Design Description

Program name : DesignReflection_Group18.pdf
Group: Group 18
Authors : Shane Dewar, Dae Hun Park, Kristin Schaefer, Matthew Wong
Date : 05-12-2019

Main Function

Description: The Main function creates a Game object and calls all of the Game functions necessary to play the Predator-Prey game.

Process:

1. Create Game object
2. Call Game member function to set starting value of number of steps as well as values necessary for extra credit option
3. Call Game member function to “initialize the world” with user-specified number of Ants and Doodlebugs and grid
4. Call Game member function to play the Predator-Prey game.

Critter Class

Description: The Critter class is the parent class of Ant and Doodlebug, thus it contains variables and functions common to Ants and Doodlebugs.

Key data members:

1. X and Y location
   - Describes the location of the Critter based on the grid.
2. Number of steps / age
   - Keeps track of the number of steps taken to help determine if the Critter will breed.

Key functions:

1. Move()
   - Virtual function
   - For every time step, the move function randomly selects an adjacent cell and checks the occupancy condition of that cell. Depending on the type of critter, the critter may or may not move to the adjacent cell. If the adjacent cell is off the grid, the critter remains in the same cell.
2. Breed()
   - Virtual function
   - If a critter survives a specific amount of steps, an adjacent cell is randomly selected and checked if it is empty. If it is empty, a new critter will be created to occupy that cell. If that cell is occupied, the other cells are checked. If no cells are available, no breeding occurs.

Ant Class

Description: The Ant class is a derived class of the Critter class. In the Predator-Prey game, Ants are the prey of Doodlebugs. Ants may move around the board and breed if they survive (not eaten by a Doodlebug) three time steps.

Key data members:

1. X and Y location
   - Inherited from Critter class.
   - Describes the location of the Critter based on the grid.
2. Number of steps / age
   - Inherited from Critter class.
   - Keeps track of the number of steps taken to help determine if the critter will breed.

Key functions:

1. Move()
   - Virtual function, overidden
   - For every time step, the move function randomly selects an adjacent cell and checks the occupancy condition of that cell. If the adjacent cell selected is occupied by a doodlebug, the ant stays in the same cell. If the adjacent cell is off the grid, the ant remains in the same cell.
   - Generate random number to move up, down, left or right in the grid.
2. Breed()
   - Virtual function, overidden
   - If the ant survives 3 time steps, an adjacent cell is randomly selected and checked if it is empty. If it is empty, a new ant will be created to occupy that cell. If that cell is occupied, the other cells are checked. If no cells are available, no breeding occurs.

Design Description

Program name : DesignReflection_Group18.pdf
Group: Group 18
Authors : Shane Dewar, Dae Hun Park, Kristin Schaefer, Matthew Wong
Date : 05-12-2019

Doodlebug Class

Description: The Doodlebug class is a derived class of the Critter class. In the Predator-Prey game, Doodlebugs are the predator of Ants. Doodlebugs move around the board and breed if they survive (manage to eat an Ant in three time steps) eight time steps.

Key data members:

1. X and Y location
   - Inherited from Critter class.
   - Describes the location of the Critter based on the grid.
2. Number of steps / age
   - Inherited from Critter class.
   - Keeps track of the number of steps taken to help determine if the critter will breed.
3. Starve point
   - Not inherited from Critter class.
   - Keeps track of the number of days that it takes for the Doodlebug to starve.

Key functions:

1. Move()
   - Virtual function, overidden
   - For every time step, the move function randomly selects an adjacent cell and checks the occupancy condition of that cell. If there are ant(s) in the adjacent cell, the doodlebug will move to that cell. If the adjacent cell is empty the doodlebug will not move to that cell. If the adjacent cell is off the grid, the doodlebug remains in the same cell.
   - Generate random number to move up, down, left or right in the grid.
2. Breed()
   - Virtual function, overidden
   - If a doodlebug survives 8 steps, an adjacent cell is randomly selected and checked if it is empty. If it is empty, a new doodlebug will be created to occupy that cell. If that cell is occupied, the other cells are checked. If no cells are available, no breeding occurs.
3. Starve()
   - Not inherited from Critter class.
   - If a doodlebug has not eaten an ant within three time steps, at the end of the third time step it will starve and die. The doodlebug should then be removed from the grid of cells.

