Below are all the components for the DoodleBug group project as part of the design:

Main

(INCLUDES EXTRA CREDIT)

The main functions operate with a continuous loop for each iteration through the game's functions. This is controlled via the three instances of the Menu class that control the Main Menu, Options, or Next Tur:n.

The menu system uses a switch case that will initialize the game with the generic values of board size, ants, doodlebugs and number of rounds. However, each of these values is editable through the options menu. As a whole, the menu will continue to reprompt for variable values until either "Start Game" or "Exit" is selected by the user.

At this time, it checks that there enough viable board spaces for the number of creatures, has the use correct this if the number of spaces is exceeded, then creates the board and populates it with the given or base number of critters in random locations.

Finally the runBoard() function is called which runs the simulation. Upon its completion the board is deleted freeing any memory that was declared on the heap, and the user is asked if they wish to play again.

If so, the process repeats, otherwise the program ends.

Ant

The Ant class is a simplified version of the DoodleBug with only the capability to move, breed, and be eaten. The movement picks a direction at random and moves to the space. However, if the space is occupied it works through the other directions in a clockwise design. Should the Ant try to move off the board, it displays a message saying so and does not move instead.

For breeding, the Ant keeps track of the number of moves that have occurred, and once the number has reached a minimum of 3 turns, it attempts to create a new Ant in an adjacent space. If none exist at the time, then the counter continues to rise until it successfully breeds, at which time the counter is reset back to 0.

Finally, the eaten portion of the Ant is handled through the DoodleBug class itself, which simply calls the Ant destructor and sets the space to NULL.

Board

Board is the set of functions that print out the location of all critters and empty spaces, tracks the location of objects, places the critters at random locations on the board upon starting the game, and handles the memory management for both expanding or clearing when critters breed or are eaten and starve.

Critter

The critter class is a base parent class that both Ant and DoodleBug are polymorphically derived from, to contain movement and breeding functions that are called the same but produce different results.

The critter class contains a string to represent the symbol of what type of critter occupies a space within the board. As well as a Space pointer that holds the latest position of the critter's location for the purposes of movement, hunting, and symbol placement.

Function wise, the accessors and mutators are also include that return space and symbol position and type respectively. RemoveCritter() calls itself for the space that that critter object currently occupies, and sets the pointer to NULL creating an blank position. The last function is the generic movement function that takes in a direction and checks for the appropriate conditions to be met. If correct, it changes the position of the creature and sets the original position to NULL. Otherwise it continues to check surrounding for both of two cases, then moves if possible to a legal position.

DoodleBug

DoodleBug is a derived class from the base Critter class that contains a more complicated movement system involving hunting for ants, the ability to starve if the prior ability has not been used within a set timeframe, and an extended time between breeding.

For hunting and movement, the class checks each adjacent space for ant, then with an if else statement determines if any ants were eaten. If so, the order of priority is in a clockwise design, with an ant above taking the highest. If no ants are found, then it moves as normal. Additionally, if now ant has been eaten within a count of three turns, the DoodleBug starves and its object is deleted, returning the space back to empty.

Menu

The menu classes are called by the main functions, with multiple objects of the menu in existence at runtime. The class includes the big four as well as the ability to add and remove items from the menu. By doing so you can create or limit the number of options within each individual menu.

The additional functions within the class display the various options, remove any memory created on the heap, or set the name of the menu for both input and output purposes.

Menu_item

These set of files are used by the Menu class to create the "items" that are the options that change the variables for board size, critter count, number of turns and whether the game will be started, exited, or played again.

Menu_item itself is a class that contains the necessary sets of data to create an option within the menu. For each data type that could be needed to use for the menu options, and individual constructor is designed. The types include: Int, Double, Bool, Char, String, and Clicks.

Additionally, the associated accessors and mutators are created for each variable, with the mutators using switch statements to change depending on the data type that is being changed.

My_lib

My_lib contains general utility functions that are used throughout the menu_item and menu classes, in order to generate the menus and check that input values are of the correct data type or value. Additionally, there are templates used by the menu_item and menu classes that generate, delete, copy between, add elements, and remove elements from 1D and 2D dynamic arrays.

The other utility functions within my_lib allow for direction reversal, character counting within strings, integer/double/float validation, conversion of data types and input collection. Each of these can be called individually or are used together based on the need in menu.

Files included

- Ant
  - Ant.h
  - Ant.cpp
- Board
  - Board.h
  - Board.cpp
- Critter
  - Critter.h
  - Critter.cpp
- DoodleBug
  - DoodleBug.h
  - DoodleBug.cpp
- Menu
  - Menu.h
  - Menu.cpp
- Menu_item
  - Menu_item.h
  - Menu_item.cpp
- My_lib
  - My_lib.h
  - My_lib.cpp
- CMakeLists.txt
- README.m