VIETNAM NATIONAL UNIVERSITY – HOCHIMINH CITY

THE INTERNATIONAL UNIVERSITY

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

FINAL REPORT

Escape Dungeon

Course: Object Oriented Programming - ITIT23UN41

INSTRUCTOR:

Assoc. Prof. Tran Thanh Tung and Nguyen Trung Nghia

No.	Full Name	Student's ID
1	Mai Văn Vinh	ITITDK22117
2	Phạm Đăng Minh	ITITIU22103
3	Nguyễn Tiến Anh	ITITDK22128
4	Ngô Tùng Chương	ITITDK22116

Table of Content

ABSTRACT

Pixel Escape Dungeon is an amazing 2D pixel-art platformer designed to deliver fast-paced action and precision gameplay. Drawing inspiration from legendary platform games such as Contra, Mario, and Pokemon, it combines the nostalgia of classic gaming with modern mechanics to create an engaging experience for players of all ages.

In this adventure, players navigate through a dungeon filled with traps, enemies, and hidden treasures, using agility and strategy to escape.. Players must rely on their reflexes, strategic thinking, and quick decision-making to survive, and find the key to escape the dungeon.

Pixel Escape Dungeon offers nostalgia and excitement, appealing to gamers looking for a familiar yet thrilling new adventure.

CHAPTER 1: INTRODUCTION

1. Objectives

1.1 Gaming in the fields:

2D platform games, a cornerstone of the gaming industry's early days, are experiencing a significant revival in recent years. While their simplistic mechanics and pixelated graphics once seemed outmoded in the face of 3D and open-world innovations, a renewed appreciation for their charm, creativity, and gameplay depth has brought them back to the forefront.

One key trend driving this resurgence is the nostalgia factor. Many players who grew up with classics like **Super Mario Bros** and **Sonic the Hedgehog** now seek similar experiences, either to relive their childhood or introduce these timeless games to younger generations. Modern developers are capitalizing on this by creating games with retro aesthetics, blending them with contemporary mechanics. Titles such as Celeste and Hollow Knight showcase intricate level designs, fluid animations, and compelling narratives, demonstrating that 2D platformers can be just as engaging as their 3D counterparts.

1.2 About the game project:

"Escape dungeon" offers players an enchanting experience through the lens of a squire on a noble quest. This mini-level design game takes you on an adventurous journey to locate a long-lost treasure and restore it to its rightful owner, the King. Combining a relaxing ambiance with moments of thoughtful challenge, the game strikes a perfect balance for players seeking both tranquility and excitement.

The story places you in the boots of a young squire, entrusted with an important mission by the kingdom. As you embark on this journey, you traverse through beautifully crafted levels, each designed to test your problem-solving skills, agility, and sense of exploration. The game's level design is its standout feature, offering a
variety of puzzles, obstacles, and hidden secrets that keep the adventure engaging. Every step brings you closer to uncovering the treasure, but not without requiring wit and perseverance.

"Reaching the Treasure" is a gem for those who appreciate thoughtful level design and an engaging narrative. It invites players to immerse themselves in a world of discovery, all while providing the satisfaction of overcoming challenges. For anyone seeking a game that blends relaxation with a sense of accomplishment, this quest promises to be a treasure worth finding.

1.3 Project Purpose

This project aims to apply Java programming and Object-Oriented Programming (OOP) principles in a practical context, bridging the gap between theoretical knowledge and real-world application. It focuses on enhancing technical skills while fostering teamwork and a mindset of continuous improvement through ideation, creation, and iteration. Additionally, the project provides foundational knowledge of game development concepts, combining technical and creative elements to design engaging gameplay experiences. Overall, it serves as a comprehensive learning opportunity to strengthen programming expertise, collaboration skills, and innovation.

2. The Tools Used

- **IDE for programming and debugging: IntelliJ, VSCode, Eclipse.**
- **Design: GIMP.**
- **Java Development Kit: 21.**
- **Mean of code version management: GitHub.**
- **Means of contacting: Facebook**

3. Developer Team

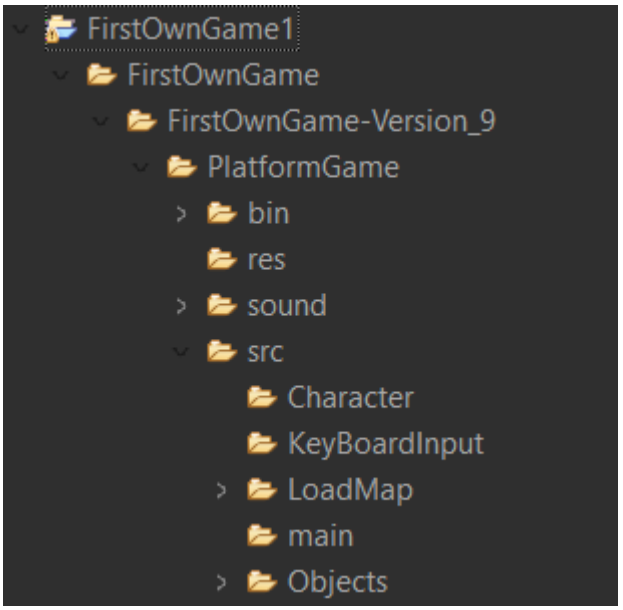
Name	ID	Contributes
Mai Văn Vinh	ITITDK22117	➢ Write Report, Slide, Presentation ➢ Develop Code: Class Enemy, Playing Menu, Die and Pause Menu.. ➢ Fix Bug
Ngô Tùng Chương	ITITDK22116	➢ Write Report, Slide, Presentation ➢ Design Map ➢ Develop Code ➢ Fix Bug
Nguyễn Tiến Anh	ITITDK22128	➢ Write Report, Slide, Presentation ➢ Find Asset ➢ Design Player Character, Cannon,... ➢ Develop Code: UI, Key System,..
Trần Đăng Minh	ITITIU22103	➢ Write Report, Slide, Presentation ➢ Develop Code: UI, NPC,... ➢ Fix Bug

CHAPTER 2: SYSTEM DESIGN

1. Package Structure

The project's current design is the outcome of thorough iteration and problem-solving, resulting in this optimized setup. The thoughtfully designed layout ensures the smooth operation of the game algorithms, which power our latest creation and enhance the interactive experiences within the game.

To simplify the management of classes, we have categorized them into distinct groups or packages, such as:



Here is the link to the GitHub repository for our project:

https://github.com/TienAnh0108/OOP_Project

2. List of Class and Responsibility

Package	Class Name	Responsibility
character	CheckHitBox	Provides a general class with static methods so that the player and enemy entities can interact with each other and also the environment.
	Crab_Spawn	It will display each crab in the map using Graphics and drawImage.
	Crab	This acts like a blue print to create a crab (enemy) storing the coordinates
	Enemy	This is also an abstract class which takes responsibility for controlling the algorithm of all enemies.
	Entity	This is an abstract class connecting with the CheckHitBox class and it will take care of the positions of the player and enemies.
	Player	This takes care of the coordinates, actions of the player when the user directs the player character. It also manages all attributes such as health, getting hit
keyBoardInput	KeyBoardAction	Handles key events in our game. It implements the KeyListener interface, which allows it to listen for and respond to key presses, releases
	Sound	Manages sound effects when player does any actions and also control the music in the game

	SwitchAction	Handles actions when the user directs the player, it will switch the animation of each action.
loadMap	Load	It will initialize and load the map (image) along with other objects including traps, enemies, cannons, door and key.
	Map	This just acts an intermediary class to get the map level
	MapManager	Handles each map level and calculate the offset when player and other objects move around the environment
main	DieMenu	Show up the die menu when the player is out of health along with buttons for the player to interact with
	Game_JFrame_Window	Create the frame for the game
	Game_JPanel	Inherits from the JPanel class and , is responsible for managing and displaying the game interface.
	Main_Class	This class just calls the MainGame class to run the program
	MainGame	This class is considered as a hub which creates every object in the game and controls all them. It also updates the game's state in every second,sets up the main game window,

