Program name: Group 8 project - Predator-Prey game
Authors: Sheila Babayan, Becky Chao, Elizabeth Donato, Devin Gendron, Ronald Walters
Date: 2/19/2018
Description: This is the project plan and reflection document for Group 8.

Introduction

We created a 2D predator-prey simulation that 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. 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 writing pseudocode to ensure that our ideas were clearly described and that all assignment requirements had been met. From this pseudocode, we created diagrams that would be used to make 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.

Tools
Chat: Slack - https://join.slack.com/t/cs162-group8/signup?x=x-311791454866-312644845526
Code sharing: Github - https://github.com/GendronD/Predator-Prey-Game
Project plan and reflection: Google docs
Live demos and quick tests: https://www.onlinegdb.com

Expected process
1. Write pseudocode
2. Round 1 - Devin and Ron tackle stuff on GIthub, Sheila makes testing plan
3. Round 2 - Join the pieces together and fix all the stuff that breaks
4. Round 3 - Run through testing plan and fix all the stuff that breaks
5. Write reflection

Pseudocode

Set up the gameboard

- *Print message stating we did the extra credit*
- Ask the user whether they want a standard (20x20, set # critters) gameboard or whether they want to set the size and # critters themselves

Input val: Must be an integer, 1 or 2

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”

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)

- Ask the user to enter the number of time steps to run

Input validation: Must be an integer greater than 0

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 time steps to run, or to enter “0” to exit

Input validation: Must be an integer

- If they enter 0, exit the program
- If they enter a different int, save this in numSteps and go back the the play function

Diagrams - expected classes

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()

Testing plan

Function to test: inputValidation() in the start menu
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	As expected
2	Input should be accepted Program should move on to ask the user to define the gameboard	As expected
1.5	Error message should appear User should be prompted to try again	As expected
abc	Error message should appear User should be prompted to try again	As expected
-42	Error message should appear User should be prompted to try again	As expected
#	Error message should appear User should be prompted to try again	As expected

(space character)	Error message should appear User should be prompted to try again	As expected
(newline / return)	Error message should appear User should be prompted to try again	As expected

Function to test: All userPrompt functions, plus, Board::printBoard()
What it does: Displays the current/starting 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 with standard 5 doodlebugs and 100 ants Steps: 1	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	As expected
User chose custom board with: 50 rows 50 cols 100 ants 5 bugs Steps: 1	Initial 50x50 board should print Board should look like a square grid Board should display 100 ants and 5 bugs, distributed randomly, and blank spaces for the rest of the cells	As expected
User chose custom board with: 10 rows 10 cols 10 ants 10 bugs Steps: 1	Initial 10x10 board should print Board should look like a square grid Board should display 10 ants and 10 bugs, distributed randomly, and blank spaces for the rest of the cells	As expected
User chose custom board with: 30 rows 10 cols 50 ants 1 bug Steps: 1	Initial 30x10 board should print Board should look like a rectangular grid Board should display 50 ants and 1 bug, distributed randomly, and blank spaces for the rest of the cells	As expected

Function to test: gameboard->simulation(numSteps)
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

User input to test	**Expected result**	**Actual result**
Any board configuration, standard or custom 1 step	The board should print twice: Once to show the initial board, and once to show the 1 step that was moved	Standard config did not print starting board. Updated main.cpp to enable printing of initial board for both menu options.
Any board configuration, standard or custom 10 steps	The board should print 11x: Once to show the initial board, and 10x to show the steps	As expected, after making the edit above

Function to test: standard game play
What it does: Runs the game simulation
Why this test: Check to see if any weird behavior is happening with our critters
Assumed previous input: User chose standard 20x20 board with standard 5 doodlebugs and 100 ants

User input to test	**Expected result**	**Actual result**
Steps: 1	Standard board should print Board should have 5 “X”s and 100 “O”s 1 step should run Neither critter should “win” since there aren’t enough steps. Depending on the distribution, some ants may get eaten.	As expected
Steps: 10	Standard board should print Board should have 5 “X”s and 100 “O”s 10 steps should run Ants should win because they breed more quickly and don’t starve	As expected. By the 10th step, the “O”s had almost taken over the board
Steps: 100	Standard board should print Board should have 5 “X”s and 100 “O”s 100 steps should run	Ants began taking over the board, but after 50 steps or so the doodles made a comeback. End result was about even. There were also “rivers” in the pattens.

