){
    lightingOn(33,29);
}
```

Altre Classi

Lighting: gestisce luce e buio.

KeyHandler e MouseHandler: implementano per ogni stato del gioco i vari tasti del mouse o della tastiera che servono al giocatore per interagire.

UI: gestisce tutta l'interfaccia utente nei vari stati del gioco.

Config: permette di modificare il volume di musica e suoni e di scegliere se mettere il gioco a tutto schermo oppure no (salva tutte queste informazioni in un file di testo config.txt)

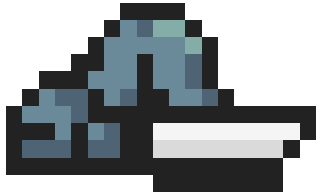
Map: stampa a video la minimappa.

Entità

Ogni oggetto, ogni mostro e il personaggio sono entità perché estendono la classe entity.

Ognuno però ha le sue caratteristiche.

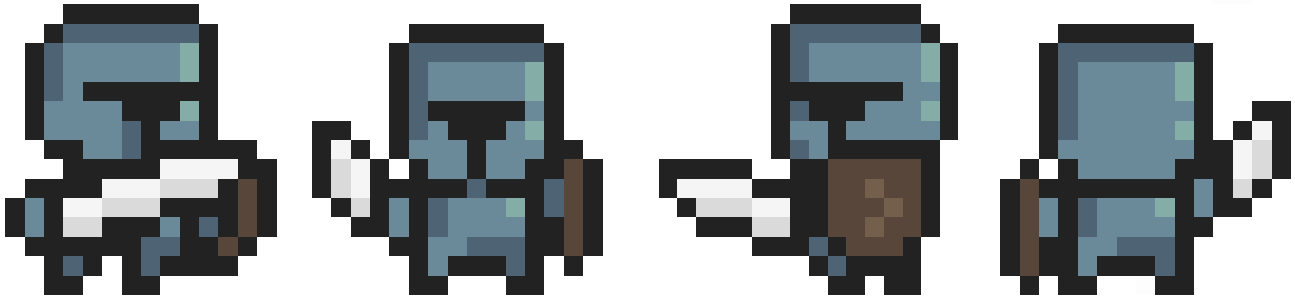
Player



Il player è il personaggio principale, quello guidato dal giocatore con l'utilizzo di "WASD".

Oltre a settare con diversi metodi i suoi valori (per salvataggio), egli può attaccare i nemici (premendo tasto sinistro del mouse quando gli è vicino) e può interagire con gli oggetti (metodo pickUpObject()).

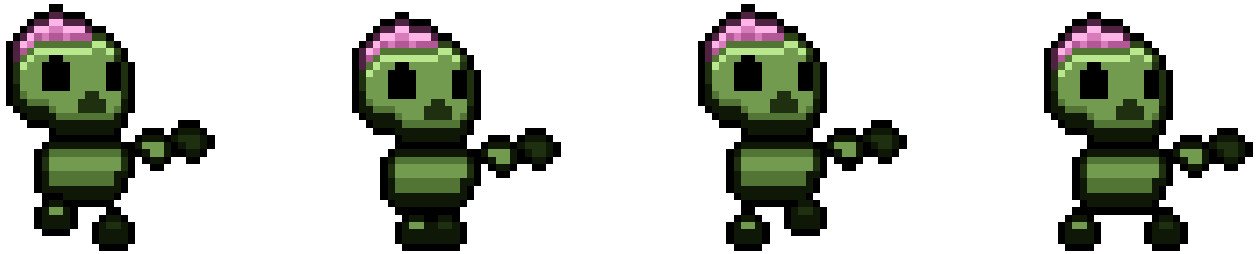
Può anche aprire l'inventario (che è un Arraylist di oggetti) e usare gli oggetti raccolti.



Zombie

Uno zombie, lento e con basso raggio d’azione.

Fa 1 di danno.

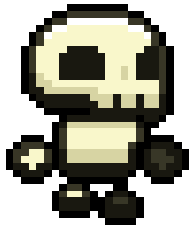


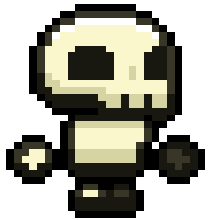
Skeleton e Arrow

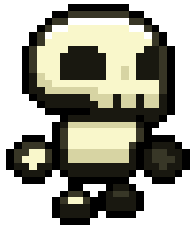
Uno Scheletro, quello con più vita.