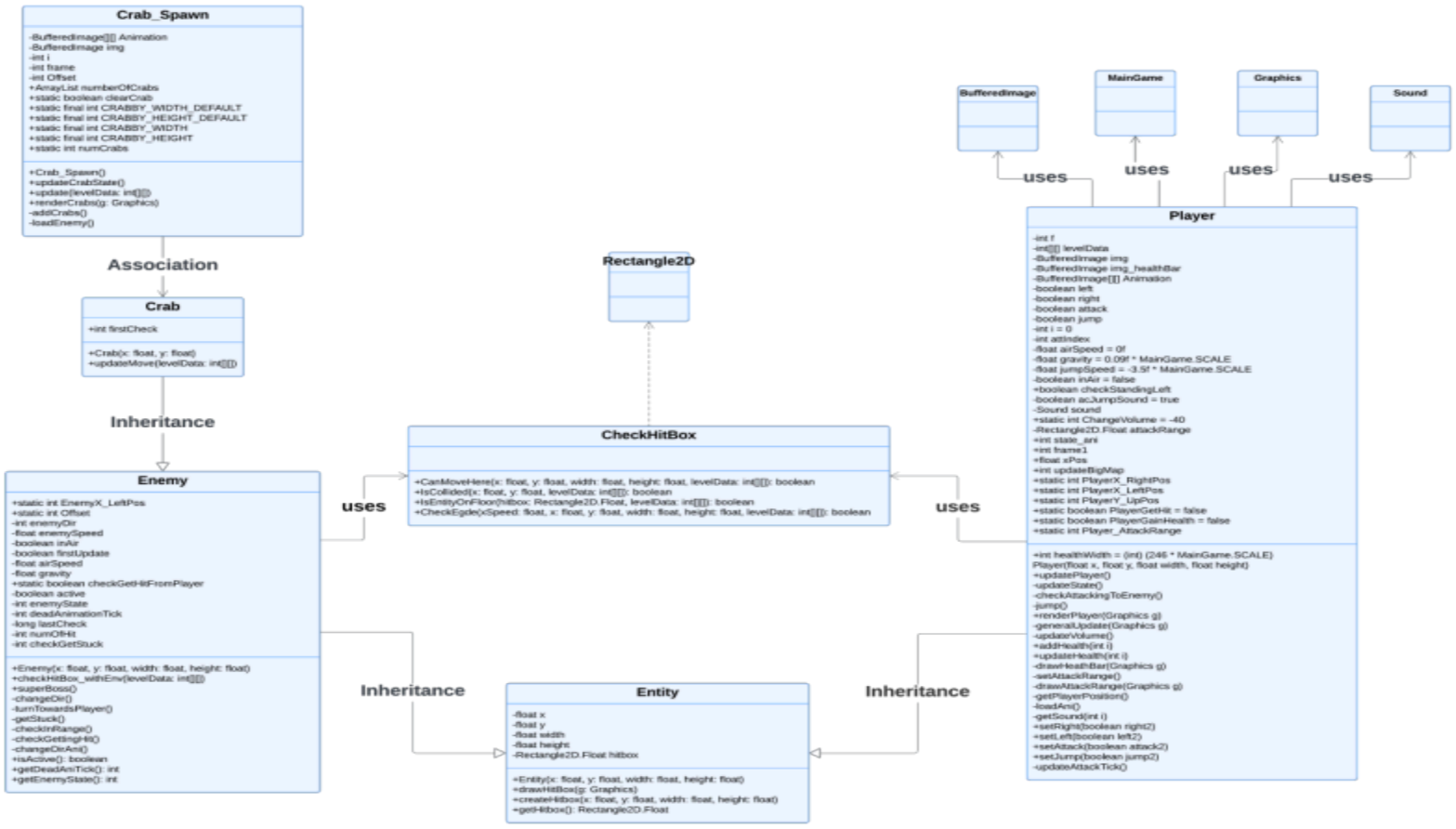
		configures its properties, and starts the game loop to begin gameplay.
	PauseMenu	After entering p button on the key board, this menu will show up and pause the game state
	PlayingMenu	Display the playing menu along with buttons for users to interact with
objects	Ball	Takes care of the coordinates of the balls when the cannon shoots
	Cannon_Spawn	This handles the rendering of cannons in the map along with balls.
	Cannon	This class is considered as a blue print storing the coordinates of each cannon in the game, also the algorithm to check whether the player in range to shoot
	Door	This class takes care of the coordinates of the door and key in the map. It also checks whether the player has a key to win or not
	HealthPotion_Spawn	This will display the image of the health Potions in the game for player
	HealthPotion	This class stores the coordinates of the health potions in the game and interacts with the player

	Trap_Spawn	Render all the traps in the game
	Traps	Handles the position of each trap and the hitting range in the game
	NPC	Acts as a blueprint for the NPC storing the coordinates of this object when the game is started
	NPC_Spawn	Handles the rendering of NPC and also updates the animation during the game

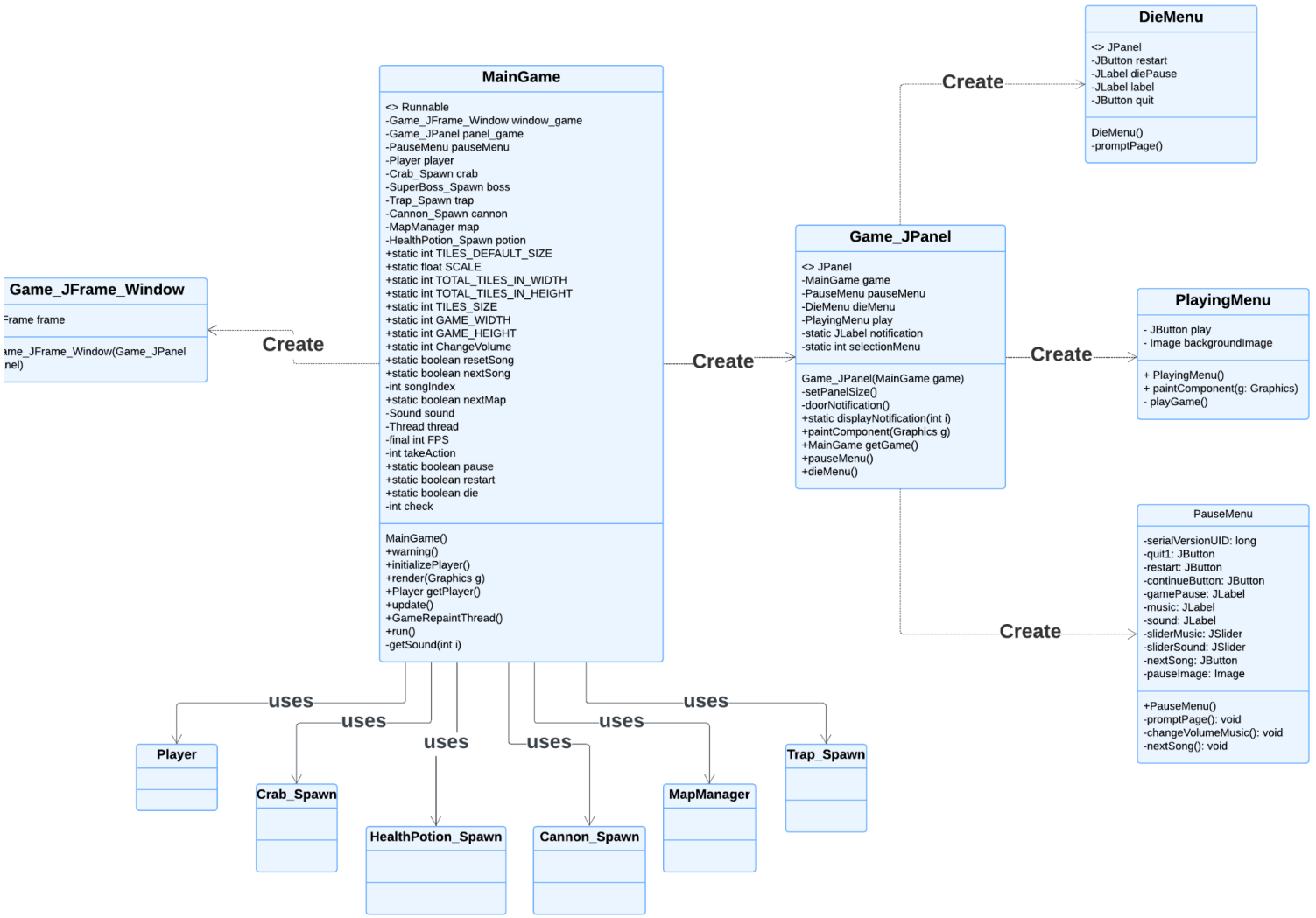
3. UML Diagram

UML diagrams allow software developers and programmers to work on the same project effectively and conveniently. This enhances collaboration and logic in code processing. It also helps identify potential problems early in the design process.

