CS162 400 Group 17

Anthony Giuliano
Jacob Seawell
Jamil Akram
Charlie Ylijoki
Danielle McIntosh

Design:

Critter (base)

Protected:

- int age
- bool hasMoved
- string type
- char symbol

Public:

- Critter constructor
- virtual Critter destructor
- virtual Critter* move(vector<Critter*> neighborsVector)
- virtual Critter* breed(vector<Critter*> neighborsVector)
- void addDay()
- string getType()
- bool hasEmptySpace(vector<Critter*> neighborsVector)
- char getSymbol()

Ant (derived)

Public:

- Static int numberOfAnts
- Ant::Ant() { numberOfAnts++ }
- Ant::~Ant() { numberOfAnts-- }
- virtual Critter* move(vector<Critter*> neighborsVector)
- virtual Critter* breed(vector<Critter*>neighborsVector)

Doodlebug (derived)

Public:

- Static int numberOfDoodles
- Doodlebug::Doodlebug() { numberOfDoodles++ }
- Doodlebug::~Doodlebug() { numberOfDoodles-- }
- virtual Critter* move(vector<Critter*> neighborsVector)
- virtual void breed(vector<Critter*>neighborsVector)
- bool hasNeighboringAnt(Critter *&neighbor)
- bool starve()
- void feed()

Game

Private:

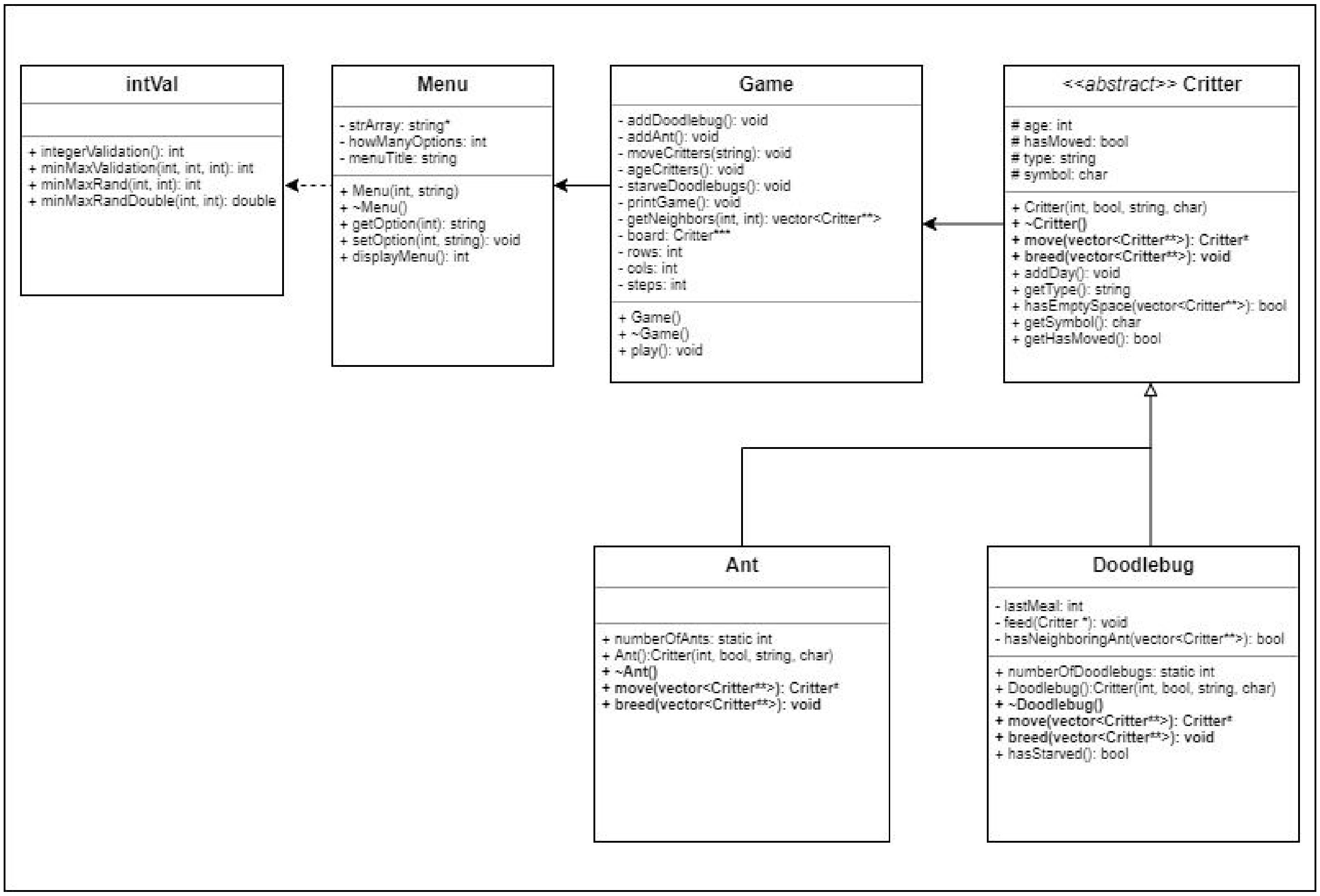
- Critter ***critters
- int rows
- int cols
- int steps

