Group 11
Samantha Schrock
Clarissa Gasiciel
Shawn Hamersley
Brittany Dunn
Esther Park

Reflection: Group Project - Predator-Prey Game

Design Description

The challenge for this project was to create a predator-prey simulation, which was a step up from our first project langton's ant. We had to code two different critters with different behaviors using polymorphism and inheritance. In addition, the critters had to alter their behavior when they ran into each other, not just the perimeter of the borders.

Working as a group remotely to complete an assignment was part of the challenge as well. We used Google hangouts to chat with each other and Canvas file sharing to upload files. A google spreadsheet had a list of bugs and functions that needed working on, and people would assign themselves these tasks so there would be no overlap in work. Samantha would integrate all files and revisions/additions to a master set of files she had.

Initial Design Plan

Critter (base class)
Protected:
stepCounter
Char type
// for location tracking
row
Col
Public:
Critter(int r, int c) {row = r; col = c; stepCounter = 0;}
Critter(int r, int c, int s) {row = r; col = c; stepCounter = s;}
Virtual void move()=0;
Virtual void breed()=0;
Virtual char getType()=0;

Ant(derived class)
Public:
Ant(int r, int c) : Critter (r, c)

{
Type = ‘O’
}
Ant(int r, int c, int s : Critter (r, c, s)
Void move() override
{
	Get random number for direction
	Test for edge cases, if new space empty, move ant
	Movement is done by creating an ant in new element passing the stepCounter to the constructor, then deleting the ant in the old element and set the pointer to nullptr
	Increment stepCounter
}

Void breed() override
{	*Check if its been 3 steps, if so breed*
	Set a breed flag to false
	For each corner, if the rest of the spaces are full, set breed flag to true
	For each edge case, if the rest of the spaces are full, set breed flag to true
	If all the spaces are full, set breed flag to true
	While breed flag is false
		Get a random direction to breed in
		Test if edge, if not, add new Ant to that space, set breed flag to true and reset the original ant’s step counter to 0.
}

Char getType {return ‘O’;}

Doodlebug(derived class)

Public:
Void move() override
Doodlebug(int r, int c) : Critter(r, c)
{
Type = ‘X’
}
Doodlebug(int r, int c, int s : Critter (r, c, s)
Void move()
{
	Test for ant in each direction (watch for edge cases)
	If find an ant, eat it (delete the Ant obj & create the new Doodlebug in that element), reset lastEating to 0
	If no ants, get random number for direction to move, test for edge case, and if new space is empty, move doodlebug

Movement is done by deleting the element in the new location, creating a doodlebug in new element passing the stepCounter to the constructor, then deleting the doodlebug in the old element and set the pointer to nullptr

Increment stepCounter

Void Breed() override
{ *Check if its been 8 steps, if so breed*

Set a breed flag to false

For each corner, if the rest of the spaces are full, set breed flag to true

For each edge case, if the rest of the spaces are full, set breed flag to true

If all the spaces are full, set breed flag to true

While breed flag is false

Get a random direction to breed in

Test if edge, if not, add new Doodlebug to that space, set breed flag to true and reset the original ant's step counter to 0.

}

Void Starve()
{

//check if its been three steps

If (lastEating == 3)

Delete the doodlebug, set pointer to nullptr

Char getType {return 'X';}

Board Class

-has a dynamically allocated array of Critter pointers
-has antCounter
-has doodlebugCounter
Functions:
-constructor: create board, set elements to nullptr
-printBoard: function that prints out the board, if nullptr prints blank space, else calls the getType function and prints the return
-randomAnts: randomly places ants on the board, if nullptr place ant & add to ant counter, else get new random numbers
-randomDoodlebugs: randomly places doodlebugs on the board, if nullptr place doodlebug & add to doodlebug counter, else get new random numbers
-moveDoodlebugs: loop through array and call move function for each Doodlebug
-moveAnts: loop through array and call move function for each Ant
-starveDoodlebugs: loop through array and getType for each pointer, if type Doodlebug, call starve function
-breedCritters: loop through array and call breed function for each critter

Game(main)

```
Do //while user wants to keep playing
{

//Get number of steps
“How many steps?
Cin << steps

//Start game
for( int i = 0; i < steps; ++i)
{
        Call move Critters functions

        Call starveDoodlebugs function

        Call breedCritters function

        Call printBoard function
}

Show menu for if player want to (1) add more steps or (2) exit game

}
while(choice == play again)
```