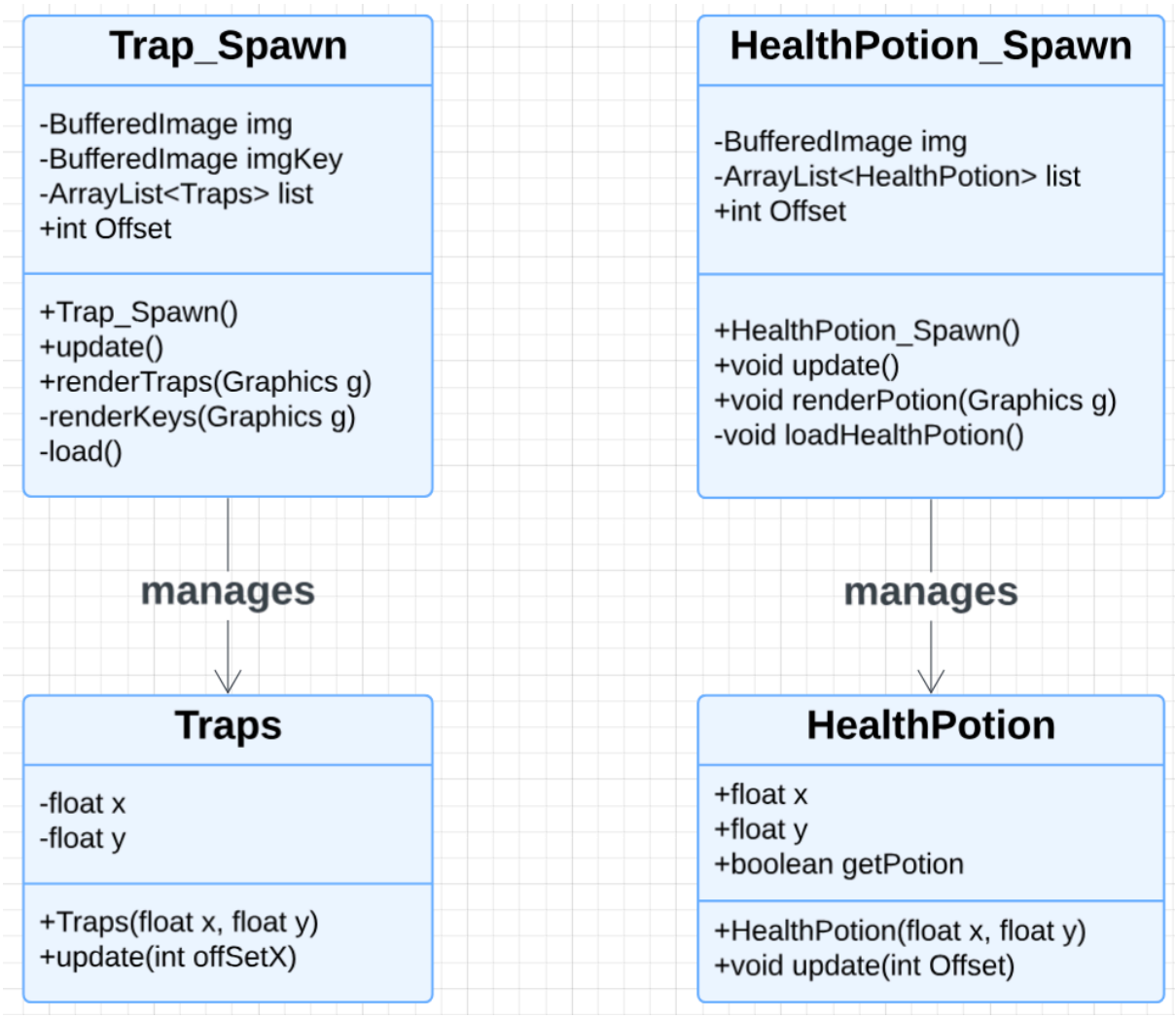
Classes extends from Entity class



MainClass Diagram

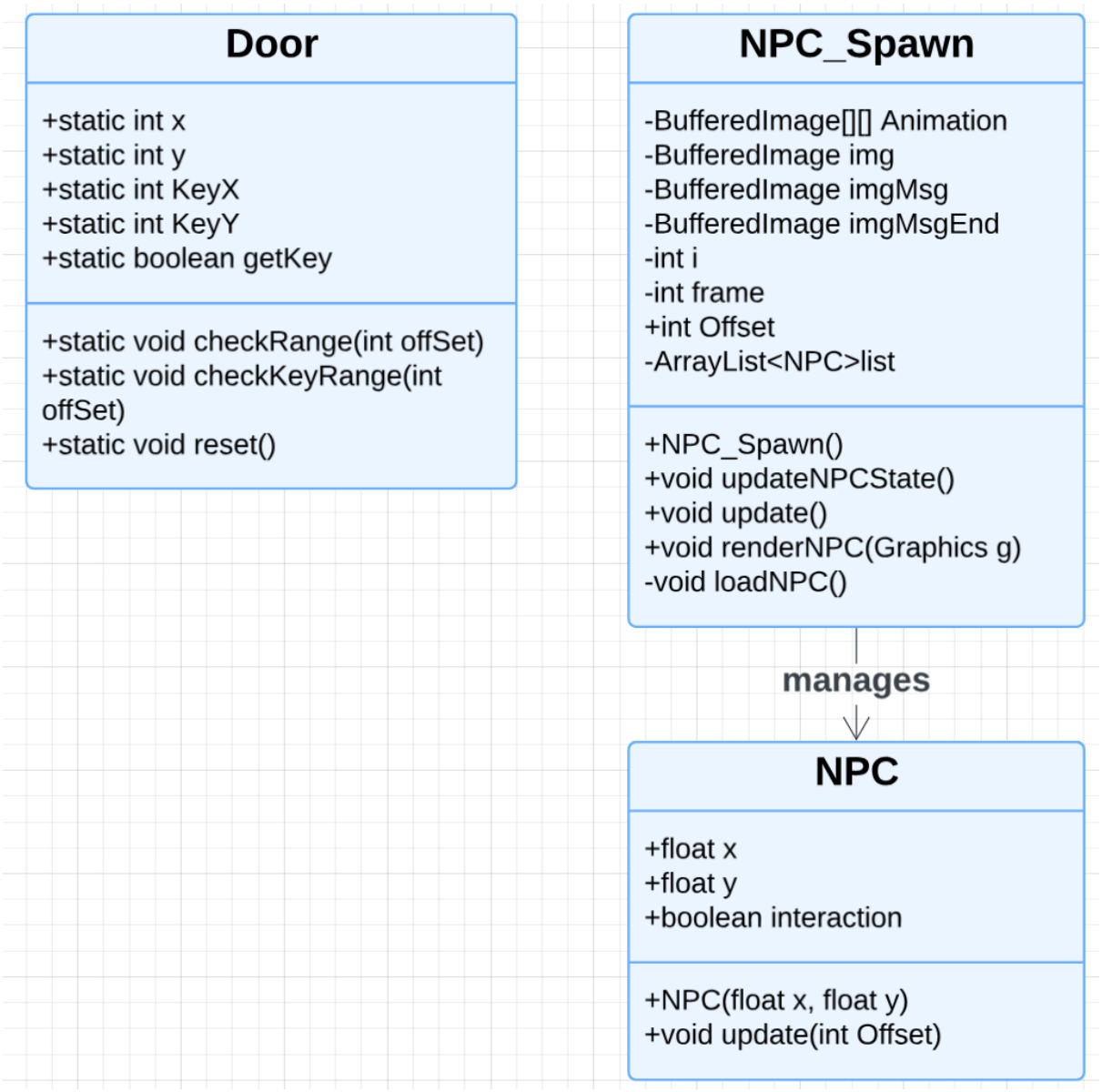


Trap_Spawn and HealthPotion_Spawn Diagrams



Cannon_Spawn Diagram

Door and NPC_Spawn Diagram



4. Code flow

Start from class MainGame, create a screen, and initialize GamePanel along with JFrame. In Game_Panel, call the render method ofMainGame to set up the game before it starts. Then call the GameRepaintThread to start the game.

1. MainGame class to set up

```
65      public MainGame() {
66
67          sound = new Sound();
68          getSound(songIndex);
69
70          initializePlayer();
71
72          panel_game = new Game_JPanel(this);
73          window_game = new Game_JFrame_Window(panel_game);
74          GameRepaintThread();
75
76      }
77
78      public void warning() {
79
80      }
81
82      public void initializePlayer() {
83          Load.mapControler();
84          map = new MapManager();
85          if(check == 1) {
86          potion = new HealthPotion_Spawn();
87          npc = new NPC_Spawn();
88          trap = new Trap_Spawn();
89          cannon = new Cannon_Spawn();
90          player = new Player(30,300, (int) (96*SCALE), (int) (84*SCALE) );
91          crab = new Crab_Spawn();
92          }
93      }
94
95      public void render(Graphics g) {
96          if(check == 1)
97             map.player_xPos = player.xPos;
98          map.get_Map_Level(g);
99
100         if(!pause && check == 1) {
101            player.updateBigMap = map.xLvlOffset;
102            player.renderPlayer(g);
```

```
103
104             crab.Offset =  map.xLvlOffset;
105             crab.renderCrabs(g);
106
107             trap.Offset =  map.xLvlOffset;
108             trap.renderTraps(g);
109
110             potion.Offset = map.xLvlOffset;
111             potion.renderPotion(g);
112
113             npc.Offset = map.xLvlOffset;
114             npc.renderNPC(g);
115
116
117             cannon.Offset =  map.xLvlOffset;
118             cannon.renderGeneral(g,player);
119         }
120
121         Door.checkRange(map.xLvlOffset);
122         Door.checkKeyRange(map.xLvlOffset);
123
124     }
125
```

