Implementation of AntFarm Simulation using Design Patterns

Semester Assignment 02

Session 2024-2025



|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  |  | | --- | --- | | **Name:** | Gulfam Afzal | | **Roll Number:** | NUM-BSCS-2023-31 | | **Submitted To:** | MS.Asiya Batool | |  |
|  |  |  |
|  |  |  |

Department of Computer Science

Namal University Mainwali

28 December 2024

**Introduction**

This document outlines the design and implementation of an AntFarm Simulation System. The system simulates the behavior and management of ants within a colony, incorporating several design patterns including Creational, Structural, and Behavioral patterns. The goal is to create an interactive simulation where ants perform different tasks, interact with each other, and the colony itself evolves based on actions such as foraging, battling, and building. The system is built using C++ and focuses on modularity, flexibility, and extensibility.

**Overview of the System**

The AntFarm simulation system is designed to simulate the behavior of ant colonies, including activities such as spawning new ants, performing various tasks (e.g., foraging, hunting), and interacting with other colonies. The simulation is tick-based, where each "tick" represents the passing of time, and ants perform actions based on their role and the current state of the colony.

**Core Design Patterns Used**

1. **Creational Patterns**:
   * **Singleton**: The Meadow class is implemented as a singleton, ensuring that there is only one instance managing all ant farms in the simulation.
   * **Factory Method**: The AntFactory class is used to create specific types of ants (e.g., Warrior, Queen, Drone, etc.). The factory pattern allows the creation of ants of different types without directly instantiating them in the AntFarm class.
   * **Builder Pattern**: The process of constructing an ant farm involves building rooms that serve different purposes (e.g., WorkerRoom, WarriorRoom). The Builder pattern is used to modularly construct the ant farm by adding various room types.
2. **Structural Patterns**:
   * **Decorator Pattern**: Ants can have additional attributes (e.g., enhanced strength or health) using the Decorator pattern, which allows the extension of ant attributes without modifying the base Ant class.
   * **Template Method**: The AntFarm is templated to work with different types of ants, providing flexibility in the type of ants it holds and manages.
3. **Behavioral Patterns**:
   * **Mediator Pattern**: The simulation is tick-based. The Meadow acts as a mediator that manages the interactions between multiple AntFarm instances during each tick. This allows for better control of the simulation's flow, as ants can interact with other ants, colonies, and their environment based on their roles.

**Class Structure**

**1. Meadow Class (Singleton)**

* **Responsibility**: Manages the overall environment, ensuring that there is only one instance of the Meadow class. It spawns and tracks multiple ant farms and colonies.
* **Key Methods**:
  + **getInstance(int studentRollNumber**): Returns the singleton instance of the Meadow class.
  + **spawnColony(int x, int y, const string& species**): Spawns a new ant farm (colony) at a specified location with a specified species of ants.
  + **getAntFarm(int id):** Retrieves a specific ant farm by its unique ID.

**2. AntFarmBase Class**

* **Responsibility**: A base class for ant farms, which defines the essential methods that each derived ant farm must implement.
* **Key Methods**:
  + **tick():** Advances the simulation by one tick, performing actions for all ants in the farm.
  + **displaySummary():** Displays the current state of the colony.
  + **giveResource(const string& resource, int amount):** Adds a specified amount of resources to the colony.

**3. Ant Class (Base Class for All Ants)**

* **Responsibility**: A base class that represents a generic ant and defines its basic attributes (e.g., species, health, strength, etc.).
* **Key Methods**:
  + **performAction():** Defines the action that the ant should perform during each tick (e.g., hunting, foraging).
  + **battle(shared\_ptr<Ant> otherAnt):** Allows an ant to engage in battle with another ant.
  + **rest(int food**): Simulates an ant resting and consuming food to recover health.
  + **isDead():** Checks if the ant is dead (health ≤ 0).

**4. Ant Species Classes (e.g., Warrior, Drone, Queen, etc.)**

* **Responsibility**: These classes inherit from the Ant base class and represent specific species with different attributes (e.g., health, strength, food cost) and behaviors.
* **Key Methods**:
  + **performAction():** Each species has a unique action that it performs during the simulation. For example, Warrior hunts, while Drone forages for food.

**5. Room Class and Subclasses (e.g., WorkerRoom, WarriorRoom)**

* **Responsibility**: Represents a room in the ant farm, which serves a specific purpose (e.g., housing worker ants, warriors).
* **Key Methods**:
* **build():** Defines the process of building a room in the farm.

**6. AntFactory Class**

* **Responsibility**: A factory class that creates specific types of ants based on the given type and species.
* **Key Methods**:
  + **createAnt(const string& type, const string& species):** Creates and returns a specific ant object.

