Group Project: Doodlebugs Vs Ants
CS 162 Spring 2018
May 13, 2018
Kimberly Broz, John Casey III, Kevin Ohrlund, Kelly Usenko, Scott Wickersham

Division of Labor

Project Coordinating & Strategy, Design Decisions
All group members

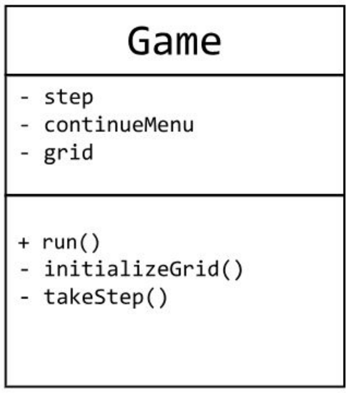
Coding Strategy & Implementation
John Casey III, Scott Wickersham

Code Review, Test Design
Kimberly Broz & Kelly Usenko

Test Case Execution, Debugging & Validation
Kimberly Broz

Design Documentation & Review
Kelly Usenko

Design



Game Class

The Game class builds and initializes required elements to implement the Predator and Prey simulation. Creates and gathers user input for simulation parameters, builds grid, places critters randomly on the grid, then moves critters in step increments according to game logic. After running the specified number of steps the Game class continues or ends the simulation according to user input or game end case. (update with what end case is)

Responsible for

- Creating a menu for gathering game parameters
- Implementing the menu and getting user input

- Initializing grid according to user input
- Placing critters on the grid
- Executing simulation step logic to move, breed, starve and age critters on the grid
- Printing the grid to the screen
- Continuing simulation until simulation is over or user does not want to continue.

Logic

Menu

Allows user to enter an integer for how many steps the simulation should take

Only accepts integers between 1 and the upper limit of the integer type on the given system the program is run on

Grid Initialization

Creates grid according to row and column specified.

Initializes all cells in grid to null

Critter Placement

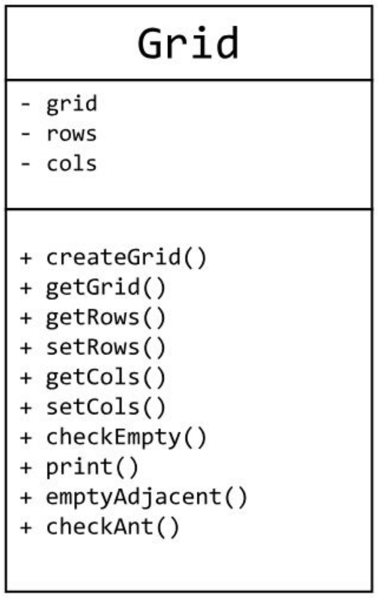
Creates and places user-specified number of Ants in random grid locations first

Creates and places user-specified number of Doodlebugs in random grid locations

Game Take Steps

Implements main simulation logic in the following order:

- Moves Doodlebugs to adjacent empty cell in grid
    - Executed by Doodlebug class
- Moves Ants to adjacent empty cell in grid
    - Executed by Ant class
- Breeds all Critters - Ants and Doodlebugs
- Starves Doodlebugs
    - Checks Doodlebug property of days since it last ate an ant, if greater than or equal to 3 the Game starves the Doodlebug and removes from grid
- Ages remaining critters one day
- Prints grid to screen



Grid Class

The Grid class represents a two dimensional grid or ‘board’ that is occupied by Critters -- Ants or Doodlebugs. The default Grid size is 20x20 or can be set by user input.

Responsible for

- Initializing a two dimensional array of pointers to Critter objects and sets initial values to null
- Maintaining and returning (set and get) number of array rows and columns
- Determining if a given space is occupied by a Critter or is null
- Determining if there are empty adjacent cells relative to a given array location (row and column)
- Printing grid to screen
- Checking if an ant occupies a given array location (row and column)

Logic

Creating Grid

Initializes triple pointer array of Critter objects with null values. Includes check if grid was initialized by grid constructor with row and column parameters.

