Group 39
Group Project
5/12/2018

Group Members:
Amena Choudhury
Monique Glasgow
Shifra Schectman
Rebecca Vickers

**Program Design**
Problem: Use cellular automata to create a 2D predator–prey simulation in your program. The preys are ants and the predators are doodlebugs.

**From the Project Requirements:**
***Rules***

Ants and doodlebugs live in a 20 * 20 grid of cells. Only one critter may occupy a cell at a time. The grid is enclosed and no critter may move off the grid. Time is simulated in steps. Each critter performs some actions every time step:

Ant's actions:

- Move – every time step, ant randomly moves up, down, left or right. If neighboring cell is occupied or if move would cause ant to fall off grid, then ant stays in current cell.
- Breed - If an ant survives for three time steps (not been eaten by doodlebugs), at the end of the time step, the ant will breed a **n**ew ant in an adjacent (up, down, left, or right) cell that is empty randomly. If the cell is not empty, another cell will randomly be chosen. If there is no empty cell available, no breeding occurs. Ants breed after every three time steps.

Doodlebugs actions:

- Move – doodlebug will move to a cell containing an ant and eat the ant. If no ants in adjacent cells, the doodlebug moves same as ant.
- Breed – if doodlebug survives for 8 time steps, the doodlebug will breed in same manner as ant.
- Starve – if doodlebug has not eaten an ant within 3 time steps, the doodlebug starves and dies (doodlebug is removed from board).

**Program Structure**

There is a class named Critter that contains data and functions common to ants and doodlebug. The program also contains an enumerator, menu function and input validation functions.

**Program Structure: Classes**

**Board class:**

| Board |
| --- |
| Critter **critterBoard<br>int numberOfRows<br>int numberOfColumns<br>int numberOfAnts<br>int numberOfDoodlebugs |
| Board(int, int)<br>~Board()<br>int getNumberOfAnts<br>int getNumberOfDoodlebugs<br>int getNumberOfRows<br>int getNumberOfColumns<br>Critter* getSpace(int, int)<br>void setNumberOfRows(int)<br>void setNumberOfColumns(int)<br>void setSpace (int, int, Critter*)<br>void setNumberOfAnts<br>void setNumberOfDoodlebugs<br>void displayBoard<br>void moveCritters<br>void breedCritters<br>void fillBoard(int, int)<br>void resetTimeStepDone |

Board::Board(int numberOfRows, int numberOfColumns)

- Initializes board of size numberOfRows by numberOfColumns
- Each space on board is pointer to a Critter

Board::~Board()

- Deletes all pointers to critters

int Board::getNumberOfAnts

- Returns integer value of the number of ants on board

int Board::getNumberOfDoodlebugs

- Returns integer value of the number of doodlebugs on board

int Board::getNumberOfRows

- Returns integer value of the number of rows on board

int Board::getNumberOfColumns

- Returns integer valur of the number of columns on board

void Board::setNumberOfRows(int numberOfRows)

- Sets the numberOfRows data member

void Board::setNumberOfColumns

- Sets the numberOfColumns data member

void Board::fillBoard(int numberOfAnts, int numberOfDoodlebugs)

- Fills board with specified number of ants and doodlebugs

void Board::displayBoard

- Loops through all pointers and prints the character type for any critter on the board
- Displays in grid format

Critter* Board::getSpace(int rowNumber, int columnNumber)

- Returns the pointer to Critter at the Critter's position on the board

void Board::setSpace(int rowNumber, int columnNumber, Critter* newCritter)

- Assigns Critter* position on board specified by parameters

void Board::breedCritters

- Loops through all pointers to critters in board and calls breed function
- Update number of ants and doodlebugs on board

void Board::moveCritters

- Loops through all pointers to critters in board and calls move function.
- Update number of ants and doodlebugs on board

void Board::resetTimeStepDone

- Resets the timeStepDone data member

void Board::setNumberOfAnts

- Updates numberOfAnts data member with the number of ants currently on the board

void Board::setNumberOfDoodlebugs

- Updates numberOfDoodlebugs data member with the number of doodlebugs on the board

**Critter class:**

| Critter |
| --- |
| int currentRow<br>int currentCol<br>bool timeStepDone<br>char critterType<br>int breedStepCount<br>int eatenStepCount |
| <setter/getter function><br><constructor(s)><br>~Critter<br>virtual void breed<br>virtual bool move<br>int getBreedStepCount<br>vector canMoveorBreed<br>virtual bool canBreed<br>int canEat<br>virtual void copyCritter |

