Group 20
Sandro Aguilar, Kuljot Biring, Ryan Gross, Jeesoo Ryoo, Rachel Schlick
CS 162-400-Winter 2019 – Introduction to Computer Science II
Professor Luyao Zhang
02/17/2019

Group Project Reflection

Predator-Prey Game

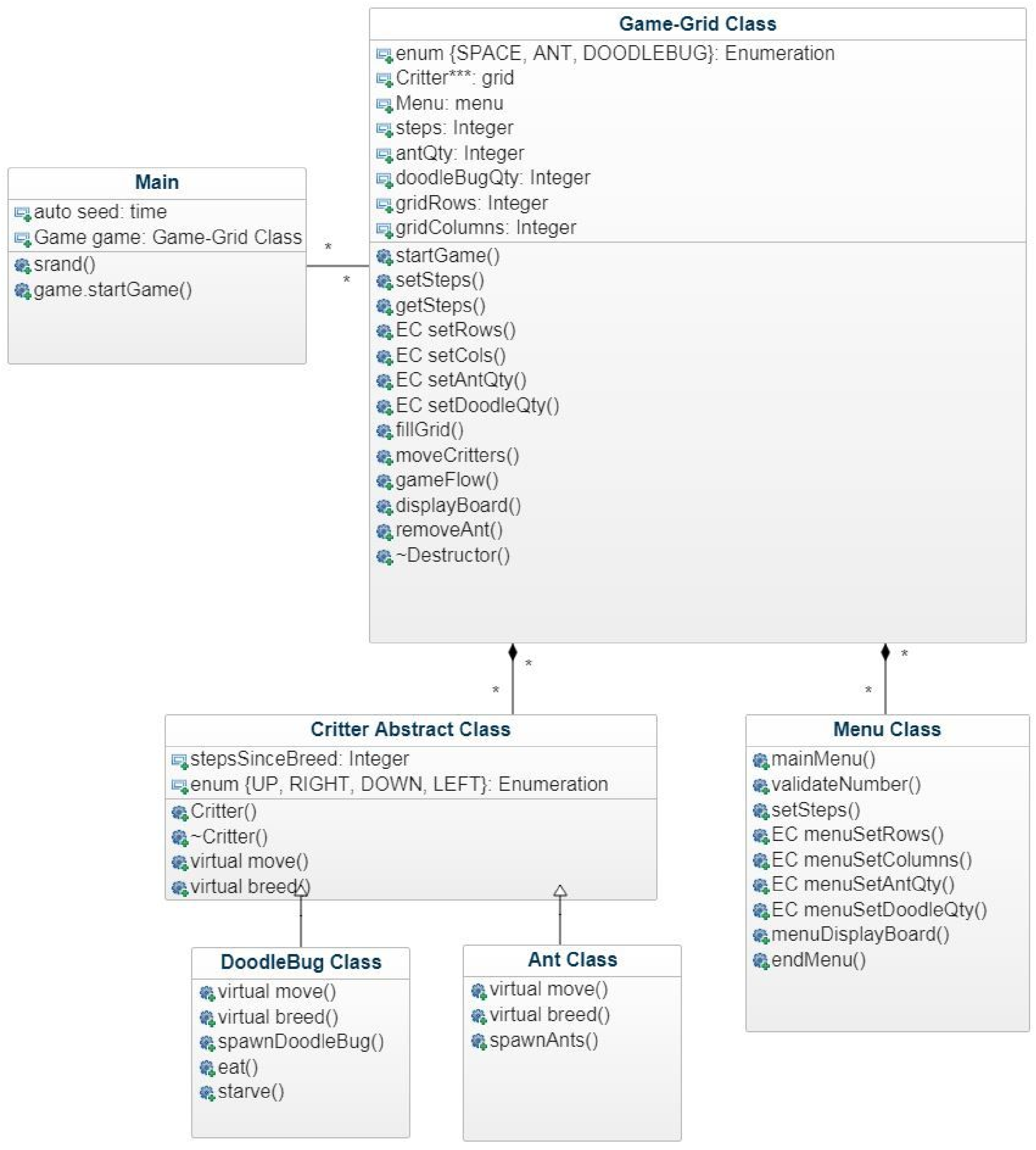
Predator-Prey Game: Project Plan Design

The design of the project was made early on in the process in order to establish the relationships between the classes in the program. This allowed us to debate the structure of the program and ways we can simplify the work involved. The team members held a meeting early on and prepared a chart to reflect classes, functions and program flow (see below).

