DESIGN AND REFLECTION

Cassidy Bullock. CS 162 Online Section

DESIGN

Classes:

Superclass: critters

Functions:

- Just get and set functions to interact with protected members

Data: Protected

- Bool movement: keeps track of whether or not an object has moved in current time step
- Int eat_ctr: keeps track of how many timesteps it has been since a critter has eaten
- Int breed_ctr: keeps track of how many timesteps it has been since critter has bred
- String type: keeps track of whether object is an ant or a doodlebug

Subclass: ants

Functions:

- Void Move: if object has not moved yet in the time step, generate a random direction, should check if the new space is outside the grid or occupied by some other critter, if the space is clear should copy over all the data in the object to the new spot and iterate the eat and the breed counter and remove the data from the previous space, it should also change movement to true
- Bool Eat: should check where the eat counter is at and if it has gotten to 10. If it has it should return true to signify that it should die
- Void breed: should check the breed counter and if it has reached 3 create a new ant in the previous space as well as reset the breed counter.
- Void die: if the eat function returned true, this function should remove all the ant's data from the board

Data: Inherited from critters class

List

Functions:

- Void Move: if object has not moved yet in the time step, generate a random direction, should check if the new space is outside the grid or occupied by some other doodlebug, if the space is clear should copy over all the data in the object to the new spot and iterate the eat and the breed counter and remove the data from the previous space, if the space had an ant it should reset the eat counter, it should also change movement to true
- Bool Eat: should check where the eat counter is at and if it has gotten to 3. If it has it should return true to signify that it should die
- Void breed: should check the breed counter and if it has reached 8 create a new doodlebug in the previous space as well as reset the breed counter.
- Void die: if the eat function returned true, this function should remove all the doodlebug's data from the board

Data: Inherited from critters class

Main:

- Create 2D array for the environment of type critters
- Have a function to randomly generate all the ants and doodlebugs
- Have a function to reset the movement Boolean so that all of the objects have not moved at the beginning of each timestep
- Have a function to print the board—ants will be shown by a '*' and doodlebugs will be shown by a '#' but blank spots are just blank
- Loop through the grid and if the is an ant call the move, eat, breed, and die functions for ants, and do the same for doodlebugs if there is a doodlebug
- Loop through a certain number of time steps

REFLECTIONS

Testing:

*Function: move(int&, int&, critters**)*

Test Case	**Input Values**	**Expected Outcome**	**Actual Outcome**	**Comments**
critter tries to move off board	location on grid (0,2)	critter will not move	critter did not move	works as expected
critter tries to move into another critter	ant at (4, 13) tries to move onto another ant on (4, 14)	will not move	did not move	works as expected

critter moves to empty space	current location (7, 3) to new location (7,2)	should move over data to (7,2) and iterate counters	critter successfully moved to new spot and counters were iterated	works as expected
doodlebug moves on top of ant	doodle bug at (15,6) onto ant at (14,6)	doodlebug replaces ant and eat counter gets reset	doodlebug replaces ant and eat counter gets reset	works as expected

*Function: eat(int, int, critters**)*

Test Case	Input Values	Expected Outcome	Actual Outcome	Comments
eat counter below maximum	eat counter at 3	should return false	returned false	works as expected
eat counter at maximum	10 (this was for an ant)	should return true	returned true	works as expected

*Function: breed(int, int, int, int, critters**)*

Test Case	Input Values	Expected Outcome	Actual Outcome	Comments
breed counter indicates not to breed	breed_ctr at 3 (for a doodlebug)	no breeding happens	nothing happened	works as expected
breed counter indicates it is time to breed	breed_ctr at 3 (for an ant)	new ant object appears in the previous spot	new ant object appears in the previous spot	works as expected

*Function: die(bool, int, int, critters**)*

Test Case	Input Values	Expected Outcome	Actual Outcome	Comments
critter should die	1	critter's data gets removed from the grid	critter's data gets removed from the grid	works as expected
critter should live	0	does nothing	does nothing	works as expected

General Other Thoughts:

Most things work as expected. There was a lot of trouble trying to get the critters to move only once, and I tried several different approaches including several if statements but in the end I ended up creating a Boolean data member to store whether a critter has moved yet or not. To make sure each critter is behaving correctly I included several cout statements that explain what the critter is doing and supposed to be doing. I have commented these out because they clutter up the screen.

I preferred to use a specific number of time steps so I could control how long the program ran. I found that when I initialized the board with 100 ants and 5 doodlebugs the doodlebugs died out and the ants took over the board. I then tried increasing the number of doodlebugs to 10 but leaving the ants the same, but again the doodlebugs died out and the ants took over the board. I suspected that the doodlebugs were mostly just dying before they had time to breed so I decided to increase the ants to 150 and keep doodlebugs at 10 but

the same thing happened again. Throughout each of these trials the number of time steps was 20. It was only when I increased the ants to 200 and the doodlebugs to 10 that doodlebugs survived to timestep 20. Upon extending the number of time steps, doodlebugs eventually did die out though. Through these trials I also did not seed the rand() function to time and just let it generate the same way each time that way I had a better picture of what was going on. For the turned in file I seeded the rand() function, put the number of time steps to 100, the starting number of ants 200, and the starting number of doodlebugs 10.

To Use the Makefile:

Type **make -f make.txt** into the command line, then **./newmain** to run the program