DREW WOLFE
CS 162 SUMMER 2016
DESIGN DOCUMENT: Assignment 2

GOALS

1. Develop classes from given requirements
2. Implement those classes using object-oriented programming techniques

REQUIREMENTS

- **Overview**
  - Design, implement, and test a cellular automata program.
    - The program will simulate a simple predator-prey model. You will have a 2D array populated by Ants, Doodlebugs and empty spaces.

- **Class hierarchy**
  - Critter base class
  - Ant subclass
  - Doodlebug subclass

- **Movement:**
  - The 2D array will have a data type of pointer to Critter.
    - That allows either type of Critter to be in any element of the array.
  - Movement is random.
    - The move function just returns one of the 4 possible directions the Critter can move, determined at random by the object.
  - The main function will hold the array.
  - If the Critter cannot move, it remains in the same element.
  - The Critter will have no knowledge of its location; it just wants to move.
  - Doodlebug cannot share a cell with another Doodlebug; a move attempt fails.
  - Ant cannot share a cell with another Ant; a move attempt fails.
  - Ant cannot share a cell with Doodlebug, a move attempt fails.
  - Doodlebug moves to cell with Ant, Ant is eaten.

- **Breeding**:
  - The Critter must know the number of turns since it last bred.
  - Before a Critter moves the main program must call its breed function.
  - When the move counter has reaches its breed trigger, a new Critter object must be left in the original position of the parent Critter.
    - 3 for an Ant
    - 8 for a Doodlebug
  - The Critter does not breed if it cannot move.
  - Reset the breed count to 0 after breeding.
    - When the breed function is called and the counter is not at the limit, the function will simply increment it by one.
  - Breed increments the time to breed counter or resets it.
  - It returns false if nothing happens, or true if it's time for a new Critter.
  - The main loop will know that the Critter didn't move so it will not create the new Critter.

DREW WOLFE
CS 162 SUMMER 2016
DESIGN DOCUMENT: Assignment 2

- **Eating**:
  - Doodlebugs eat Ants.
    - When eaten the pointer to the Ant object is freed in the main program.
  - Programs calls the eat function.
    - This function resets the Move counter if the Doodlebug ate an Ant.
    - The function increments the counter for the Critter when it does not eat.
    - Use a Boolean parameter to indicate if the Doodlebug ate an Ant.
  - When eat is called for an Ant it will ignore the parameter.

- **Death**:
  - After the Move and Eat functions have been called, the main program must call the die function.
  - IFF a Critter's Move counter exceeds the TTL counter THEN Critter dies and is removed from the grid.
    - 10 moves for Ants
    - 3 moves for Doodlebugs
  - Use a Boolean function.

- **The Program**:
  - Use a nested loop to run through the array.
  - For each occupied element call the move, breed, eat, and die functions
  - Included the class hierarchy with the behavior of the functions specified for each class and subclass.
    - Functions have different limits for their counters.
    - Eat for a Doodlebug may also reset the time to live counter.
  - IMPORTANT NOTE: All behavior for a class must be contained within that class. If you are doing some activity for a class outside the class then you are not enforcing encapsulation.
  - The main program will manage the array.
  - Use a 20x20 static array.
  - Display the grid with Ants, Doodlebugs and empty spaces at each time step.
    - Pause the display at each step
  - All the counters will start at 0.
  - You will start with 5 Doodlebugs and 100 Ants.
  - For debugging start each Critter in the same location.
    - Adjust the starting number of each Critter in case the grid fills up, or one or both Critters consistently die out.
    - Remember that if you do that you would also need to add or remove Critters from the starting configuration.