2. Game_Panel class with render method to run the game

```
27      public Game_JPanel(MainGame game) {
28          this.game = game;
29
30          pauseMenu = new PauseMenu();
31          pauseMenu.setVisible(false);
32
33          dieMenu = new DieMenu();
34          dieMenu.setVisible(false);
35
36          play = new PlayingMenu();
37          play.setVisible(true);
38
39          add(pauseMenu);
40          add(dieMenu);
41          add(play);
42          doorNotification();
43          setPanelSize();
44
45       }
46
47      private void setPanelSize() {
48          Dimension size = new Dimension(MainGame.GAME_WIDTH, MainGame.GAME_HEIGHT);
49          setMinimumSize(size);
50          setPreferredSize(size);
51          setMaximumSize(size);
52
53      }
54
55      public void paintComponent(Graphics g) {
56          super.paintComponent(g);
57          game.render(g);
58
59      }
60
61
62      public MainGame getGame() {
63       return game;
64      }
```

3. run method to update the game in MainGame

```
@Override
public void run() {

    double nanoFPS = 1000000000.0  / FPS;
    long lastTime = System.nanoTime();
    long now = System.nanoTime();
    long loadframe = System.currentTimeMillis();
    long nowframe = System.currentTimeMillis();

    while(true) {
        checkSound();
        now = System.nanoTime();

        if(now - lastTime >= nanoFPS) {
            panel_game.repaint();
            lastTime = now;
        }

        if(winning && Door.getKey) {
            if(winningPrompt()) {
                Door.reset();
                initializePlayer();
            }else
                System.exit(0);

        }
        winning = false;

        nowframe = System.currentTimeMillis();
        if(nowframe - loadframe >= 150) {
            update();
            loadframe = System.currentTimeMillis();

        }
    }

}
```

In order to load the map and other objects in the game, the initializePlayer() method will be called to load the resource then it will create Load and MapManger objects.

4.These methods in Load class has to load the map and objects

```
79     public static int[][] GetMapLevelData() {
80
81         BufferedImage img = LoadImage(mapControler.get(index));
82         int[][] levelData = new int[img.getHeight()][img.getWidth()];
83
84
85         for (int j = 0; j < img.getHeight(); j++) {
86             for (int i = 0; i < img.getWidth(); i++) {
87                 Color color = new Color(img.getRGB(i, j));
88                 int value  = color.getRed();
89                 initializeObjects(i,j,color.getBlue(),color.getGreen());
90
91                 if (value >= 48)
92                     value = 0;
93
94                 if(value == 23) {
95                     Door.x = i*MainGame.TILES_SIZE;
96                     Door.y = j*MainGame.TILES_SIZE;
97                 }
98
99                 levelData[j][i] = value;
100            }
101
102        }
103
104        return levelData;
105
106    }
107
```

```
108    private static void initializeObjects(int i, int j, int value1,int value2) {
109        switch (value1) {
110        case 0:
111            getTraps.add(new Traps(i * MainGame.TILES_SIZE, j * MainGame.TILES_SIZE));
112            break;
113
114        case 5:
115            getCannon.add(new Cannon(i * MainGame.TILES_SIZE, j * MainGame.TILES_SIZE));
116            break;
117
118        case 6:
119            Door.KeyX = i * MainGame.TILES_SIZE;
120            Door.KeyY = j * MainGame.TILES_SIZE;
121            break;
122
123        case 7:
124            getHeathPotion.add(new HealthPotion(i * MainGame.TILES_SIZE, j * MainGame.TILES_SIZE));
125            break;
126
127        case 8:
128            getNPC.add(new NPC(i * MainGame.TILES_SIZE, j * MainGame.TILES_SIZE));
129            break;
130        }
131
132        if (value2 == 0)
133            getCrab.add(new Crab(i*MainGame.TILES_SIZE, j*MainGame.TILES_SIZE));
134
135    }
136
137 }
```

These methods in MapManager class will load the image of Tile Blocks to set up the map and also calculate the offset in the game which helps displaying the screen for the player

```
65      public void get_Tile_Block() {
66          image = Load.LoadImage(Load.TILE_2D);
67          index = new BufferedImage[48];
68
69          int value = 0;
70
71            for(int i = 0; i < 4; i++) {
72                for(int j = 0; j < 12; j++ ) {
73                    index[value] = image.getSubimage(j*32, i*32, 32, 32);
74                    value++;
75
76                }
77            }value = 0;
78      }
79
80      public void checkMoveLeftAndRight() {
81          int playerX = (int) player_xPos;
82          diff = playerX - xLvlOffset;
83
84          if (diff > rightBorder)
85              xLvlOffset += diff - rightBorder;
86          else if (diff < leftBorder)
87              xLvlOffset += diff - leftBorder;
88
89          if (xLvlOffset > maxLvlOffsetX)
90              xLvlOffset = maxLvlOffsetX;
91          else if (xLvlOffset < 0)
92              xLvlOffset = 0;
93
94
95      }
96
```

```
99      public void get_Map_Level(Graphics g) {
100         checkMoveLeftAndRight();
101
102
103         if(firstCheck) {
104           g.setColor(Color.DARK_GRAY);
105           g.fillRect(0 , 0 , 1248, 672);
106         }else
107           g.drawImage(scaledImage, 0, 0, 1248, 672, null);
108
109         for (int j = 0; j < MainGame.TOTAL_TILES_IN_HEIGHT; j++)
110             for (int i = 0; i < levelData_Player[0].length; i++) {
111                 int num = level.get_levelData_index(i, j);
112                 g.drawImage(index[num], MainGame.TILES_SIZE * i - xLvlOffset, MainGame.TILES_SIZE * j , MainGame.TILES_SIZE, MainGame.TILES_SIZE,
113             }
114
115
116     }
117
118
119
120
121 }
```

After loading the map and all objects ,they will be updated in every second inside the MainGame class

```
120    public void update() {
121
122        takeAction = SwitchAction.action;
123        player.frame1 = (SwitchAction.GetFramesAction(takeAction));
124        npc.updateNPCState();
125        crab.updateCrabState();
126        player.updatePlayer();
127        cannon.updateAttackTick();
128
129
130        if(restart) {
131            Game_JPanel.selectionMenu = 0;
132            panel_game.dieMenu();
133            restart = false;
134            die = false;
135            pause = false;
136            initializePlayer();
137        }
138
139        if(die) {
140          pause = true;
141          panel_game.dieMenu();
142        }
143
144
145    }
```

When the player moves in any direction , the updateState method of Player class is updated and gets checked with CheckHitBox class

```
86      private void updateState() {
87          createHitbox(x + 60, y + 40, width-120, height - 74);
88          setAttackRange();
89
90          if(SwitchAction.action == 0) state_ani = 0;
91          else if (SwitchAction.action == 1)state_ani = 1;
92
93          if (jump) {
94              if(acJumpSound) getSound(0);
95              acJumpSound = false;
96              jump();
97
98          }
99
100
101         if (left && !right) {
102             x -= 2.5;
103             state_ani = 3;
104         } else if (right && !left) {
105             x += 2.5;
106             state_ani = 2;
107         }
108
109
110         if(attack) {
111             if( (right && !left) || state_ani == 0)
112               state_ani = 4;
113             else if((left && !right) || state_ani == 1)
114               state_ani = 5;
115         }
116
117
118         if (!inAir)
119             if (!CheckHitBox.IsEntityOnFloor(hitbox, levelData))
120                 inAir = true;
```

```
117
118         if (!inAir)
119             if (!CheckHitBox.IsEntityOnFloor(hitbox, levelData))
120                 inAir = true;
121
122         if (inAir) {
123             if (CheckHitBox.CanMoveHere(hitbox.x, hitbox.y + airSpeed, width-120, height - 74, levelData)) {
124                 y += airSpeed;
125                 airSpeed += gravity;
126                 acJumpSound = false;
127
128             }
129             else {
130                 airSpeed += gravity + 0.12;
131                 if (airSpeed > 0) inAir = false; airSpeed = 0; acJumpSound = true;
132             }
133         }
134
135
136         if(!CheckHitBox.CanMoveHere(x + 60, y + 40, width-120, height - 74,levelData)) {
137             if(right) x -= 2.5;
138             if(left) x += 2.5;
139         }
140
141
142
143     }
```

