

GROUP PROJECT : Reflection, Design and Test Table

Group 39

Jeff Porter

Luis Renteria

Maryum Shabazz

James Walter

David Vega

5-2019

CS 162_400_Spring 2019

INTRODUCTION

This reflection was compiled from slack chats, minutes typed up from voice chats during hangouts sections, code snippets from chat. The reflection follows a day by day format. With each day showing how our logic and ideas changed. Our group was formed shortly after the group project opened. We finished up project 2 and lab 4 before beginning.

April 30th-May 1st(am)

We started out with setting a meeting time to discuss the project in detail that would work with everyone's schedules using a WhenIsGood app. Thursday was set for the meeting time. In between the official meeting time we discussed project design and code in the chats that were copied over from Slack in the discussion area of our group project. We immediately decided to go for extra credit and would base our further ideas and designs keeping the implementation of the extra credit items in mind.

May 1st (evening)

We discussed movement of the ants and doodlebugs and It was suggested to have a random move for ants and doodlebugs, unless the doodlebug has found an ant to eat. It was further suggested to make move a virtual function in critter.

We discussed how best to divide code. It was immediately clear after discussing various functions that each person has a different idea on implementation. The point of having one cohesive game plan was clearly apparent then. Several of us uploaded our parts of the entirety of our ant programs to github to compare code and determine what would be the best options for generic code like menus and input validation.

It was pointed out that layout of function calls was crucial to knowing what functions to code and we would need to decide that soon.

A group member proposed initializing the board by having a random function to place X's and O's on the board and then a loop for an array of each insect that would read either X or O [depending on bug type] and replace the X or O with the critter.

It was pointed out this was not feasible as instructions stated we would need to have the

board filled with critter pointers to begin with.

We finalized the meeting time for the official meeting of all group members based on the WhenIsGoodApp and pointed out discussion was needed on canvas for grading purposes, Due to Canvas's lack of an interface that made discussion easy , the group decided to screenshot and copy comments from slack periodically to the discussion in Canvas. A group agreed to take responsibility for copying and pasting discussions.

A scrap design document was started, and ideas were added by group members, as they were thought of. This was not an official plan yet, but rather a way to serve as a group notepad and promote brainstorming.

There was thinking that we would need to look through the board for objects. Which it was stated would be hard to do. It was suggested, using a `getCharacter();` function to return the character (X or O of each object) and to look for that.

The discussion then turned to how to move ants or doodlebugs. There was the suggestion of creating critter objects and replacing those in ants or doodlebugs when a move was made then generating a new critter object in the space the doodlebug or ant has vacated.

A group member suggested using a voice chat for the meeting on 5/2/19, a online whiteboard (AWW app) was also suggested to draw/type out ideas during the meeting.

One group member agreed to keep minutes for the meeting since it would be a voice chat.

May 2 2019 (in slack prior to official voice chat)

Code was added to github for `main.cpp` and `game.cpp/hpp`, testing out some ideas. A group member stated that they agreed the 20x20 array should be all critters objects, some generic critters and others ants and doodle bugs but also agreed swapping and moving the various entities would be something different. The group member also stated that when the game class is constructed, loops are done in main through the game function calls.

A group member stated that moving the ants and doodlebugs was transferring a pointer but the issues was what to with the pointer it was moving from.

Another group member wondered could we copy the pointer and assign a new blank

critter object to the old one. It was then stated that an issue would be with us “bugs” unless we could delete the dropped bug somehow.

A snippet to illustrate the idea of delete/reassign then allocate new critter.

(fig 1)

```
delete board[1][2]; //deletes the critter (or ant if doodlebug eats)

board [1][2] = board[1][1]; //moves the ant/doodlebug

//makes a new critter which will be displayed as ' ' since it will have a token
//dataMember
board[1][1] = new Critter;
```

There was discussion on how to delete the board without leaks and various code snippets were proposed, but many had leaks. A group member solved the issue with

```
Game::~Game()
{
    for (int i = 0; i<rows; i++)
    {
        for (int j = 0; j<columns; j++)
        {
            delete grid[i][j];
        }
    }

    for (int i=0; i<rows; i++)
    {
        delete [] grid[i];
    }

    delete [] grid;
}
```

(fig 2)

Previous code snippets were only deleting the objects but not the dynamically allocated board. The code proposed by the above does both and when tested was without leaks.