- **Submit**
  - Design document
  - Test plan and results
  - Source and header files
    - Critter

- Ant
- Doodlebug
- main
- Makefile

- Submit all in a zip file

FUNCTION DEFINITIONS (STUBBING OUT)

```
class Critter
{
public:
     bool move();
     void eat();
     void breed();
     void die();

private:

}

class Ant : public Critter
{
public:


private:
     int moveCount = 0;
     int breedCount = 0;
     int currentCoord [][];
     int newCoord [][];
     int oldCoord [][];

};

class Doodlebug : public Critter
{
public:


private:
          int moveCount = 0;
     int breedCount = 0;
     int currentCoord [][];
     int newCoord [][];
     int oldCoord [][];

}
```

DREW WOLFE
CS 162 SUMMER 2016
DESIGN DOCUMENT: Assignment 2

```
//Number of ants in the 2D array
int antCount;

//Number of doodlebugs in the 2D array
int dBugCount;

//Used by gridTravel() to determine where it is within the grid
int currentGridCoord [][];

//Used by gridTravel() to determine where within the grid it will move
next
int newGridCoord [][];

//Used by Critter to store current grid location
int currentCoord [][];

//Used by Critter to store where within the grid it will move next
int newCoord [][];

//Used by Critter to store coordinate from which the Critter has moved
int oldCoord [][];

//20 X 20 2D array holds Critters. Spaces will initially be filled with "
"
int critterArray [][];

//Variable to track moves scoped to each Critter object
int moveCount;

//Variable to track moves/turns till breed
int breedCount;

//randomly generates a column and row coordinate within the range of 0 &&
19
//returns array coordinates
int randoCoord();

//Function to create and assign the location of Critters within the 2D
array
char fillArray();
/*
fillArray(ant)
     call randoCoord()
          IFF value of space from randoCoord() return == " "
               Create new Ant object
               Assign return from randoCoord() to Ant object's
currentCoord
               Update value in currentCoord space from " " to "A"
               Increment antCount
```

```
          IFF antCount < 100
               Restart fillArray(ant)
          IFF antCount >= 100
               Exit fillArray(ant)
*/

//Function that determines whether grid is completely full or completely
empty
//in order to exit the program. Critters will be assigned a value of 1
//and spaces " " will be assigned a value of 0
bool gridCheck();
/*
Assign char "A" value of 1
Assign char "D" value of 1
Assign char " " value of 0
Sum function of 2D array
     IFF sum of values in grid < 200
          THEN all spaces are not yet filled
               Return bool YES
     IFF sum of values in grid >= 200
          THEN all the spaces are filled
               Return bool NO
     IFF sum of values in grid = 0
          THEN all Critters are dead
               Return bool NO
*/

/*gridTravel() Moves through each coordinate (by index) in the grid and
 initiates the appropriate functions to run against the grid's
 spaces depending on whether its empty, contains an Ant, or
 contains a Doodlebug.*/
void gridTravel();
/*
Run gridCheck()
     IFF return == YES
          Continue gridTravel() loop
     IFF return == NO
          Exit program
Increment column value of currentCoord
     IFF newCoord column value > 20
          Increment row value && set column value to 0
               Update newCoord to currentCoord
                    IFF currentCoord row value > 20
                         Set newCoord column && row value to 0, 0
                              Update newCoord to currentCoord
                                   Restart gridTravel() loop
     IFF newCoord column value < 20
          IFF value in newCoord space == " "
               Update newCoord to currentCoord
```

```
                    Restart gridTravel() loop
          IFF value in newCoord space == "A"
               Update newCoord to currentCoord
                    Run critterLife(Ant)
                         move(Ant)
                         eat(Ant)
                         breed(Ant)
                         die(Ant)
                              Restart gridTravel() loop
          IFF value in newCoord space == "D"
               Update newCoord to currentCoord
                    Run critterLife(Doodlebug)
                         move(Doodlebug)
                         eat(Doodlebug)
                         breed(Doodlebug)
                         die(Doodlebug)
                              Restart gridTravel() loop