Game Class

Description: The Game class initializes the world with Ants and Doodlebugs and creates a 2d grid consisting of “O” for Ants, “X” for Doodlebugs and “ “ for empty cells. The Game class collects user-specified values for the number of time steps, the size of the grid (extra credit) and the number of Ants and Doodlebugs (extra credit). Most importantly, the Game class controls the simulation of the Predator-Prey game.

Key data members:

1. Number of time steps
   - Represents the number of times / loops the simulation runs for.
   - To be specified by the user, and input validated
2. Number of grid columns and grid rows
   - Hold the number of rows and columns in the char grid
   - To be specified by the user, and input validated (extra credit)
3. Dynamic array to represent char grid
   - Dynamic array of chars, representing the Predator-Prey game board.
   - “O” represents an Ant, “X” represents a Doodlebug, “ “ represents an empty cell.
4. Dynamic array of critters

Key functions:

1. setUserValues()
   - Gets and validates the number of time steps, grid rows, grid columns, Ants and Doodlebugs
2. initializeWorld()
   - Initializes the board and the Critters, and randomly places the Critters on the board.
3. playGame()
   - Controls one time step simulation.

- Moves Doodlebugs, then Ants one step in the board.
- If a Doodlebug moves to a cell with an Ant, it eats the Ant.
- Checks if Critters can breed, and creates new Critters if conditions are met.
- If a Doodlebug has not eaten for 3 turns, it needs to be deleted (starves).
- print Board.

4. printBoard()
   - Prints the current board, indicating the position of the Ants, Doodlebugs and empty spaces.

Testing Plan

Program name : DesignReflection_Group18.pdf
Group: Group 18
Authors : Shane Dewar, Dae Hun Park, Kristin Schaefer, Matthew Wong
Date : 05-12-2019

File	Function	Variable(s)
game.cpp	setStartValue() inputVal(1,10000)	int numSteps

	Test 1 : int too high	Test 2 : int too low	Test 3 : int negative	Test 4 : float	Test 5 : char	Test 6 : string	Test 7 : mix	Test 8 : mix + space
	20000	0	-100	123.60	x	quit	fdsa;15	1 abc
Expected Outcome	pass input validation	pass input validation	pass input validation	pass input validation	pass input validation	pass input validation	pass input validation	pass input validation
Observed Outcome	pass input validation	pass input validation	pass input validation	pass input validation	pass input validation	pass input validation	pass input validation	pass input validation

Testing Plan

Program name : DesignReflection_Group18.pdf
Group: Group 18
Authors : Shane Dewar, Dae Hun Park, Kristin Schaefer, Matthew Wong
Date : 05-12-2019

File	Function	Variable(s)
game.cpp	setStartValue() inputVal(1,1000)	int gridRow, gridCol

	Test 1 : int too high	Test 2 : int too low	Test 3 : int negative	Test 4 : float	Test 5 : char	Test 6 : string	Test 7 : mix	Test 8 : mix + space
	20000	0	-100	123.60	x	quit	fdsa;15	1 abc
Expected Outcome	pass input validation	pass input validation	pass input validation	pass input validation	pass input validation	pass input validation	pass input validation	pass input validation
Observed Outcome	pass input validation	pass input validation	pass input validation	pass input validation	pass input validation	pass input validation	pass input validation	pass input validation

Testing Plan

Program name : DesignReflection_Group18.pdf
Group: Group 18
Authors : Shane Dewar, Dae Hun Park, Kristin Schaefer, Matthew Wong
Date : 05-12-2019