A group member added bounds checking to the menu to prevent the user from requesting more ants or doodlebugs than spaces.

It was suggested to have main.cpp call an antMove() function (fig 3) which would go through all the ants and call the move() function for each of them, the same could be done for doodlebug; Ants or doodlebugs would be found using the getCharacter function to search for X or O. A snippet of sample

code to illustrate this idea was posted by a group member.

```
void Game::moveAnts()
{
    for (int i=0; i<rows; i++)
    {
        for (int j=0; j<columns; j++)
        {
            if(grid[i][j]->getCritterChar() = 'O')
            {
                grid[i][j]->move();
            }
        }
    }
}
```

(fig 3)

There was discussion on how to move the doodle bug if two ants were nearby. The suggestion was to check the same order of directions each time and eat the first one it sees.

May 2nd (Hangouts voice chat, minutes being kept)

Further discussion of design to ensure all group members are on the same page. To do list was made and tasks were divided. Specifics of each function were discussed in detail particularly what variables were needed. Whiteboard app was used during meeting to facilitate explanation.

Remaining Work was divided as follows:

Maryum : Reflection, Starve

James: Compilation, testing and submits in zip file

Luis: Breed(Base critter function)

Jeff: Move, Eat

David: Aging/Breed

May 3rd

We spent this time testing and writing code and discussing basic elements of the project which functions would need flags for true or false and then call other functions based on the flag such as the breed function. We discussed how the ant or doodlebugs would check for occupied areas and made sure everyone was in agreement to use the critters character of 'X' or 'O' to decide if occupied or not.

May 4th

While writing code we also found we would have to remember to pass max rows and columns to the move function and breed functions for bounds checking. Also we found that according to the requirements we would have some redundant code since specs were forcing us to use a virtual move function in critter but then redefine the function in ant and doodlebug.

May 4th

There was discussion on using Nullptrs to check for critters instead of using space characters and the get token function instead of critter objects since instantiating multiple critter objects creates more of a burden on memory. With larger projects memory would be a huge issue but for now in the interest of having a project that works and allowing time to complete our other class requirements we decided to continue with using critter objects. At this point changing how we were coding would have necessitated re-coding multiple functions.

It was determined we needed to add a references to the grid, otherwise inside the doodlebug and ant classes we could not access the reference.

May 5th

More testing was done on the program now that all the main components were completed. It was found that the breeding and starving functions had issues. The ant didn't start breeding until they hit the edge of the grid when tested on a 50x50 grid, and the doodlebugs were not starving either.

It was found when the ants or doodlebugs move up (north) or left (south) they would be skipped. Because movement was happening first which causes the pointer to redirect the ant moves first.

To fix this issue the ants and doodlebugs needed separate loops in the move function. It was also decided we kept the functions in the critter classes we would've needed 5 parameters, one of them being a reference to a triple pointer. So they were moved them to game and the '.' changed to a ->. In game class they need no parameters.

May 6th

More testing was done and it was discovered doodlebugs were jumping more than one space and ants were moving more spaces then they should have (see fig 4 and fig 5)

[illegible]

(fig 4)

(fig 5)

```

|X      x x|
|      |
|X      x  |
|      |
|      x   |
|      x   |
|X      |
|      |
|xx     |
|      |
-----
Simulating step 2 of 3
-----
|X      x x|
|X      |
|      |
|      x   |
|      xx  |
|X      |
|      x   |
| x     |
|      |
-----
Simulating step 3 of 3
-----
|X      x  |
|      x   |
|      x   |
|X      |
|      xx  |
|X      |
|      x   |
|      |
|X      |

```

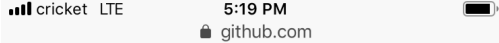
The issue was found that if a ant or Doodlebug moved down a row then It was reprocessessed again through The loop when the loop reached the new row. So a flag was determined

to

be needed to mark when either an ant or doodlebug had moved and once the loop was finished it would reset.

