CS 162 Group 10 Project Reflection

Brian Yi | Catrina Joos | Chen Yan | Clinton Hawkes | Victoria Dmyterko
Oregon State University, Corvallis, OR

Overall Gameplay Design

Create a Gameboard object in main
Create a Menu object in main
Menu prompts user for input (Extra Credit)
Update Gameboard object with user input
Call Gameboard playGame function
While(numSteps <= maxSteps)
    Tell doodlebugs to move
    Tell ants to move
    Tell doodlebugs to breed
    Tell ants to breed
    Tell doodlebugs to starve
    Print board
Ask user to play again or exit program
If user plays again
    Add extra steps to maxSteps counter
    Repeat game loop
Else
    Call Gameboard cleanup function to delete objects
    Exit program

Gameboard Design

Gameboard::initialize
Used for Gameboard constructors
Validate that inputted row and column is >= 20
Set currentSteps and maxSteps counters to 0
Creates the Critter pointer array using validated row and column size

Gameboard::printBoard
Use nested for loop to print the board
If isEmpty() returns true
    Print white space
If hasAnt() returns true
    Print ant (O)
If hasBug() returns true

Print doodlebug (X)

Gameboard::playGame

Using a for loop

Print step counter
Tell doodlebugs to move
Tell ants to move
Tell doodlebugs to breed
Tell ants to breed
Tell doodlebugs to starve
Print board

Gameboard::actOnCritters

If critter type is ant

Go through ant vector with for loop
If flag is ‘m’
Call moveCritter
If flag is ‘b’
Call breedCritter

If critter type is bug

Go through bug vector with for loop
If flag is ‘m’
Call moveCritter
If flag is ‘b’
Call breedCritter
If flag is ‘s’
Call starveCritter

Gameboard::starveCritter

Call Critter object’s starve function

Gameboard::moveCritter

If critter != nullptr

Call Critter object’s move function
If isValid is true for Point returned by Critter
Get Critter’s new row and column
Update board with new location
Get Critter’s old row and column
Set old location to nullptr

Gameboard::breedCritter

Create new Critter pointer equal to the return Point of the Critter object’s breed function

If new pointer != nullptr

Call addCritter

Gameboard::cleanup
Using nested for loop
    If column location != nullptr
        Delete column
    Delete rows
Delete board
Clear ant vector
Clear bug vector

Gameboard::addCritter
Get current location of Critter
If isEmpty returns true
    Get row
    Get column
    Update board at that location
    If type is ant
        Add pointer to ant vector
    If type is bug
        Add pointer to bug vector
    Return true
Else
    Delete ptr

Gameboard::removeCritter
Get row
Get column
If isEmpty returns false
    Call findAndRemoveFromVector on that location
    Delete board pointer at that location
    Set to nullptr
    Return true
Else
    Return false

Gameboard::findAndRemoveFromVector
If critter type is bug
    Set bool bugFound to false
    Set i = 0
    While bugFound == false and i < bug vector size
        Get location of bug at i
        If bug at i is equal to bug to be removed
            Erase bug from vector

Set bugFound to true
i++
If critter type is ant
Set bool antFound to false
Set i = 0
While antFound == false and i < ant vector size
Get location of ant at i
If ant at i is equal to ant to be removed
Erase ant from vector
Set antFound to true
I++

Gameboard::isValid
Returns true if the point is within the board's limits
Else returns false

Gameboard::isEmpty
Returns true if isValid returns true and if that location on the board is equal to nullptr
Else returns false

Gameboard::hasAnt
If isEmpty returns true
Return false
If isValid returns false
Return false
If object at point has type ant
Return true
Else
Return false

Gameboard::hasBug
If isEmpty returns true
Return false
If isValid returns false
Return false
If object at point has type bug
Return true
Else
Return false

Ant/Doodlebug Move Design

Create Point object with the same coordinates as the critter

Increase age of critter
Generate random number between 0 and 3
Switch based on random number
    If 0
        If isValid and isEmpty return true
            Move critter location up one cell
    If 1
        If isValid and isEmpty return true
            Move critter location to the right one cell
    If 2
        If isValid and isEmpty return true
            Move critter location down one cell
    If 3
        If isValid and isEmpty return true
            Move critter location to the left one cell
If randomly chosen cell is not valid/empty
    Return -1,-1 and critter doesn't move