Public:

- Constructor Game::Game
    - {Construct critter board}
- void playGame()

    - void moveDoodles()
    - void moveAnts()
    - void breedDoodles()
    - void breedAnts()
    - void starveDoodles()
    - void printGame()
    - Void addDay()
- Destructor Game::~Game();

Critter Project UML Class Diagram:


intVal
+ integerValidation(): int
+ minMaxValidation(int, int, int): int
+ minMaxRand(int, int): int
+ minMaxRandDouble(int, int): double
Menu
- strArray: string*
- howManyOptions: int
- menuTitle: string
+ Menu(int, string)
+ ~Menu()
+ getOption(int): string
+ setOption(int, string): void
+ displayMenu(): int
Game
- addDoodlebug(): void
- addAnt(): void
- moveCritters(string): void
- ageCritters(): void
- starveDoodlebugs(): void
- printGame(): void
- getNeighbors(int, int): vector<Critter**>
- board: Critter***
- rows: int
- cols: int
- steps: int
+ Game()
+ ~Game()
+ play(): void
<<abstract>> Critter
# age: int
# hasMoved: bool
# type: string
# symbol: char
+ Critter(int, bool, string, char)
+ ~Critter()
+ move(vector<Critter**>): Critter*
+ breed(vector<Critter**>): void
+ addDay(): void
+ getType(): string
+ hasEmptySpace(vector<Critter**>): bool
+ getSymbol(): char
+ getHasMoved(): bool
Ant
+ numberOfAnts: static int
+ Ant():Critter(int, bool, string, char)
+ ~Ant()
+ move(vector<Critter**>): Critter*
+ breed(vector<Critter**>): void
Doodlebug
- lastMeal: int
- feed(Critter *): void
- hasNeighboringAnt(vector<Critter**>): bool
+ numberOfDoodlebugs: static int
+ Doodlebug():Critter(int, bool, string, char)
+ ~Doodlebug()
+ move(vector<Critter**>): Critter*
+ breed(vector<Critter**>): void
+ hasStarved(): bool

Test Table

Test Case #	Test Title	Steps	Expected Results	Actual Results	Pass?
1	Input validation - main menu - lower bounds	1. Open the program 2. Enter “-1”	Invalid entry message displays - prompts with a valid range of ints	Invalid input message displays	Pass

