Game repport

shadow adventure

january 2025

present by:
BOUMECHHOUR Sabrine
CHARIF Mehdi Zoheir

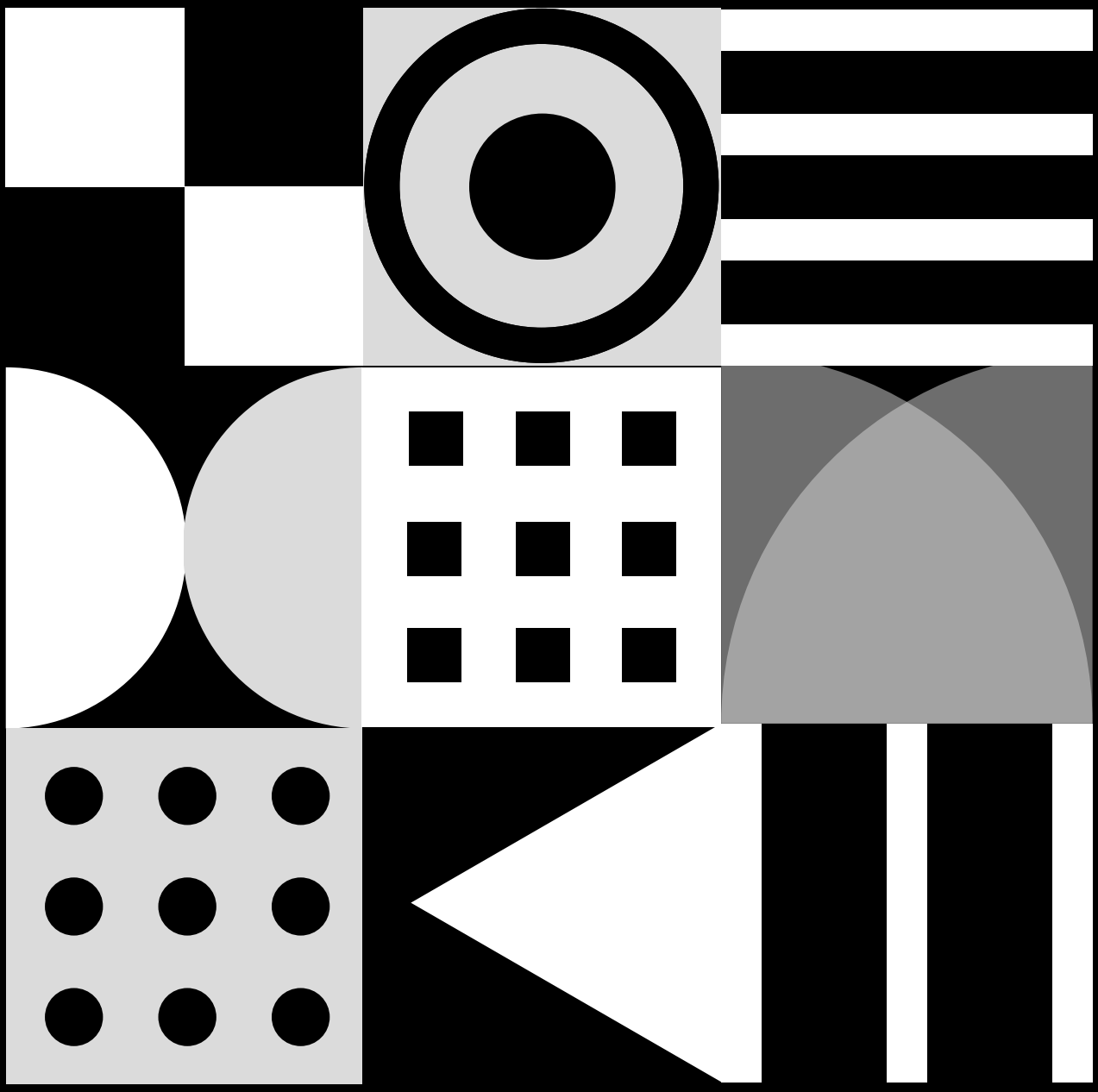
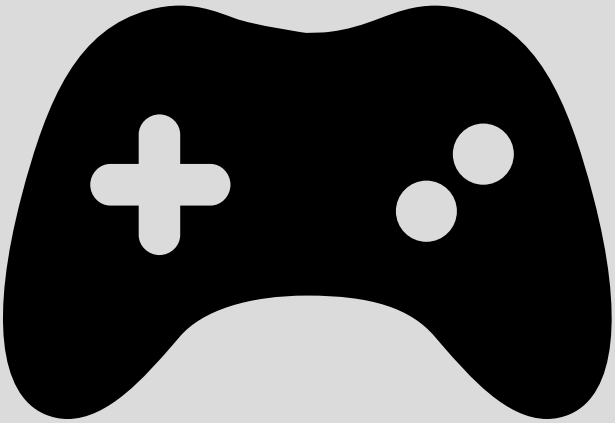


Table of Contents

Introduction

This report provides a detailed analysis of the 2D game developed in Java. The game features simple yet engaging gameplay, with a playable main character, enemies, dynamic animations, and customizable functionalities. The primary goal of the game is to find a hidden treasure while overcoming various obstacles and enemies.