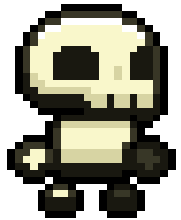
Fa 1 di danno, ma randomicamente lancia nella direzione in cui guarda una freccia, che vola finche non colpisce qualcosa, finisce la sua vita (che decrementa ogni secondo) o viene distrutta dal player.

La freccia fa 2 di danno





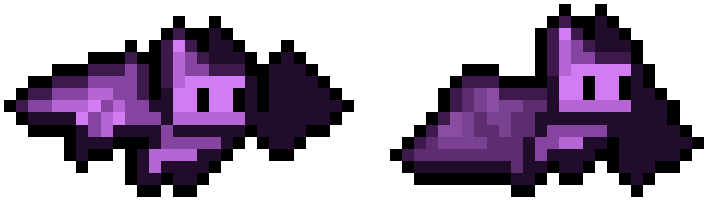




Bat

Un pipistrello, con poca vita ma molta velocità.

Fa 1 di danno.



Pozioni

Rossa: dà 2 di vita al giocatore.

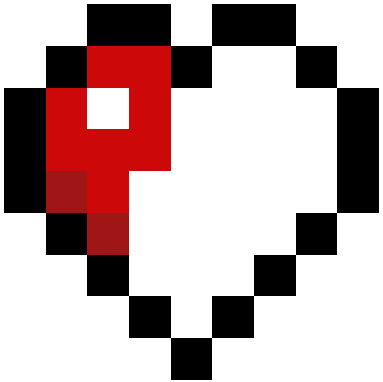
Blu: per 1 minuto, l’attacco del player aumenta.

Verde: per 1 minuto, la velocità del player aumenta.

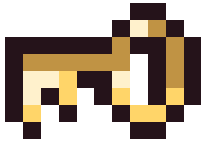
Pezzo di cuore

Unico drop dei mob per ora aggiunto.

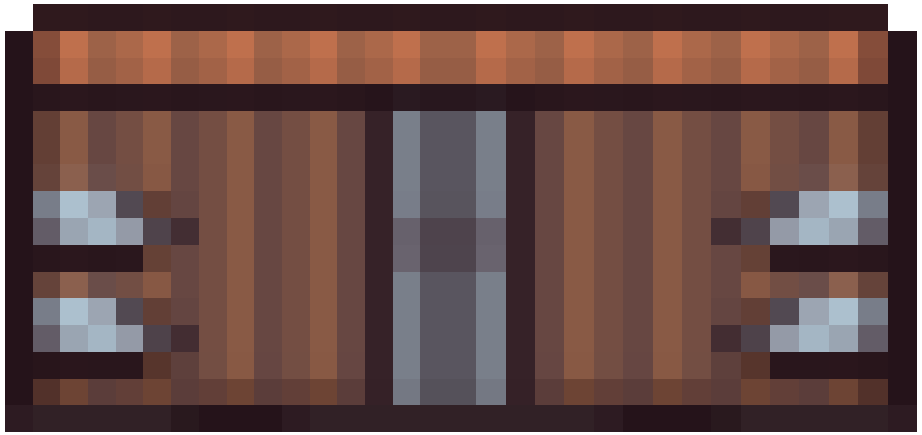
Dà al player 1 di vita una volta raccolto.



Porta



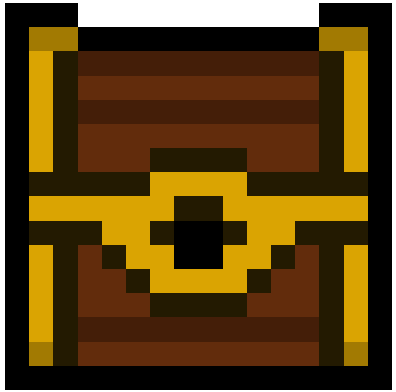
Blocca il passaggio; si può aprire solo se si possiede almeno una chiave (non specifica).



Chest

Una volta aperta col tasto “E” si può raccogliere quello che c’è al suo interno.

ATTENZIONE: se l’inventario è pieno, non si potrà raccogliere tutto!!



Cartello

Premendo “E” si può leggere il suo contenuto.