Group 2

Group Project: Predator-Prey Game

# **Program Design**

Problem to be solved: In this lab, you will use cellular automata to create a 2D predator–prey simulation in your program.  The preys are ants and the predators are doodlebugs.

## **Structure of Program**

**Menu Selection**

* When the game starts the menu should prompt the user to do one of the following:
  + 1. Play Predator-Prey
  + 2. Exit game
* If the user decided to “Exit game”, the program should quit.
* If the user decides to “Play Predator-Prey”, the program should prompt user for the following information:
  + How many time steps do you want to run?
    - What should the min and max steps be?

**Game Play**

BEGIN GAME:

* Initialize a board object to be a 20 \* 20 grid
  + Each cell should be assigned a “ “ (empty space).
* Randomly assign 5 cells to be doodlebugs (“X”)
* Randomly assign 100 cells to be ants (“O”)
* User is prompted to enter the number of time steps for the game
* Start steps

FOR EACH TIME STEP:

* Doodlebug moves first
  + move();
    - Check adjacent cells for ants (bool checkForAnts)
    - If checkForAnt == true\*, eatAnt();
      * move to the cell harboring the ant
      * replace the ant with a doodlebug (“O” becomes “X”)

\*If there are more than one adjacent cells harboring ants, how should we choose to move? Random? First cell we find with an ant?

* + - Else, move according to the following rules:
      * Randomly select a cell to move up, down, left, or right
      * If the selected cell is occupied, OR will move the critter off the grid, stay put
      * Mark the cell the critter ended up in with the ASCII character “X”
      * Increment the stepCounter for the critter
  + breed();
    - If stepCounter % 8 == 0, spawn off a new doodlebug
      * Check for any empty adjacent cells (up, down, left, or right)
      * If empty adjacent cells exist:
        + randomly select one to put a new doodlebug in
      * Else, do not breed
  + starveAndDie();
    - Check if the doodlebug has eaten an ant.
      * If ( stepCounter % 3 == 0 ) && ( !hasEatenAnt ):
        + Doodlebug cell == “ “;
* Ant moves second
  + move();
    - Randomly select a cell to move up, down, left, or right
      * If the selected cell is occupied, OR will move the critter off the grid, stay put
    - Mark the cell the critter ended up in with the ASCII character “O”
    - Increment the step count for the critter
  + breed();
    - If stepCounter % 3 == 0, spawn off a new ant
      * Check for any empty adjacent cells (up, down, left, or right)
      * If empty adjacent cells exist:
        + randomly select one to put a new ant in
      * Else, do not breed
* Display resulting grid (after moves, eating, breeding, and starving functions have been completed for both doodlebugs and ants)

MAX TIME STEPS REACHED:

* This should be the while part of the do, while loop in main():
  + do { timeStep(); } while( !maxSteps );
  + When maxSteps evaluates to true display a menu to prompt the user to do one of the following:
    - 1. Play again?
    - 2. Exit Program

**Modular Components**

**Menu Class**

* This class will print menu options to the screen for the user and allow the user to enter input based on the menu options displayed. The input will be validated before allowing the user to continue through the program.
* Some key functions and data members:
  + vector<string> menuOptions;
  + createMenu( vector<string> options)
  + displayMenu()
  + makeSelection( Menu )
    - This function will verify the user’s input and will return the validated input back to the program
  + getMenuOptions() // added during implementation

**Input Validation Class**

* This class will validate user input
* Some key functions and data members:
  + isValidInt – verifies that the input is a valid integer
  + isValidFloat – verifies that the input is a valid float

**Critter – Virtual Base Class**

* Virtual base class to create a critter object.
* Some key functions and data members:
  + virtual void move() = 0;
  + void breed();

**Ant Class – Derived from Critter**

* Derived from Critter
* Some key functions and data members:
  + virtual void move();

**Doodlebug Class – Derived from Critter**

* Derived from Critter
* Some key functions and data members:
  + virtual void move();

**Board Class**

* Some key functions and data members:

# **Test Table**

|  |  |  |
| --- | --- | --- |
| **Test Scope** | **Test Case** | **Expected Output** |
| Menu | When the program starts the menu will provide 2 options. | Options will be:   1. Play game 2. Exit game |
| Menu | User enters invalid data at the first menu. Invalid data is anything other than a 1 or a 2.   * Float * Characters * Negative number * 0 | Input is requested again until the user inputs valid data. |
| Menu | User chooses option 2, the program should quit. | Program quits. |
| Menu | User chooses option 1, the game should begin. | Game begins.  Program prompts the user for the following information needed for the game:   * Enter a number of time steps |
|  |  |  |
|  |  |  |
|  |  |  |

# **Reflection**