General Description of the Game



To move right, press D.
To move left, press A.
To move up, press W.
To move down, press S.
Display the menu, press Esc.
Display the player's data, press C.
Throw fireballs, press F.
To attack , press Enter.
To defend, press Space.
To change arm ,press Enter after select the new arm and press enter.

Genre : 2D Adventure Game.

Plateforme : PC.

Langage de programmation : Java.

Objectif principal : Explore the world to find a treasure guarded by enemies (and for us, the programmers, to gain experience in object-oriented programming).

game mechanics

01 | Main Character:

- Playable Character: Only one character is playable.
  1.Animations:
  2.Walking
  3.Attacking
- Taking damage
  1.Dying
  2.Health Points (HP):
- Reduced by enemy attacks.
- Regenerated with potions found in the world.

02 | Enemies:

- Type of Enemy: Only one type of enemy.
- Behavior: Attacks the player to reduce HP.

03 | Inventory:

- Items: The player can possess multiple weapons, such as an axe.
- Access: The inventory can be accessed at any time to switch weapons or check collected items.

04 | Customization:

Game Settings: The player can modify certain parameters like sound volume.

main features

Health Regeneration:

- Regeneration Potions: Regeneration potions are scattered throughout the world.
- Effect: When a potion is collected, it increases the player's health points.

Combat System:

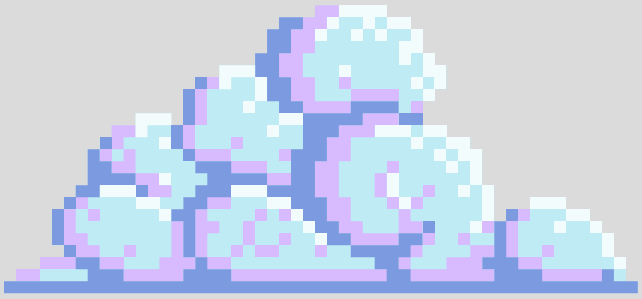
- Attacks: The player can attack enemies with weapons available in the inventory.
- Enemy Response: Enemies react to attacks and deal damage to the player on contact

Exploration:

- World Design: The world is designed to encourage exploration.
- Treasure: The treasure is hidden in an area protected by enemies.
- Collectibles: The player can collect coins.

Animations:

- Smooth Animations: Fluid animations enhance immersion in the game.
- Action Animations: Every action of the character (walking, attacking, dying, etc.) is accompanied by a corresponding animation.



Technical Features:

Graphics:

- Style: Minimalistic 2D style.
- Sprites: Use of sprites for characters and enemies.

Sound:

- Sound Effects: Sound effects for main actions (attacking, taking damage, etc.).

```
public Sound() {

    soundURL[0] = getClass().getResource("/sound/BlueBoyAdventure.wav");
    soundURL[1] = getClass().getResource("/sound/coin.wav");
    soundURL[2] = getClass().getResource("/sound/fanfare.wav");
    soundURL[3] = getClass().getResource("/sound/unlock.wav");
    soundURL[4] = getClass().getResource("/sound/powerup.wav");
    soundURL[5] = getClass().getResource("/sound/hitmonster.wav");
    soundURL[6] = getClass().getResource("/sound/receivedamage.wav");
    soundURL[7] = getClass().getResource("/sound/levelup.wav");
    soundURL[8] = getClass().getResource("/sound/cursor.wav");
    soundURL[9] = getClass().getResource("/sound/burning.wav");
    soundURL[10] = getClass().getResource("/sound/gameover.wav");
    soundURL[11] = getClass().getResource("/sound/stairs.wav");
    soundURL[12] = getClass().getResource("/sound/blocked.wav");
    soundURL[13] = getClass().getResource("/sound/parry.wav");
}
```

- Class Using the Game Sounds: Sound Class: A class that handles the different sounds in the game.
- Customizable Soundtrack: Customizable Soundtrack: The soundtrack can be customized via the game settings
-

Game Engine:

- Collision Management: Handling collisions between the character, enemies, and objects.