These methods in CheckHitBox class makes sure the interaction between player and map run properly by taking the player's coordinates and comparing them with the map

```
7  public class CheckHitBox {
8
9      public static boolean CanMoveHere(float x, float y, float width, float height, int[][] levelData) {
10         if (!IsCollided(x, y, levelData))
11             if (!IsCollided(x, y + height, levelData))
12                 if (!IsCollided(x + width, y, levelData))
13                     if (!IsCollided(x + width, y + height, levelData))
14                         return true;
15
16         return false;
17     }
18
19
20     public static boolean IsCollided(float x, float y,int[][] levelData) {
21         if(x < 0 || x >= levelData[0].length*MainGame.TILES_SIZE) return true;
22         if(y < 0 || y >= MainGame.GAME_HEIGHT) return true;
23
24
25
26         float xIndex = x / MainGame.TILES_SIZE;
27         float yIndex = y / MainGame.TILES_SIZE;
28
29
30
31         int value = levelData[(int)yIndex][(int)xIndex];
32
33         if (value >= 48 || value < 0 || value != 11)
34             return true;
35         return false;
36     }
```

When the player gets attacked or gets the health potion, its health will be updated .If the health bar is zero, the game over JPanel will be displayed

```
212    public void addHealth(int i) {
213        if(healthWidth + i > (int) (246*MainGame.SCALE))
214            healthWidth = (int) (246*MainGame.SCALE);
215        else
216            healthWidth += i;
217    }
218
219    public void updateHealth(int i) {
220        healthWidth -= (0.25)*i;
221        if(healthWidth == 1) {
222            Game_JPanel.selectionMenu = 1;
223            MainGame.die = true;
224            healthWidth = 0;
225        }
226    }
```

When the player wants to attack the enemies, this method will handle this action if the animation of the player is 3 which is in the attack state.

```
147     private void checkAttackingToEnemy() {
148         if(attack) {
149             if(attIndex == 3){
150                 Enemy.checkGetHitFromPlayer = true;
151                 Cannon.GetHit = true;
152             }else {
153                 Enemy.checkGetHitFromPlayer = false;
154                 Cannon.GetHit = false;
155             }
156 
157         }else
158             Enemy.checkGetHitFromPlayer = false;
159 
160     }
161 
```

Then the enemy's health will be calculated by this method in Enemy class. The enemy will be dead if it gets 6 hits

```
143     private void checkGettingHit() {
144         int dx = (int) ((Player.Player_AttackRange) - EnemyX_LeftPos);
145         int dy = (int) (Player.PlayerY_UpPos - y);
146         if(dx > -20 && dx < 44 && Math.abs (dy) < 100) {
147             if(Player.PlayerX_RightPos > EnemyX_LeftPos)
148                 x -= 20;
149             else
150                 x += 20;
151 
152             numOfHit++;
153             if(numOfHit > 6) {
154                active = false;
155                Crab_Spawn.numCrabs--;
156                numOfHit = 0;
157             }
158         }
159 
160     }
161 
```

To make the game more interesting, we also create a sound class. This MainGame will create an object of this class when the game is started

```
13 public class Sound {
14
15     Clip clip;
16     public FloatControl fc;
17     public URL soundURL[] = new URL[30];
18
19     public Sound() {
20         soundURL[0] = getClass().getResource("/jump.wav");
21         soundURL[1] = getClass().getResource("/attackSound.wav");
22         soundURL[2] = getClass().getResource("/Pitifulface.wav");
23         soundURL[3] = getClass().getResource("/VuaHanVuaYeu.wav");
24
25
26     }
27
28     public void getSound(int i,int volume) throws UnsupportedAudioFileException, IOException, LineUnavailableException {
29             AudioInputStream audioStream = AudioSystem.getAudioInputStream(soundURL[i]);
30             clip = AudioSystem.getClip();
31             clip.open(audioStream);
32             fc = (FloatControl) clip.getControl(FloatControl.Type.MASTER_GAIN);
33             fc.setValue(volume);
34             clip.start();
35     }
36
37     public void reset() {
38         clip.setFramePosition(0);
39     }
```

To win this game, the player has to kill all enemies and find the key, open the door. In order to know the key is taken or not, we have to

```
 8 public class Door {
 9
10     public static int x;
11     public static int y;
12     public static int KeyX;
13     public static int KeyY;
14
15
16     public static boolean getKey = false;
17
18     public static void checkRange(int offSet) {
19
20         double range = x - Player.PlayerX_RightPos - offSet;
21         if(range <= 27 && !getKey )
22             Game_JPanel.displayNotification(1);
23         else
24             Game_JPanel.displayNotification(0);
25
26         if(range <= 27)
27             if(getKey && !Crab_Spawn.clearCrab)
28                 System.out.println("Kill all enemies");
29             else
30                 MainGame.winning = true;
31
32     }
33
34     public static void checkKeyRange(int offSet) {
35         double rangeX = KeyX - Player.PlayerX_RightPos - offSet;
36         double rangeY = KeyY - Player.PlayerY_UpPos;
37
38         if(Math.abs(rangeX) < 25 && rangeY > 45)
39             getKey = true;
40
41     }
```

check the position of player(x,y) along with KeyX and KeyY

Then the prompt will show up to ask whether the player wants to play again or not ?

```
223     private boolean winningPrompt() {
224         int result = JOptionPane.showConfirmDialog(
225                 null,
226                 "You Win, Wanna play again?", // Message
227                 "Confirmation",            // Title
228                 JOptionPane.YES_NO_OPTION, // Option type
229                 JOptionPane.QUESTION_MESSAGE // Message type
230             );
231 
232 
233             if (result == JOptionPane.YES_OPTION)
234                 return true;
235             else
236                 return false;
237     }
238 
```

CHAPTER 3: ENTITY, NPCs, OTHER OBJECT DESIGN

1. Player

Asset:

Figure : Player Asset

we choose the asset player as a knight with the purpose of immersing the player in the role of a mighty one striving to break free from the dungeon

How to load:

The loadAni method is responsible for loading and processing the animation resources for the player character. Here's a breakdown of its functionality:

Open Resource Stream:

- An InputStream (is) is used to access the sprite sheet file (/knight.png) from the resources folder.

Load Images:

- The ImageIO.read() method reads the sprite sheet into a BufferedImage object (img).

Split Sprite Sheet into Frames:

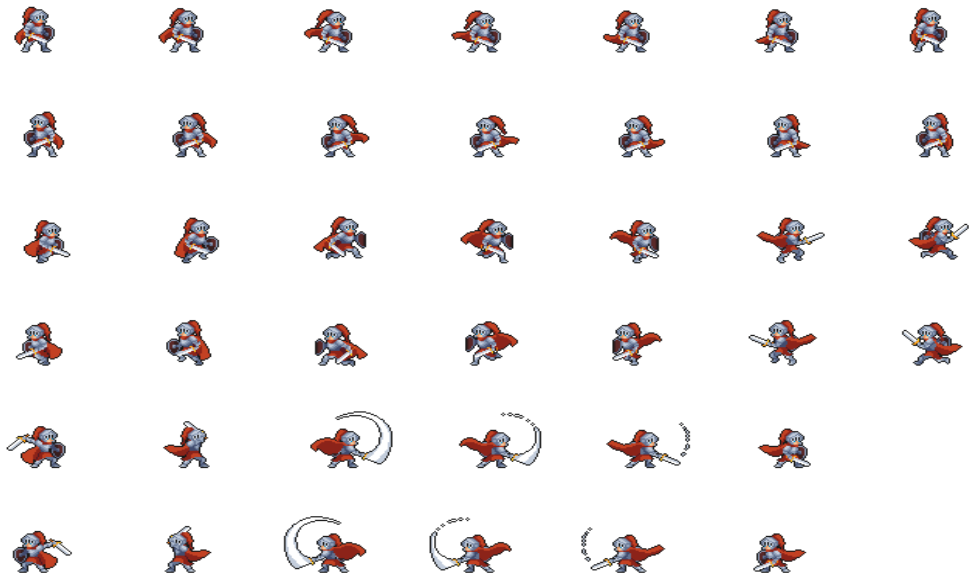


Figure : Player Animations

- Initializes a 2D array (Animation) to hold the animation frames. It has dimensions [6][7], representing 6 rows of animations, each containing 7 frames.
- The nested for loops iterate through each row (i) and column (j) of the sprite sheet.
- Each frame is extracted using getSubimage(j * 96, i * 84, 96, 84):
    - j * 96 and i * 84 specify the top-left corner of each frame.
    - 96 (width) and 84 (height) represent the size of each frame.

