Assignment 2

Reflections and Test Plan

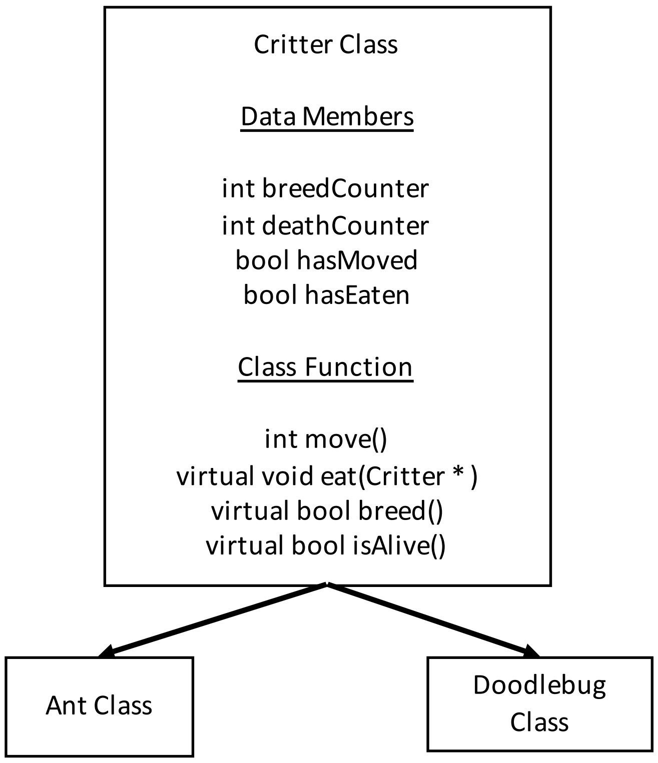
Student: Joaquin Saldana

Program Description/Design

The purpose of this assignment is to practice inheritance. It's mission is for us to create a base class, with two derived classes. The base class being the Critter class, and the derived classes being the Ant and Doodlebug classes.

We will never instantiate the Critter class but use its base class as a pointer to the derived class. The derived classes will be placed in a 20x20 "grid" and will move around the grid and perform certain actions based on rules given to us.

My first instinct was to create a list of the rules so to then identify my member data variables, set/get functions, and more importantly my object class unique functions. Below is a table of the items I identified for the Critter, Ant, and Doodlebug class.



In addition, I drafted a table listing the unique characteristics of each class is it's object. Although they would be sharing the same functions, their definitions were different.

Function	Ant Class	Doodlebug Class

Eat (virtual)	Cannot eat other Critters this includes other Ants and Doodlebugs	Can only eat Ants (if the space their moving into has an Ant). If the space their moving into has another Doodlebug then the bug does not move at all
isAlive (virtual)	Ants die after 10 steps/moves no matter what	If after 3 steps, the Doodlebug has not eaten it has dies. If it has eaten once getting to 3 steps (or before) then the counter is reset.
Move	Neither Critter will know it's position on the "grid". It will only generate a random movement with the direction of going left, right, up and down. This function was not defined as virtual since it will perform the same for all classes, including the children classes.	Neither Critter will know it's position on the "grid". It will only generate a random movement with the direction of going left, right, up and down. This function was not defined as virtual since it will perform the same for all classes, including the children classes.
Breed (virtual)	Upon moving, there will be a breed counter that is incremented. For the Ant, if the counter reaches 3 , the function returns true and resets the counter. This will signal the program/main to create a new Critter Ant in the previous location of the Critter.	The same is also true for the Doodlebug. However the breed counter must reach 8. It's offspring is another Doodlebug.

I also included my own personal utilities class in the Critter class so I had access to my personal functions used to validate user input and more.

The unique class functions of breed, death, and death were identified as virtual functions. The purpose being each class will need to identify them in their own way. However, the move function will operate the same for all derived classes so it was decided not to identify it as a virtual function.

Also, in each class I included a critter name function which returns a string and critter type. Example: for Ant it returns a string that reads as "Ant". This was done to help the main program identify when the object was either an Ant or a Doodlebug.

In addition, the data members for the Critter class were identified as protected. This was done to allow access to them when the required by the derived classes.

Below is the pseudocode for the main program :

1. Create the necessary objects, this includes the 2d array of data type pointer to Critter
2. Introduction to the program and rules
3. Request user input on how many Ants and Doodlebugs and validate input
4. Placement of critter's in random areas of the Grid
5. Print the grid to show the critter's starting positions
6. Call of class functions and logic
    a. While loop that continues until the user enters zero. Inside the loop it holds the program logic and call to the class functions
        i. Index values variables are declared and current index values are stored in these variables
        ii. If statement that checks if the pointer is pointing to NULL or pointing to an object. If pointing to an object and that has yet to move:
            1. Call move function to determine if it's going to move left, right, etc.
            2. Checking to ensure the new indexes are not out of bounds
                a. If statement checking if the next space is pointing to NULL
                    i. Calls the eat function, and checks if the critter needs to breed. If yes, it leaves it's offspring in the old space. If not, it simply moves the Critter to the new position and then c alls the alive function to determin if it needs to die.
                b. If the next space is not pointing to NULL, call the eat function and verify if the current bug is a doodlebug, and if the object in the next space is an Ant, if so then it will be eaten. Standard class functions are called
                    i. If the object cannot be eaten, then it simply calls the eat, breed, and is alive function to verify if the object needs to be destroyed.
            3. If the new column or row is out of bounds, then the object is not moved and the class functions are called to check if the object needs to be destroyed/die.
        iii. Reset all of the Critter's has moved Boolean variable this allows us to generate a move for the critter.
        iv. Print the grid again to show the new moves
        v. Prompt the user to enter 0 if they wish to terminate the program
    b. End of while loop
7. Deallocating the array and freeing up the memory for loop to destroy the 2d dynamic array and end of program