The game class is where the simulation of the program is controlled. It dictates overall game parameters, the flow of the game, the movement of the ants, eating and breeding, and whether one wants to run the simulation again. The initial design we made shows that we used object composition in regards to the Critter class and the Menu class. This allows for easy access to objects of those classes within the Game class. The specifications require that the Critter class be an abstract class and so the class hierarchy shows that it contains some virtual functions that the derived classes Doodlebug and Ant inherit.

After the initial design of the program hierarchy, we held a discussion on Canvas and mostly on Slack to go over any questions or suggestions on how to modify the initial design. Included in the next page is the final initial design plan that we agreed on. We knew at that time that the initial design plan would most likely change as we encountered additional unexpected issues or design modifications during the implementation of the game, however, we knew that this designed covered the main operations of the program and we did not expect any significant changes to the program architecture.

Original Project Plan Chart - Planning Phase



Work Distribution

During our first phone conference, we discussed work distribution and tools that would be used for collaboration. Since our group is composed of five group members, collaborating on all aspects of the project was challenging and we needed to find a way to work together as best as possible.

We initially agreed to use a code repository tool that would allow all individuals access to the code base while allowing one to work on any features without affecting the master branch. In our case, we chose GitHub as our source control. In addition, we collaborated throughout the project period using a Slack channel and Google Hangouts teleconferences.

We then discussed how to partition the work. However, this part was rather difficult because at that time we were not sure how all the pieces of our project would fit together and we were only estimating who would work on what. So based on our rough estimate and design diagram, we decided to split the work based on the main functions required by the project.

- Overall program flow [Sandro]
- Menu Functions [Kuljot, Rachel, Sandro]
- Validation [Kuljot, Rachel]
- Move Functions [Jeesoo, Sandro]
- Eat Functions [Jeesoo]
- Breed Functions [Jeesoo, Sandro]
- Starve Functions [Jeesoo]
- Clean-up Ops [Rachel, Sandro, Jeesoo]
- Debugging [Rachel, Kuljot, Ryan, Sandro, Jeesoo]
- Commenting [Ryan, Rachel, Jeesoo]
- Write Up [Rachel, Kuljot, Ryan, Sandro, Jeesoo]

Design Implementation

While one could argue that the assignment mimicked a previous assignment, it became apparent that the design had to be deliberate and precise as our group decided to implement the optional features of the game as extra features needed to be added. Not only were critters manipulated on a gameboard, but doodlebugs had the ability to eat ant objects and remove them from play and also breed more of themselves as they ate ants (unless they starved). This, along with the additional input validation required for the extra credit, required the game design to be deliberate and concise.

The group utilized logic from a similar assignment where a 2D array is used as a row-and-column based game board. This allowed the design and the initialization of the board during runtime to be relatively intuitive. However, as per the specified technical requirements, said board needs to hold a pointer to a critter object. As a result, our group decided to implement a triple pointer; the board itself points to nested row and column pointers that contain pointers to these critter objects. The complexity of dynamically instantiating and initializing the board required our team to implement multiple conditional methods that iterated through these array pointers for checking if a critter object was at a certain “row and column” in the board, seeing what available options exist for a particular objects movement, and even verifying that a gamespace on the board is free. By setting these gamespaces to a null pointer by default, it made it easier to deal with the board. The team also had to account for memory deallocation within the board arrays, making the game susceptible to leaks.