Ant/Doodlebug Breed Design

Create Point object of (-1,-1)
Create bools (state, north, east, south, west) all set to false
If age < 3
    Return nullptr (critter did not breed)
Set point to critter's current location
Check cell above critter
If isValid and isEmpty returns true
    Set north to true
    Set state to true
Check cell east of critter
If isValid and isEmpty returns true
    Set east to true
    Set state to true
Check cell below critter
If isValid and isEmpty returns true
    Set south to true
    Set state to true
Check cell west of critter
If isValid and isEmpty returns true
    Set west to true
    Set state to true
If state equals false
    Set Point to (-1, -1)

```
        Return nullptr
While state equals true
        Generate a random number between 0 and 3
        Switch based on random number
                If 0
                        If north equals true
                                Move Point up by one cell
                                Set state to false
                If 1
                        If east equals true
                                Move Point right by one cell
                                Set state to false
                If 2
                        If south equals true
                                Move Point down by one cell
                                Set state to false
                If 3
                        If west equals true
                                Move Point left by one cell
                                Set state to false
Reset critter's age to 0
Create new critter and pointer in new Point location
Return critter pointer
```

Doodlebug Starve Design

```
Create Point object of (-1,-1)
Set point to doodlebug's location
If starveAge < 3
        Return point
Else
        Call board object's removeCritter function
        Return point
```

Test Table

Test Case	**Input Values**	**Expected Outcome**	**Observed Outcome**
User inputs an invalid row size	4, 0, 101, a, 12b	Re-prompts the user for a new input until it is valid	Re-prompts the user for a new input until it is valid

User inputs a valid row size	50, 5	Creates a board with the specified number of rows	Creates a board with the specified number of rows
User inputs an invalid column size	4, 0, 101, a, 12b	Re-prompts the user for a new input until it is valid	Re-prompts the user for a new input until it is valid
User inputs a valid column size	50, 5	Creates a board with the specified number of columns	Creates a board with the specified number of columns
User inputs an invalid number of steps	0, -5, 501, b	Re-prompts the user for a new input until it is valid	Re-prompts the user for a new input until it is valid
User inputs a valid number of steps	100	Runs the simulation for the specified number of steps	Runs the simulation for the specified number of steps
User inputs an invalid number of ants	-2, c, 12c, (# greater than board size)	Re-prompts the user for a new input until it is valid	Re-prompts the user for a new input until it is valid
User inputs a valid number of ants	5	Creates the specified number of ants on the gameboard.	Creates the specified number of ants on the gameboard.
User inputs an invalid number of doodlebugs	-5, c, 12c, (# greater than board size)	Re-prompts the user for a new input until it is valid	Re-prompts the user for a new input until it is valid
User inputs a valid number of doodlebugs	10	Creates the specified number of doodlebugs on the gameboard.	Creates the specified number of doodlebugs on the gameboard.
At the end, user chooses to continue the simulation and adds more steps	1 - to continue then 50 to add extra steps	*Continues* the simulation from where it left off for the additional number of steps	*Continues* the simulation from where it left off for the additional number of steps
At the end, user chooses to exit the program	2	Simulation ends	Simulation ends

Reflection

Originally our Critter class only contained member variables for the Critter's current location. After integrating the Gameboard class, Catrina realized that we needed to have member variables that also kept track of the old location of the critters. Another change we made to the Critter class was to have the member function move() return a Point object, breed() to return a Critter pointer, and the starve() function to return a bool when originally the functions were returning void. Another function we added was getType() which uses enums declared in the BugType.hpp header file to keep track of whether the Critters are Ants or Doodlebugs. We also made changes to our original design of Gameboard. During our design discussions we decided it would be helpful for the Gameboard to return values to the Critters so that they know the state of the board. We added helper functions such as isEmpty() and hasAnt() to the design so that we could let the Critters know whether or not the cells they wanted to move or breed into were valid spots on the board.

One of our biggest issues was figuring out how to get the Critters and the Gameboard to interact with one another without creating an inclusion loop. Initially we had each header file include the other class, with each file also including members that were of the other class type. This ended up creating errors at compile time. Luckily, Brian caught onto the issue and sent a helpful page from stackoverflow which explained the problem of having header files including one another. With some editing and a forward declaration we were able to solve the problem and get our Critter and Gameboard to compile. Other minor issues include making sure that all our of our return types matched for the overridden functions, making sure curly braces and semicolons are in the correct spots, and getting our different classes to all fit together. In order to solve these common coding issues we held regular meetings throughout the week, tested the code frequently, and helped one another debug our classes.

Work Distribution

Brian Yi

- Menu
- Main
- Input validation

Catrina Joos

- Gameboard
- makefile
- Testing

Chen Yan

- Doodlebug

Clinton Hawkes

- Ant
- Testing

Victoria Dmyterko

- Critter
- Reflection