File	Function	Variable(s)
game.cpp	setStartValue() inputVal(1,1000)	int amountAnt, amountDood

	Test 1 : int too high	Test 2 : int too low	Test 3 : int negative	Test 4 : float	Test 5 : char	Test 6 : string	Test 7 : mix	Test 8 : mix + space
	20000	0	-100	123.60	x	quit	fdsa;15	1 abc
Expected Outcome	pass input validation	pass input validation	pass input validation	pass input validation	pass input validation	pass input validation	pass input validation	pass input validation
Observed Outcome	pass input validation	pass input validation	pass input validation	pass input validation	pass input validation	pass input validation	pass input validation	pass input validation

Program name :	DesignReflection_Group18.pdf
Group:	Group 18
Authors :	Shane Dewar, Dae Hun Park, Kristin Schaefer, Matthew Wong
Date :	05-12-2019

Changes in design, problems encountered and how problems were solved

One of the main design challenges our group faced for the Predator-Prey game was determining how to implement the dynamic array to represent the grid and to control the actions of the Ants and Doodlebugs. Our initial design plan was to create two dynamic arrays: one 2d-char array to display the board, the other a 2d-array of pointers to Critter objects. After starting to write the Ant, Doodlebug and Game functions we realized that reducing to one dynamic 2d-array would be more simple to use. We explored the possibilities of using either a dynamic 2d-char array or a dynamic 2d-array of pointers to Critter objects, and ultimately we chose to use a dynamic 2d-array of pointers to Critter objects. To replace the 2d-char array a char variable representing the type of Critter (‘a’ for Ant or ‘d’ for Doodlebug) and a getType() function were created in order to check the type of Critter in the cell.

Although we intended in the design plan to override the move() and breed() functions in both the Ant and Doodlebug classes, we tried to avoid overriding both functions for both classes. However, this proved to not be possible due to the different move and breed conditions for the classes, and we reverted to our original design plan. Our team also decided to create a getRandomNumber function for the initial placement and the random movements of the Ants and Doodlebugs. It suited our design better to make the getRandomNumber function independent of the Critter class in order to use the function with the Game class as well.

Our group opted to do the extra credit option, which in addition to prompting the user to enter the number of time steps, asked the user to also enter the size of the grid rows and columns and the number of Ants and Doodlebugs. Initially our input validation functions appeared to be working well with the program, however, after using the testing plan we had issues with the values going out of bounds. We then updated our input validation by creating an independent inputval function to be flexible enough to check all of our user input values. The inputval function was designed to take two integers, representing the range to check, as parameters. The function consisted of a do-while loop which took the user input as a string, checked if the string was a valid input, and then converted the user input to an integer and returned the validated integer to the Game class member functions. This improvement was helpful in that it allowed us to pass the values such as “gridCol * gridRow” as a range parameter so that the user input of number of Ants and Doodlebugs could not be higher than the number of cells in the board.

Working on such a challenging project with a group was a great experience. It would have been a very difficult project to complete alone, and thanks to great communication(via Campus) and lots of effort, our group completed the project with great results.

Distribution of Work

Program name :	DesignReflection_Group18.pdf
Group:	Group 18
Authors :	Shane Dewar, Dae Hun Park, Kristin Schaefer, Matthew Wong
Date :	05-12-2019

1. Combine all files, comments, makefile, reflection doc., test on flip, submit .zip file on Canvas

- Shane Dewar
- Dae Hun Park
- Kristin Schaefer

2. Critter class

- Shane Dewar
- Dae Hun Park
- Kristin Schaefer
- Matthew Wong

3. Ant class and Doodlebug Class

- Shane Dewar
- Dae Hun Park

4. Simulation / Dynamic array / Display the result

- Shane Dewar
- Dae Hun Park

5. Menu / Input validation. + Extra Credit

- Shane Dewar
- Dae Hun Park
- Kristin Schaefer
- Matthew Wong