We also noticed that we would need to change the order of functions we had for the game based on the assignment specs. The original order was (see fig 6)

(fig 6) The new proposed order was Age->MoveD->MoveA->Breed->Starve->Print->Reset.



```
or (int n = 0; n < steps;
    std::cout << "Simulating step "
               << n << std::endl;
    g1.starveCritter();
    g1.breedAllCritters();
    g1.incrementAllAges();
    g1.moveDoodleBug();
    g1.moveAnt();
    g1.printGrid();
    g1.resetMoveFlag();
```

We also noticed while testing that no ants or doodlebugs could be placed in the last row or column initially. We discovered it was an issue with the random number statement where instead of having for example `rand() % 6+1` the code was `rand()%6-1`.

May 7th

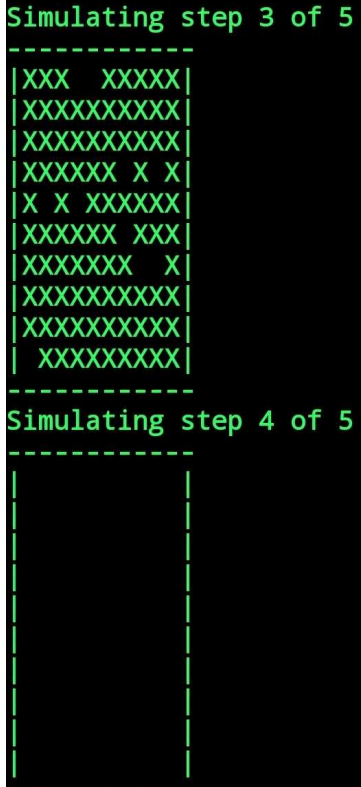
More testing was done. And this latest round of testing did not find any bugs. The latest rounds of data from testing were added to the test table.

May 8th

```
std::cout << "Simulation complete"
          << std::endl;
```

While testing an issue was found with the starve function where all the doodlebugs were dying on the

same day. This issue was fixed by moving the days since last fed variable to the move function. Originally the counter was being reset if they fed during move but then it was increasing during starve. So doodlebugs that hadn't ate still showed as eaten or because the starve function was increasing the counter to one because it reset to zero earlier in the loop. (see fig 7) prior to change. The change described



```
Simulating step 3 of 5
-----
|XXX  XXXXX|
|XXXXXXXXXX|
|XXXXXXXXXX|
|XXXXXXX X X|
|X X XXXXXX|
|XXXXXXX XXX|
|XXXXXXX X |
|XXXXXXXXXX|
|XXXXXXXXXX|
| XXXXXXXXX|
|XXXXXXXXXX|
-----
Simulating step 4 of 5
-----
|          |
|          |
|          |
|          |
|          |
|          |
|          |
|          |
|          |
|          |
|          |
-----
```


above fixed the starve error (see fig 8). We spent time (fig 8) commenting the program and cleaning. There was discussion on making move a pure virtual function which would have led to rewriting of the program since we would no longer be able to instantiate critter objects. A group member raised the concern that it made more sense for move to be pure virtual since base class critters weren't supposed to move, only the specific types of ants or doodlebugs. The way the program was set up with instantiating critter objects it wasn't possible to make a pure virtual function in the critter class.

A decision was made to move forward with making Critter an abstract class and not using critter objects but instead using nullptrs. The previous program design was working but instead of instantiating so many critter objects a more efficient allocation of memory could be made by not using critter objects. An attempt to switch from instantiating critter objects to an abstract base class was made.

The coding for the switch to nullptrs went well mostly. The logic was already planned out so there were minor changes needed to make the full switch. There were a few minor bugs that were worked out, namely small logic errors and variables that needed to be moved as critter class was made abstract.