Function to test: custom game play
What it does: Sets custom game with board rows (10-100) and board cols (10-100). The user can then fill the board with up to 50% ants and 25% doodlebugs.
Why this test: Check to see if any weird behavior is happening with our critters
Assumed previous input: User chose custom board

User input to test	**Expected result**	**Actual result**
Ants: 5000 Doodles:1250 Board size: 10000 Steps: 15000	Custom board should print Board should have 5000 “X”s and 1250 “O’s” 15000 steps should tun	Board begins with proper size and number of ants, doodlebugs, and steps. There is no clear winner, the board appears to have equalized.
Ants: 5000 Doodles:1250 Board size: 10000 Steps: 105000	Custom board should print Board should have 5000 “X”s and 1250 “O’s” 105000 steps should tun	Board begins with proper size and number of ants, doodlebugs, and steps. There is no clear winner, the board appears to have equalized.
Ants: 312 Doodles:156 Board size: 2500 Steps: 15000	Custom board should print Board should have 312 “X”s and 156 “O’s” 15000 steps should tun	Board begins with proper size and number of ants, doodlebugs, and steps. There is no clear winner, the board appears to have equalized.

Function to test: End menu, play further or exit
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 user is prompted to enter how many steps they'd like to run The game continues	As expected
2	The program exits	As expected
3	Error message should appear User should be prompted to try again	As expected
0	Error message should appear User should be prompted to try again	As expected
?	Error message should appear User should be prompted to try again	As expected

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	Files compile with no errors Executable program file appears
make clean	Executable program file and all *.o files should be deleted from flip	Executable program file and all *.o files deleted from flip

Reflection

Actual work distribution

Planning

Pseudocode and initial planning: Devin, Ron, Sheila
Class diagrams: Liz

Coding, round 1

Menus and input validation: Devin
Critter, Ant, and Doodlebug: Ron
Integration and compile errors: Sheila

Coding, round 2

Menus and input validation: Devin, Sheila
Critter, And, and Doodlebug: Ron, Devin, Becky
Integration and compile errors: Ron, Sheila, Devin
Makefile: Becky
Version-2 branch: Liz

Testing and bug fixing

Fixing critter movements: Ron
Menus and input validation: Devin
Integration and compile errors: Ron, Sheila, Devin
Finish and implement testing plan: Sheila, Devin

Final files

Finish project plan: Sheila
Final testing on flip: Everyone
Team reflection: Everyone
Turn the project in: Devin

Team reflection

What changed from our initial plan

- We initially planned to do a board of chars, but realized that wouldn't accomplish what we wanted. We had to change the board to Critter pointer objects in order to get the polymorphism to work correctly, where each Critter object had a member variable with its corresponding symbol (' ', 'X', or 'O')
- Once we started testing, we realized that we had no way to track bugs that had already moved. This meant that if a bug moved east or south, it would end up moving multiple times during each time step. To fix that, we had to implement a variable to track whether the bug had moved during that time step or not.
- Initially we'd planned a Game class from which to run the simulation, but later we decided it wasn't necessary and was actually adding complexity to the program.

What went well

- Our code compiles and runs!
- We were able to get the initial round of general code completed very quickly to start testing and debugging.
- Google docs was good for group collaboration on the project plan
- When we were able to collaborate, a lot was accomplished in a short time.
- Setting up github and adding collaborators spurred a lot of work to be done and was an excellent tool to get things rolling.

What didn't go well

- Time zones and schedules made working together at the same time tough.
- It was hard to get in contact with our teammates. The first 1.5 weeks was spent trying to gather everyone into one communication space
- Lots of last-minute debugging and changes
- Contribution amongst team members was significantly unbalanced
- Members attempting to implement work that diverged from the rest of the groups implementation entirely.

What we'd change next time in order to improve the process or outcome

- Get the team communications into Slack asap
- Start planning as soon as at least 2 people on the team are communicating. Don't wait for the entire team to join in, as we can't control other people's availability.
- Divide the work not only by function, but also by time. Assign owners for specific time periods and ask them to push the work forward, upload all the latest files, and then tell the team what changed and what the next steps should be.
- Create a specific task list and ensure that workload is divided evenly and that all group members contribute to the project.
- Ensure that design decisions and project changes are clearly communicated to all group members.