Name: Bakhtishod Umurzokov
Neptun code: Y62UF3
Assignment 3rd. Task 2nd.
Group: № 8.

Task

We have a rattlesnake in a desert, and our snake is initially two units long (head and rattler). We have to collect with our snake the foods on the level, that appears randomly. Only one food piece is placed randomly at a time on the level (on a field, where there is no snake).
The snake starts off from the center of the level in a random direction. The player can control the movement of the snake's head with keyboard buttons. If the snake eats a food piece, then its length grow by one unit.
It makes the game harder that there are rocks in the desert. If the snake collides with a rock, then the game ends. We also lose the game, if the snake goes into itself, or into the boundary of the game level.
In these situations show a popup messagebox, where the player can type his name and save it together with the amount of food eaten to the database. Create a menu item, which displays a highscore table of the players for the 10 best scores. Also, create a menu item which restarts the game.

Description of the task

The objective of this project is to create a user-friendly Snake game implemented using Java Swing. The game should follow an object-oriented approach, with the following requirements:

- **Gameplay**: The player controls a snake using the WASD or arrow keys to eat food, avoiding collisions with obstacles, walls, and itself.
- **Levels**: The game should include 10 predefined levels, each loaded from a file, with varying obstacle patterns.
- **Timer**: Tracks the elapsed time for each game session.
- **Scoring**: Tracks the player's score based on the snake's length.
- **Persistence**: Stores high scores in a database and displays them.
- **UI**: Includes a menu screen, game screen, and a score panel. The menu bar is visible only during gameplay.

Analysis

1. **User Interface**:
    - Menu screen with options to start the game, view high scores, and select levels.
    - Game screen with a grid-based display and a score panel.
    - Dynamic menu bar visible only during gameplay.
2. **Game Mechanics**:
    - The snake moves on a 20x20 grid, eating food to grow in size.
    - Collision detection determines when the game ends.
3. **Levels**:
    - Levels are predefined in `.txt` files, stored in a `levels` folder.
    - Each file specifies the positions of obstacles, the snake's starting point, and food.
4. **High Scores**:
    - Stored in a MySQL database and displayed in the high scores screen.

Classes and Responsibilities

1. SnakeGame:
    - Handles the overall UI and game flow.
    - Manages the main panel with screens for the menu and game.
    - Dynamically adds/removes the score panel and menu bar.
2. GameLogic:
    - Core game mechanics, including movement, collision detection, and food spawning.
    - Handles game states like running or stopped.
3. GamePanel:
    - Responsible for rendering the game grid and drawing the snake, obstacles, and food.
4. ScorePanel:
    - Displays the current score and elapsed time during gameplay.
5. DatabaseManager:
    - Manages high scores, including storing and retrieving scores from the database.
6. HighScoresManager:
    - Retrieves high scores from DatabaseManager and displays them.

Connection of event handlers:

Key Press:

- Arrow/WASD keys change the snake's direction via KeyAdapter.

Button Click:

- Start button triggers startGame().
- High Scores button calls showHighScores().

Timer Events:

- Game timer (gameTimer) drives the game loop.
- Elapsed time timer updates the ScorePanel.

Interesting Algorithms

Level Loading:

Levels are loaded from .txt files. Each file represents a 20x20 grid where:

- 1 indicates an obstacle.
- S is the snake's starting position.
- F is the food position. The GameLogic class parses these files and initializes the grid.

Collision Detection:

