|  |  |
| --- | --- |
|  | **Namal University Mianwali**  **Department of Computer Science** |

**Assignment 2**

|  |  |  |  |
| --- | --- | --- | --- |
| Course | Software Engineering | | |
| Instructor | Asiya Batool | Session / Semester | 2023-2027 (3rd) |

|  |  |
| --- | --- |
| **Name** | **Roll no** |
| Laiba Shaheen | NUM-BSCS-2023-16 |

### ****Implementation Plan for AntFarm Simulation****

### ****1. Class Definitions****

#### **Meadow (Singleton Pattern)**

**Purpose**: Central controller for the simulation.

**Responsibilities**:

* Ensure only one Meadow instance exists.
* Maintain a collection of AntFarm instances.
* Manage global ticks across all AntFarms.

**Key Methods**:

* spawnAntFarm(int x, int y, std::string species): Creates and adds a new ant farm to the meadow.
* tick(int t = 1): Advances the simulation by t ticks for all farms.
* summary(int id): Provides details about the state of a specific ant farm.
* allocateResources(int id, int resources, int ants): Allocates resources and assigns additional ants to a specific farm.

#### **AntFarm (Builder Pattern)**

**Purpose**: Represents an individual ant colony.

**Responsibilities**:

* Contains a collection of AntRoom instances.
* Holds a specific species of ants (template-based to allow flexibility).

**Key Methods**:

* buildRoom(int capacity): Constructs a new room for the ant farm with the given capacity.
* addAnt(Ant\* ant): Adds a new ant to the farm.
* tickActions(): Executes the farm's actions during a simulation tick, including room construction and ant activities.

#### **AntRoom (Factory Pattern)**

**Purpose**: Represents a room within an ant farm, such as a resting area, food storage, or egg-laying chamber.

**Responsibilities**:

* Tracks its capacity, workers assigned to it, and construction progress.

**Key Methods**:

* assignWorkers(int workers): Assigns worker ants to the room.
* build(): Increments the construction progress based on the number of workers assigned.
* isComplete(): Checks if the room has been fully constructed.

**Ant Creation**:

Factory methods create Drone and Warrior ants to populate the farm.

#### **Ant (Decorator Pattern)**

**Purpose**: Represents an individual ant, with attributes enhanced dynamically using decorators.

**Responsibilities**:

* Base class for all ant types, including drones, warriors, and queens.
* Allows additional attributes (e.g., strength, efficiency) to be layered dynamically.

**Key Methods**:

* getStrength(): Returns the ant's total strength.
* getEfficiency(): Returns the ant's total efficiency.
* addDecorator(int bonusStrength, int bonusEfficiency): Dynamically adds bonuses to the ant's attributes.

#### **Queen (Factory Pattern)**

**Purpose**: Specialized ant responsible for spawning eggs and managing the colony.

**Responsibilities**:

* Acts as the leader of the colony, ensuring its growth and survival.
* Spawns new ants at intervals.

**Key Methods**:

* spawnAnt(): Generates a new ant for the colony.

#### **Simulation Mediator (Mediator Pattern)**

**Purpose**: Coordinates actions between ants and farms during a simulation tick.

**Responsibilities**:

* Ensures proper sequencing and execution of actions for all ants and farms.
* Randomizes the order of ant actions to simulate real-world variability.

**Key Methods**:

* handleTick(std::vector<AntFarmBase\*>& farms): Executes all actions for a single tick, including battles and resource management.
* initiateBattle(Warrior\* attacker, Warrior\* defender): Handles battles between warriors.

### ****2. Key Features****

#### **Tick-Based Simulation (Mediator Pattern)**

**Description**:

* + - * Each tick represents a discrete step in the simulation.
      * During a tick:
        + Worker ants construct rooms or gather food.
        + Warriors defend the colony or engage in battles.
        + The queen spawns eggs to grow the colony.
      * The mediator ensures the random execution order of these actions.

**Implementation**:

* + A global tick() function triggers actions across all farms.
  + Random number generation determines the sequence of events.

#### **Battle Mechanism**

**Description**:

* Warriors can engage in battles with ants from other colonies.
* Battles determine a winner based on the strength attribute.
* The loser is removed from the colony, and the winner inherits the loser's strength and efficiency as a bonus.

**Implementation**:

* initiateBattle(Warrior\* attacker, Warrior\* defender):
  + - * Compare strengths to decide the winner.
      * Update the winner’s attributes and remove the loser from the farm.

#### **Room Construction**

**Description**:

* Workers collectively build rooms for the colony.
* Progress depends on the number of workers assigned and the duration of construction.

**Implementation**:

* + Each tick increases the room's construction progress based on the assigned workers.
  + Completed rooms provide benefits like increased resting capacity or storage space.

#### **Resting and Food Consumption**

**Description**:

* + Ants rest in available rooms, consuming food proportional to their efficiency.
  + Resting ants regain energy to perform their roles effectively in subsequent ticks.

**Implementation**:

* + A rest() method cycles ants into available rooms.
  + Food is deducted from the farm’s resources.

### ****3. Interface Implementation****

#### **Command-Line Interface (CLI)**

**Purpose**:

* + Provides a simple text-based interface for user interaction.

**Commands**:

1. **spawn X Y T**:
   * + Creates a new colony at position (X, Y) with species T.
     + Calls Meadow::spawnAntFarm() to add the colony.
2. **give I R A**:
   * + Allocates R resources and A additional ants to colony I.
     + Calls Meadow::allocateResources() for resource allocation.
3. **tick [T]**:
   * + Advances the simulation by T ticks (default is 1).
     + Calls Meadow::tick(T) to trigger farm actions.
4. **summary I**:
   * + Displays the status of colony I, including its resources, ants, and room progress.
     + Calls Meadow::summary(I) to retrieve details.