Key Method

updatePlayer()

- Updates the player's state each frame.
- Handles attack animations and sound effects using updateAttackTick().

updateState()

- **Collision Detection**:
  - Creates a hitbox for the player and updates the attack range.
  - Checks if the player is standing on the ground using CheckHitBox.IsEntityOnFloor().
- **Movement**:
  - Handles left and right movement based on flags (left, right).
  - Updates animation states (state_ani) based on movement and actions.
- **Jumping**:
  - If the jump flag is set (jump), the player jumps unless already in the air.
  - Airborne motion is controlled by airSpeed and gravity.
- **Collision Response**:
  - Prevents the player from moving into solid tiles by adjusting x based on the movement direction.

checkAttackingToEnemy()

- Detects whether the player is attacking enemies or objects like cannons.
- Uses the attack index (attIndex) to determine whether to enable enemy hit detection (Enemy.checkGetHitFromPlayer).

renderPlayer(Graphics g)

- Draws the player on the screen using the current animation state and frame.
- Resets player-specific flags like PlayerGetHit and PlayerGainHealth.

generalUpdate(Graphics g)

- Handles updates like:
    - Player state (updateState()).
    - Position tracking (getPlayerPosition()).
    - Health bar rendering (drawHeathBar()).
    - Attack range drawing (drawAttackRange()).
    - Damage and health gain mechanics.

Health Management

- **addHealth(int i)**: Increases the player's health. Caps health at the maximum value.
- **updateHealth(int i)**: Reduces the player's health. If health reaches zero, it triggers game-over logic.

Sound Effects

- getSound(int i): Plays a sound based on the action (e.g., jumping or attacking).
- updateVolume(): Adjusts the volume dynamically.

Animations

- loadAni(): Loads the player sprite sheet and extracts subimages for animation frames.
- updateAttackTick(): Handles the progression of attack animations and resets the attack flag when the animation finishes.

Helper Methods

- setAttackRange(): Updates the position and size of the attack range based on the player's direction and state.
- getPlayerPosition(): Updates the player's position in the game world for accurate collision and rendering.

2. **Enemy**

   Asset:

Figure : Crab Asset

- The Crab class is a specific type of enemy in the game, extending the Enemy class. It represents the behavior and properties of an individual crab enemy

Key Method:

updateMove(int[][] levelData):

- Updates the crab's movement logic:
  - Calls the method checkHitBox_withEnv(levelData):

Handles Movement:

❖ On the first update, check if the enemy is on the floor using CheckHitBox.IsEntityOnFloor().

❖ If airborne, adjust my position using airSpeed and gravity and check for collisions below.

Horizontal Movement:

- ❖ Moves the enemy in its current direction (enemyDir) using enemySpeed.
- ❖ Calls CheckHitBox.CanMoveHere() to validate potential movement.
- ❖ Detects edges using CheckHitBox.CheckEgde() and adjusts direction or state if needed.

Handles Stuck States:

- ❖ Tracks whether the enemy gets stuck in the environment and disables it if stuck for too long.

Player Interaction:

- ❖ Checks if the enemy is within range of the player and reacts accordingly.
- ❖ The Crab_Spawn class is responsible for managing the spawning, updating, and rendering of "Crab" enemy characters

Key Properties:

BufferedImage[][] Animation:

- Stores the animation frames for the crabs.
- Each frame corresponds to a specific enemy state (e.g., idle, moving, or dead) and animation step.

ArrayList<Crab> numberOfCrabs:

- Holds the list of all crab enemies in the game.

Core Methods:

addCrabs():

- Populates numberOfCrabs using the map's crab data (Load.GetCrabs()).
- Updates numCrabs to reflect the number of crabs added.

updateCrabState():

- Updates the animation frame index (frame) in a loop for smooth animations.

update(int[][] levelData):

- Iterates through all crabs and updates their movement based on the level data.
- Only updates active crabs.

renderCrabs(Graphics g):

- Handles rendering crabs on the screen:
    - For active crabs, display their current animation frame.
    - For dead crabs, displays a "dead" animation for a certain number of ticks before stopping.
- Checks if all crabs are cleared (numCrabs == 0) and sets clearCrab to true.

loadEnemy():

- Loads the crab spritesheet from a file (/e.png).
- Splits the image into sub-images for different animation states and frames.

3. **NPC**

Figure : NPC Asset

The characters that appear with speech when the player interacts with the NPCs are shown above. Like other light RPGs, the dialogue system is straightforward.

Load Image:

Figure : NPC Animations

4. **Trap**

Asset:

Spike | Tiny-face Ball | Cannon

Figure : Trap Asset

Traps in video games serve several purposes. They provide challenges that keep players engaged and excited, offer variety in gameplay that requires strategic thinking, and showcase players' skills and strategic thinking

For specific,

- ☐ **Spike**: make the player lose blood if touching it

  Key method:

  Update(int offSetX), appears to manage collision detection or interaction between a game object and the player. The method checks whether the object is close enough to the player vertically and horizontally. If the object meets the proximity conditions:

- It flags Player.PlayerGetHit as true, indicating that the player is affected by the object (e.g., takes damage or triggers an event).

☐ **Cannon**: fires tiny-face balls at the player, and coming into contact with these bullets will result in the player losing health.

Key method:

Update(int Offset):

- Calculate dx (horizontal distance) and dy (vertical distance) relative to the player's position, considering the scrolling offset.

- Compute the Euclidean distance between the object and the player:

  float distance = (float) Math.sqrt(dx * dx + dy * dy);

  - If the object is within a radius of 500 units, checkInrange is set to true; otherwise, false.
  - If the object is to the left of the player's position, it is marked as not in range:

    if (Player.PlayerX_RightPos > (x - Offset))

    checkInrange = false;
  - If the object is currently being hit (GetHit is true), the method checkGettingHit(Offset) is invoked to process hit behavior. Otherwise, GetHit is set to false.

5. Other Object



Key Door Health Potion

Figure : Other Object

The above objects do the following activities when player comes into contract:

- Door: Restricts players from passing through unless they both have a key and kill all enemies.
- Key: To open door.
- Heart: Helps players to recover their missing health.

CHAPTER 4: MAP DESIGN

1. Asset:

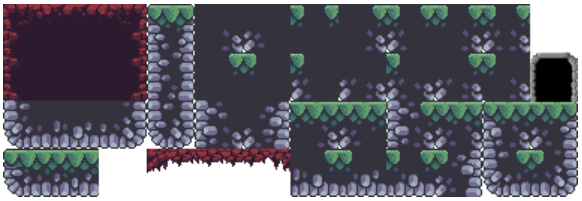


Figure : Tile Asset

The provided text file, "2D Tiles.png," lists image filenames along with boolean values. A true value indicates that the associated image represents a 2D box collider in the map design.

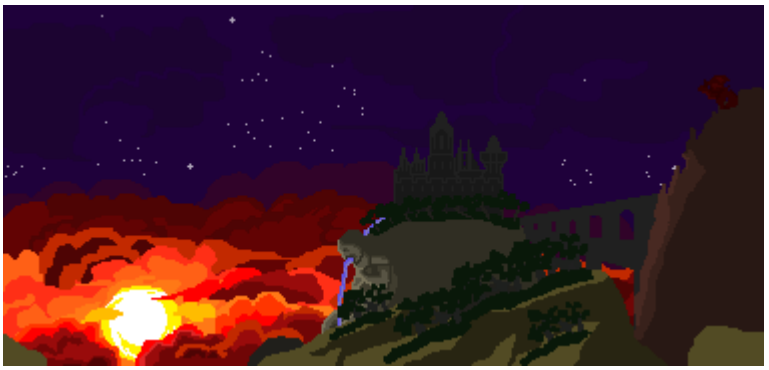


Figure : Background

We added a background to the game to create an environment that feels like being in a dungeon.

2. How to design:

Figure : Map

Our map was created using GIMP - GNU Image Manipulation Program, a tool that allows for freely placing and drawing squares. The software employs a method that converts images into digital characters for storing and loading maps.

3. How to load:

The GetMapLevelData() method processes a map image and extracts data representing the level's layout, such as terrain, objects, or interactive elements.

```
public static int[][] GetMapLevelData() {

    BufferedImage img = LoadImage(mapControler.get(index));
    int[][] levelData = new int[img.getHeight()][img.getWidth()];


    for (int j = 0; j < img.getHeight(); j++) {
        for (int i = 0; i < img.getWidth(); i++) {
            Color color = new Color(img.getRGB(i, j));
            int value = color.getRed();

            if (value >= 48)
                value = 0;

            if(value == 23) {
                Door.x = i*MainGame.TILES_SIZE;
                Door.y = j*MainGame.TILES_SIZE;
            }

            levelData[j][i] = value;
        }

    }

    return levelData;

}
```

Figure : GetMapLevelData() method

Image Loading:
The method begins by loading an image corresponding to a level using the LoadImage() function. The image is fetched based on an index from the mapControler list.

Array Initialization:
A 2D integer array, levelData, is initialized to store the processed level information. Its dimensions match the image's width and height.
Loop Through Image Pixels:

- **Outer Loop** (j): Iterates over the image's height (rows).
- **Inner Loop** (i): Iterates over the image's width (columns).

