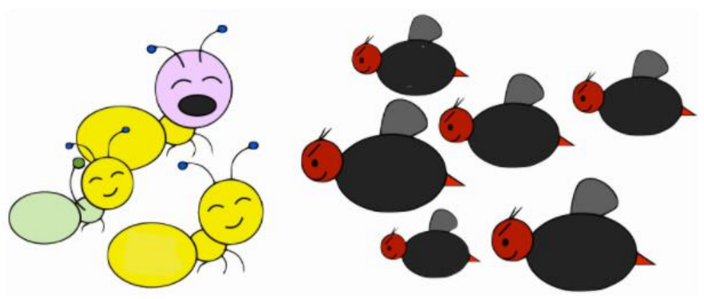
CS 162 Assignment #4 Ants vs. Some Bees

Design Document due: Sunday, 5/15/2022, 11:59 p.m. (Canvas)

Assignment due: Sunday, 5/22/2022, 11:59 p.m. (TEACH)

Demo due: 6/3/2022 (without penalties)



The bees are coming!

Create a better soldier

With inherit-ants.

Goals:

- Practice good software engineering design principles:
  - Design your solution before writing the program
  - Develop test cases before writing the program
- Practice Object-Oriented Programming
- Implement "Is-a" relationship using Inheritance
- Implement polymorphism and practice STL template by using vector
- Practice file separation and create Makefile
- Use functions to modularize code to increase readability and reduce repeated code

Motivation

The goal of this assignment is to start working with polymorphism and C++ template classes from the STL (standard template library). Follow the directions below and be sure to submit a TAR file containing only your .h, .cpp, and Makefile

Introduction

Ants vs. Some Bees, based on the game Plants vs. Zombies, is a tower defense game. The player controls the ant colony and must defeat the bees. If the bees get to the queen, the bees win the game. If all bees are defeated from the game, then the ants win.

Game Setup:

The game is played on a one-dimensional board of size 10. The queen ant resides on the left side of the board while the bees are generated on the right side from the beehive (See the picture below). The ant colony will begin with a food bank of 20. **Multiple insects (ants/bees) may reside on a square of the board.**

1	2	3	4	5	6	7	8	9	10
Queen Ant									Bee Hive

Game Elements:

Bee details:

Bees are the insect adversaries in this game. They have an armor of 3 when they are generated. They try to move one square, right to left, on the board each turn. **The bee cannot move to the left if the square <u>the bee is on</u> is occupied with an ant.** The ant on that square must be killed before the bee can move. Multiple bees may occupy a square. If multiple bees occupy a square, and the first bee kills the ant, then all other bees on the square (not including the bee who killed the ant) may proceed forward during that turn. When a bee attacks an ant, it inflicts 1 damage.

Ant details:

There are multiple kinds of ants. All ants have an initial food cost that is incurred when they are placed on the board. All ants have armor, but the armor value varies per ant. Once ants are placed, they are fixed to that location while they are alive. A new ant may be generated and placed each turn if the food bank permits. An ant may be placed on any square including those occupied by a bee, **but you cannot place an ant on Queen or Beehive square (i.e., square 1 or 10).** An ant may not share a square with another ant unless the other ant is a bodyguard. The following table outlines the different ants:

Ant	**Food Cost**	**Armor**	**Action**
Harvester	2	1	Adds one food to the colony each turn
Fire	4	1	Upon the death of this ant (when armor is 0 or less), it kills all bees on the same square
Thrower	3	1	Inflicts 1 damage on the bee which is closest to it
Snow Thrower	3 (Or 5 to make the game more playable)	1	Inflicts 1 damage on the bee which is closest to it and freeze the bee for 2 turns
HungryAnt	4	1	Eats one whole bee from its square, and spends 2 turns digesting before eating again
Wall	4	4	Occupies a square to block the bees. Cannot attack bees
Ninja	6	1	Cannot block bees and bees cannot attack the ninja. Bees pass by the ninja and incur 1 damage
Bodyguard	4	2	The only ant which may occupy a square which has an ant on it. The bodyguard will be the ant to incur the damage from any bee which also occupies the square.

			When the bodyguard ant dies, the next bee attack will hit the other ant on the square. There can be at most one bodyguard on a square

Game Play:

Five things happen each turn:

1. A bee is generated on the right side of the board, i.e., at square 10.
2. The player can generate an ant and place it on the board. Note: The player can choose not to generate an ant if they don't want to or have insufficient food.
3. The ants attack the bees. (Order of ant attacks occur left to right).
4. The bees either attack an ant (order of attack is left to right) which is blocking them or pass through to the next square on the board if they are not blocked by an ant.
5. Check to see if the bees have reached the queen (square 1) or if there are any bees left in play, declare a winner and end the game if either condition is true. Note: ants win by killing all the bees on the board, including the newly generated one from step 1 in the same turn.

