Assignment 4 - Game Engine Design - UML

# Student Name:

Olesia Mykhailyshyn

# Group:

ОПД 2

# Task:

The task involves designing and implementing a Snake game engine using UML (Unified Modeling Language) diagrams to describe the structure and behavior of the system. The game engine manages the core logic, including player controls, snake movement, food interaction, game win/lose conditions, and the scoreboard system for tracking high scores.

System design: structural and behavioral diagrams were created to describe the game engine components using UML.

Class diagram: Design the main classes (e.g., Game, Snake, Food, Board, Scoreboard), their attributes, methods, and relationships.

Sequence diagram: Illustrate the interactions between objects during gameplay.

Activity and state diagrams: Detail the process flow and state transitions in the game.

Use case diagram: Describe player interactions with the game.

The game includes unique features, such as different types of food (e.g., normal, poisonous), and a scoreboard system to track and display high scores.

**System Model:**

# 

# Зображення, що містить текст, почерк, чорнило, папір Автоматично згенерований опис

# Solution Description:

1. System Design

* Game: The main controller of the system that initializes and manages the game loop, creating the snake, board, food, and scoreboard.
* Board: Responsible for rendering the game environment, including drawing the grid and updating the display based on the state of the snake and food.
* Snake: Represents the snake, consisting of multiple Point objects, with methods to move, grow, shrink, and change direction. The snake's behavior changes depending on the food it eats, which transforms it into different types of snakes (e.g., BigSnake, SmallSnake).
* Food: Abstract base class representing food, with two derived classes (NormalFood and PoisonousFood). Each type of food applies different effects on the snake (e.g., growth, shrinkage).
* Point: Represents a coordinate on the board, used for the snake's segments and the food position.
* Scoreboard: Manages the saving, loading, and displaying of scores. It tracks the player's performance across games and maintains a high score list.
* GameState: Handles the state of the game, such as when it is running, paused, or over, allowing the player to pause or end the game at any time.

2. Design Approach

The game was designed using several relations between classes:

* Composition: The Game class is composed of Board, Snake, Food, and Scoreboard components. The board and snake exist only as long as the game is running, which ensures tight coupling between these elements.
* Aggregation: The Scoreboard is loosely coupled with the game, as the results persist beyond the game session. It is responsible for saving and loading scores across multiple games.
* Inheritance: The Snake and Food classes follow an inheritance model where different types of snakes and food extend the base classes with specific behavior.

3. Behavioral Logic

* Main game loop: The game continuously updates the state of the snake and the board. It processes user inputs to move the snake and updates the display accordingly.
* Score tracking: After the game ends, the player's score is saved to a file, and the top scores are displayed to encourage replayability.

4. Development Process

The software was developed iteratively, with a focus on modularity, making it easy to extend or modify individual components without affecting the whole system. UML diagrams (class, sequence, activity, and state diagrams) were used throughout the process to visualize the relationships between classes, manage the complexity of interactions, and ensure consistency in the design.

5. Key Features

* Dynamic food effects: The snake’s behavior changes based on the type of food consumed. For example, eating poisonous food reduces its size and alters its appearance.
* Score management: The system tracks and displays the player’s score and maintains a history of high scores.
* Pausing and quitting: The game supports pausing and resuming the game, as well as quitting early, with all scores being saved automatically.

# Appendices:

<https://www.youtube.com/watch?v=xtZHJxYA6q8&list=PLiZZKL9HLmWMF8PlzvZu2WOC9kjs1zzhm>

<https://www.youtube.com/watch?v=eXbYHcOkZV4&t=1658s>

**DIAGRAMS**

**CLASS DIAGRAM**

@startuml

class Game {

- Board board

- Snake snake

- Food food

- Scoreboard scoreboard

- GameState state

+ Run()

+ HandleInput()

+ CreateNewFood()

+ SaveScore()

}

class Board {

+ void DrawBoard()

+ void Clear()

}

class Snake {

- vector<Point> body

- char direction

- char symbol

- string color

+ Move(width, height) virtual

+ AddCell() virtual

+ RemoveCell() virtual

+ Point GetHead() virtual

+ Draw() virtual

+ SetDirection(direction) virtual

}

class BigSnake {

+ AddCell() override

+ RemoveCell() override

}

class SmallSnake {

+ AddCell() override

+ RemoveCell() override}

class Point {

+ SetPoint(x, y)

+ GetX()

+ GetY()

+ Draw()

+ IsEqual(Point)?

