David Anderson, Zane Hendrickson, Matthew Kueper, Christopher Lundeberg, Oliver Solorzano
Group 3
Group Project, CS162
February 17, 2019

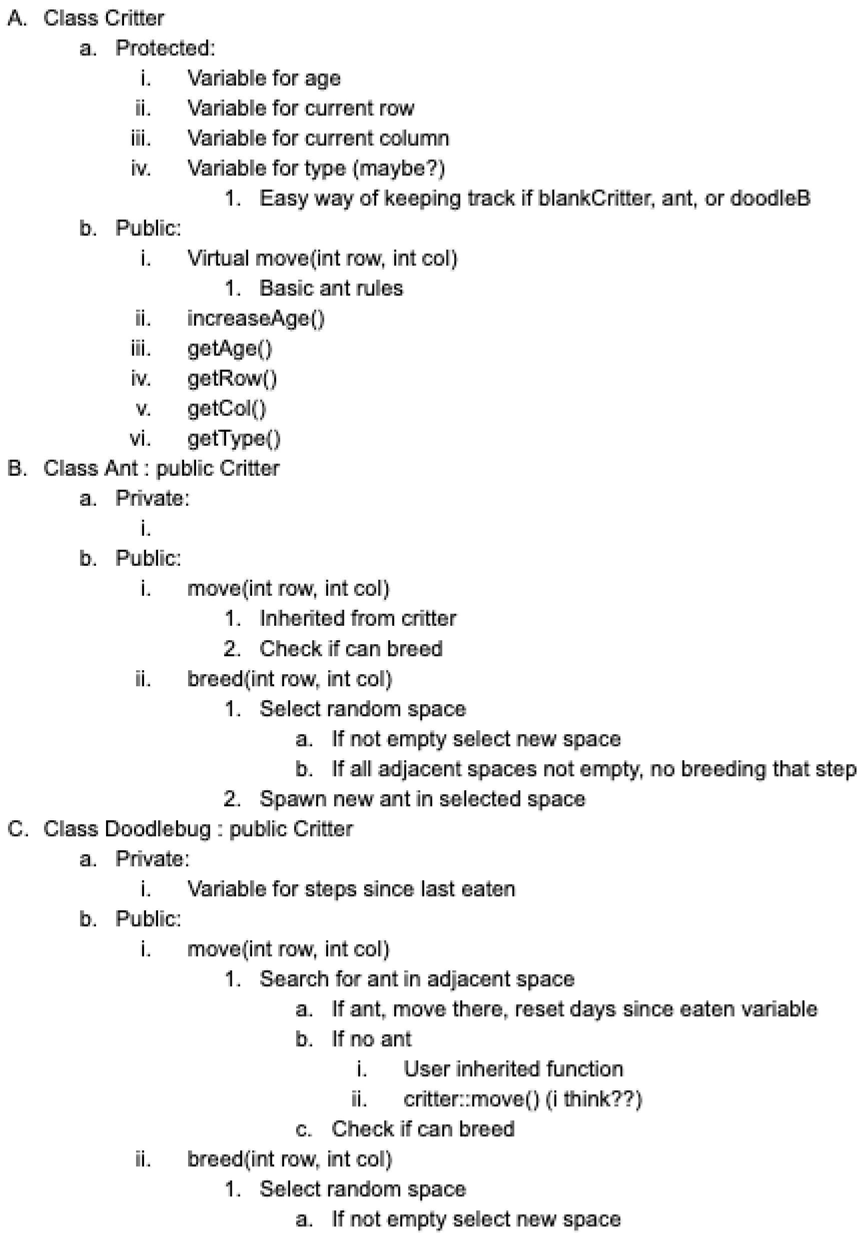
Group #3 Reflection

Problem to be Solved:

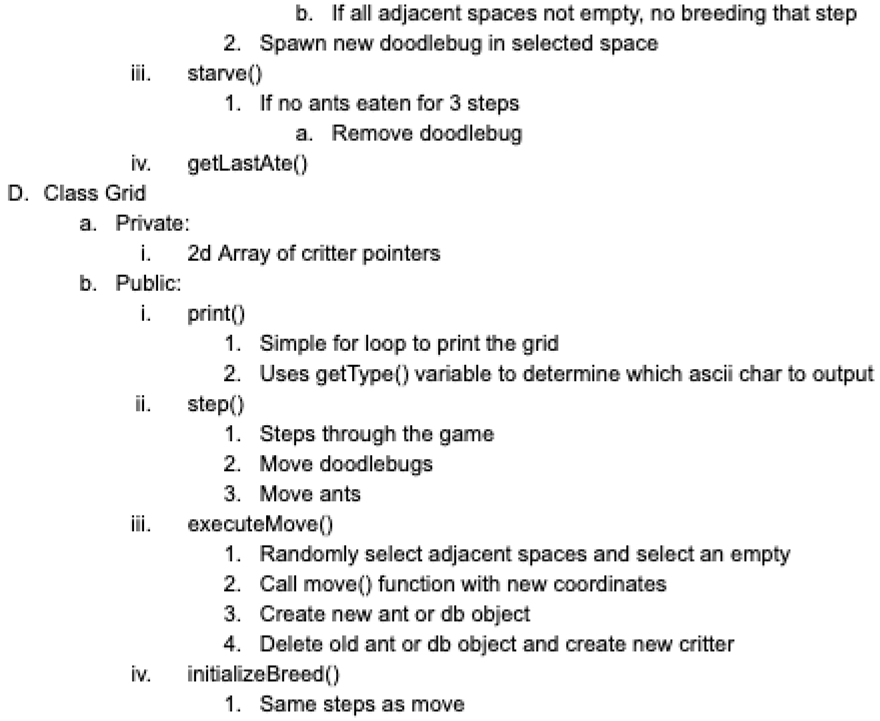
The primary problem that we were trying to solve for this assignment was how to create this board to print the Doodlebugs/Ants which were subclasses of the Critter class.