**Functional Requirements**

1. **Ant Farms and Species**:
   * Ants are organized into species, and each species has unique attributes and behavior.
   * The Meadow can contain multiple ant farms, each representing a distinct colony.
2. **Battle Mechanism**:
   * Ants can battle each other, with the winner gaining the attributes of the defeated ant.
3. **Resource Management**:
   * Ant farms can be supplied with resources (e.g., food, warriors), and ants consume food when they rest.
4. **Tick-based Simulation**:
   * The simulation progresses in discrete time steps, with each tick representing an action phase where ants perform tasks like resting, foraging, or hunting.
5. **Colony Merging**:
   * If one colony kills another colony's queen, the winner assumes control of the defeated queen's colony, merging the populations.
6. **Room Construction**:
   * Each room in the ant farm serves a specific purpose and must be built with worker ants. The construction takes multiple ticks based on the number of workers assigned.

**Interface Commands**

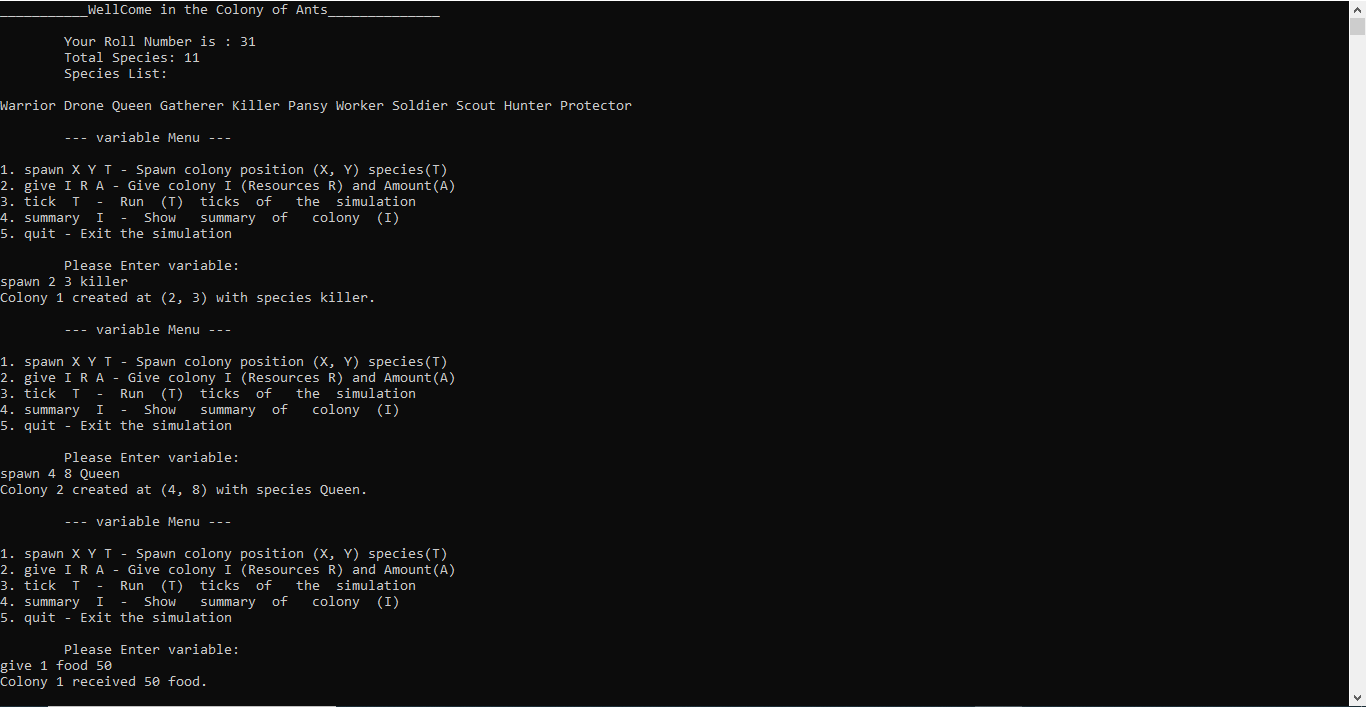
The AntFarm Simulation System provides a command-line interface (CLI) that enables users to interact with and control the simulation. This section describes the available commands, their formats, and provides examples for clarity.