*/

//Returns a random direction
int randoDirection();
/*
Randomly chooses up, down, left, or right
     Up = row de-increment by 1
     Down = row increment by 1
     Left = column de-increment by 1
     Right = column increment by 1
increments or de-increments the row || column of Critter currentCoord by 1
returns array coordinates
*/

//goDontGo() determines whether the output from randoDirection() exceed
the limits of the grid
bool goDontGo();
/*
IFF the column || row value from randoDirection() return > 20
     THEN the coordinates are outside of the grid
          Return bool NO (as in do not move)
IFF the column || row value from randoDirection() return < 20
     THEN the coordinates are inside the grid
          Return bool YES
*/

bool move();
/*
IFF goDontGo() return == YES
     IFF grid coordinate value ==
          "D" Doodlebug in space
               Cannot move
```

```
                Increment moveCount
                move() return NO
          “A” Ant in space
                IFF Critter self == Ant
                     Increment moveCount
                     move() return NO
                IFF Critter self == Doodlebug
                     Assign currentCoord value of “ “
                     Update newCoord to currentCoord
                     Increment moveCount
                     move() return YES
          “ “ in space
                IFF Critter self == Ant
                     Assign currentCoord value of “ “
                     Assign newCoord value of “A”
                     Update newCoord to currentCoord
                     Increment moveCount
                     move() return YES
                IFF Critter self == Doodlebug
                     Assign currentCoord value of “ “
                     Assign newCoord value of “D”
                     Update newCoord to currentCoord
                     Increment moveCount
                     move() return YES
*/

void eat();
/*
IFF Critter self == Doodlebug
     IFF value of currentCoord == “A”
          Change space value from “A” to “D”
          Set moveCount to 0
          Exit eat()
     IFF value of currentCoord == “ “
          Exit eat()
     IFF Critter self == Ant
          Exit eat()
*/

void breed();
/*
IFF move() return == YES
     IFF Critter self == Doodlebug && breedCount == 8
          Create new Doodlebug object
          Assign new Doodlebug object to oldCoord
          Update value in space oldCoord from “ “ to “D”
          Reset breedCount to 0
          Exit breed()
     IFF Critter self == Doodlebug && breedCount < 8
```

```
          Increment breedCount
          Exit breed()
     IFF Critter self == Ant && breedCount == 3
          Create new Ant object
          Assign new Ant object to oldCoord
          Update value in space oldCoord from " " to "A"
          Reset breedCount to 0
          Exit breed()
     IFF Critter self == Ant && breedCount < 3
          Increment breedCount
          Exit breed()
ELSE IFF move() return == NO
     IFF Critter self == Doodlebug && breedCount == 8
          Reset breedCount to 0 (this logic is pulled from Mr. Rooker's
email from 7/6/2016 "The counter is still restarted")
          Exit breed()
     IFF Critter self == Doodlebug && breedCount < 8
          Increment breedCount
          Exit breed()
     IFF Critter self == Ant && breedCount == 3
          Reset breedCount to 0
          Exit breed()
     IFF Critter self == Ant && breedCount < 3
          Increment breedCount
          Exit breed()
*/

void die();
/*
IFF Critter self == Doodlebug && moveCount == 3
     Delete Doodlebug object and release memory
     Replace value of currentCoord w/ " "
     Exit die()
ELSE IFF Critter self == Doodlebug && moveCount < 3
     Exit die()
IFF Critter self == Ant && move-counter == 10
     Delete Ant object and release memory
     Replace value of currentCoord w/ " "
     Exit die()
ELSE IFF Critter self == Ant && moveCount < 10
     Exit die()
*/
```

REFLECTION

Holy crap that was hard!

I obviously didn't stick all that close to my design, but I blame that largely on my lack of syntax understanding AND not having fully understood the power / utility of pointers going into the planning.

DREW WOLFE

CS 162 SUMMER 2016
DESIGN DOCUMENT: Assignment 2

Seriously though. This was hard. I learned a lot. Wished I could've finished on-time, but the extra day was super worth it. I refactored a lot of superfluous code out and worked (tirelessly) to ensure encapsulation.