Original Design: This is our initial design that we planned on doing.

David Anderson, Zane Hendrickson, Matthew Kueper, Christopher Lundeberg, Oliver Solorzano
Group 3
Group Project, CS162
February 17, 2019



- A. Class Critter
  - a. Protected:
    - i. Variable for age
    - ii. Variable for current row
    - iii. Variable for current column
    - iv. Variable for type (maybe?)
      - 1. Easy way of keeping track if blankCritter, ant, or doodleB
  - b. Public:
    - i. Virtual move(int row, int col)
      - 1. Basic ant rules
    - ii. increaseAge()
    - iii. getAge()
    - iv. getRow()
    - v. getCol()
    - vi. getType()
- B. Class Ant : public Critter
  - a. Private:
    - i.
  - b. Public:
    - i. move(int row, int col)
      - 1. Inherited from critter
      - 2. Check if can breed
    - ii. breed(int row, int col)
      - 1. Select random space
        - a. If not empty select new space
        - b. If all adjacent spaces not empty, no breeding that step
      - 2. Spawn new ant in selected space
- C. Class Doodlebug : public Critter
  - a. Private:
    - i. Variable for steps since last eaten
  - b. Public:
    - i. move(int row, int col)
      - 1. Search for ant in adjacent space
        - a. If ant, move there, reset days since eaten variable
        - b. If no ant
          - i. User inherited function
          - ii. critter::move() (i think??)
        - c. Check if can breed
    - ii. breed(int row, int col)
      - 1. Select random space
        - a. If not empty select new space



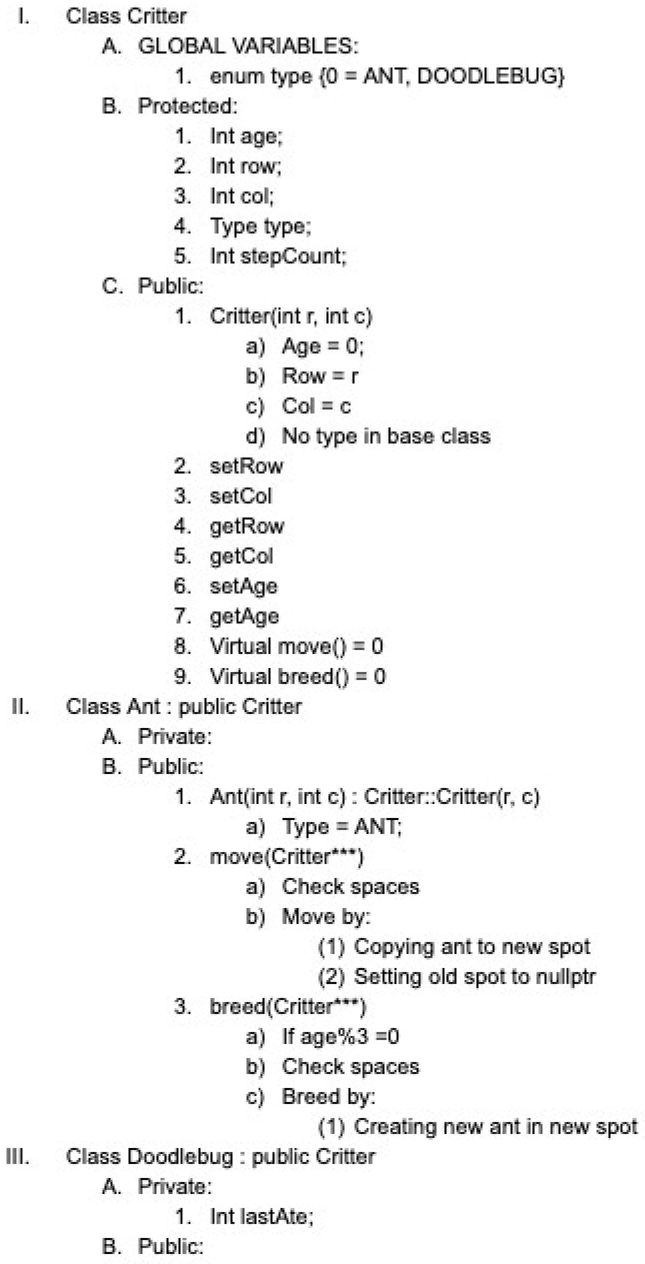
- b. If all adjacent spaces not empty, no breeding that step
- 2. Spawn new doodlebug in selected space
- iii. starve()
    - 1. If no ants eaten for 3 steps
        - a. Remove doodlebug
- iv. getLastAte()