Check Space: Empty

Takes who parameters representing row and column. Returns true if null or false otherwise.

Check Space: Empty Adjacent

Takes who parameters representing row and column. Cycles through all surrounding spaces and returns true if at least one is null.

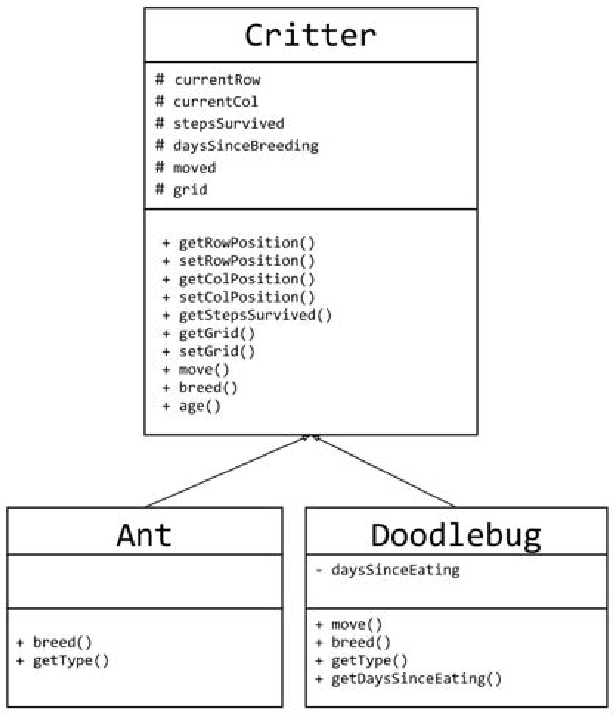
Check Space: Ant

Takes who parameters representing row and column. Utilizes Critter getType function to determine if array location points to Ant object.

Print Grid

Prints grid to screen with the following symbols:

- ‘ ‘ - empty space
- o - Doodlebug occupied space
- x - Ant occupied space



Critter Class

The Critter class is a parent class to Ant and Doodlebug with virtual abstract functions for breeding logic and critter type determination. It contains attributes and implements functions common to Ants and Doodlebugs.

Responsible for:

- Setting position of critter (row and column) (Constructor or manual with setRowPosition() and setColPosition()
- Returning position of critter (row and column)
- Determining base logic for critter movement
- Updating critter location on grid (move)
- Tracking and updating Critter attributes
  - Simulation grid it is on
  - Position (row and column) on grid
  - Age (steps survived)
  - Days since critter spawned/bred

Logic

Movement

The virtual move function determines the next space on the simulation grid to move the critter to via the following checks:

- A direction is populated at random relative to the critter's current position.
- Next space is chosen based on critter's current position + random direction
- Adjacent cell in the chosen direction is not occupied by another critter (empty/null)
- Adjacent cell is within boundaries of the grid

If the adjacent cell based on the random direction is not empty or not within bounds of the grid, the critter

- Does not move
- Continues to increment days since breeding attribute

Ant Class

The Ant class is a child class of Critter. It utilizes Critter's public attributes and implemented methods. The Ant class implements Breeding logic and a function to return the type of critter the object is.

Responsible for

- Tracking Critter "type" (Ant) and days since last spawn (breeding)

Logic

Breeding

If an ant's number of days since it last bred (daysSinceBreeding) is greater than or equal to 3, and there is an open adjacent cell relative to the Ant's current position, the Ant will spawn a new ant in a randomly chosen direction.

If the random direction chosen does not result in open cell, a new random direction will be chosen until the open cell is found.

If there are no open adjacent positions then the Ant's breed function does not evaluate direction and continues to increment days since last spawn.

Doodlebug Class

The Doodlebug class is a child class of Critter. Similar to the Ant class, it utilizes Critter's public attributes and implemented methods except for Movement logic. The Doodlebug class overrides the move function in the Critter parent and implements it's own breed function.

Responsible for:

- Tracking Critter "type" (Doodlebug) days since last spawn (breeding), and days since eating.

Logic

Breeding

The Doodlebug's breeding follows the same logic as the Ant class with the exception of the interval it breeds.