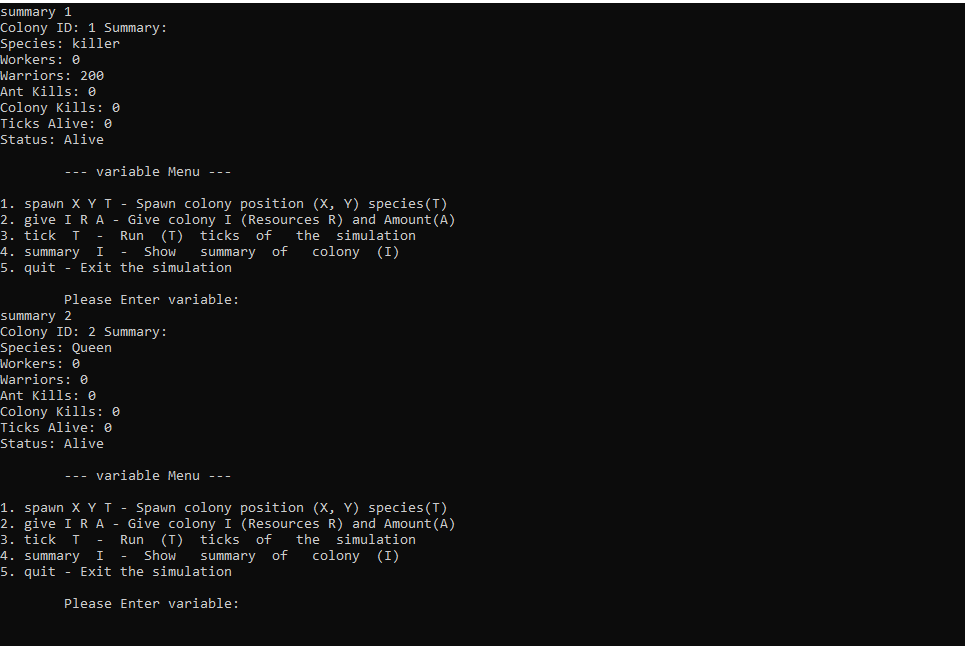
1. **spawn X Y T**
   * **Description**: This command creates a new ant colony at specified coordinates (X, Y). The colony is identified by its species type T, which dictates the behavior and characteristics of the ants within that colony.
   * **Parameters**:
     + **X** (int): The x-coordinate where the colony will be spawned.
     + **Y** (int): The y-coordinate where the colony will be spawned.
     + **T** (string): The type of species for the colony (e.g., Killer, Worker, Pansy).
   * **Example**:
     + spawn 14 -32 Killer
   * **Output**:
     + Colony 1 created at (14, -32) with species Killer.
2. **give I R A**
   * **Description**: This command allows the user to supply a specific colony with various resources. The command specifies the colony ID (I), the resource type (R), and the amount (A) to be provided.
   * **Parameters**:
     + **I** (int): The unique ID of the colony receiving the resources.
     + **R** (string): The type of resource to supply (e.g., food, warriors).
     + **A** (int): The quantity of the specified resource to be supplied.
   * **Example**:
     + give 1 food 50
   * **Output**:
     + Colony 1 received 50 food.
3. **tick [T]**
   * **Description**: Advances the simulation forward by a specified number of ticks (T). If no parameter is provided, the default behavior is to advance by 1 tick. Each tick represents a discrete time unit in the simulation where ants perform their actions.
   * **Parameters**:
     + **T** (int, optional): The number of ticks to advance the simulation. Defaults to 1 if not specified.
   * **Example**:
     + tick 10
   * **Output**:
     + The simulation advanced by 10 ticks.
   * **In the tick process, you may see additional outputs reflecting the actions taken by ants**:
     + Warrior is foraging.
     + Warrior is hunting.
4. **summary I**
   * **Description**: Displays a comprehensive summary of the specified colony, including current resources, number of ants, and overall health status. This command provides the user with vital information about the colony's state and performance.
   * **Parameters**:
     + **I** (int): The unique ID of the colony to summarize.
   * **Example**:
     + summary 1
   * **Output**:
     + Colony ID: 1 Summary:
     + Species: Killer
     + Warriors: 10
     + Resources:
       - Food: 50
       - Warriors Available: 20
     + Status: Alive
     + Ticks Alive: 12
5. **quit**
   * **Description**: Terminates the current simulation session and exits the command-line interface. This command should be used when the user wishes to conclude their interactions with the simulation.
   * **Example**:
     + quit
   * **Output**:
     + Exiting the simulation. Thank you for playing!

**Example Command Sequence**

To demonstrate the use of these commands in a typical workflow, consider the following sequence:

1. spawn 14 -32 Killer // Creates a "Killer" ant colony at (14, -32)
2. give 1 food 50 // Supplies 50 food to colony 1
3. tick 10 // Advances the simulation by 10 ticks
4. summary 1 // Displays a summary of colony 1
5. quit // Exits the simulation

**Sample Output**



**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**