- D. Class Grid
    - a. Private:
        - i. 2d Array of critter pointers
    - b. Public:
        - i. print()
            - 1. Simple for loop to print the grid
            - 2. Uses getType() variable to determine which ascii char to output
        - ii. step()
            - 1. Steps through the game
            - 2. Move doodlebugs
            - 3. Move ants
        - iii. executeMove()
            - 1. Randomly select adjacent spaces and select an empty
            - 2. Call move() function with new coordinates
            - 3. Create new ant or db object
            - 4. Delete old ant or db object and create new critter
        - iv. initializeBreed()
            - 1. Same steps as move

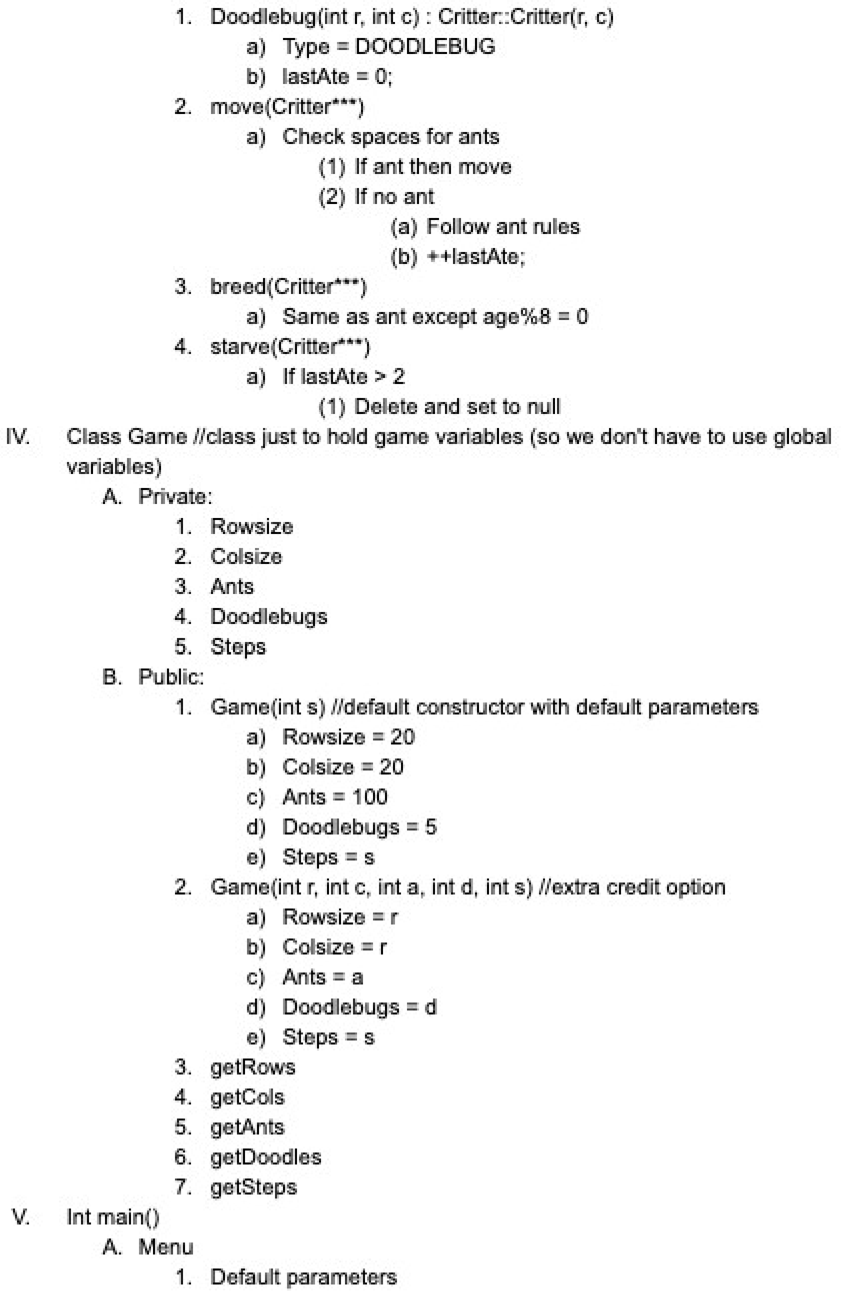
Implementation Issues:

We ran into implementation issues with the original design, so we had to pivot to another design shown below.