If a Doodlebug's number of days since it last bred (daysSinceBreeding) is greater than or equal to 8, and there is an open adjacent cell relative to the Doodlebug's current position, the Doodlebug will spawn a new Doodlebug in a cell based on a randomly generated direction.

Movement

The Doodlebug Class overrides the parent Critter class' move function and behaves as follows:

- The Doodlebug move logic checks if an Ant occupies *any* adjacent cells to the Doodlebug's current grid position or whether adjacent cells are open within grid boundaries.
    - If at least one Ant is found, the Doodlebug will generate a random direction and determine a new position to move to.
    - The new position is checked to confirm if it has an Ant and is within bounds of the grid.
        - If it does, the Doodlebug will eat the ant, and move into the new position
        - If it does not, the Doodlebug will continue to evaluate new positions via random Direction generation until it finds a grid space containing an Ant to eat.
    - If no Ants are found, the Doodlebug moves to an open adjacent cell that is within bounds of the grid via random direction generation similar to the parent Critter move function.

Test Plan

Test Category: Input Validation

Test Scope	Set Up	Steps	Expected Result	Actual Result
Menu choices	User prompted for number of rows	User enters letters (cy) when prompted for row size	Program continues to prompt user until valid input is received, then proceeds	Invalid input. Please enter an integer (until valid input.)
Menu choices	User prompted for number of rows	User enters number less than 20 (19) when prompted for row size	Program continues to prompt user until valid input is received, then proceeds	Let's user know # is 'too large or small' until valid input.
Menu choices	User prompted for number of rows	User enters number greater than 200 when prompted for row size	Program continues to prompt user until valid input is received, then proceeds	Let's user know # is 'too large or small' until valid input.
Menu choices	User prompted for number of columns	User enters letters when prompted for column size	Program continues to prompt user until valid input is received, then proceeds	Invalid input. Please enter an integer (until valid input.)
Menu choice	User prompted for number of columns	User enters number less than 20 (-1) when prompted for column size	Program continues to prompt user until valid input is received, then proceeds	Let's user know # is 'too large or small' until valid input.
Menu choice	User prompted for number of columns	User enters number greater than 200 (201) when prompted for column	Program continues to prompt user until valid input is received, then proceeds	Let's user know # is 'too large or small' until valid input.
Menu choice	User prompted for number of rows	User enters 30	Should be accepted as valid input.	Accepted as valid input.
Menu choice	User prompted for number of rows	A non-integer, (20.2) is input.	Invalid input error and prompt user for new input.	Game interprets it as invalid input and prompts for new input.
Menu choice	User prompted for number of columns	A non-integer, (40.5) is input.	Invalid input error and prompt user for new input.	Game interprets it as invalid input and prompts for new input.
Menu choice	User prompted for number of columns	User enters 20	Should be accepted as valid input.	Accepted as valid input.

# of Ants	User prompted for number of ants	User enters negative number (-1)	User is asked to enter a number between 1 and 1 less than the number of grid locations.	User is told the number is too small and re-prompted until valid input is received.
# of Ants	User prompted for number of ants	User enters a number larger than 599 in a 20X30 grid	User is asked to enter a number between 1 and 1 less than the number of grid locations (599 in this case).	User is re-promted to enter a number between 1 and 599 (for a 600 spot grid).
# of Ants	User prompted for number of ants	User enters 10 in a 20X30 grid	Input should be accepted.	Input accepted.
# of Doodlebugs	User prompted for number of doodlebugs	User enters 0.	User must enter at least 1 and less than (size of grid - number of ants).	User is re-prompted to enter a number between 1 and 595 (for a 600 spot grid with 5 ants).
# of Doodlebugs	User prompted for number of doodlebugs	User enters 30 in a 20X30 grid.	Input should be accepted.	Input accepted.
# of Doodlebugs	User prompted for number of doodlebugs	User enters 1.5	User will be re-prompted to enter a valid integer.	User is told it is not a valid integer and re-prompted.
# of Doodlebugs	User tries to put a Doodlebug in 1 remaining spot	20X20 grid with 399 ants and 1 doodlebug	Game should allow user and find this one remaining spot for the doodlebug.	Works as expected. Next step doodlebug eats adjacent ant.
Continue Game	User is able to continue running the game after first block of rounds.	Input: 1 (to continue)	User should then be prompted for how many rounds for the continuation.	User prompted for number of rounds as expected.
Continue Game	User is able to exit game instead of continue.	Input: 2 (to exit)	Game should quit.	Game says, ‘goodbye’ and quits.
Continue Game	User is prompted for number of rounds to continue.	Input: letters	Game will accept input, and prompt user to enter/return to continue.	Game continues running simulation for # of input time steps after user presses enter.

