CS 2337 – PROJECT 1 – Army of Ants

Pseudocode Due: 9/12 by 11:59 PM

Core Implementation Due: 9/21 by 11:59 PM

Final Submission Due: 9/28 by 11:59 PM

KEY ITEMS: Key items are marked in red. Failure to include or complete key items will incur additional deductions as noted beside the item.

Submission and Grading:

- All project source code will be submitted in Zybooks.
  - Projects submitted after the due date are subject to the late penalties described in the syllabus.
- Programs must compile using gcc 7.3.0 or higher with the following flags enabled
  - -Wall
  - -Wextra
  - -Wuninitialized
  - -pedantic-errors
  - -Wconversion
- **Type your name and netID in the comments at the top of all files submitted. (-5 points)**

Objective: Implement object-oriented programming in C++ with inheritance and polymorphism.

Problem: People Fun, a mobile game developer located in Richardson, has begun working on a new mobile game called Army of Ants. The game is a simple, casual game where players control a colony of ants and must evade an invading colony of beetles. People Fun has hired you as an intern and assigned you to a group working on the artificial intelligence for the game. To get you involved in the project, the team has asked you to develop the novice level of AI for the program.

Pseudocode: Your pseudocode should describe the following items

- For each function, identify the following
  - Determine the parameters
  - Determine the return type
  - Detail the step-by-step logic that the function will perform
- Functions
  - Ant class
    - Breed
    - Move
  - Beetle class
    - Breed
    - Move
    - Starve
  - Main
    - Main function

        - Any additional functions that you plan to create
- A list of at least 10 test cases you will check during testing (other than the examples below)
    - Specific input is not necessary
    - Describe what you are testing
    - Examples:
        - Forcing an ant to go north when at the northern edge of the grid
        - Checking ant movement when beetles are equidistant east and west of ant
        - Checking beetle movement when ants are east, west and south, but south ant is closest and east and west ants are equidistant but further away than south ant

Zybooks Information:

- You will have multiple source files
    - `main.cpp`
    - `Creature.h` – base class
    - `Ant.h` and `Ant.cpp` – derived class
    - `Beetle.h` and `Beetle.cpp` – derived class
- Core implementation has unlimited submissions
    - This will help you make sure the basic actions of your program are working properly in the environment
- Final submission is limited to 15 submissions
    - White space will be checked

Core Implementation

- Read the input file and store it in the grid
- Reproduce the first 5 turns of the sample given
    - Basic beetle movement to the right and up
        - Moving toward closest ant
    - Basic ant movement to the right and down
        - Moving away from beetle
        - Not moving off grid or stepping on other ants
    - Ant breeding (turn 3)
- There are no ties
- The sample does not cover every possible use case for movement or breeding
- No starving check
- No beetle breeding check

Classes

- Base class
    - Named **`Creature.h`**
        - Abstract class
        - Methods

        - Move (pure virtual) (-5 points if not)
        - Breed (pure virtual) (-5 points if not)
- Derived Classes
    - Ant
        - Name files **`Ant.h`** and **`Ant.cpp`**
        - Methods
            - Move
            - Breed
    - Beetle
        - Name files **`Beetle.h`** and **`Beetle.cpp`**
        - Methods
            - Move
            - Breed
            - Starve
- The move, breed and starve functions will provide information to main to perform the operation
    - These functions do not modify the grid
    - It wouldn't make sense for an object in an array to use the containing array as an argument to an object method
    - I have intentionally left the implementation of these methods up to you. If the methods do not modify the array, what data could they possibly tell you for that object?
- You can add any member variables necessary for each class
- All files must be submitted in ZyLabs
    - Based on your design, your cpp file may be empty

Details:

- The game will be played on a 10 x 10 grid
    - You decide how the grid will be stored in memory
    - This grid will be created in main
    - This grid will not be passed into any of the ant or beetle objects
- A file will be used to populate the grid
    - See sample file
    - a = ant
    - B = beetle
- Grid traversal should be by column first then row
    - Evaluate a column top to bottom before moving the next column
    - Creatures to the left perform actions before creatures to the right
- User will specify number of turns to watch
    - There will not be any user interaction in the game
    - The goal is to test the AI, so the game will play against itself
- All movement is orthogonal (N, S, E, W)
    - Movement is limited to one space per turn

Turn Order:

1. Beetles move
2. Ants move
3. Beetles starve
4. Ants breed (every 3 turns)
5. Beetles breed (every 8 turns)

Each phase of the turn order will be applied to the entire grid before starting the next phase

Ant Details:

- **Move**
  - Each ant will attempt to move in the opposite direction of the nearest orthogonal beetle
  - If no orthogonal beetle, ant stands still
  - If there is a tie for nearest beetle, prioritize movement
    - Move ant in direction of no beetle if possible
    - If beetles in all directions, move toward farthest beetle
    - If there are multiple possible directions to move (either multiple clear pathways or multiple beetles furthest away) use the following movement priority: N, E, S, W
  - Ants cannot move to an occupied space
    - If space moving to is occupied, no movement happens
  - Cannot move off grid
  - If an ant cannot move in the chosen direction, it does not move
    - The ants are not smart enough to seek alternate routes
- **Breed**
  - Every 3 turns, ants breed
  - Add ant in adjacent orthogonal space
    - Start with north space and check clockwise around ant until empty orthogonal space found
    - If no empty spaces, no breeding
  - Ant may not breed again unless it survives another 3 turns

Beetle Details:

- **Move**
  - Move toward nearest orthogonal ant
  - If there is a tie for nearest ant, prioritize movement
    - Move toward ant with most adjacent ant neighbors (orthogonal and diagonal)
    - If still tied, move toward ant with most ant neighbors using the following priority: N, E, S, W
  - If no ant in orthogonal direction, move toward farthest edge from the beetle
    - If there is a tie for farthest edge, use the following priority: N, E, S W
  - A beetle cannot move into a space with another beetle

    - A beetle can move into a space with an ant and eat it
- **Breed**
    - If beetle survives for 8 turns, it breeds
    - Use the same breeding algorithm used for ants
    - Beetle cannot breed again unless it survives another 8 turns

- **Starve**
    - If a beetle does not eat an ant in 5 turns, it dies
    - When a beetle eats, reset its timer
    - Starting with turn 5, beetle starvation needs to be checked every turn

User Interface: The user will be prompted for the following information in the order listed

- Initial grid filename
- Character to represent ant in output
- Character to represent beetle in output
- Number of turns

Input:

- The initial grid will be populated from file input.
- Prompt the user for the input filename
- There will be no need for input validation.
- The last row in the file may or may not have a newline character at the end

Output:

- All output will be sent to the console.
- Print the current state of the grid to the console window for each turn.
    - Display the turn header followed by a newline
        - `TURN<space><turn number>`
    - Display each row of the grid followed by a newline
        - Each ant and beetle is represented by the user-defined character
        - An empty cell in the grid is represented with a space
    - Display a blank line after the last row of the grid