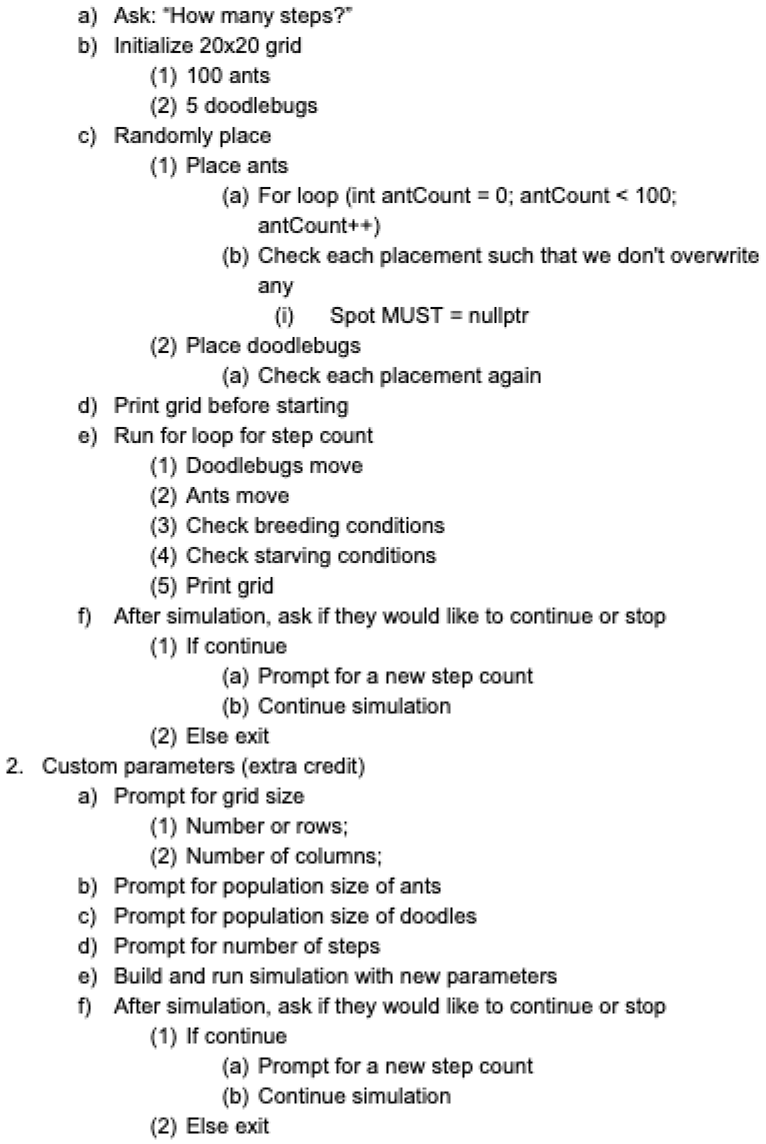
David Anderson, Zane Hendrickson, Matthew Kueper, Christopher Lundeberg, Oliver Solorzano
Group 3
Group Project, CS162
February 17, 2019



- I. Class Critter
  - A. GLOBAL VARIABLES:
    - 1. enum type {0 = ANT, DOODLEBUG}
  - B. Protected:
    - 1. Int age;
    - 2. Int row;
    - 3. Int col;
    - 4. Type type;
    - 5. Int stepCount;
  - C. Public:
    - 1. Critter(int r, int c)
      - a) Age = 0;
      - b) Row = r
      - c) Col = c
      - d) No type in base class
    - 2. setRow
    - 3. setCol
    - 4. getRow
    - 5. getCol
    - 6. setAge
    - 7. getAge
    - 8. Virtual move() = 0
    - 9. Virtual breed() = 0
- II. Class Ant : public Critter
  - A. Private:
  - B. Public:
    - 1. Ant(int r, int c) : Critter::Critter(r, c)
      - a) Type = ANT;
    - 2. move(Critter***)
      - a) Check spaces
      - b) Move by:
        - (1) Copying ant to new spot
        - (2) Setting old spot to nullptr
    - 3. breed(Critter***)
      - a) If age%3 =0
      - b) Check spaces
      - c) Breed by:
        - (1) Creating new ant in new spot
- III. Class Doodlebug : public Critter
  - A. Private:
    - 1. Int lastAte;
  - B. Public:



```
            1. Doodlebug(int r, int c) : Critter::Critter(r, c)
                a) Type = DOODLEBUG
                b) lastAte = 0;
            2. move(Critter***)
                a) Check spaces for ants
                    (1) If ant then move
                    (2) If no ant
                        (a) Follow ant rules
                        (b) ++lastAte;
            3. breed(Critter***)
                a) Same as ant except age%8 = 0
            4. starve(Critter***)
                a) If lastAte > 2
                    (1) Delete and set to null
IV.   Class Game //class just to hold game variables (so we don't have to use global
      variables)
        A. Private:
            1. Rowsize
            2. Colsize
            3. Ants
            4. Doodlebugs
            5. Steps
        B. Public:
            1. Game(int s) //default constructor with default parameters
                a) Rowsize = 20
                b) Colsize = 20
                c) Ants = 100
                d) Doodlebugs = 5
                e) Steps = s
            2. Game(int r, int c, int a, int d, int s) //extra credit option
                a) Rowsize = r
                b) Colsize = r
                c) Ants = a
                d) Doodlebugs = d
                e) Steps = s
            3. getRows
            4. getCols
            5. getAnts
            6. getDoodles
            7. getSteps
 V.   Int main()
        A. Menu
            1. Default parameters
```



- a) Ask: "How many steps?"
- b) Initialize 20x20 grid
    - (1) 100 ants
    - (2) 5 doodlebugs
- c) Randomly place
    - (1) Place ants
        - (a) For loop (int antCount = 0; antCount < 100; antCount++)
        - (b) Check each placement such that we don't overwrite any
            - (i) Spot MUST = nullptr
    - (2) Place doodlebugs
        - (a) Check each placement again
- d) Print grid before starting
- e) Run for loop for step count
    - (1) Doodlebugs move
    - (2) Ants move
    - (3) Check breeding conditions
    - (4) Check starving conditions
    - (5) Print grid
- f) After simulation, ask if they would like to continue or stop
    - (1) If continue
        - (a) Prompt for a new step count
        - (b) Continue simulation
    - (2) Else exit

2. Custom parameters (extra credit)
    - a) Prompt for grid size
        - (1) Number or rows;
        - (2) Number of columns;
    - b) Prompt for population size of ants
    - c) Prompt for population size of doodles
    - d) Prompt for number of steps
    - e) Build and run simulation with new parameters
    - f) After simulation, ask if they would like to continue or stop
        - (1) If continue
            - (a) Prompt for a new step count
            - (b) Continue simulation
        - (2) Else exit