May 9th

Testing showed by changing from instantiating the critter class to having an (fig 9) abstract class of critter , memory went from 5 million allocs to 1.1 million for a 50x50 board with 300 ants and 50 doodlebugs. Move, breed and starve were made pure virtual. The

```
Initial grid layout
-----
|XXXXXXXXXX|
|XOXXXXXXXX|
|XXXXXXXXXXX|
|XXXOXXXXXX|
|XOXXXXXXXX|
|XOXXXXXXX|
|XXXXXXXXXXX|
|XXXXXXXXXXX|
|XXXXXXXXXXO|
|XXXXXXXXXX|
|-----|
Simulating step 1 of 5
-----
|XXXXXXXX X|
|XXXX XXXX|
|XXXX XXXX|
|X XXXXXXX|
|XXX XX XXX|
|XXXXXXXXXX|
|XXXXXXXXXX|
|XXXXXXX X|
|XXXX XXXX|
|XXXXXX XXX|
|-----|
Simulating step 2 of 5
-----
|XXXX XXXX|
|XXXXXXXX X|
|XXXXXX XXX|
|XXXXXXXXXX|
|XXX XXX XX|
|XX XXXXXX|
|XXXXXXXXXX|
|XXXX XX XX|
|XXXXXXXXXX|
|XXXXXX XXX|
|-----|
Simulating step 3 of 5
-----
|  X  X  |
|X  X   |
|  X  X  |
|           |
|           |
|           |
|           |
|           |
|           |
|           |
|-----|
```

```
-----
| SIMULATION COMPLETE |
|-----|
|1 - Play more steps |
|2 - Quit             |
|-----|
Enter selection: 2
==12601==
==12601== HEAP SUMMARY:
==12601==    in use at exit: 0 bytes in 0 blocks
==12601== total heap usage: 18,447 allocs, 18,447 frees, 520,333 bytes allocated
==12601==
==12601== All heap blocks were freed -- no leaks are possible
==12601==
==12601== For counts of detected and suppressed errors, rerun with: -v
==12601== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

final program showed all heap blocks were freed, no memory leaks present and no errors on Valgrind the conversion was successful. see(fig 9).

May 10th

Final compilation and testing was completed. Documentation was reviewed by group.

Summary

Overall the project was a success but the biggest learning experience wasn't the coding but the group aspect of the project. Not all members of the group were familiar with Github and this made everyone working together to code a little difficult since some group members were trying to learn the new platform and keep up with the coding.

Also our schedules varied greatly making Slack a great idea for discussing rather than directly using canvas. Slack gave us the tools to easily and quickly share snippets of code, ideas and even pictures. It also gave us a better sense of when people were available by using the status indicator next to their name. The mobile aspect of slack allowed on the go and up to date review of discussions even if someone was not able to directly participate. Slack usage greatly enhanced team cohesiveness.

We also did well with everyone being willing and enthusiastic about whatever part of the project they were contributing. Sometimes people's ideas didn't turn out like they planned initially or someone else took over but overall there was respect for everyone.

More planning prior to starting would have been useful. Even though we did some planning once we started coding it became apparent that everyone's ideas of what would work were somewhat different. More ideas weren't necessarily a bad thing but when coding by yourself you have one vision to work from, with 5 people there was more varied ideas which can be a blessing or a curse.

Group 39 , achieved the end goal of completing the project and learned a few keystones to the importance of teamwork in the computer science field along the way.

Design

1. CRITTER CLASS

- a. Parent class that will be an abstract class for both Ant/Doodlebug to inherit. This will include the shared data members, getter/setter functions, and the three purely virtual member functions.
 - i. Integer to track the time in steps since the Critter last fed. This will be used by Doodlebug.
 - ii. Character to be used to distinguish between Ants and Doodlebugs when printing the board.
 - iii. Integer to track the age in steps of the Critters. This is what will allow breeding at appropriate times.
 - iv. Boolean to flag when a Critter has moved within a simulation step. This will stop Critters from moving multiple times when we loop through the grid.
 - v. Member function: incrementAge() used to increment the age of each Critter by 1 at the beginning of a simulation step.
 - vi. Member function: resetMovedStepFlag() will be used at the end of each simulation step to allow the Critters to move again within the next simulation step.
 - vii. Three purely virtual member functions to be defined within Ant/Doodlebug: move(), breed(), and starve().

2. ANT CLASS