}

class Food {

- vector<Point> body

+ Point position virtual = 0

+ ApplyEffect(Snake& snake) virtual = 0

+ Draw() virtual = 0

}

class NormalFood {

+ ApplyEffect(Snake& snake) override

+ Draw() override

}

class PoisonousFood {

+ ApplyEffect(Snake& snake) override

+ Draw() override

}

class Scoreboard {

+ void SaveToFile()

+ void LoadFromFile()

+ void AddScore(playerName, snakeLength, gameTime)

+ void ShowTopScores()

}

enum GameState {

RUNNING

PAUSED

GAME\_OVER

}

Food <|-- NormalFood : override

Food <|-- PoisonousFood : override

Snake <|-- BigSnake : override

Snake <|-- SmallSnake : override

Game \*-- Board : Composition

Game \*-- Snake : Composition

Board \*-- Snake : Composition

Board o-- Food : Aggregation

Snake \*-- Point : Composition

Food \*-- Point : Composition

Game \*-- GameState : uses

Game \*-- Board : Composition

Game \*-- Snake : Composition

Game o-- Food : Aggregation

Game o-- Scoreboard : Aggregation

Game ..> GameState : dependency

@enduml

Зображення, що містить текст, схема, План, ряд

Автоматично згенерований опис

**SEQUENCE DIAGRAM**

@startuml

actor Player

participant Game

participant Board

participant Snake

participant Food

participant Scoreboard

participant Point

== Use Case: Start Game ==

Player -> Game: Presses key to start the game

activate Game

Game -> Board: Creates the board

activate Board

deactivate Board

Game -> Snake: Creates the snake

activate Snake

deactivate Snake

Game -> Food: Creates the food

activate Food

deactivate Food

Game -> Scoreboard: Loads previous results

activate Scoreboard

deactivate Scoreboard

== Use Case: Main game loop ==

Player -> Game: Starts the game loop

loop Main game loop

Game -> Snake: Receives input (direction)

activate Snake

Snake -> Snake: Moves the snake

Snake -> Point: Updates segment coordinates

activate Point

deactivate Point

deactivate Snake

Game -> Board: Clears the board

activate Board

Game -> Board: Draws the board

deactivate Board

Game -> Snake: Draws the snake

activate Snake

deactivate Snake

Game -> Food: Draws the food

activate Food

deactivate Food

alt Snake eats the food

Snake -> Food: Checks for collision with food

activate Food

Food -> Snake: Applies food effect

deactivate Food

Game -> Food: Creates new food

end

alt Snake collided with itself

Snake -> Game: Snake dies

Game -> Scoreboard: Saves the result

activate Scoreboard

deactivate Scoreboard

Game -> Player: Displays game result

Game -> Scoreboard: Shows top scores

end

end

== Use Case: End Game ==

Player -> Game: Ends the game

Game -> Board: Cleans up

Game -> Snake: Cleans up

Game -> Food: Cleans up

Game -> Scoreboard: Cleans up

deactivate Game

@enduml

Зображення, що містить текст, знімок екрана, схема, Паралель

Автоматично згенерований опис

**Activity Diagram: Game Flow**

@startuml

start

:Initialize Game;

:Create Board;

:Create Snake;

:Create Food;

:Load Scoreboard;

:Start Game Loop;

while (Is Game Over?)

