Group Project – Predator-Prey Game
CS162 – Intro to Computer Science – Section 400
Winter, 2018
Group 8 - Sheila Babayan, Becky Chao, Elizabeth Donato, Devin Gendron, Ronald Walters

Introduction

For the purposes of this assignment, we have created a 2D predator-prey simulation using cellular automata. The simulation uses polymorphism to allow objects to move and behave differently on the grid. A base Critter class establishes characteristics common to both predators and prey, including age, position, and representation on the grid. Additionally, the base class contains virtual member functions that can be modified to produce different behavior by the predators and the prey. From the base Critter class, a Doodlebug class is derived to represent the predators on the grid and an Ant class is derived to represent the prey. Each of the derived classes establishes characteristics unique to the predators and prey as specified by the assignment requirements. The grid is created using a Board class that constructs and maintains a two-dimensional board and allows for placement of Critters during the simulation. The simulation is managed through a Game class that contains information about objects and standard utility functions.

Project Plan

Our group approached this project by separating our work into three phases: project design and planning, class coding and debugging, and program testing and reflection. During the initial project design and planning phase, we set up collaborative workspaces through Slack, Github, and Google Docs. We then began drafting pseudocode to ensure that our ideas were clearly described and that all assignment requirements had been met. From this pseudocode, we drafted diagrams that would be used when creating our classes to ensure that all variables and functions were compatible and interacted correctly. We also drafted a testing plan during this phase of the project to ensure that our design would be both correct and complete. Our initial design pseudocode, class diagrams, and testing plan are included at the end of this document. During the coding and debugging phase of the project, we created the required classes and utility functions along with a simple client program that would be used to test simulation components. As components were completed and added to the project, all group members contributed to the debugging process by providing input about design decisions, discussing problems and potential solutions, and updating files as needed. Finally, during the testing and reflection phase we executed the testing plan that we had created during the initial phase of the project. As issues were encountered, we discussed how these issues should be resolved and made any necessary changes. Once we had completed testing, we each provided our thoughts about the project and these thoughts were compiled into the reflection below.

The workload for this project was initially distributed as follows:

Sheila Babayan – Initial Testing Plan
Becky Chao – File Compatibility and Debugging
Elizabeth Donato – Draft Design Document and Final Reflection
Devin Gendron – Class and Utility Functions (input validation, user-input, menu)
Ronald Walters – Critter, Doodlebug, and Ant Class
All Group Members – Design, Testing, Reflection Input

Initial Design Pseudocode

Set up the gameboard

- *Print message stating we did the extra credit*
- Ask the user if they want a standard (20x20, set # critters) gameboard or if they want to
    - If static:
        - Dynamically generate a 2D array of chars, size 20x20 (“board”?)
        - Each array element will be a pointer to a Critter
        - World initialized with 5 doodle bugs & 100 ants (rand Loc)
            - Doodle Bug = “X”
            - Ant = “O”
            - User input val for number of steps
    - If user-generated:
        - Prompt Menu for user input:
        - User input val for x & y columns
        - User input val for number of ants/doodlebugs/etc* (controlled by board size)
        - User input val for number of steps
- Ask the user to enter the number of time steps to run
    - If the input is >0, save this in an int (“numSteps”?)

Place the Critters

- For (1 to 5 doodlebugs)
    - Get random x/y values;
        - Check if values are taken
        - If taken, get new values
    - Set bugs at x/y value;
- For (1 to 100 ants)
    - Get random x/y values;
        - Check if values are taken
        - If taken, get new values
    - Set ants at x/y value;

Run the game

- While there are still moves left in the game, do this:
    - Move the doodlebugs
    - Breed the doodlebugs
    - Starve the doodlebugs
    - Move the ants
    - Breed the ants
    - Print updated board
    - Decrement numMoves

Play again or exit

- Ask the user to enter another number of steps to run, or to enter “0” to exit
    - If they enter 0, exit the program
- If they enter a different int, save this in numSteps and go back the the play function