- a. Child class of Critter. Inherits all the necessary data members and uses the constructor to set its character to 'O'. Defines the purely virtual functions from Critter.
 - i. move() takes in a pointer to the grid, the row and column value that the ant is currently at, and the max rows and columns of the grid to avoid segmentation faults and moving the Ant off the board. Will create a random number for direction and try to move one spot in that direction on the grid. This will include boundary checking, looking for other critters, and swapping the flag to has moved.
 - ii. breed() is the only other action that an Ant takes. Takes in the same parameters of move() and then tries to create a new Ant in an adjacent spot to the Ant. Includes bounds checking and looking for

other Critters in the way. Only takes place if $\text{age} \% 3 = 0$.

3. DOODLEBUG CLASS

- a. Child class of Critter. Inherits all the necessary data members and uses the constructor to set its character to 'X'. Defines the purely virtual functions from Critter.
 - i. `move()` takes in a pointer to the grid, the row and column value that the ant is currently at, and the max rows and columns of the grid to avoid segmentation faults and moving the Doodlebug off the board. This is also where the Doodlebug handles eating though. First it will check all 4 directions in a clockwise pattern and will eat the first Ant it sees. Moves to the Ants location, deletes the Ant, and resets the `timeLastFed` to 0. If no Ants are available to eat, increments `timeLastFed` and moves the same as the Ant class.
 - ii. `breed()` acts the same as Ant but only happens if the age is a multiple of 8. Only breeds if $\text{age} \% 8 = 0$.
 - iii. `starve()` is defined within Doodlebug as it needs to eat at least once every 3 steps. This is called on each Doodlebug and if `timeSinceFed` is ≥ 3 , then the Doodlebug is deleted.

4. GAME CLASS

- a. Game class contains the logic of the whole simulation step handling. It has 3 data members, the 2D dynamic array (grid) of Critter pointers, integer for rows of the grid, and integer for columns of the grid. Has a parametrized constructor, four member functions, and a destructor.
 - i. Constructor takes 4 integers from the user for extra credit. The number of rows, the number of columns, the number of Ants, and the number of Doodlebugs. The number and Ants + number of Doodlebugs cannot exceed the size of the grid specified.
 - ii. `printGrid()` will loop through the 2D array printing the character of the Critter within the space or a ' ' character if the space is equal to `nullptr`.
 - iii. `incrementAllAges()` loops through the grid and calls the `incrementAge()` function on the objects within the grid.
 - iv. `resetMoveFlag()` is called at the end of each simulation step and loops through the grid to reset the `movedStepFlag` on each object so it can move again next step.
 - v. `runStep()` is the main logic handling member function of the Game.

This is function that handles the logical order of each simulation step. First loops through the grid calling move() on all Doodlebugs, then calls move() on all Ants to conclude the first step. It will then call starve() on all Doodlebugs to prevent the ones that haven't eaten from breeding. The last step on the day will be to loop through the grid calling breed() on all eligible Critter objects.

- vi. A default constructor is needed to make sure the board is cleared of all memory allocation. This prevents any memory leak within our program.

Test Table

Test Scope	Description	Expected Results	Actual Results
Input Validation	Check all bounds within the menus and inputs	Non-integers and very large numbers are rejected	As expected, user can't crash program from incorrect menu input.
Can Board be filled to max capacity	Set board at 10x10, attempt to fill with 100 doodle bugs or 100 ants	The board should fill with either 100 doodle bugs or 100 ants	Board fills to max capacity of 100 spaces filled with either ants or doodle bugs and program runs.
Ants breed correctly	Watch ants for several steps and see if ants breed if able every 3rd step	At step 1 if one ant, at step 3 if room there should be 2, at step 6 there should be 4 and step 9 there should be 8 ants	Step 1: one ant Step 3: two ants Step 6: four ants Step 9: eight ants