In addition to the board, the input validation that we took on for extra credit required consideration as well. Instead of a predetermined size for it, our implementation allows the user to choose not only the length and width of the game board, but the specific amount of each type of critter that is to go in it. To do so, our group needed to validate that the amount of doodlebugs and ants that the user wants to put in the board doesn’t overflow it. The team not only had to check that the input given could be converted into an integer, but we also had to validate that the input given for the quantity of a critter did not exceed a certain range.

Changes in Design

In the Critter Class, we added 2 new variables (newRow, newCol) indicating the potential row and column that critter is going to move. In the original design, we did not have variables indicating the critter's potential location. This caused two main problems; first, we had to create a lot of duplicated code to check the type of the potential new cell, second, we had to pass new row and column information to all related function calls such as move, breed, eat and starve. After we changed the design, we could easily check adjacent cells, and manage move/spawn/eat functions.

We also changed the design of eat functionality of the Doodlebug Class. In the original design, we were passing the Ant object as a parameter to remove the eaten ant from the board. However, we encountered a problem with updating the eaten ant's cell on the grid to nullptr, thus memory leak. To resolve this issue, instead of passing an Ant object as a parameter, we passed the grid and used newRow and newCol variables to delete the Ant object and reset the cell to nullptr.

Challenges Faced in Code Implementation

In the implementation of our design we encountered a few notable problems that required further debugging and review. In the initial phases we had acknowledged that we were going to do the extra credit which allows for the user to choose the size of the board as well as the number of ants and doodlebugs present on the board. This required some scaling that needed to be performed as we realized that having a set up such as a smaller board with too many ants or doodlebugs would become problematic as the ants and bugs could be placed on top of each other creating major problems in the execution of the code. This issue would require scaling the total number of bugs based on the board size. To solve this issue we first set floors to the three parameters of; board row/col size, number of ants, and number of doodlebugs. And then implementing controls that would limit the min and max values allowed for the bugs the user could enter with robust input validation.

Another issue we encountered was with the spawning of doodlebugs. Originally, our code was encountering segmentation faults with respect to the breeding function since on occasion the doodlebugs were attempting to spawn outside of the array boundaries resulting in the segfaults.

Another significant issue we initially had was a memory leak we were encountering when doodlebugs were breeding. The new keyword was being used to instantiate new doodlebugs however our tests would show that a leak was occuring. It was identified that the delete keyword was not being used. However using the delete keyword was of no help in this situation because it would delete the doodlebug we had just created.

One minor issue we experienced was determining the most appropriate way to display changes to the board (i.e., ants bred, ants, eaten, doodlebugs bred, doodlebugs starved, total ants, and total doodlebugs). The initial implementation included a print statement for each location of each occurrence. This was helpful during debugging, but cumbersome for game play.

The initial validation function did not stop the game from proceeding when a user inputted bad values. An error message would display, but the game would go to the next step.

Resolving Issues

The memory leak was resolved after further review of how we were passing the board to the Doodlebug class breed function. We realized that passing a pointer to the game board was the correct thing to do i.e., ***grid however, this passes the pointer by value. Passing the pointer by value allows you to modify what the value in the pointer holds, yet it does not allow you to modify what the pointer actually points to. Therefore, we changed the way we passed the board to the function by passing it by reference instead i.e., ***&grid. This allowed us to create new doodlebugs while at the same time getting rid of the memory leak with the destructor in the game class.

We resolved the issue of tracking game changes by implementing a status display board each step. This required feeding data through various functions and changing return types.