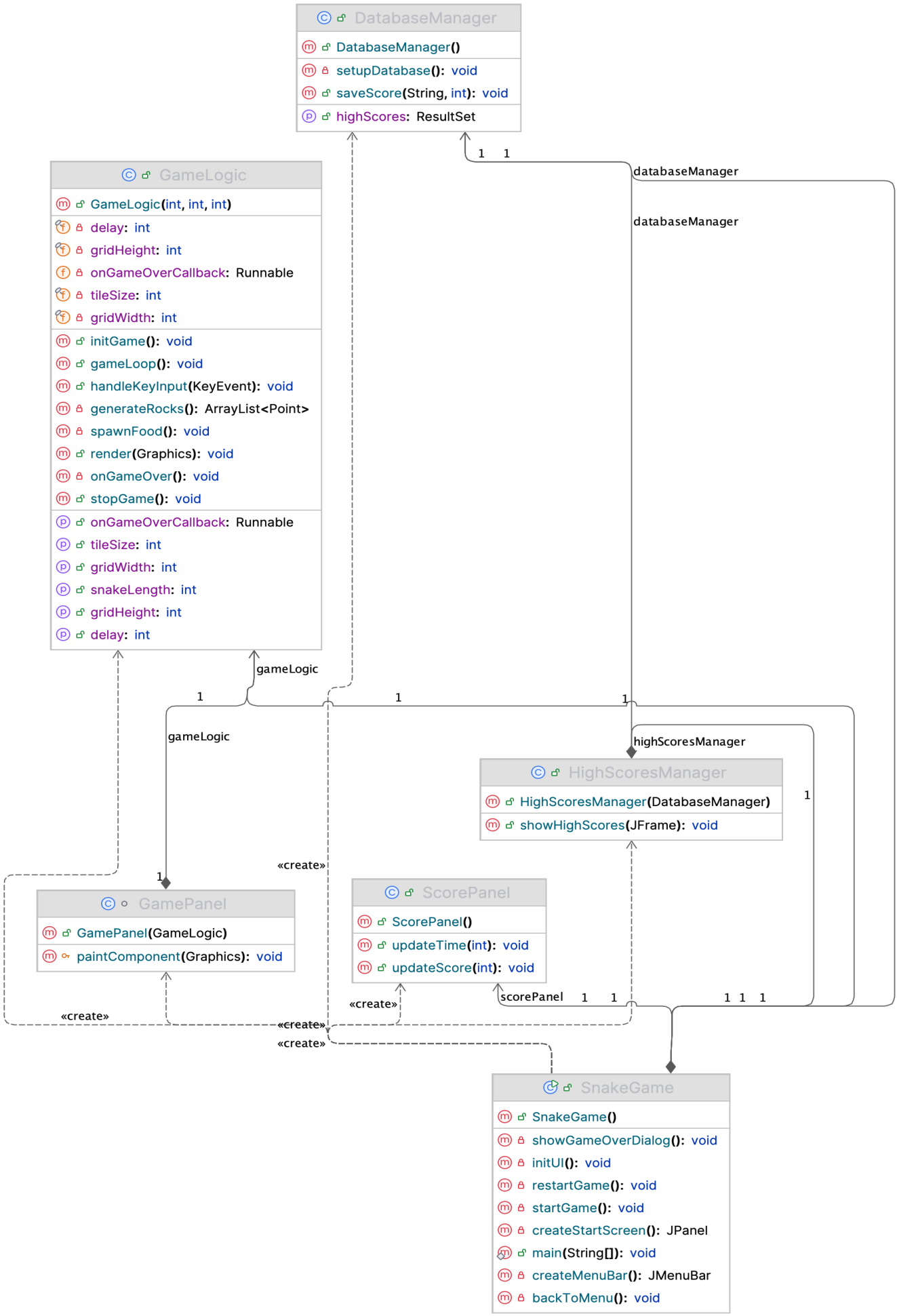
Detects when the snake collides with:

- Walls: The head moves outside the grid.
- Itself: The head occupies the same position as another segment.
- Obstacles: The head moves into a rock.

Timer and Score Updates:

The Timer updates elapsed time and score dynamically via the ScorePanel.

Class diagram



Testing

1. Start Game Successfully

- **AS A** user
- **I WANT TO** start the game
- **GIVEN** the menu screen is visible
- **WHEN** I click the "Start Game" button
- **THEN** the game starts, the snake appears on the grid, and the score panel is visible.