Test Category: Game Flow

Test Scope	Set Up	Steps	Expected Result	Actual Result
Initialized Grid	Manually count initialized grid, ants, and Doodlebugs.	20X20 Grid with 10 ants and 20 doodlebugs	User will be shown initial grid with correct number of rows, columns, ants, and doodlebugs.	Works as expected.
Take One Step	Grid prints same size. Doodlebugs and ants have moved logically.	Call move functions for critters (either during game or manually)	Game moves doodlebugs prior to moving ants. Same number of doodlebugs, but possible decrease in ant population. All doodlebugs should have moved but not necessarily all ants.	2 ants eaten. 10 doodlebugs remain, as expected. Can see that eaten ant spots are now occupied by the moved doodlebugs. All doodlebugs have moved.
Take Four Steps	Grid prints 4 times (after the first initialization). Doodlebugs and ants have moved logically each step. Some or all of the doodlebugs now die off. Ant population starts breeding.	Manual check of critters.	Possible decrease in doodlebug population. Increase not possible. Ant population could go in either direction, but will most likely double. Some might have been eaten and some might have spawned. All remaining doodlebugs should have moved, but not necessarily all ants.	Majority of doodlebugs have died off from not having eaten. Ants have doubled. Their breeding shows on the fourth step.

Test Category: Class Logic

Test Scope	**Set Up**	**Steps**	**Expected Result**	**Actual Result**
Take 9 steps	20X20 grid with 100 ants and 5 doodlebugs	9	Board prints 9 times (after initial board). Doodlebugs have bred if they have ‘eaten’ in last 3 steps.	All Doodlebugs have died off. Must create a different test to observe Doodlebug breeding.
Take 3 steps, attempt to kill off Doodlebugs by 3rd step.	20X20 grid. Take 3 steps, but put only 1 ant on board and 10 Doodlebugs, so they are sure to die off.	3	Doodlebugs will have died off completely, or nearly completely.	As expected, no Doodlebugs remaining.
Set up to see doodlebug breeding in 9th step.	20X20 Grid with at least 50 ants (plenty of food), and 50 doodlebugs.	9	Some doodlebugs will breed, and we expect this to show on the 9th step.	Doodlebug spawn shown on the 9th step.
Move - Ants - Open adjacent cells	Grid has at least one ant with open adjacent cells prior to move	Move ants	Ant moves to open cell directly next to prior spot. Ant does not move more than one cell away same number of ants prior and post move on grid	As expected.
Moves - Ants - No Open Adjacent Cells	Grid has at least one ant completely surrounded by either doodlebugs or other ants	Move ants	Ant that is surrounded does not move to new location. Ant is either eaten or stays put.	As expected, no ant movement of surrounded ant.
Breed - Ants - Open Adjacent Cells	Grid with ants whose days since breeding are at least 3 & have adjacent open cells	Next step is taken in game play (or days incremented manually)	Ants with days since last breed >= 3 spawn new ant in open adjacent cell Ants with less than 3 days since last breed do not spawn new ants	Breeding occurs at end of 3rd step. Shows on the 4th step.
Breed - Ants - No Open Adjacent Cells	Grid with ants whose days since breeding are at least 3 & have no adjacent open cells	Next step is taken in game play (or days incremented manually)	Ants with days since last breed >= 3 do *not* spawn new ant if no open adjacent cell. Days since last breed increments 1 day	Breeds on next turn in the event of open adjacent cell. Validated working as expected.