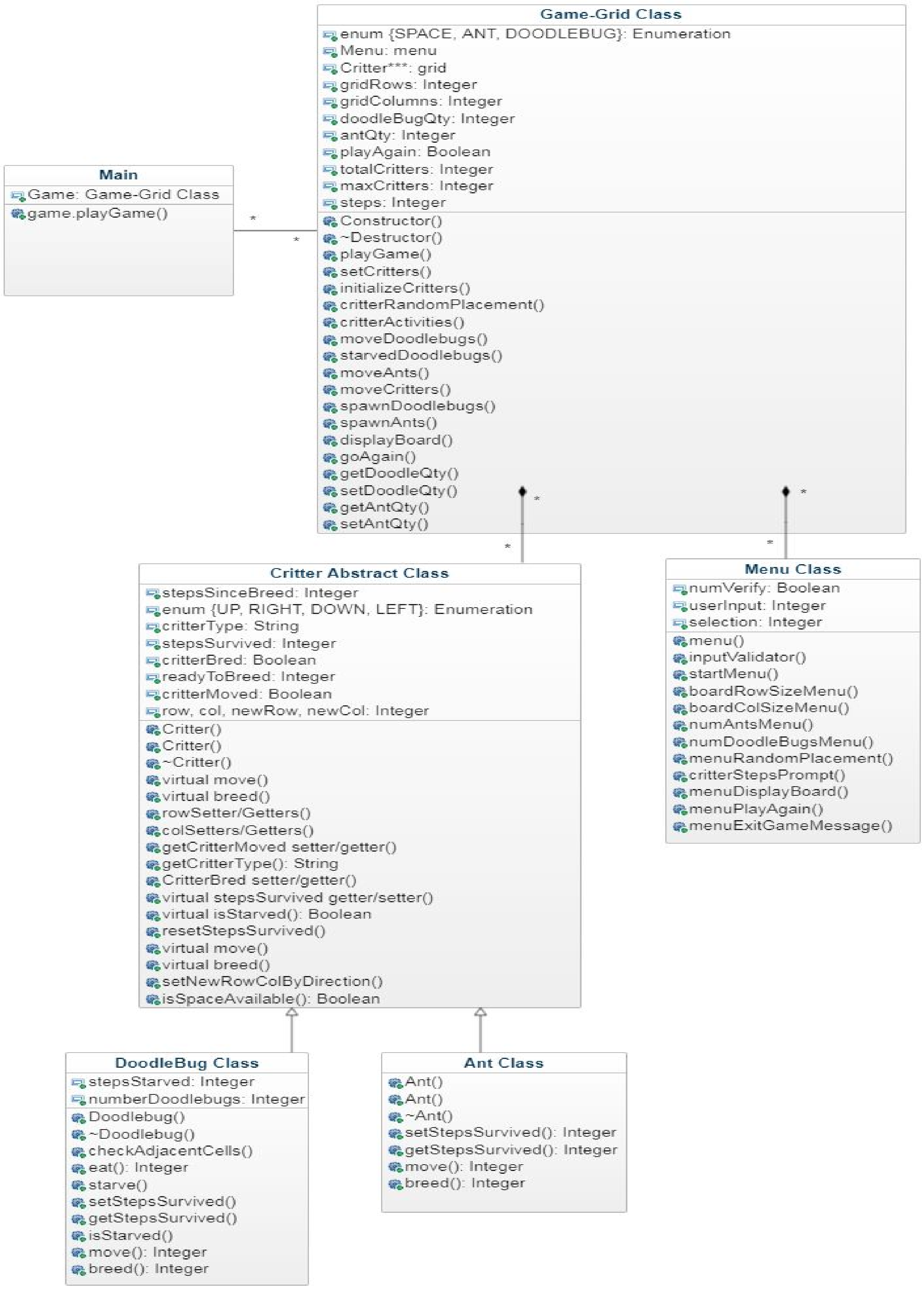
The validation function error appeared to be associated with the accepting no input as a valid entry. This was corrected by augmenting the validation function to check for that.

In the next page, the hierarchy chart shows how the design of our project evolved as well as how we resolved the challenges above. The chart shows additional data variables and functions that were added since as we began creating the program, we realized we needed additional ways to track information throughout the program as well as new methods in processing the information.

