The American University in Cairo

Project Design Sheet

Assignment # 3

Dr. Ghada Hassan

CSCE310201 - Programming in Java

Fall 2024

Hassan Mohamed

Omar Youssef

Mohamed Ayoub

1. Workflow of Program

The program begins with a welcoming interface where the player is introduced to the game. From this screen, users can customize two key parameters: the number of mines and the cannon's aggressiveness. These inputs are adjusted using sliders. Once the player finalizes their settings, they can click the "Start Game" button to proceed. All of this setup is managed within the “MainGame” class.

Upon starting the game, the Main class initializes the “GamePanel”, passing the user-defined inputs to its constructor, and makes it visible. The game then launches, presenting the player with a tank they can control. The tank moves in all eight directions using the arrow keys and fires projectiles with the spacebar. The game environment also includes a house, which acts as an enemy and targets the tank, as well as mines that are randomly placed based on the user-specified count. Additionally, a mountain serves as an obstacle, adding complexity to navigation.

The “GamePanel” class is responsible for most of the game's logic, with threading playing a pivotal role. Threading ensures the game runs smoothly at 60 frames per second, synchronizing all components such as the tank's movement and the house's targeting. This design also allows the tank to move and fire simultaneously, maintaining fluid gameplay. The GamePanel also methods from other classes such as the Tank, House, Mine, HealthBar, Missile and Obstacle each class containing its respective logic that can then be called by the GamePanel

The game continues until one of three endgame conditions is met:

- The tank collides with a mine.
- The tank's health (displayed via a health bar) is depleted.
- The tank successfully lands three hits on the house, resulting in a player victory.

If the tank's health reaches zero or it hits a mine, the player sees a game-over screen. Conversely, if the house is destroyed, the player is rewarded with a victory screen.

2. Classes

MainGame:

- Sets up the initial game window and displays a start menu panel where users can configure two settings: the number of mines and the aggressiveness of cannon missiles using sliders. The class includes a start button that, when clicked, transitions from the start menu to the main game by replacing the start panel with a GamePanel, passing the configured settings to it.

GamePanel:

- This class manages the core gameplay logic and rendering component for the "Tank Game." It extends JPanel and implements KeyListener to handle user input and control gameplay elements. The panel initializes and manages game objects such as a Tank, a House (enemy base), Mines, and an Obstacle. It also includes a HealthBar to track the tank's health.

- The game loop is managed by a background thread that processes active key presses (e.g., tank movement and rotation), updates game states (such as missile movement), checks for collisions, and repaints the graphics. Collision detection ensures interactions between missiles, mines, and objects like the tank and house. If a collision leads to a win or loss condition, the game ends with an appropriate message displayed on the screen.

- The paintComponent method handles rendering the game scene, including the tank, house, mines, and obstacle, while also displaying the game-over message when the game ends. User inputs (keyboard and mouse) allow for tank control, including movement and missile firing.

Tank:

The Tank class defines the behavior and attributes of the player's tank in the "Tank Game." It is responsible for the tank's position, movement, direction, health, and interactions with other game elements like missiles, obstacles, and the enemy house. The class also handles rendering the tank and its missiles.

1. **Attributes:**
   - The tank's position (x, y), direction, health, and speed.
   - A list of Missile objects that represent projectiles fired by the tank.
   - References to other game objects (House, Obstacle, and HealthBar) for interaction and collision logic.
   - A tankImage to visually represent the tank, updated based on its direction.
2. **Movement:**
   - moveForward and moveBackward: Move the tank in the current direction while ensuring it doesn't collide with obstacles (House or Obstacle) or move out of bounds.
   - rotateLeft and rotateRight: Change the tank's direction with a cooldown to prevent rapid rotation. The tank's image is updated based on its new direction.
3. **Missile Handling:**
   - fireMissile: Creates and launches a missile in the current direction, ensuring a cooldown between consecutive shots.
   - moveMissiles: Updates the positions of active missiles.
4. **Health Management:**
   - takeDamage: Reduces the tank's health when hit and updates the associated HealthBar.
   - isDestroyed: Returns whether the tank has been destroyed.
5. **Collision Avoidance:**
   - The tank stops moving if it collides with the House or Obstacle.
6. **Rendering:**
   - draw: Draws the tank and its missiles on the game panel.

Overview:

The `Tank` class encapsulates the player's vehicle, providing comprehensive methods for movement, combat (firing missiles), and health management. It ensures smooth interaction with other game objects and integrates visually with the game environment.

House:

The House class represents a stationary enemy base that fires missiles at the player's tank and can take damage until destroyed.