Test Table (test plans, test results)

Test Case	Input Values	Driver Functions	Expected Outcomes	Observed Outcomes
Input too low	Input < 0	inputValidation() do...while input < 0	Values are entered into validation function (inputValidation.cpp/.cpp) and do-while loops until getInt function receives an integer greater than 0	Loop back to the question prompting user for input
Input not a number	Input != integer (ex. a, t, $, etc)	inputValidation() do...while input != integer	Values are entered into validation function (inputValidation.cpp/.cpp) and do-while loops until getInt function receives a valid integer input	Loop back to the question prompting user for input
Input at 0	Input = 0	inputValidation() do...while input is in valid range	Values are entered into validation function (inputValidation.cpp/.cpp) and do-while loops until getInt function receives a valid integer put	Loop back to the question prompting user for input
Edge cases are accounted for	Observe critter behavior (no specific input req'd for this test case)	move()	Critter stays in current cell if at edge and would move off the board.	Early version of move() function did not test whether or not a critter was at the board edge before checking to see if an adjacent space had an ant to eat (doodlebug). If no edge cases aren't accounted for before checking

				adjacent spaces, memory outside the array would be accessed which causes error. Corrections were made to account for edge cases.
Ensure dynamically allocated memory is freed	No additional input is req'd	board::~board ()	Destructor frees dynamically allocated memory	Memory is freed
(Extra Credit) Input Critters > board spaces	Ant: 40 Doodlebug: 40 Board: 20x20	main()	Values are entered into	Program would get stuck in an infinite loop of trying to populate the ants & doodlebugs on spaces that were not available because they're already populated. Corrections were made to make sure number of critters were less than available spaces on board.
(Extra Credit) Make sure board is always visible on screen. Input for board dimensions is small enough to be on screen.	Input > what screen is capable of showing.	inputValidation()	Put an upper limit on board size so that board will never be too large for a screen so user can always see the simulation properly	Upper limit on board size allows simulation to be viewed properly.

Reflection

Brittany kicked things off with a design plan that we based the structure of our project on. Samantha realized that this project would not utilize just a 2D array of pointers, but we needed a pointer to a pointer to a pointer to a double (pointer to a double pointer), because the board would be an array of Critter pointers. This changed the way we had to approach many of the functions but was not a big issue as we just added a bit more to the initial design plan, and Samantha suggested using Critter objects:

"Having Critter objects won't effect how the compiler calls the functions because they're still virtual. Without any objects of the Critter class, we will have to have elements in the board array that are just pointers set to nullptr which means that we have to test for nullptr in our loops before we can call a function with that element because the element isn't pointing to anything. If you had Critter objects you could fill the empty spaces of the board with Critter objects and then you could loop through the array without having to test for nullptr for each object before calling a function.

Either way, we should be able to make the program work. My first instinct was just to use Critter objects as blank space holders is all."

Samantha coded Ant and Critter classes first, which then Brittany used those as a framework for the doodlebug class. Shawn contribute the board class files after that and also created the shared spreadsheet and group members listed bugs that needed to be fixed or functions that still needed adding. Clarissa contributed the helper functions/files that included validation functions for example. At this point the board with critters was able to be printed, but was not looping or being simulated. Esther attempted to get that loop going in the board class, but had difficulty understanding the "triple pointer" so Samantha jumped in and completed this while providing an in-depth explanation. Esther was able to contribute the destructor function at this point, and the group had a running simulation.

Extra credit features and changes were added in by Shawn. Some modification needed to be made such as making sure more critters weren't being added than available spaces on the board. In addition, to always ensure the board would be able to be small enough for any screen, an upper limit was added so that the board was never bigger than the screen, allowing the user to view the simulation properly. Esther kept track of the testing attempts and issues that came up with certain cases and documented them on the reflection. At the end she created a final reflection, implementing the first design plan and initial approach the group took.

Work Distribution

- Samantha Schrock* (updated and revised master set of files)
    - Critter.cpp/.hpp
    - Ant.cpp/.hpp
    - Extra Credit
- Brittany Dunn
    - Group Design Plan
    - Doodlebug.cpp/.hpp

- Shawn Hamersley
    - Group spreadsheet
    - board.cpp/.hpp
    - Extra Credit
- Clarissa Gasiciel
    - helper.cpp/.hpp (inputValidation.cpp/.hpp)
- Esther Park
    - Reflection