Group 16 : Predator-Prey Game
Members : Tomiko Hansen, Roland Nyamoga, Brandon Schultz, Tik Ho Wong, Adam Wright

Project Design:

The problem in the predator prey simulation is to create two classes which inherit from critter and then react in the correct ways. In our program, the four potential directions of movement for each critter are stored as the integers 1 through 4. In order to randomly choose from these four directions, we seed the random function first in our main, outside of the menu loop in order to prevent the seed from resetting.
This main menu loop then collects input from the user, validates this input and then both initializes and deletes the board before prompting the user to try again. This same logic is applied when the user chooses to specify whether the board size, ant and doodlebug count. The simulations steps themselves are started from this main menu, while the movement and directions of the Critters are detailed in their respective class definitions. The Critter class is an abstract class that the Ant and Doodlebug inherit from.

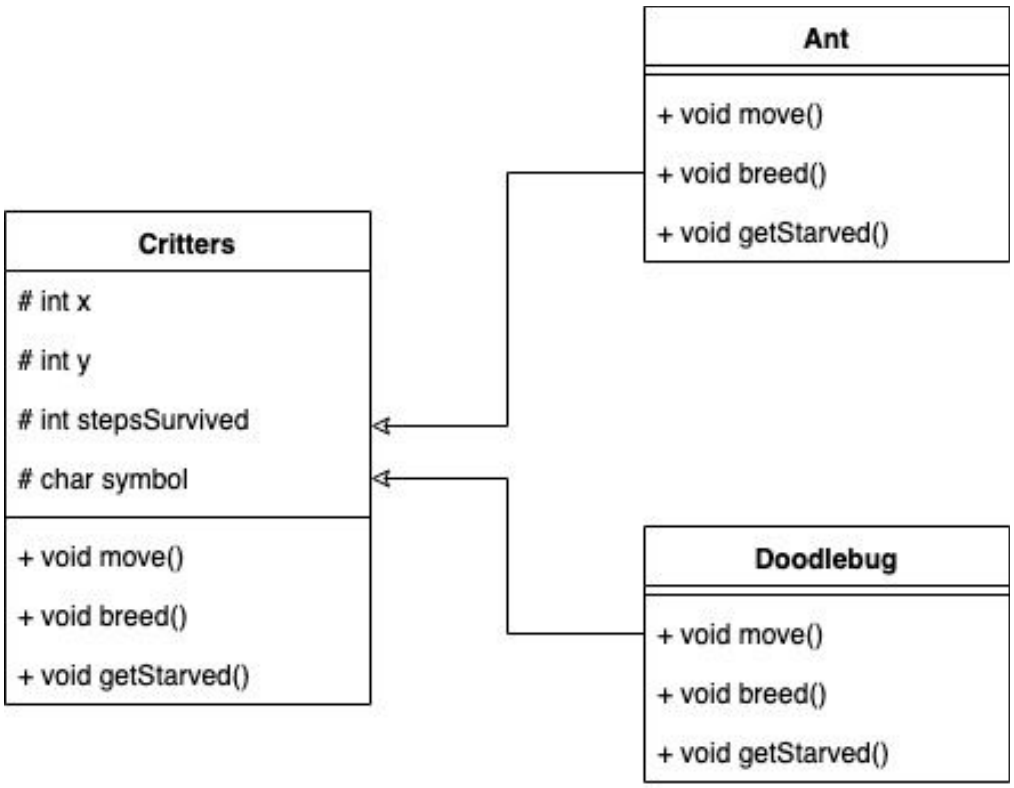
Group member responsibilities:

Adam Wright: Main lead on the project, code and class design and writing
Tomiko Hansen: code review and attempt at extra credit, debugging, testing and reflection
Roland Nyamoga: code review and extra credit logic, debugging, testing and reflection
Brandon Schultz: contributor, testing
Tik Ho: Contributor, testing

Pseudocode:

do while loop wrapping the game to allow restarting and exit
cout a greeting
  entering invalid input will relaunch the menu
  entering 1 continues to the round count prompt
    Ask the user if they would like to specify board size, ant and doodlebug count
    If yes:
      Ask the user for the desired board size (<100)
      Ask the user for the ant and doodlebug counts (< board size/2)
    If no:
     Prompt the user for the number of rounds
      entering invalid input will relaunch the round count prompt
      entering a valid input launches the program
        A board of Critters is made and initialized to null
        A number of Ants and Doodlebugs are dynamically made in the board
        The Critters then move based on how many steps the user inputted

The board is printed for each round
Board is deleted to clear memory
User prompted to restart or quit the program, closing the menu loop
entering 2 will exit the application



Testing Table:

Testing Input	**Expected Output**	**Actual Output**
Call main	Cout greeting	Cout greeting
User enters 1 to enter game	Prints steps prompt	Prints steps prompt
User enters 2 to exit	Back to console	Back to console
User enters non valid input at menu	Rejects input, prints menu again	Rejects input, prints menu again
User enters invalid steps value	Rejects input, prints steps prompt again	Rejects input, prints steps prompt again
User enter 10 steps	Board prints 10 times	Board prints 10 times
User enters 1 to play again	Prints steps prompt	Prints steps prompt

User enters 10 steps again	Board prints 10 times	Empty board prints 10 times
Starve doodlebug	Doodlebug dies after not eating in 3 turns	We saw a decrease in doodbug population accordingly
Ant breed (remove randomized bug population)	Observe surviving ant breed after 3 turns of survivals	Ants bread accordingly after survival
Doodlebug breed (remove randomized bug population)	Observe doodlebug breed after 8 turns of survival	Doodlebugs bred after survival
Specify board size	Board size should be equal to the specified int	Board size was the specified int
Specify ant and doodlebug count	Ant and doodlebug initial count equal to specified int	And and doodlebug count were equal to the int

Reflection:

This project had a number of classes to balance and several functions to struggle to create from the given requirements.

In terms of the design, we didn't go with a board class since we thought we could easily create it as part of the main class. However, this led to major complications when trying for extra credit since the sizes had been hardcoded and we didn't have class flexibility in creating the board. This was a good learning moment for the team. When we attempted to add extra credit requirements, we had conditional jumps, segmentation faults, and memory leaks and were hard to retrace since the main file had the entire game logic, including board creation.

In the end, after extensive troubleshooting, we didn't manage to add the needed code for extra credit.

One notable bug arose when trying to get the game to print out the critters for any subsequent rounds in the program. Based on the test cases, we expected the game to simply relaunch the program once the user was prompted to make a choice. While it did correctly launch the prompt for entering the steps, the boards that were printed to the screen were always blank, with no critters appearing on the board. Since the boards were filled with the '.' character, representing a null value, we initially believed that deleting the boards at the end of the first round may have been causing the error. However, this proved to not be the case, and most notably, modifying the delete sections of the code were reintroducing the memory leaks that they were made to prevent in the first place. This meant that the deleting was not the problem we were looking for.

One of the difficulties of debugging these types of problems is that we can often lose sight of where bug may occur after spending so much time working with the code, similar to losing sight of the forest for its trees. In this sense, working as a group came as a welcome benefit when it came to debugging the code; the change in perspective was key in solving what ultimately was a rather simple bug. During a conference call with the team members, we were finally able to fix this particularly annoying bug: it turns out that the initialized variables from the first round of the program were not included in the whole menu loop. This meant that any loops after the first round would not have those variables set correctly, and thus the board would not be able to print the correct values because the ant and doodlebug counts were already set at their totals. SImply moving these variables into the simulation loop fixed our problem and did not introduce any new ones to our relief.