Class Diagrams

Critter Class	
Member Variables	***Member Functions***
int daysAlive	Critter()
int xPos	virtual void move(Critter***, int, int)
int yPos	virtual void breed(Critter***, int, int)
char symbol	virtual char getSymbol();

Ant Class Derived from Critter Class	
Member Variables	***Member Functions***
	Ant(int, int, int)
	virtual void move(Critter***, int, int)
	virtual void breed(Critter***, int, int)

Doodlebug Class Derived from Critter Class	
Member Variables	***Member Functions***
int daysStarving	Doodlebug(int, int, int, int)
	virtual void move(Critter***, int, int)
	virtual void breed(Critter***, int, int)
	void starve(Critter**);

Board Class	
Member Variables	***Member Functions***
int rows	Board()
int cols	Board(int, int)
const int	~Board()
boardRows	void createBoard()
constInt	void printBoard()
boardCols	void setRows(int)
Critter***	void setColumns(int)
board	int getRandom(int, int)

Game Class	
Member Variables	***Member Functions***
char **board	Game()
	~Game()
	void startMenu()
	void simulation()
	int inputValidation()

Initial Testing Plan

Function to Test: inputValidation()
What It Does: Asks user if they want standard (20x20) board or if they want to customize it. Program should accept an integer that is either 1 or 2
Why This Test: Make sure our input validation works, despite the user's best efforts to break it
Assumed Previous Input: None

User input to test	**Expected result**	**Actual result**
1	Input should be accepted Program should move on to generate the gameboard and ask user for number of steps	
2	Input should be accepted Program should move on to ask the user to define the gameboard	
1.5	Error message should appear User should be prompted to try again	
Abc	Error message should appear User should be prompted to try again	
-42	Error message should appear User should be prompted to try again	
#	Error message should appear User should be prompted to try again	
(space character)	Error message should appear User should be prompted to try again	
(newline / return)	Error message should appear User should be prompted to try again	

Function to Test: Board::printBoard()
What It Does: Displays the current gameboard
Why This Test: Ensure that the board is constructed and printed correctly at the start of the game, with the correct number ants and bugs distributed randomly.
Assumed Previous Input: User chose either standard or custom in start menu, and inputted the number of moves in the game

User input to test	**Expected result**	**Actual result**
User chose standard 20x20 board	Initial 20x20 board should print Board should look like a square grid Board should have 5 doodlebugs("X") and 100 ants("O"), and blank spaces for the rest of the cells	

User chose custom board with: 50 rows 50 cols XX ants XX bugs	Initial 50x50 board should print Board should look like a square grid Board should display XX ants and XX bugs, distributed randomly, and blank spaces for the rest of the cells	
User chose custom board with: 5 rows 5 cols XX ants XX bugs	Initial 5x5 board should print Board should look like a square grid Board should display XX ants and XX bugs, distributed randomly, and blank spaces for the rest of the cells	
User chose custom board with: 30 rows 10 cols XX ants XX bugs	Initial 30x10 board should print Board should look like a rectangular grid Board should display XX ants and XX bugs, distributed randomly, and blank spaces for the rest of the cells	

Function to Test: game play - not sure what the function name is yet
What It Does: The game should run for the correct number of moves
Why This Test: Ensure that the number of moves selected by the player is the number of moves that the game actually plays
Assumed Previous Input:

User input to test	Expected result	Actual result

Function to Test: End menu, play again or exit - not sure the name of the function yet
What It Does: Asks the user if they'd like to play again or exit the game
Why This Test: Making sure that our end menu loops or exits appropriately and that input validation is still happening
Assumed Previous Input: User has successfully run at least one game

User input to test	Expected result	Actual result
1	The game restarts	
2	The program exits	
3	Error message should appear User should be prompted to try again	
0	Error message should appear User should be prompted to try again	

?	Error message should appear User should be prompted to try again	

Item to Test: Our makefile
What It Does: Compiles all the program files on flip
Why This Test: Check to make sure the makefile works before we zip up the files

User input to test	**Expected result**	**Actual result**
make	Files should compile with no errors Executable program file should appear	
make clean	Executable program file and all *.o files should be deleted from flip	