---

2. Select a Level

- **AS A** user
- **I WANT TO** select a specific level
- **GIVEN** the level selector dropdown is visible
- **WHEN** I select a level and start the game
- **THEN** the corresponding level loads, showing the appropriate obstacles, snake position, and food.

---

3. Snake Collides with a Wall

- **AS A** player
- **I WANT TO** end the game if the snake collides with a wall
- **GIVEN** the snake is moving near the grid boundary
- **WHEN** the snake's head moves outside the grid
- **THEN** the game ends, and the game-over dialog is displayed.

---

4. Snake Collides with Itself

- **AS A** player
- **I WANT TO** end the game if the snake collides with itself
- **GIVEN** the snake is moving in a tight space
- **WHEN** the snake's head moves into its own body
- **THEN** the game ends, and the game-over dialog is displayed.

---

5. Snake Collides with Obstacles

- **AS A** player
- **I WANT TO** end the game if the snake collides with obstacles
- **GIVEN** a level with obstacles
- **WHEN** the snake's head moves into an obstacle
- **THEN** the game ends, and the game-over dialog is displayed.

---

6. Eating Food

- **AS A** player
- **I WANT TO** grow the snake's size and increase the score when eating food
- **GIVEN** food is present on the grid
- **WHEN** the snake's head moves onto the food tile
- **THEN** the snake grows, the score updates, and a new piece of food spawns.

---

7. Restart the Game

- **AS A** player
- **I WANT TO** restart the game after a failure
- **GIVEN** the game-over dialog is visible
- **WHEN** I click the "Restart" option from the menu
- **THEN** the game restarts, and the snake returns to the starting position.

---

8. Open High Scores

- **AS A** user
- **I WANT TO** view the high scores
- **GIVEN** the menu screen is visible
- **WHEN** I click the "High Scores" button
- **THEN** the high scores dialog opens, showing the top scores.

---

9. High Score Saving

- **AS A** player
- **I WANT TO** save my score after the game ends
- **GIVEN** the game-over dialog is displayed
- **WHEN** I enter my name
- **THEN** my score is saved to the database.

---

10. Return to Menu During Gameplay

- **AS A** player
- **I WANT TO** return to the menu screen during gameplay
- **GIVEN** the game is in progress
- **WHEN** I click "Back to Menu" from the menu bar
- **THEN** the game stops, and the menu screen is displayed.

---

11. Timer Stops on Game Over

- **AS A** player
- **I WANT TO** stop the timer when the game ends
- **GIVEN** the game is running
- **WHEN** the snake collides with a wall, obstacle, or itself
- **THEN** the timer stops, and the elapsed time is recorded.

---

12. Resume Level Selection After Returning to Menu

- **AS A** user
- **I WANT TO** select a new level after returning to the menu
- **GIVEN** I returned to the menu screen from the game
- **WHEN** I select a different level and start the game
- **THEN** the selected level loads, and gameplay begins.

---

13. Handle Invalid File Load

- **AS A** developer
- **I WANT TO** handle cases where a level file is missing or corrupted
- **GIVEN** a corrupted or missing level file
- **WHEN** the game tries to load the level

- **THEN** an error is logged, and the game returns to the menu.

---

14. Snake Movement After Quick Direction Change

- **AS A** player
- **I WANT TO** ensure the snake doesn't move into itself after a quick direction change
- **GIVEN** the snake is moving
- **WHEN** I quickly press opposite direction keys (e.g., UP and DOWN)
- **THEN** the snake ignores the invalid direction and continues moving safely.

---

15. Snake Growth Boundaries

- **AS A** player
- **I WANT TO** ensure the snake grows properly without exceeding the grid
- **GIVEN** the snake is eating food near the edge of the grid
- **WHEN** the snake grows
- **THEN** it avoids collisions and remains within valid boundaries.

Test cases:

1) https://drive.google.com/file/d/1-3-hRJMT2SGXTmGnHZu9jMArM0WUT-KQ/view?usp=sharing
2) https://drive.google.com/file/d/10jyDwNsSKuv5y5T7ANV1TNMYoF2MHufO/view?usp=sharing