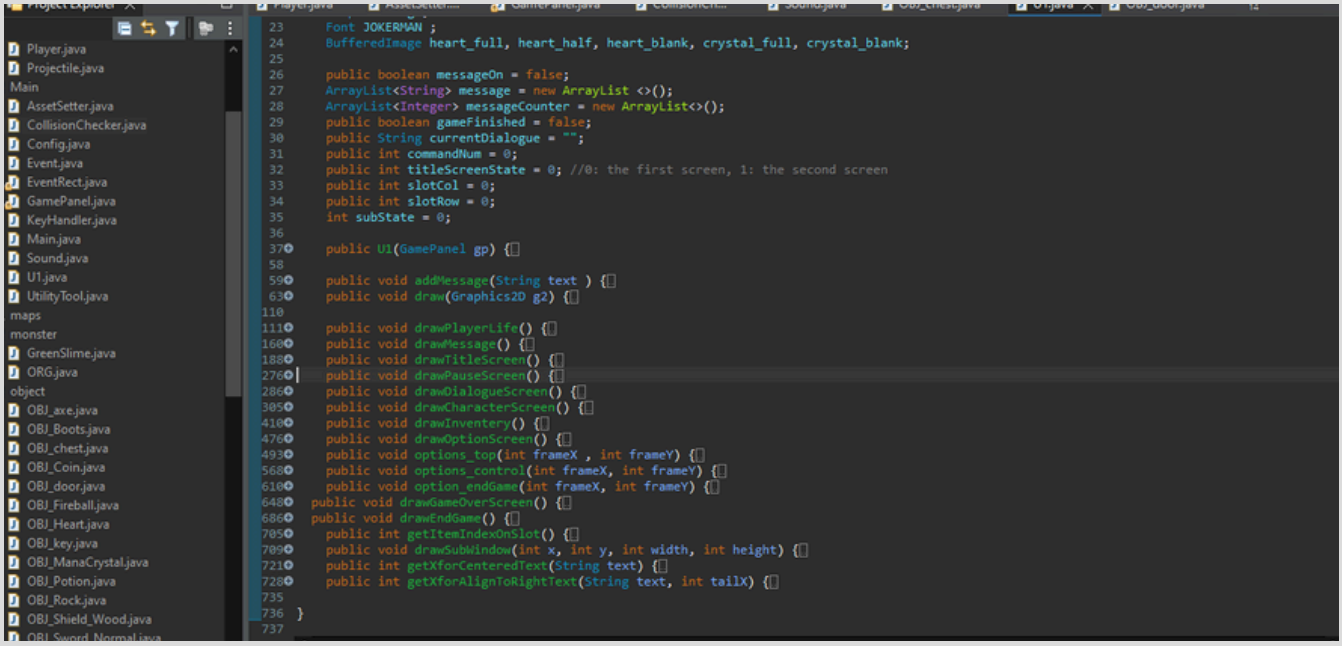
```
public int checkObject(Kentity kentity, boolean player) {

    int index = 999;

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

            // get kentity's solid area position
            kentity.solidArea.x = kentity.worldX + kentity.solidArea.x;
            kentity.solidArea.y = kentity.worldY + kentity.solidArea.y;
            //get the object's solid area position
            gp.obj[i].solidArea.x = gp.obj[i].worldX + gp.obj[i].solidArea.x;
            gp.obj[i].solidArea.y = gp.obj[i].worldY + gp.obj[i].solidArea.y;

            switch(kentity.direction) {
            case"up":    kentity.solidArea.y -= kentity.speed;   break;
            case"down":  kentity.solidArea.y += kentity.speed;   break;
            case"left":  kentity.solidArea.x -= kentity.speed;   break;
            case"right":kentity.solidArea.x += kentity.speed;   break;
                }

            if(kentity.solidArea.intersects(gp.obj[i].solidArea)) {
                if(gp.obj[i].collision == true) {
                    kentity.collisionOn = true;
                }
                if(player == true) {
                    index = i;
                }
            }

            kentity.solidArea.x = kentity.solidAreaDefaultX;
            kentity.solidArea.y = kentity.solidAreaDefaultY;
            gp.obj[i].solidArea.x = gp.obj[i].solidAreaDefaultX;
            gp.obj[i].solidArea.y = gp.obj[i].solidAreaDefaultY;
        }
```

```
switch(direction) {
case"up" :
    KentityTopRow = (kentityTopWorldY - kentity.speed)/gp.tileSize;
    tileNum1 = gp.tileM.mapTileNum[KentityLeftCol][KentityTopRow];
    tileNum2 = gp.tileM.mapTileNum[KentityRightCol][KentityTopRow];
    if(gp.tileM.tile[tileNum1].collision == true || gp.tileM.tile[tileNum2].collision == true ) {
        kentity.collisionOn = true;
    }
    break;
case"down" :
    KentityBottomRow = ( kentityBottomWorldY + kentity.speed)/gp.tileSize;
    tileNum1 = gp.tileM.mapTileNum[KentityLeftCol][KentityBottomRow];
    tileNum2 = gp.tileM.mapTileNum[KentityRightCol][KentityBottomRow];
    if(gp.tileM.tile[tileNum1].collision == true || gp.tileM.tile[tileNum2].collision == true ) {
        kentity.collisionOn = true;
    }
    break;
case"left":
    KentityLeftCol = ( kentityLeftWorldX - kentity.speed)/gp.tileSize;
    tileNum1 = gp.tileM.mapTileNum[KentityLeftCol][KentityTopRow];
    tileNum2 = gp.tileM.mapTileNum[KentityRightCol][KentityBottomRow];
    if(gp.tileM.tile[tileNum1].collision == true || gp.tileM.tile[tileNum2].collision == true ) {
        kentity.collisionOn = true;
    }
    break;
case"right":
    KentityRightCol = ( kentityRightWorldX + kentity.speed)/gp.tileSize;
    tileNum1 = gp.tileM.mapTileNum[KentityRightCol][KentityTopRow];
    tileNum2 = gp.tileM.mapTileNum[KentityRightCol][KentityBottomRow];
    if(gp.tileM.tile[tileNum1].collision == true || gp.tileM.tile[tileNum2].collision == true ) {
        kentity.collisionOn = true;
    }
    break;
}
```