Design aside, this was the first collaborative assignment that we have done in this program, which was somewhat daunting. Rather than doing something such as forwarding emails of the assignment back-and-forth or delegating the code to be written by a specific team member. The group decided to add another layer of complexity to the assignment and added source control via Git. Although it’s outside of the course material, we immediately knew that the benefits of it exceeded any challenges learning it. By having the project up on a private repository, the project became more flexible; features could be delegated to a specific person and said person could work on it at any time by pulling down a copy of the latest code, implementing their work, and then submitting a request for it to be merged. Removing the need to work on a “master” copy of the codebase prevented one feature from breaking another, which is useful when a half dozen people are simultaneously working on the same project. We felt that staying in constant communication, reviewing each other’s code regularly, and planning phases of development in increments helped us achieve great results in an organized manner.

With this methodology we were able to divide the work and ensure that each member’s contributions didn’t alter anybody else’s. This, combined with designing a simplistic approach toward object manipulation, allowed our group to succeed as a whole.

Final Hierarchy Chart



Predator-Prey Game: Test Table

Test Case(s)	Input Values	Driver Function(s)	Expected Outcome(s)	Observed Outcome(s)
Invalid entry: character, string, string with space, out of range	“One” “G” “This one” 0 400	Main.cpp Game::playGame Menu::startMenu Menu::inputValidator	Prompt to re-enter option.	Prompted to re-enter in each case.
Enter Game	1	Main.cpp Game::playGame Menu::startMenu Menu::inputValidator	Prompt to enter board size	Prompted to enter board size
Exit Game	2	Main.cpp Game::playGame Menu::startMenu Menu::inputValidator Menu::menuExitGameMessage	Exit message and game exits	Exit message displayed and game exited
Board and initial critter count	Board size: 5 Ants: 1 Doodlebugs:1	Game::initializeCritters Game::displayBoard	5x5 board, 1 ant, 1 doodlebug displays	5x5 board, 1 ant, 1 doodlebug displayed
Board and initial critter placement	Board size: 10 Ants: 1 Doodlebugs: 1	Game::initializeCritters Game::displayBoard Game::critterRandomPlacement	2 similar runs have the same dimensions, but different ant and doodlebug locations	1st: 10x10 board, ant at (0,10), doodlebug at (1,4) 2nd: 10x10 board, ant at (6,9), doodlebug at (1,0)
Critter Selection doesn’t exceed number of board spaces	Board size: 10 (10x10 = 100 spaces)	Game::setCritters	Prompt user to select from 1 to 99 ants (must be at least 1 doodlebug)	Prompted to select from 1 to 99 ants
Critter Selection doesn’t exceed	Board size: 10 (10x10 = 100	Game::setCritters	Prompt user to select 1 doodlebug	Prompted to select from 1 to 1

number of board spaces	spaces) Ants: 99			doodlebugs
Number of steps	10	Menu::critterStepsPrompt	Prompt user for number of steps; iterate through those steps	As expected. 10 steps.
Status Menu - displays current step	N/A	Menu::menuDisplayBoard	Current displayed correctly.	As expected.
Status Menu - displays total Ants	N/A	Menu::menuDisplayBoard Game	Number of ants depicted on board will match count in status display	As expected
Status Menu - displays total Doodlebugs	N/A	Menu::menuDisplayBoard	Number of doodlebugs depicted on board will match count in status display	As expected
Status Menu - Number Ants Bred	N/A	Menu::menuDisplayBoard Ant::breed	Number will match with cout statement in breed ants function	As expected
Status Menu - Number Doodlebugs Bred	N/A	Menu::menuDisplayBoard Doodlebug::breed	Number will match with cout statement in breed doodlebugs function	As expected
Status Menu - Number Doodlebugs Starved	N/A	Menu::menuDisplayBoar Doodlebug::starved	Number will match with cout statement in starve doodlebugs function	As expected
Board - test frame maximum	100	Game::initializeCritters	100x100 board with frame should appear	As expected
Board - test frame minimum	5	Game::initializeCritters	5x5 board with frame should appear	As expected

Continue game correctly	1	Game::goAgain	Game should continue from previous board display	As expected
Exit game correctly	2	Game::goAgain	Game should exit	Game exited
Memory Leaks	N/A	Game::~Game Doodlebug::eat Doodlebug::breed Ant::breed	There should be no memory leaks	No memory leaks were detected