Pixel Color Extraction:
Pixel colors are retrieved (img.getRGB(i, j)), converted to Color objects, and their red channel values determine the tile/feature type

Value Adjustment:

- If the value is greater than or equal to 48, it is reset to 0. This might signify non-collidable or empty space.
- If value equals 23, it indicates a door. The door's position is calculated in game-world coordinates by multiplying the pixel coordinates (i, j) by MainGame.TILES_SIZE, and the Door object is updated with these coordinates.

Store Data:
The processed value is stored in the levelData array at the corresponding position [j][i].

Return Result:
Once all pixels are processed, the levelData array is returned, containing the entire level's layout information.

CHAPTER 5: GAME DESIGN

1. Game Rule

Embark on an exciting dungeon exploration adventure in this thrilling game where players take control of a courageous character navigating through perilous challenges. The dungeon is filled with deadly obstacles, including menacing cannons, sharp spikes, and fearsome monsters, all designed to test the player's skills and determination. With quick reflexes and clever strategy, players must overcome these dangers while exploring the dungeon's dark corridors. The ultimate objective is to discover the hidden key that unlocks the path to freedom, escaping the dungeon's clutches and emerging victorious. Are you ready to take on the challenge?

2. Game Structure

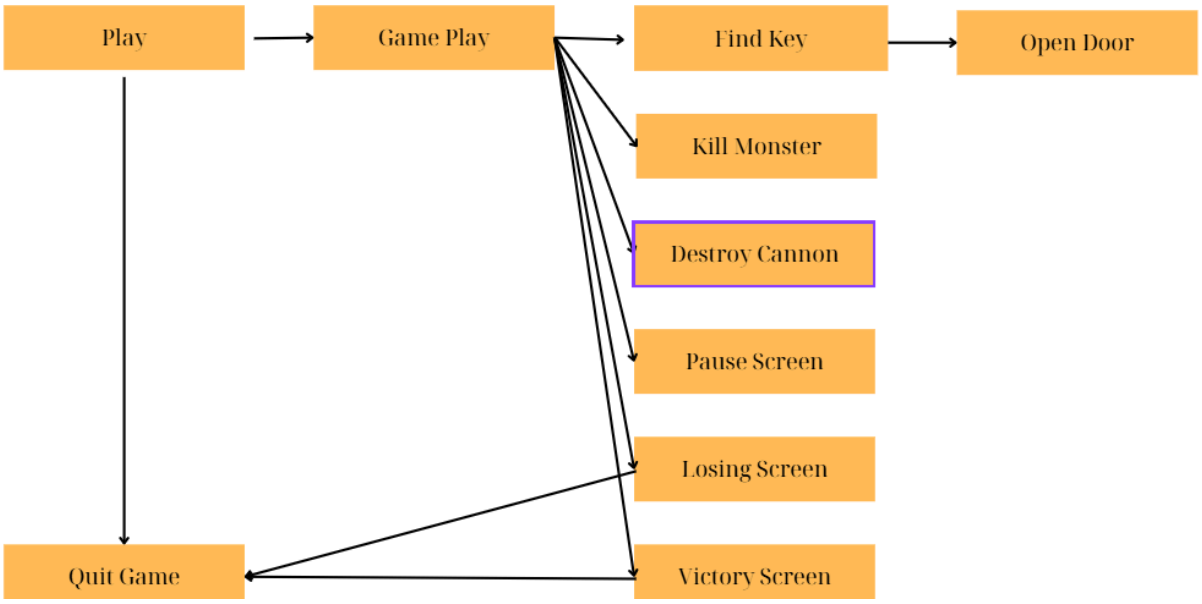


Figure : Game Structure

3. Game Play

Player Controls:

We use keyboards key such as A,W,S,D to move the player, Z to attack the enemies and SPACE to jump

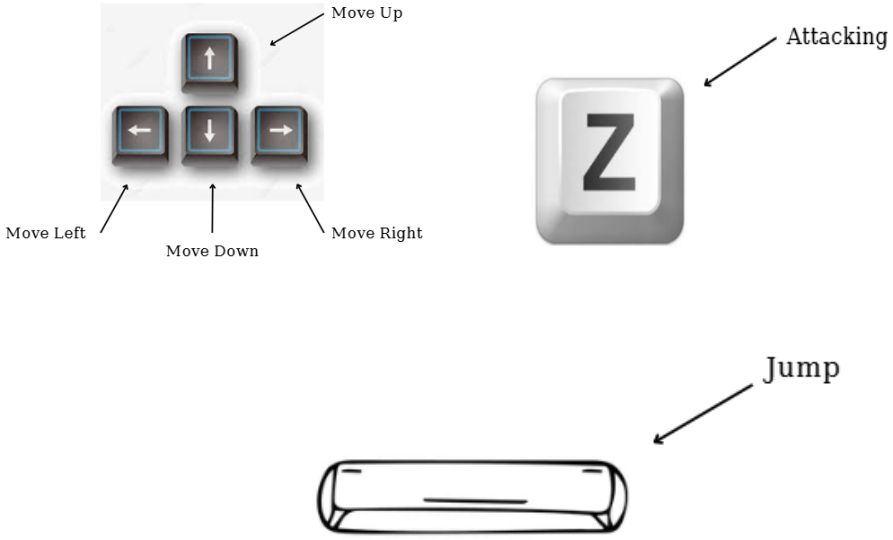


Figure : Player Controls

Game Controls:

Figure : Game Controls

4. Animation States

4.1 Player:

The player has 5 animation types including left movement, right movement, jumping, falling, and attacking. To assist players feel the game as genuinely as possible, we produce more realistic movement effects by performing player motions by reading frames throughout each player movements.

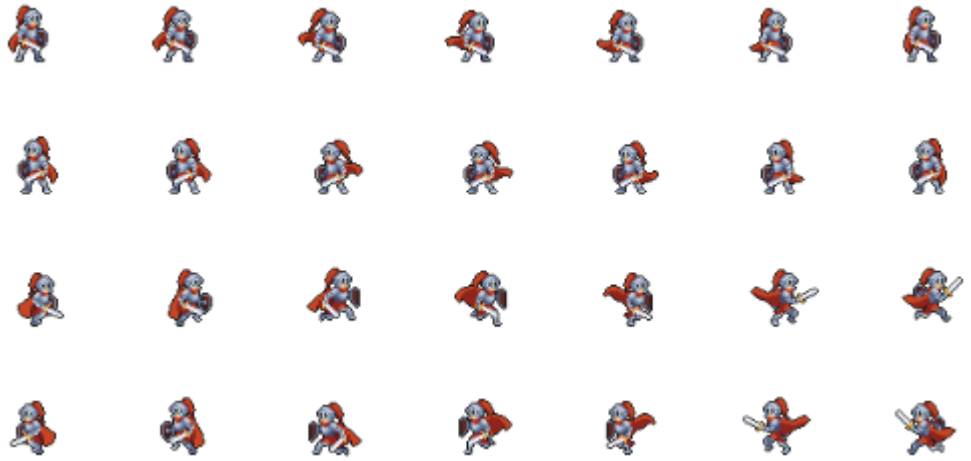


Figure: Player Animations

4.2 Monster and Traps:

The monster's movement animations are designed and built with 2 animations such as right and left movement and monsters will also accelerate attacks towards the player when within range.

About traps, we have cannon and spike traps to increase the difficulty and challenge for players when joining Escape Dungeon. The cannon includes bullets and a cannon body, the bullets fire to the left, and the body is fixed.