Critter::Critter(int currentRow, int currentColumn)

- Constructor initializes data members of object

Critter::~Critter

- Virtual destructor

char Critter::getCritterType

- Returns critterType

Bool Critter::getTimeStepDone

- Returns timeStepDone

int Critter::getBreedStepCount

- Returns breedStepCount

int Critter::getEatenStepCount

- Returns eatenStepCount

void Critter::setTimeStepDone

- Sets the value for timeStepDone

void Critter::setBreedStepCount

- Sets the value for breedStepCount

void Critter::setEatenStepCount

- Sets the value for eatenStepCount

vector<int> Critter::getAdjacentEmptySpaces(Board &board)

- Returns integer vector containing all directions where adjacent cells are empty

int Critter::canEat(Board &board)

- Returns integer value representing direction of 1st adjacent cell containing an ant

**Ant class:**

| Ant |
| --- |
| |
| <constructor(s)><br>~Ant<br>virtual void breed<br>virtual bool canBreed<br>bool move<br>virtual void copyCritter |

Ant::Ant(int currentRow, int currentCol) : Critter(currentRow, currentCol)

- Constructor initializing data members

Ant::~Ant

- Virtual destructor

void Ant::breed(Board &board)

- Randomly select adjacent empty space and create new ant in that space

- Reset breed count
- If no empty space available or ant hasn't survived to breed yet, function does nothing

bool Ant::move(Board &board)

- Returns boolean value indicating whether the ant is able to move or not.

bool Ant::canBreed

- Returns boolean value indicating whether the ant can breed or not.

void Ant::copyCritter

- Creates copy of ant calling in the specified space on the board

**Doodlebug class**

| Doodlebug |
| --- |
| |
| <setter/getter function><br><constructor(s)><br>~Doodlebug<br>bool move<br>virtual void breed<br>virtual bool canBreed<br>virtual void copyCritter |

Doodlebug::Doodlebug(int currentRow, int currentCol) : Critter(currentRow, currentCol)

- Constructor initializing data members

Doodlebug::~Doodlebug

- Virtual destructor

bool Doodlebug:move(Board &board)

- Returns Boolean value indicating whether doodlebug was able to move or not
    - There is an ant in adjacent cell; ant is eaten and doodlebug is created in its place
    - No ant in adjacent cell, but adjacent empty space available; copy of doodlebug is created

    - No adjacent ant and no adjacent empty space; doodlebug does not move

bool Doodlebug::canBreed

- Returns Boolean value indicating whether the doodlebug can breed or not

void Doodlebug::breed(Board &board)

- Randomly select adjacent empty space and create new doodlebug in that space
- Reset breed count
- If no empty space available or doodlebug hasn't survived to breed yet, function does nothing

void Doodlebug::copyCritter

- Creates copy of doodlebug

**Other Functions:**

Enumerator
enum Direction (UP=1, RIGHT, DOWN, LEFT)

- Used to represent the direction the critter will move

Menu function:

int Menu::promptForInteger(string prompt)

- Accepts string argument and returns integer
- Checks user's input to make sure it is an integer

int Menu::getIntegerNoPrompt

- Takes next input from user and ensure it is an integer

int Menu:promptForIntegerWithRange(string prompt, int minValue, int maxValue)

- Accepts string argument and two integer arguments for min and max values
- Checks user's input to make sure it is an integer within range

int Ment::promptForIntegerWithMin(string prompt, int minValue)

- Checks user input is an integer less than or equal to minimum value

int Menu::promptForIntegerWithMax(string prompt, int maxValue)

- Checks user input is an integer equal to or greater than the maximum value

void Menu::setPromptForChoicesMenu (string newPrompt)

- Sets prompt for multiple choice menu

void Menu::addChoice(string newChoice)

- Adds string to vector containing all multiple choice options

void Menu::alterChoice(int choiceNumber, string changedOption)
- Integer represents location of multiple choice item to be altered
- String replaces text

void Menu::printChoicesMenu
- Prints multiple choice menu prompt and all choices

int Menu::getUserSelection
- Prompts user to select a choice and validates user's choice

Input Validation functions:

int InputValidation::validateInt(string input)
- Validates if input string can be converted to an int

int InputValidation::validateRange(int input, int minValue, int maxValue)
- Verifies if input is within given range

int InputValidation::validateRangeMinOnly(int input, int minValue)
- Validates if input is greater than or equal to minimum value

int InputValidation::validateRangeMaxOnly(int input, int maxValue)
- Validates if input is greater than or equal to maximum value