Structure:

David Anderson, Zane Hendrickson, Matthew Kueper, Christopher Lundeberg, Oliver Solorzano
Group 3
Group Project, CS162
February 17, 2019

The structure of our program centers around our main.cpp file which contains the menu and other functions that utilize the rest of our files. The start function asks the user how many steps they want to have in the simulation. The next function is the customize game function which allows the user to select whether they would like to customize the board size and number of doodlebugs/ants. This function also gives the option for users to use the default settings. After that then the createGrid and populateGrid functions are called. These functions create the board, initialize it with critter pointers, and then fill it with randomly placed doodlebugs/ants based off of the customize function. The executeStep function is called until the number of steps specified has been completed. The executeStep function runs the various move, breed, and starve functions within the critter subclasses. Lastly the keepPlaying function is called to ask the user if they want to run the simulation again or if they want to quit.

Classes:

The Critter class contains the getStep(), getLastAte(), getType(), move(), breed(), starve(), and increaseAge() functions.

The Ant class contains the move(), breed(), and starve() functions.

The Doodlebug class contains the move(), breed(), and starve() functions.

The Game class contains many getters and setters that are required for the program to function.

Files:

main.cpp

David Anderson, Zane Hendrickson, Matthew Kueper, Christopher Lundeberg, Oliver Solorzano
Group 3
Group Project, CS162
February 17, 2019