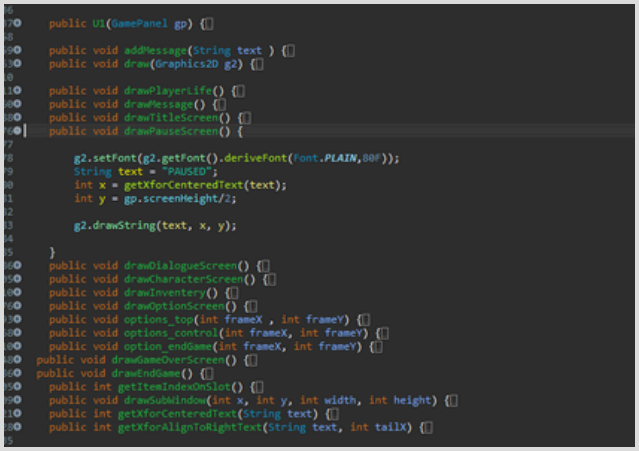
- Class for Collision Detection: A class that detects collisions with enemies and obstacles.
- Real-Time System: A system for updating animations and character states in real time.

Different Menus:

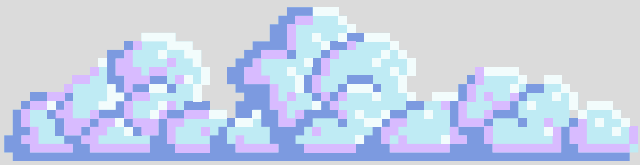
- **Death Menu:**A menu that appears upon death, giving the player the option to either retry or quit the game.
- **Pause Menu:**A pause menu that allows the player to pause the game or access the settings.



A class that generates the pause menu



Menu principal qui donne le choix de charger et de creer une nouvelle partie ou accede encore une fois aux parametres.



What We Would Have Liked to Add (Possible Improvements):

1.Add difficulty levels to make the game accessible to a wider audience.
2. Introduce different types of enemies with varied behaviors.
3. Integrate a progression system for the character, such as skills or weapon upgrades.
4. Expand the game world with new areas to explore.
5. Add a multiplayer mode to enrich the experience.
6. Add the option to choose from multiple characters with different powers.
7. Add a story to give the game another dimension.

Conclusion

The code was too long to write, so we used multiple classes to create this simple world and what it's made of.

Package Entity

Classes :
Entity: A parent class for the player, old man (NPC), projectile, and monsters.
Player: All methodes that the player use are in this classe.
NPC_OldMan: the caracterizations, walk, dialogues...
Projectile: An arm for the player in the game.

Package Main

Classes:AssetSetter : objects location.
CollisionChecker.
Event: teleportation , trap.
GamePanel.
KeyHandler.
Sound.U1: for graph.
UtilityTool

Package monster

GreenSlime.
ORG.

Package Object

describe all the objects in the game.

Package Tuile

Contains the images to build the game world