Ant breeds (cardinal position)	Does the ant breed in every cardinal position	At step 1, there are 10 ants, at step 3, there will be new ants born at each cardinal position (north, south, east and west)	At step 3: <ul style="list-style-type: none"> - 3 ants born west - 2 ants born east - 2 ants born south - 3 ants born north
DoodleBugs breed correctly	See if doodlebugs breed every 8th step	On 10x10 board 30 ants and 2 doodlebugs placed. By step 8 there should be 4, by step 16 there should be eight if able	Step1: two doodlebugs Step 8: four doodle bugs Step 16: six doodle bugs - Two were not able to breed

			correctly.																					
DoodleBugs (cardinal position)	Does the doodle bug breed in every cardinal position	At step 1, there are 10 doodlebugs and 90 ants.	At step 8: <ul style="list-style-type: none">- 3 doodlebugs born south- 4 doodlebugs born north- 2 doodlebugs born west- 1 doodlebug born east																					
Check move function	Ensure that each ant or doodlebug only moves one step at a time	Follow ants and doodlebugs for several steps and ensure they move only one step	Ants and doodle bugs move only one step a piece.																					
Check move function	Ensure that each ant and doodle bug moves each direction	Observe one ant and one doodle bug move for 50 steps (with other functions disabled)	Both ant and doodlebug moved each direction at least once after observing it move 50 steps.																					
Check Move for ant independent of breeding and starving doodlebugs	Disable breeding and starving for doodle bugs and breeding for ants , see how ant moves	Ant and doodlebugs move according to assignment specs	<table><tr><td>step</td><td>db</td><td>ant</td></tr><tr><td>0</td><td>5, 5</td><td>1,8</td></tr><tr><td>1</td><td>5, 6</td><td>1,9</td></tr><tr><td>2</td><td>5,5</td><td>0,9</td></tr><tr><td>3</td><td>6,5</td><td>0,8</td></tr><tr><td>4</td><td>6,6</td><td>0,8</td></tr><tr><td>5</td><td>6,5</td><td>1,8</td></tr></table> <p>Ant found to stay on same space from step 3-4 but this was because it tried to move out of bounds - which caused it to stay still per specifications.</p>	step	db	ant	0	5, 5	1,8	1	5, 6	1,9	2	5,5	0,9	3	6,5	0,8	4	6,6	0,8	5	6,5	1,8
step	db	ant																						
0	5, 5	1,8																						
1	5, 6	1,9																						
2	5,5	0,9																						
3	6,5	0,8																						
4	6,6	0,8																						
5	6,5	1,8																						

Test Board at edges of parameters.	<p>Set max values of:</p> <ol style="list-style-type: none"> 1. 100 rows, 100 columns, 100000 steps, 500 ants, 500 doodlebugs. 2. 100 rows, 100 columns, 10000 doodlebugs, 100000 steps, 3. 100 rows 100 columns and 10000 ants, 100000 steps 4. Set min values of 1 row 1 column 1 ant or 1 doodlebug, 1 step 	Program runs for full number of steps for entirety of program in all 4 instances.	Program runs without error to completion
Check if program allows out of bounds parameter entry	<ol style="list-style-type: none"> 1. Attempt to enter more critters then max amount of spaces on board. 2. Attempt to enter less than 0 critters 3. Attempt to enter more than 100000 steps and less than 0 steps. 4. Attempt to enter columns and rows less than 1 and more than 100 	All out of bounds parameters should be rejected and user should be prompted to enter an in bounds parameter which is accepted by program.	All out of bounds parameter are rejected and user should be prompted to enter an in bounds parameter which is accepted by program.
Check Starve	10x10 board 2 doodle bugs, 2 ants watch if doodlebugs starve correctly (18 steps)	Doodle bugs will starve at end of 3rd step if they have not eaten within 3 steps	First doodlebug starves and disappears by beginning of step 4 (has not eaten) Second doodlebug eats day 3 and lives until starving by step 6, and is no longer present on step 7.

Check board filled with Ants - No breeding/changes.	3x3 board filled with Ants.	Ants should stay on board for any number of steps.	Board did not change for any number of steps ran.
Each Ant /Doodlebug function isolated to see if it works.	Manually created boards with Ants and Doodlebugs. Checked runs with only Move, runs with only Starve, and runs with only Breed.	Each run of the simulation works as expected. Critters move within the board in all directions. Ants breed every 3 steps, Doodlebugs every 8 steps. Doodlebugs starve on step 3 with no Ants.	All results as expected.
Check for Memory Leaks	Run Valgrind --leak-check=full	No memory Leaks should be present	No memory leaks present