	testing				
2	Input validation - main menu - incorrect value	1. Open the program 2. Enter the value “abc123”	Invalid input message displays - prompts user to input a valid int	Invalid input message displays	Pass
3	Input validation - main menu incorrect value	1. Open the program 2. Enter the value “123abc”	Invalid input message displays - prompts user to input a valid int	Invalid input message displays	Pass
4	Input validation - main menu incorrect value	1. Open the program 2. Enter the value “1 1 1 1”	Invalid input message displays - prompts user to input a valid int	Invalid input message displays - prompts user to input a valid int	Pass
5	Input validation - main menu - upper bounds testing	1. Open the program 2. Enter “3”	Invalid entry. Please enter an integer from 0 to 2 - message displays	Invalid input message displays	Pass
6	Correct workflow - Exiting the program	1. Open the program 2. When prompted for an option, choose to quit (press “0”)	The program should terminate	The program terminates	Pass
7	Correct workflow - Start default game	1. Open the program 2. When prompted for an option, choose to “play game (default settings) [press “1”]	The program will enter a default game prompting the user to enter the number of time steps	The console output displays that option one is selected and is prompting the user for a number of time steps	Pass
8	Default game - input validation	1. Open the program 2. When prompted for an option, choose to “play game (default settings) [press “1”] 3. Enter a negative integer for the time steps 4. Enter “1000” for the number of steps 5. Enter “abc123” for the number of time steps 6. Enter “123abc” for the number of time steps	Invalid entry message displays - prompts with a valid range of ints	Invalid input menu displays	Pass

		7. Enter “.1” for the number of time steps			
9	Default game - correct output	1. Open the program 2. When prompted for an option, choose to “play game (default settings) [press “1”] 3. For the number of time steps, enter “10”	There will be 10 versions of the board, each at a different time step	10 steps display with the corresponding board	Pass
10	Board validation	1. From step 9, verify that the board displays 20 rows and columns. NOTE: the board’s rows and columns are based on the INDEX number	There are twenty rows and columns	20 rows and columns display	Pass
11	Default game - restart prompt - output	1. From step 10, verify that prompt asks the user to continue or quit	A message displays asking the user to play or quit after the game ended	A message displays asking the user to play or quit after the game ended	Pass
12	Default game restart prompt - validation	1. From step 11, enter “-1” 2. Enter “2” 3. Enter “abc” 4. Enter “abc123 5. Enter “1 1 1 1”	An invalid input message displays	An invalid input message displays	Pass
13	Default game restart prompt - Exiting	1. From step 12, enter “0”	The program terminates	The program terminates	Pass
14	Default game restart prompt - restart	1. From step 11, enter “1”	The game prompts the user to enter the number of time steps	The game prompts the user to enter the number of time steps	Pass
15	Default game restart - time step validation	1. From step 14, enter “-1” 2. Enter “abc123” 3. Enter “123abc” 4. Enter “1000”	An invalid input message displays	An invalid input message displays	Pass
16	Default game - restart prompt - play new game -	1. From step 15, for the number of time steps, enter “999”	The game outputs 1000 steps with a corresponding	The game outputs 1000 steps with a corresponding board	Pass

	max steps		board		
17	Default game - restart prompt - play new game - min steps	1. From step 16, press “1” to continue. 2. Enter “1” for the number of time steps.	The game outputs 1 step with a corresponding board	The game outputs 1 step with a corresponding board	Pass
18	Custom game - correct output	1. Start the program 2. Press “2” to play a custom game	The console outputs the custom game dialog, prompting the user for the number of rows and columns	The console outputs the custom game dialog, prompting the user for the number of rows and columns	Pass
19	Custom game - min row and column - max ants	1. From step 18, enter “2” for rows 2. Enter “2” for columns 3. Enter “3” for ants 4. Enter “1” for doodle bugs 5. Enter “10” for steps	The board is empty when the game is over	The board is empty when the game is over	Pass
20	Custom game - min row and column - max doodle bugs	1. Start a new custom game 2. Enter “2” for rows 3. Enter “2” for columns 4. Enter “3” for doodle bugs 5. Enter “1” for ants 6. Enter “10” for time steps	The board is empty when the game is over	The board is empty when the game is over	Pass
21	Custom game - max row and column - max ants	1. Start a new custom game 2. Enter “99” for rows 3. Enter “99” for columns 4. Enter “9800” for ants 5. Enter “1” for doodle bug 6. Enter “999” for steps	There are still ants and doodlebugs on the board	There are still ants and doodlebugs on the board	Pass
22	Custom game - max row and column - max ants	7. Start a new custom game 8. Enter “99” for rows 9. Enter “99” for columns 10. Enter “1” for ants 11. Enter “9800” for doodle bug 12. Enter “999” for steps	There are no ants or doodlebugs on the board	There are no ants or doodlebugs on the board	Pass
23	Custom	1. After step 22, verify	The program	The program exits	Pass