**Test Table**

| Test Case | Input | Expected Outcome | Actual Outcome |
|---|---|---|---|
| Validate # of time steps | -5, p, & | Prompt user to enter valid input | Prompt user to enter valid input |
| Validate # of rows | -2, r, # | Prompt user to enter valid input | Prompt user to enter valid input |
| Validate # of rows | -3, q, $ | Prompt user to enter valid input | Prompt user to enter valid input |
| Validate # of ants | -4, f, ( | Prompt user to enter valid input | Prompt user to enter valid input |
| Validate # of doodlebugs | -6, e, @ | Prompt user to enter valid input | Prompt user to enter valid input |

| | | | |
|---|---|---|---|
| Ant on board displayed by 'O' character | N/A | O | O |
| Doodlebug on board displayed by 'X' character | N/A | X | X |
| Game runs for correct number of steps | 4 | 4 time steps | 4 time steps |
| Ant move | N/A | Ant moves to adjacent empty cell | Ant moved to adjacent empty cell |
| Ant breeding | 1 ant on 3x3 board for 4 time steps | Ant breeds in adjacent cell | Ant bred in adjacent cell |
| Edge case check for Ant | N/A | Ant should stay in current cell | Ant stayed in current cell |
| Doodlebug move | N/A | Doodlebug to eat ant in adjacent cell | Doodlebug ate ant in adjacent cell |
| Doodlebug breeding | N/A | Doodlebug to breed after $8^{th}$ time step and ate ant within every 3 steps | Doodlebug bred after $8^{th}$ step and ate ant within 3 steps |
| Doodlebug starve | 1 doodlebug on 3x3 board for 4 time steps | Doodlebug gets deleted from board | Doodlebug deleted from board |
| Edge case check for Doodlebug | N/A | Doodlebug should stay in current cell | Doodlebug stayed in current cell |
| Validate menu | -7, 0, h, - | Prompt user to enter valid input | Prompt user to enter valid input |
| Validate # of more time steps | -8, v, ^ | Prompt user to enter valid input | Prompt user to enter valid input |
| Keep Playing | 1 | Prompts user to enter how many more time steps | Prompts user to enter how many more time steps |
| Exit game | 2 | Exits game | Exits game |

**Work Distribution**

- Board class, Debugging - Monique
- Critter class/ Move function/Ant class/Doodlebug class - Shifra
- Breeding function, Main, Menu, Input Validation, Makefile - Rebecca
- Reflection document – Amena

**Problems and How we solved them**

1. Character representations for ants and doodlebugs was reversed.
2. Doodlebugs were not moving on the board. A bracket in the function was not in the right place.
3. The critters on the board were moving one by one instead of the doodlebugs moving first.
4. Critters were moving more than once. For example, if a critter moved to the right, it would move again while looping through move function. Edits were made in Critter class to have each critter move once for each loop.
5. The doodlebug was not eating the adjacent ant on a 3x3 board on the first time step. Instead, it would eat the ant on the second time step. The canEat function and the getAdjacentEmptyCells were testing for > 0 instead of >= 0.
6. Negative integers were being accepted as input. Input validation functions were updated to accommodate for negative integers

**Reflection**

Out group started the project early on which gave us enough time to complete the project by the due date. We utilized Google to have video conferences and share files. The project was broken down into major functions – board, move, breed and starve. Pseudocodes were written for the functions and then translated to actual code. It was noticed that each person had their own coding style such as braces placement; one person would place the brace on its own line whereas another person did not. These differences in code writing increased our ability to read and understand another person's code.

Before the actual flow of the program was established, one video conference was spent on making sure the functions and classes written out thus far compiled. Since all members of the group used different IDE's, it was a good to see how each IDE compiled the code. Some of the noticeable differences are:

- Xcode was not recognizing the #pragma directive
- Some IDE's were giving warnings while others were not.
- One IDE would compile the code, while another one was not able to even though it was the same exact code being compiled
- Visual differences of IDE layout

We shared our screens to show the other members of the group how the IDE was compiling and the testing output results.

Once the functions and classes written out compiled, the code for main, menu and input validations were added. After all the code was written out and it compiled, the program was tested (see test table).

The program was not tested only at the end. As the code was being written, each person did their own testing to make sure what they wrote was meeting the requirements of the project. We even spent one meeting, testing the board and how the critters appeared on the board and if the critters were moving on the board. Incremental testing helped us to find errors early on.

Overall, the group worked well together to complete the project.