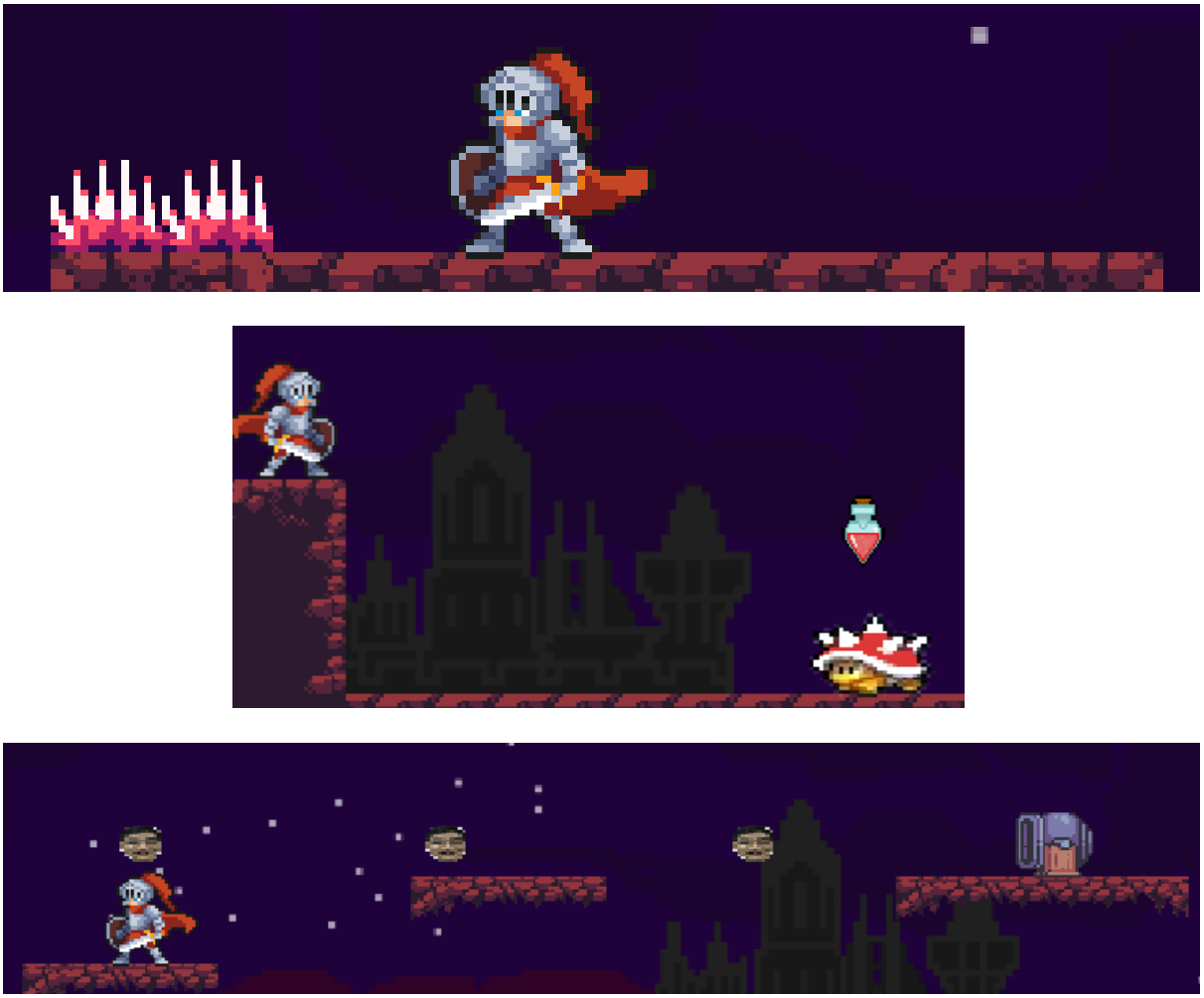


Figure: Trap Animations

4.2 NPC:

The NPC animation comprises 4 images and 1 text image to notify players about their mission. From the above images, a standing animation for the NPC is created.

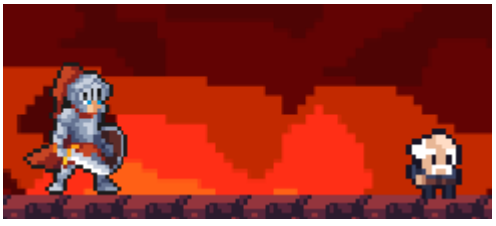


Figure: NPC Animations

5. UI

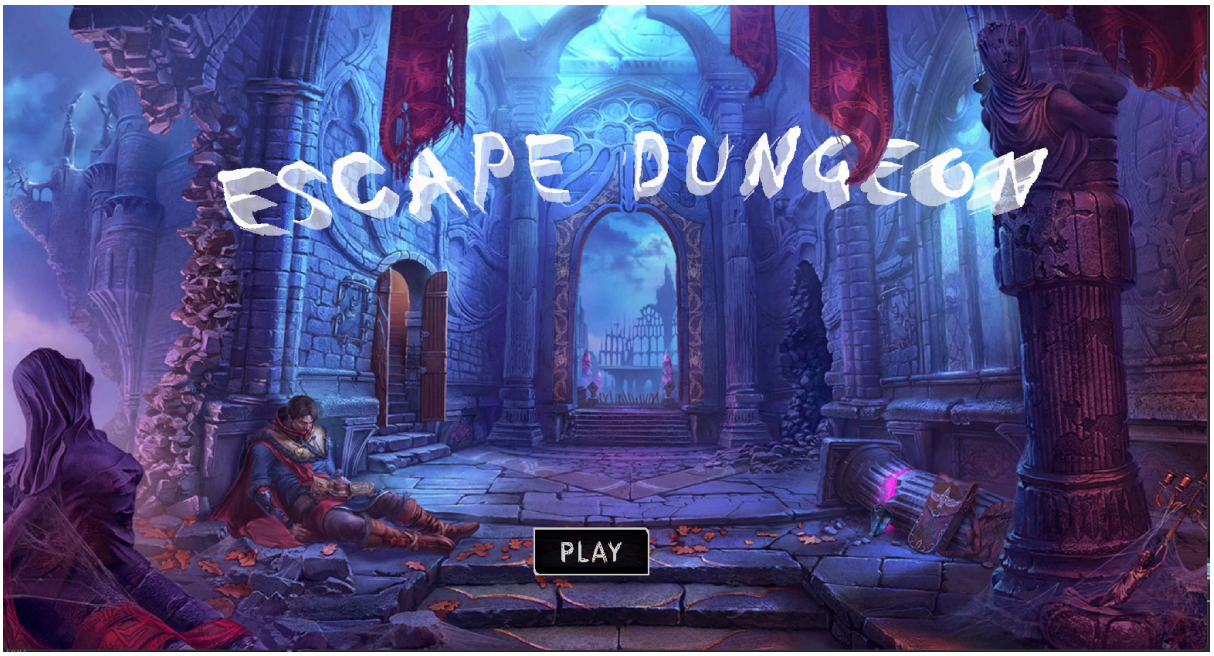


Figure: Start Game

Figure: Game over

6. Audio

The following table shows the list of audio clips with descriptions used to develop this game

Name	Description
attackSound.wav	This sound notifies player when they attack monsters
jump.wav	This sound is used for jump motion when the player jumps to pass traps and find the key.
Pitifulface.wav	First theme song
VuaHanVuaYeu.wav	Second theme song

CHAPTER 6: CONCLUSION AND FUTURE WORKS

1. Conclusion

With the game's development, the team has achieved a profound understanding of the foundational principles of Object-Oriented Programming (OOP) and the SOLID principles. This knowledge has been instrumental in refining our development skills, allowing us to create innovative features and enhance its functionality to support the game experience further. The project extensively explored encapsulation within its class structures, while inheritance, abstraction, and polymorphism were integral to the design of the enemies and player packages. These principles formed the backbone of “Reaching the Treasure”, ensuring that the game code adheres to OOP standards and integrates a key design pattern learned during our studies. This journey has not only deepened our technical expertise but also fostered a spirit of innovation and collaboration. It stands as a testament to our dedication to crafting a game that embodies both technical excellence and creative ambition.

2. Future works

Currently, due to the limitation of our skills and knowledge, we cannot bring complexity and dynamics to the gameplay along with graphics. So, in the near future, we are going after the dynamic of the world-building and the in-depth gameplay. And after that, a meaningful storyline may be the best ingredient to catch up.

Level and soundtrack are also what give the players a chance to immerse themselves into our handcrafted-world. After all, every aspect is as important as each other. But what the best we can do is try and improve each at the time to achieve the desired results.

3. Acknowledgment

In the end, we would like to express our gratitude to everyone who contributed to the development of "Reaching the Treasure".

First and foremost, we'd like to thank our instructors and mentors for guidance throughout this project. Their expertise and guidance throughout this project have been an inspiration for our game and success, helping us understand the complexities of Object-Oriented Programming (OOP) and game design principles.

Huge thanks to our teammates for making this project fun and collaborative! They brought it to life, and we couldn't have presented it without them. Thanks also to platforms like itch.io for textures, YouTube for soundtracks, and Eclipse for smooth coding.

REFERENCE

[1] Platform Game (2022)[Java]. Retrieved May 10, 2024, from https://github.com/KaarinGaming/PlatformerTutorial

[2] Java Swing Tutorial. Javatpoint. (n.d.). Retrieved December 11, 2023, from https://www.javatpoint.com/java-swing

[3] UML Diagrams Tutorial. Javatpoint. (n.d.). Retrieved January 4, 2023, from https://www.javatpoint.com/uml-diagrams

[4] Drew, O. (2022). Othneildrew/Best-README-Template. Retrieved January 4,2023, from https://github.com/othneildrew/Best-README-Template

[5] SOLID Principle in Programming: Understand With Real Life Examples. GeeksforGeeks. Retrieved December 28, 2023, from https://www.geeksforgeeks.org/solid-principle-in-programming-understand-with-real-life-examples/

[*]quintino-pixels.itch.io/tilemap-mini-dungeon, from https://quintino-pixels.itch.io/starry-night