:Receive Input;

:Move Snake;

if (Snake Eats Food?) then (yes)

:Apply Food Effect;

:Create New Food;

endif

if (Snake Collided with Itself?) then (yes)

:Set Game Over;

endif

:Draw Board;

:Draw Snake;

:Draw Food;

endwhile

:Save Score;

:Show Game Results;

:End Game;

stop

@enduml

Зображення, що містить текст, квитанція

Автоматично згенерований опис

**Activity Diagram: Snake Movement**

@startuml

start

:Player Input Direction;

if (Direction Valid?) then (yes)

:Move Snake Head;

:Update Snake Body;

if (Snake Collided with Wall or Itself?) then (yes)

:Set Game Over;

endif

else (no)

:Ignore Input;

endif

stop

@enduml

Зображення, що містить текст, знімок екрана, схема, Шрифт

Автоматично згенерований опис

**Activity Diagram: Food Interaction**

@startuml

start

:Snake Head Moves;

if (Snake Eats Food?) then (yes)

:Apply Food Effect;

if (Food is Normal?) then (yes)

:Increase Snake Length;

else (no)

:Apply Poisonous Effect;

:Reduce Snake Length;

endif

:Generate New Food;

else (no)

:Continue Game;

endif

stop

@enduml

Зображення, що містить текст, знімок екрана, схема, Шрифт

Автоматично згенерований опис

**Activity Diagram: Game Over**

@startuml

start

if (Player Ends Game or Snake Collided?) then (yes)

:Set Game Over State;

:Save Score;

:Show Game Results;

else (no)

:Continue Game Loop;

endif

stop

@enduml

Зображення, що містить текст, знімок екрана, схема, ряд

Автоматично згенерований опис

**State Diagram: Game States**

@startuml

[\*] --> GameRunning

GameRunning : Player controls the snake\nGame is updating

GameRunning --> GamePaused : Player presses 'P'

GameRunning --> GameOver : Snake collides with itself

GamePaused : Game is paused\nNo updates

GamePaused --> GameRunning : Player presses 'P'

GameOver : Game is over\nDisplay final results

GameOver --> [\*] : Player exits

@enduml

Зображення, що містить текст, схема, ряд, знімок екрана

Автоматично згенерований опис

**State Diagram: Snake Movement**

@startuml

[\*] --> MovingRight : Start with initial direction

MovingRight --> MovingUp : Player presses 'W'

MovingRight --> MovingDown : Player presses 'S'

MovingUp --> MovingLeft : Player presses 'A'

MovingUp --> MovingRight : Player presses 'D'

MovingDown --> MovingLeft : Player presses 'A'

MovingDown --> MovingRight : Player presses 'D'

MovingLeft --> MovingUp : Player presses 'W'

MovingLeft --> MovingDown : Player presses 'S'

MovingRight --> GameOver : Snake collides with itself

MovingUp --> GameOver : Snake collides with itself

MovingDown --> GameOver : Snake collides with itself

MovingLeft --> GameOver : Snake collides with itself

@enduml

Зображення, що містить схема, текст, ескіз, малюнок

Автоматично згенерований опис

**State Diagram: Food Interaction**

@startuml

[\*] --> NormalFoodPresent : Game starts with normal food

NormalFoodPresent --> FoodEaten : Snake eats the food

FoodEaten --> NormalFoodPresent : Create new normal food

NormalFoodPresent --> PoisonousFoodPresent : Random chance generates poisonous food

PoisonousFoodPresent --> FoodEaten : Snake eats the food

FoodEaten --> PoisonousEffect : Apply effect (e.g., shrink snake)

PoisonousEffect --> NormalFoodPresent : Return to normal state

@enduml

Зображення, що містить текст, схема, ескіз, ряд

Автоматично згенерований опис

**Use Case Diagram**

@startuml

actor Player

rectangle "Snake Game" {

usecase "Start Game" as UC\_StartGame

usecase "Move Snake" as UC\_MoveSnake

usecase "Pause Game" as UC\_PauseGame

usecase "Eat Food" as UC\_EatFood

usecase "Game Over" as UC\_GameOver

usecase "Save Score" as UC\_SaveScore

usecase "Show Top Scores" as UC\_ShowScores

}

Player --> UC\_StartGame

Player --> UC\_MoveSnake : Arrow keys (W, A, S, D)

Player --> UC\_PauseGame : Press 'P'

Player --> UC\_EatFood

Player --> UC\_GameOver

Player --> UC\_ShowScores

UC\_GameOver --> UC\_SaveScore

UC\_SaveScore --> UC\_ShowScores

@enduml

Зображення, що містить схема, ряд, ескіз, малюнок

Автоматично згенерований опис