Game.cpp

Game.hpp

Grid_functions.cpp

Grid_functions.hpp

Critter.cpp

Critter.hpp

Ant.cpp

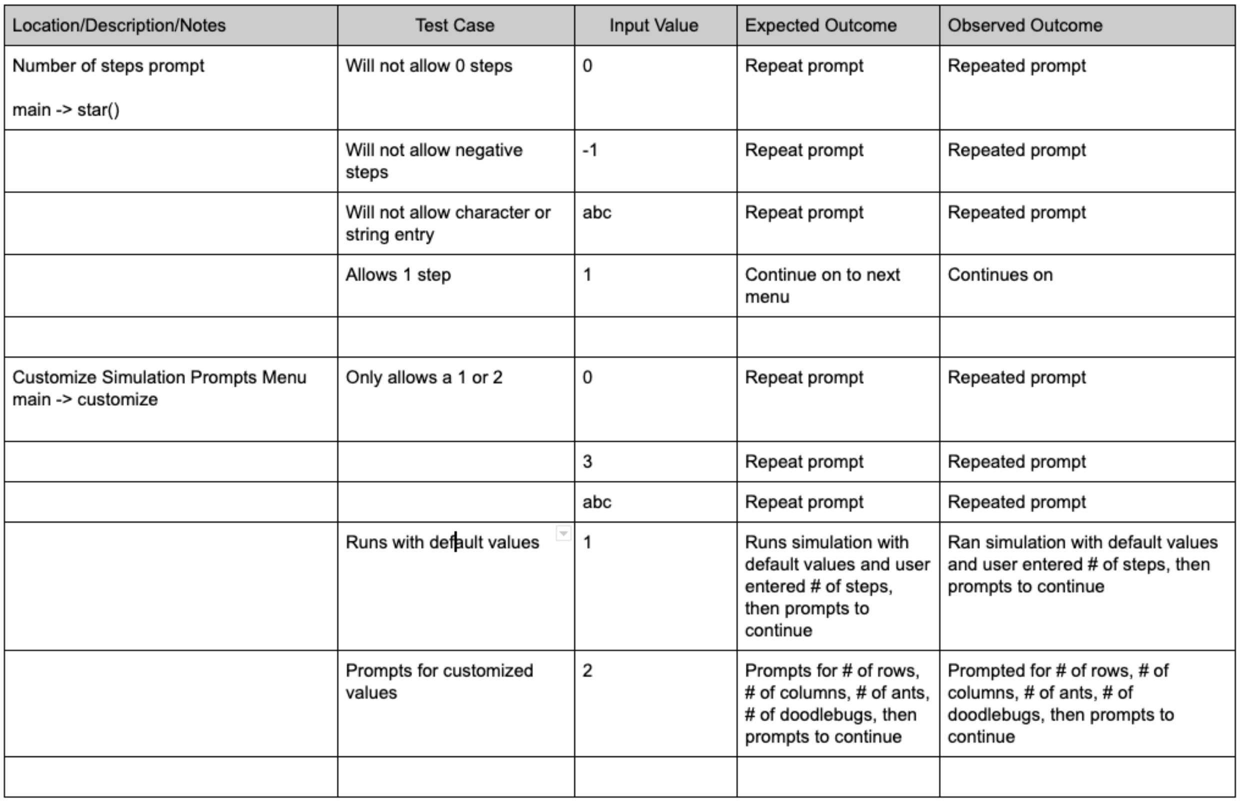
Ant.hpp

Doodlebug.cpp

Doodlebug.hpp

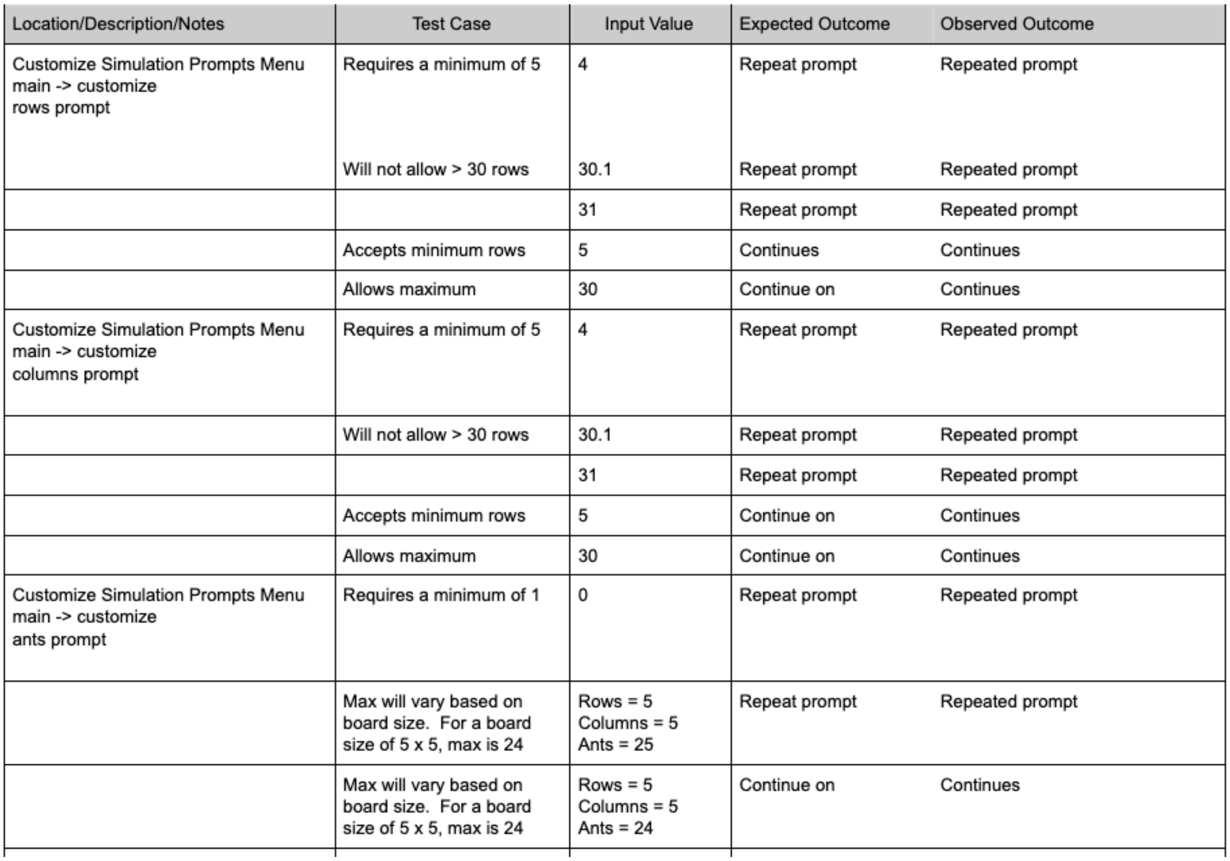
Test Plan:

David Anderson, Zane Hendrickson, Matthew Kueper, Christopher Lundeberg, Oliver Solorzano
Group 3
Group Project, CS162
February 17, 2019



Location/Description/Notes	Test Case	Input Value	Expected Outcome	Observed Outcome
Number of steps prompt main -> star()	Will not allow 0 steps	0	Repeat prompt	Repeated prompt
	Will not allow negative steps	-1	Repeat prompt	Repeated prompt
	Will not allow character or string entry	abc	Repeat prompt	Repeated prompt
	Allows 1 step	1	Continue on to next menu	Continues on
Customize Simulation Prompts Menu main -> customize	Only allows a 1 or 2	0	Repeat prompt	Repeated prompt
		3	Repeat prompt	Repeated prompt
		abc	Repeat prompt	Repeated prompt
	Runs with default values	1	Runs simulation with default values and user entered # of steps, then prompts to continue	Ran simulation with default values and user entered # of steps, then prompts to continue
	Prompts for customized values	2	Prompts for # of rows, # of columns, # of ants, # of doodlebugs, then prompts to continue	Prompted for # of rows, # of columns, # of ants, # of doodlebugs, then prompts to continue

David Anderson, Zane Hendrickson, Matthew Kueper, Christopher Lundeberg, Oliver Solorzano
Group 3
Group Project, CS162
February 17, 2019