<u>Test Plan</u>

I performed the majority of my tests in modular increments. I started by verifying the class functions and data members were working properly. Next I verified my main program was working correctly. I broke it down into sections by starting with verifying my input validation was working, next I moved to my class function implementation and flow of the program including my 2D array, and finally the printing of the Grid.

Test Case	Input Values	Driver Functions	Expected Outcome	Observed Outcomes
Verifying base class variables classified as "Protected" are accessible to derived class	N/A	N/A	Verifying the program will compile and work on both my local IDE and FLIP server	Program compiled and ran correctly on both environments
Critter pointer will correctly access the function of the derived classes. This is particular important for the virtual functions.	N/A	Output statement calling the critterName function of the two pointers. One pointer creating a dynamic Critter and another creating a dynamic Ant.	For the output to show "Critter" for the pointer creating a dynamic critter object, and "Ant" for the pointer to a dynamic Ant object.	Output did show the correct critter names when called in a cout statement. This proves the virtual functions are working correctly.
Verifying the move() function was creating random numbers and that the death function was working correctly for the Ant class	N/A	While loop which continued until bool variable holding the isAlive functions result is true. Second while which continuously called the move function and output the random number	For the alive function, output 9 zeros and the last number, 10th number, to be a 1. For the move loop, to print 15 random numbers on two runs.	The alive function worked correctly, indicating the virtual function was correct drafted. However the move function was producing the same random numbers. When taking a closer look, I seeded the incorrect random number function. I seeded the srandom()

				function rather than srand() function. This was corrected
Verifying input validation. This includes the number of Doodlebugs and Ants the user wishes to add to the Grid. User will be forced to enter numbers between 1 and 100	-123, 0, 1 (edge value), 99 (edge value), 100 (edge value), 101, 200, 188, 5000, and more	While loop and validation functions from my utilities class.	The program to prompt the user to enter the values if incorrect	Validation code worked and the user is forced to reenter a number when they provide a number outside the range or when they enter a string or characters.
Verify the Grid is properly adding the Critters to the grid in random locations.	Enter several values and edge values.	While loop which requests users number of Ants and Doodlebugs and for loop to print the grid	A standard view of chars A and D representing ants and doodlebugs and blank spaces representing NULL pointers Also verifying the Grid was properly declared as a pointer ot a 2d dynamic array	Ran the function on my local IDE and FLIP server. Found the critter's were correctly placed in random location and the function correctly verified the "cell/space" was not already occupied. However in my tests I found, using Valgrind, my program had memory leaks.

Finding what was creating a memory leak in my program	Entered various values for the number of Ants and Doodlebugs. This included edge values like 1, 100, 99, 101, 0, and various negative numbers.	For loops placing the critters in random locations and for loop printing the grid at initialization. None of the class functions are called yet.	For the program to print the grid, however expect Valgrind to throw memory leak errors.	Memory leaks were found. Suspected the leaks were occurring because the Critter's were not freed from memory prior to program terminating. Created for loop at end of program which freed each Critter individually. Afterwards tests ran showed no leaks. Success.
Added the logic which calls the eat, breed, and is alive function	Entered various values for the Doodlebug and Ants in order to overcrowd the grid. Doing so will let me know if the Critter's are acting as they should	Several nested if stmts and logic, this includes checking if the next space is pointing to NULL or an object, and actions to do once decided. This also includes out of bounds checking	Possibly seg faults for pointing to NULL, memory leaks and bad allocations	The program crashed a few times with seg faults because of attempting to access a null pointer and there were memory leaks. I realized the cause of the problem was I was assigning the grid object (grid[a][b]) by reference to another Critter pointer. I eliminated the

				use of the critter pointers and directly access it by dereferencing the 2D array. Ran more tests and it worked. No memory leaks and no seg faults on either local IDE or FLIP server.
Verified all if statements were properly nested and correct calling the necessary functions	Entered various values for the Doodlebug and Ants in order to overcrowd the grid. Doing so will let me know if the Critter's are acting as they should	Several nested if stmts and logic, this includes checking if the next space is pointing to NULL or an object, and actions to do once decided. This also includes out of bounds checking	Erratic behavior from the critters	Everything checked out. All of the critter's died when required, Doodlebugs ate the Ants when possible, and remained steady when accessing an out of bounds or unable to move.