	game - continue prompt - exit	that the continue prompt displays 2. Press “0” to exit	exits		
24	Custom game - continue prompt - more steps	1. After step 22, verify that the continue prompt displays 2. Press “1” to continue 3. Enter “999” for the steps	The program runs, there are no doodlebugs or ants on the board	The program runs, there are no doodlebugs or ants on the board	Pass

Design Changes:

The final implementation of our project is a product of tweaks and changes to our original design.

One example of a design change is how we decided to have the Critter objects check if there is a different Critter object in an adjacent space on the board. In our original design we weren’t sure if we wanted the Critter objects to always know their location on the board, thus passing coordinates as parameters, or if the board would keep track of Critter objects’ location and pass that information to the objects.

We decided to move forward with a variation of the second option. The Game class would have a function (getNeighbors) that returns a vector containing the objects surrounding a given location on the board. The vector from the getNeighbors function could then be passed to the Critter objects. The benefit of this was that we needed to have the Doodlebugs check if there was a neighboring Ant before it moved. Using the getNeighbors function allowed the Doodlebug class to have a bool type data member (hasNeighboringAnt) to check for an Ant before calling the overridden move function.

Another design change was how we were going to check to make sure that every pointer to a critter object on the board only moved once per round. Our solution was to include a bool type flag. This would allow the program to toggle the flag after each ant or doodle bug has moved.

Reflection:

This group project demonstrated the challenges that a team faces when presented with a programming problem. Figuring out how to break the larger problem down into more manageable problems, determining object data members, and moving forward with an overall design approach are considerations which we have all encountered through our previous projects. However, each group member has their own individual techniques to approach a problem, and we all have our own coding nuances. In order for our project to be successful communication was going to be fundamental.

We decided to start our communication with a shared Google document and a Google hangout. Jacob got the ball rolling with some project pseudocode, which got all of us engaged in thinking about how we wanted to approach the design. We moved forward with creating class skeletons for the Critter class, Ant class, Doodlebug class, Board class, and Menu. Each group member had an opportunity to add what data members and methods they thought were necessary for the classes. While these initial skeletons were fluid and meant to be changed (talked about below), it was very helpful for everyone to visualize the design approach and the skeletons provided insight into how other group members thought about the problem we were trying to solve. With the pseudocode and class skeletons in place, it was time to start coding. Initially, we had thought Amazon

Web Services' Cloud9 would be a user friendly option to interactively work on coding; however, we ultimately decided on GitHub, which Anthony and Jamil had some experience with.

Since we all had flexible schedules and were all in the same time zone, we decided to schedule 1-2 hour daily meetings via Google Hangouts. Each meeting usually focused on coding the header and implementation file for one or several classes and brainstorming for the next meeting. Anthony was the lead coder, while Jamil, Danielle, Jacob, and Charles contributed design changes and general programming input. We were also able to utilize Jacob's Menu for this project. All team members worked on different pieces of the reflection document and we worked as a team to review, make edits, and assemble it.

Overall, the difficulties of working in a group setting were limited due to how great the communication was throughout the two week period. The technique of coding on Google hangouts and then sharing code on Github was only feasible due to every group member being willing and able to meet regularly. Some take away lessons shared by group members were the importance of debugging, becoming more comfortable with vectors and pointers, and to always make sure memory is deallocated appropriately.