Breed - Doodlebugs - Open Adjacent Cells	Grid with doodlebugs having between 1 and > 8 days since last breed and at least one has open adjacent cell	9th step is taken.	Doodlebugs with open adjacent cells and >= 8 days since last breed spawn new Doodlebug into adjacent cell	Validated working as expected.
Valgrind Leak Check	Valgrind should be checked for possible leaks		Everything freed, no leaks possible.	All heap blocks were freed.

Reflection

<u>Project Challenges Encountered</u>

Circular dependencies - Thinking through how the class relationships between Grid and Critter as each “having” one of the other as data members was initially challenging because of the circular logic. This was overcome by implementing a forward class declaration between Grid and Critter so we were able to utilize logic with each referring to the other in our simulation.

Keeping track of which critters moved/efficiently moving all critters - The group debated a few options on how to handle updating the positions of critters as they moved during the simulation. A few ideas were evaluated including maintaining vectors of the different critters and parsing through those only (so the game wouldn’t have to check 20x20 = 400 array locations and could have a precise list instead), and simply looping through all grid spaces checking whether a space was null or occupied. Ultimately we decided on adding a data member and member function to Critter that allowed us to set a flag indicating whether a critter had been moved for the given step operation or not.

<u>Design Decisions</u>

What to do when no open adjacent cells are available for move/breed events -
Since there are simulation events that are conditional based on Critter data members’ incremented values, we had to think through how to handle the cases of when to increment these counters if the preceding operation failed. Specifically the cases we discussed were what to do when a critter tries to breed into an adjacent cell but cannot either because all surrounding cells are occupied by other critters or the critter is at the grid boundaries. The breed event is triggered by a Critter’s “days since breeding” counter. We decided to continue to increment the critter’s days since breeding count in the event it attempts to breed but cannot, and decided the critter would continue to breed at every next step opportunities (as opposed to only attempting in increments of the minimum ‘days since breeding’ value).

How to determine which Critter child class occupied a given space on the grid -
Since Doodlebug move logic is dependent on knowing whether an adjacent space is occupied by an Ant we explored different ways to determine the critter type including having stand-alone collections of each critter in the simulation (in vectors) or creating indicators within the critter class. Ultimately the group decided on creating a Critter class enum that could be called via a function that returns the type of Ant or Doodlebug.

Where in program to eliminate starved Doodlebugs -
Performing this function in the Game class made the most sense because the events that would determine Doodlebug attributes were driven by Game logic.

Reflection on Group Work

Group work challenges

Working in a group on a more complex project presented challenges outside of programming and design decisions. Communication is key to coordinating who will do what work, how and when the work will be accomplished. This is always more difficult when your team is completely virtual and conversations don't happen in real-time. Our group overcame this challenge by creating a slack channel and multiple discussions in our Canvas group to stay organized. Slack let us have more free-flow discussions that felt more natural. It was also helpful that slack let us search, pin, and share content extremely easily.

Time Zones and opposing work hours! Our group has two members in Asia, one member who works overnights and two in the Midwest. This made it even more challenging to have real-time discussions about coordinating work. Our group handled this really well by building off of each other's work done the previous day/night and collaborating in the few odd hours most of us were online together.

Group work advantages

The advantages of working in a group on this project outweighed the challenges by far. One advantage was having a wider pool of ideas to choose from when deliberating design options. Everyone in the group was open and respectful of each other's ideas. When determining how to handle keeping track of what critters had been moved, we had a few ideas around using vectors, looping through the entire grid, or adding a new mechanism or function to track. After trying out one option and running into complications (vectors) we ended up going with another group member's idea that was easy to implement and solved our problem quickly.

We experienced broader advantages from working in this group beyond the immediate project at hand. Throughout the past three weeks we had conversations comparing coding techniques, useful external learning resources and tips we've learned so far in the course that make assignments easier (menus, helper functions etc.). Our group also helped each other learn new tools like git and github that will be useful far beyond this course and made coordinating coding for the project very smooth.