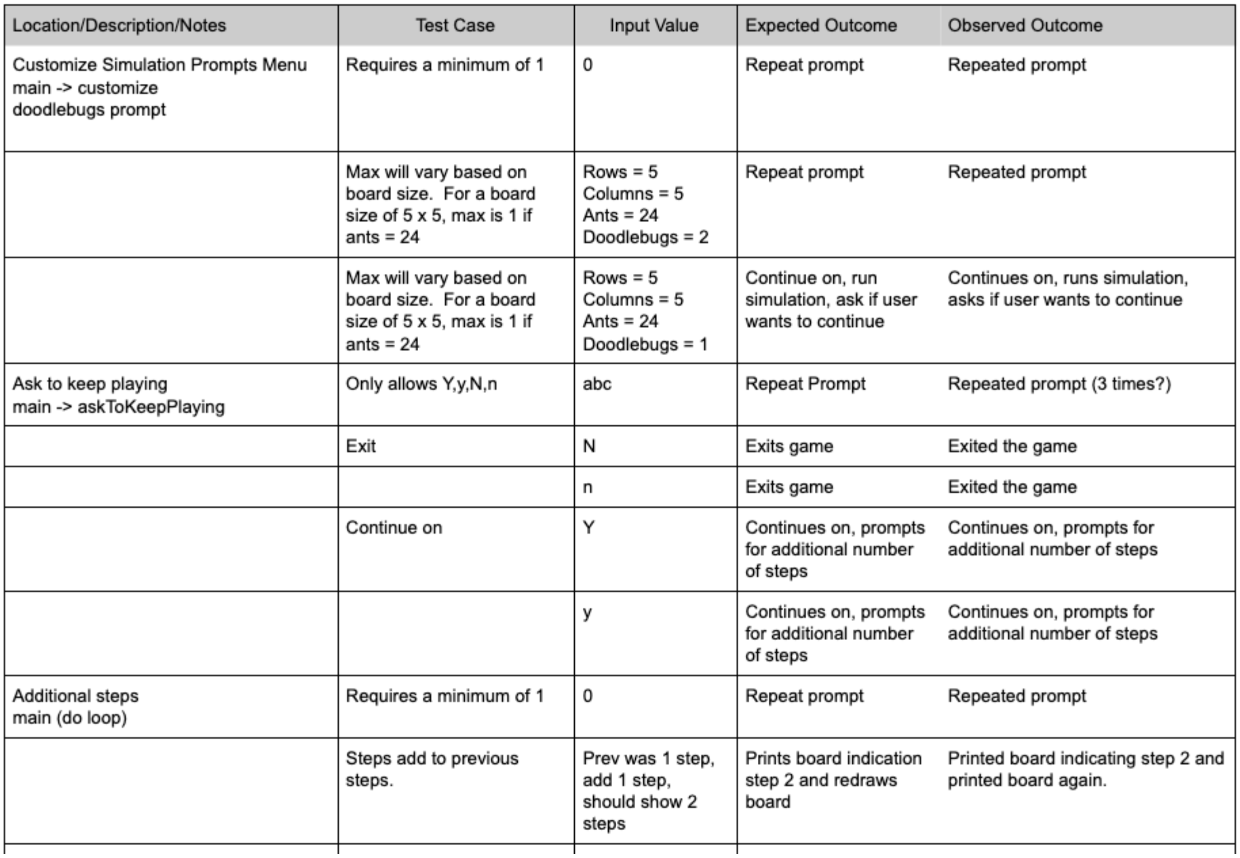


Location/Description/Notes	Test Case	Input Value	Expected Outcome	Observed Outcome
Customize Simulation Prompts Menu main -> customize rows prompt	Requires a minimum of 5	4	Repeat prompt	Repeated prompt
	Will not allow > 30 rows	30.1	Repeat prompt	Repeated prompt
		31	Repeat prompt	Repeated prompt
	Accepts minimum rows	5	Continues	Continues
	Allows maximum	30	Continue on	Continues
Customize Simulation Prompts Menu main -> customize columns prompt	Requires a minimum of 5	4	Repeat prompt	Repeated prompt
	Will not allow > 30 rows	30.1	Repeat prompt	Repeated prompt
		31	Repeat prompt	Repeated prompt
	Accepts minimum rows	5	Continue on	Continues
	Allows maximum	30	Continue on	Continues
Customize Simulation Prompts Menu main -> customize ants prompt	Requires a minimum of 1	0	Repeat prompt	Repeated prompt

	Max will vary based on board size. For a board size of 5 x 5, max is 24	Rows = 5
		Columns = 5
		Ants = 25
			Repeat prompt	Repeated prompt

	Max will vary based on board size. For a board size of 5 x 5, max is 24	Rows = 5
		Columns = 5
		Ants = 24
			Continue on	Continues

David Anderson, Zane Hendrickson, Matthew Kueper, Christopher Lundeberg, Oliver Solorzano
Group 3
Group Project, CS162
February 17, 2019