Key Features:

1. **Attributes:**
    - **Position:** x, y for the house; can_x, can_y for the cannon.
    - **Health:** Starts at 3; reduced on taking damage.
    - **Missiles:** A thread-safe CopyOnWriteArrayList stores missiles fired by the house.
    - **Aggressiveness (aggrs):** Determines firing frequency, with higher values reducing the interval.
    - **Tank Reference:** Used to aim missiles at the player.
2. **Methods:**
    - fireMissile(): Calculates the angle to the tank and fires a missile in one of eight directions.
    - moveMissiles(): Updates missile positions and removes inactive ones.
    - takeDamage(): Reduces health and logs the current status.
    - draw (Graphics g): Renders the house, cannon, and missiles using Toolkit images (house.png, cannon.png).
3. **Automated Firing:**
    - A thread continuously fires missiles at intervals based on aggrs.
4. **Thread-Safe Design:**
    - Uses CopyOnWriteArrayList for safe missile operations in a multi-threaded environment.

Overview:

The `House` is an enemy structure that attacks the tank by firing missiles while defending itself. It integrates smoothly with the game loop and reacts dynamically based on its health and aggressiveness level.

HealthBar:

- The HealthBar class represents a visual health bar component in the "Tank Game." It extends JPanel and is responsible for displaying the current health of a game entity (e.g., the tank) as a colored bar. The health bar dynamically updates its appearance based on the entity's health level.

- The class initializes with a specified initialHealth and maxHealth, ensuring health values are within a valid range. It provides a setHealth method to update the health and repaint the bar accordingly. The paintComponent method visually represents the health bar by filling it with a color that reflects the health level (e.g., green for full health, yellow for medium, orange for low, and red for critical).

Mine:

- The Mine class represents a mine in the "Tank Game." Each mine has a fixed position on the game field, defined by its x and y coordinates. The class provides getter methods (getX and getY) to access its position, making it useful for collision detection and gameplay logic.

- The draw method renders the mine on the game panel. It uses the Graphics object to draw a visual representation of the mine, typically as an image loaded from a file (e.g., "bomb.png"). The image is displayed at the mine's position (x, y) with a fixed size of 70x70 pixels

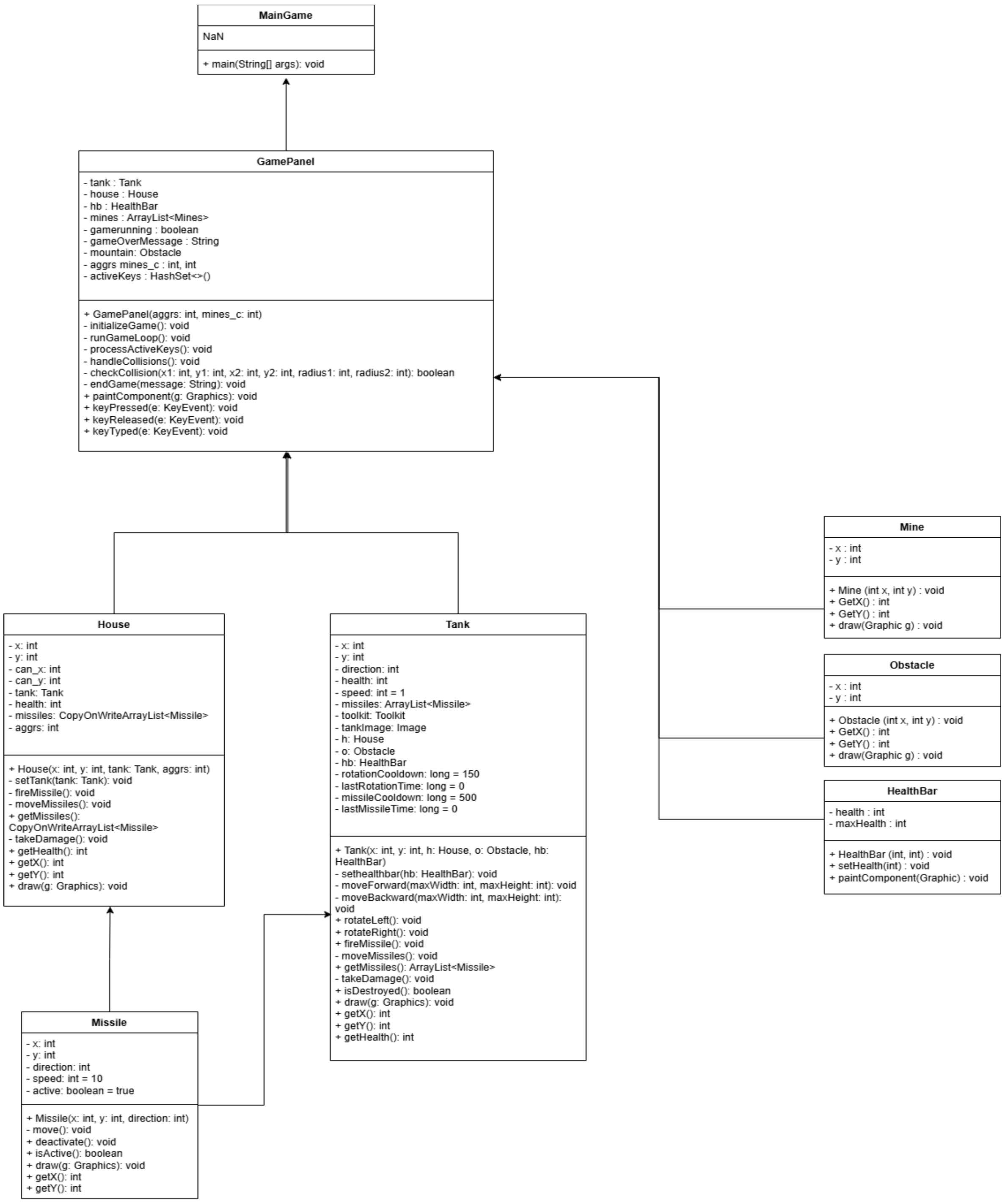
Missile:

The Missile class represents a moving projectile in a game, with attributes such as position (x, y), direction, speed, and an active state. The missile moves in one of eight possible directions, controlled by the direction attribute, and updates its position every time the move() method is called. If the missile moves off the screen or collides with an object, it deactivates itself. The draw() method renders the missile as a small red circle on the screen, but only if it's still active. The class also provides methods to manually deactivate the missile and check its status, ensuring it behaves correctly during gameplay.

Obstacle:

The Obstacle class represents an obstacle in the game, such as a mountain, that is positioned at specific x and y coordinates. It provides getter methods to retrieve these coordinates, and the draw() method renders an image of the obstacle (e.g., a mountain) at the specified location on the screen. The class utilizes the Toolkit to load the image, and the image is drawn at a fixed size of 200x120 pixels. This simple structure allows the obstacle to be drawn and interacted with in the game world.

3. Class Diagram


MainGame
NaN
+ main(String[] args): void
GamePanel
- tank : Tank
- house : House
- hb : HealthBar
- mines : ArrayList<Mines>
- gamerunning : boolean
- gameOverMessage : String
- mountain: Obstacle
- aggrs mines_c : int, int
- activeKeys : HashSet<>()
+ GamePanel(aggrs: int, mines_c: int)
- initializeGame(): void
- runGameLoop(): void
- processActiveKeys(): void
- handleCollisions(): void
- checkCollision(x1: int, y1: int, x2: int, y2: int, radius1: int, radius2: int): boolean
- endGame(message: String): void
+ paintComponent(g: Graphics): void
+ keyPressed(e: KeyEvent): void
+ keyReleased(e: KeyEvent): void
+ keyTyped(e: KeyEvent): void
Mine
- x : int
- y : int
+ Mine (int x, int y) : void
+ GetX() : int
+ GetY() : int
+ draw(Graphic g) : void
Obstacle
- x : int
- y : int
+ Obstacle (int x, int y) : void
+ GetX() : int
+ GetY() : int
+ draw(Graphic g) : void
HealthBar
- health : int
- maxHealth : int
+ HealthBar (int, int) : void
+ setHealth(int) : void
+ paintComponent(Graphic) : void
House
- x: int
- y: int
- can_x: int
- can_y: int
- tank: Tank
- health: int
- missiles: CopyOnWriteArrayList<Missile>
- aggrs: int
+ House(x: int, y: int, tank: Tank, aggrs: int)
- setTank(tank: Tank): void
- fireMissile(): void
- moveMissiles(): void
+ getMissiles(): CopyOnWriteArrayList<Missile>
- takeDamage(): void
+ getHealth(): int
+ getX(): int
+ getY(): int
+ draw(g: Graphics): void
Tank
- x: int
- y: int
- direction: int
- health: int
- speed: int = 1
- missiles: ArrayList<Missile>
- toolkit: Toolkit
- tankImage: Image
- h: House
- o: Obstacle
- hb: HealthBar
- rotationCooldown: long = 150
- lastRotationTime: long = 0
- missileCooldown: long = 500
- lastMissileTime: long = 0
+ Tank(x: int, y: int, h: House, o: Obstacle, hb: HealthBar)
- sethealthbar(hb: HealthBar): void
- moveForward(maxWidth: int, maxHeight: int): void
- moveBackward(maxWidth: int, maxHeight: int): void
+ rotateLeft(): void
+ rotateRight(): void
+ fireMissile(): void
- moveMissiles(): void
+ getMissiles(): ArrayList<Missile>
- takeDamage(): void
+ isDestroyed(): boolean
+ draw(g: Graphics): void
+ getX(): int
+ getY(): int
+ getHealth(): int
Missile
- x: int
- y: int
- direction: int
- speed: int = 10
- active: boolean = true
+ Missile(x: int, y: int, direction: int)
- move(): void
+ deactivate(): void
+ isActive(): boolean
+ draw(g: Graphics): void
+ getX(): int
+ getY(): int