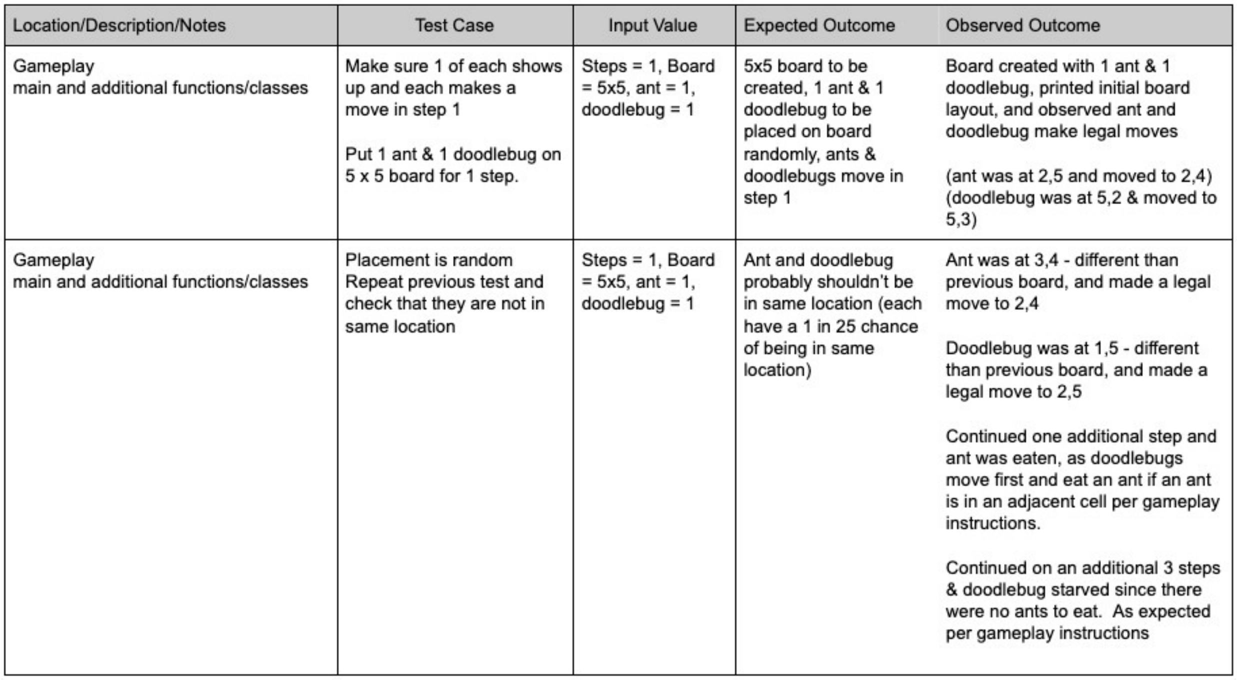
Location/Description/Notes	Test Case	Input Value	Expected Outcome	Observed Outcome
Customize Simulation Prompts Menu main -> customize doodlebugs prompt	Requires a minimum of 1	0	Repeat prompt	Repeated prompt

	Max will vary based on board size. For a board size of 5 x 5, max is 1 if ants = 24	Rows = 5
		Columns = 5
		Ants = 24
		Doodlebugs = 2
			Repeat prompt	Repeated prompt

	Max will vary based on board size. For a board size of 5 x 5, max is 1 if ants = 24	Rows = 5
		Columns = 5
		Ants = 24
		Doodlebugs = 1
			Continue on, run simulation, ask if user wants to continue	Continues on, runs simulation, asks if user wants to continue

Ask to keep playing main -> askToKeepPlaying	Only allows Y,y,N,n	abc	Repeat Prompt	Repeated prompt (3 times?)
	Exit	N	Exits game	Exited the game
		n	Exits game	Exited the game
	Continue on	Y	Continues on, prompts for additional number of steps	Continues on, prompts for additional number of steps
		y	Continues on, prompts for additional number of steps	Continues on, prompts for additional number of steps
Additional steps main (do loop)	Requires a minimum of 1	0	Repeat prompt	Repeated prompt
	Steps add to previous steps.	Prev was 1 step, add 1 step, should show 2 steps	Prints board indication step 2 and redraws board	Printed board indicating step 2 and printed board again.

David Anderson, Zane Hendrickson, Matthew Kueper, Christopher Lundeberg, Oliver Solorzano
Group 3
Group Project, CS162
February 17, 2019



Location/Description/Notes	Test Case	Input Value	Expected Outcome	Observed Outcome
Gameplay main and additional functions/classes	Make sure 1 of each shows up and each makes a move in step 1 Put 1 ant & 1 doodlebug on 5 x 5 board for 1 step.	Steps = 1, Board = 5x5, ant = 1, doodlebug = 1	5x5 board to be created, 1 ant & 1 doodlebug to be placed on board randomly, ants & doodlebugs move in step 1	Board created with 1 ant & 1 doodlebug, printed initial board layout, and observed ant and doodlebug make legal moves (ant was at 2,5 and moved to 2,4) (doodlebug was at 5,2 & moved to 5,3)
Gameplay main and additional functions/classes	Placement is random Repeat previous test and check that they are not in same location	Steps = 1, Board = 5x5, ant = 1, doodlebug = 1	Ant and doodlebug probably shouldn't be in same location (each have a 1 in 25 chance of being in same location)	Ant was at 3,4 - different than previous board, and made a legal move to 2,4 Doodlebug was at 1,5 - different than previous board, and made a legal move to 2,5 Continued one additional step and ant was eaten, as doodlebugs move first and eat an ant if an ant is in an adjacent cell per gameplay instructions. Continued on an additional 3 steps & doodlebug starved since there were no ants to eat. As expected per gameplay instructions

Reflection:

This group project was challenging because for many of us it was our first time collaborating on a coding project. Because of how intertwined all of the pieces were required to be for this project it was initially difficult for everyone to contribute. We had a zip file that was being passed around, but it meant only one person could work on it at a time, and it was slow-going. Eventually we switched over to using a version control system, which helped everyone contribute at the same time without overwriting other people's work. This change along with using slack for additional communication was very helpful. It allowed problems to get identified and resolved more quickly, and allowed people who are in different time zones and have different hours available to make sure they have the most current code version before making any changes. A huge benefit of collaborating, however, was the chance to see different coding styles and the approaches other programmers take when trying to solve a problem, and the ability to gain different perspectives.

Group Work Distribution:

David Anderson: Debugging, maintained gitlab/source files, bug fixes

Zane Hendrickson: Developed major portions of the program, constructed test plan

Matthew Kueper: Bug fixes, debugging, input validation, rand function

Christopher Lundeberg: Architect for program, developed major portions of the program

Oliver Solorzano: Developed major portions of the program