The game continues so long as there are bees on the board at the end of the turn and they have not reached the queen (square 1).

Program Requirements:

- Your program should allow the user to play Ants vs. Some Bees, as described above.
- The user interface of your program must be readable and understandable
- Your code must have the following classes: `Ant`, `Bee`, `Square`, `Game`, `Harvester`, `Thrower`, `Snow_Thrower`, `Wall`, `HungryAnt`, `Ninja`, `BodyGuard`, and `Fire`. You may create more classes if they would be helpful.
- The `Ant` class **must be abstract** (i.e. it must contain purely virtual functions), and the `Harvester`, `Thrower`, `Snow_Thrower`, `Wall`, `HungryAnt`, `Ninja`, `BodyGuard`, and `Fire` classes should all be derived from `Ant`.
- **You must use the `Ant` classes polymorphically**.
- The game board, ants/bees on a square should be implemented using the `std::vector` class.
- Your program may not have any memory leaks. I strongly recommend that you use valgrind to occasionally test your program as you develop it (even when everything seems to be working).
- The Big 3 must be implemented as appropriate.

- Your program must be factored into interface, implementation, and application. Specifically, you should have one header file and one implementation file for each class, and you should have a single application file containing your `main()` function. You should also include a Makefile that specifies compilation for your program.
- Lack of correct coding style will incur an automatic 10-point deduction. You must follow the spirit of the assignment.

Hints

- Polymorphism only works when you are working with references or pointers. If you use the value of an object directly, it may not select the correct member function.

Extra Credit

In addition to the requirements above, you may earn extra credit as follows:

- (10 points) Expand the board from 1D of size 10 to 2D of size n x 10 (n rows of size 10 each, n is an integer greater than 1), where n is determined by the programmer. Each row has a queen on the left side and a beehive on the right side. One bee is generated from a randomly selected beehive each turn. The bees win by defeating one of the queens whereas the ants win by defeating all bees.
- (10 points) Design at least 3 different kinds of bees that are inherited from the `Bee` class and use them polymorphically. The `Bee` class must be an abstract class.

Implementation Details:

As with the previous program, you have additional freedom to design your own implementation. There are certain baseline requirements as specified in this document. However, the exact member functions and member variables are up to you.

Be sure to design your program on paper. Use scratch paper with flow charts or pseudo-code. In particular, think of how your program will flow. What will the constructors do? How will you pass objects and variables between different classes? Taking the time to do a thorough program design can save you hours of frustrating debugging time!

On the topic of debuggers, I strongly recommend that you choose a debugger and get familiar with its operation (GDB, Visual Studio, etc). As your programs get more complex it becomes even more useful to instantly know the value of all your variables and memory contents.

Programming Style/Comments

In your implementation, make sure that you include a program header. Also ensure that you use proper indentation/spacing and include comments! Below is an example header to include. Make sure you review the style guidelines for this class, and begin trying to follow them, i.e. don't align everything on the left or put everything on one line!

```
/******************************************************
** Program: ants_vs_bees.cpp
** Author: Your Name
** Date: 5/10/2022
** Description:
** Input:
** Output:
******************************************************/
```

Design Document – Due Sunday 5/15/2022, 11:59pm on Canvas
Refer to the Example Design Document – Example_Design_Doc.pdf

Understanding the Problem/Problem Analysis:

- What are the required classes, member variables for each class, etc.?
- What assumptions are you making?
- What are all the tasks and subtasks in this problem? What is the game flow?

Program Design:
At this point, I highly recommend you create a design using both flowcharts and pseudocode!!!

- What does the overall big picture of the problem? (Flowcharts)
  - What functions will you create for each class?
  - What functions would be virtual?
  - How would you modularize the program?
  - How do they interact with each other?
  - How will you apply inheritance? What member variables will be shared?
  - What class would be abstract class?
  - What class needs big 3?
- What each function looks like? (Pseudocode)
  - What is the task of this function?
  - What are the steps to get the task done?
- What kind of bad input are you going to handle?

Based on your answers above, list the **specific steps or provide a flowchart** of what is needed to create. Be very explicit!!!

Program Testing:
Create a test plan with the test cases (bad, good, and edge cases). What do you hope to be the expected results?

- What are the good, bad, and edge cases for ALL input in the program? Make sure to provide enough of each and for all different inputs you get from the user.

Electronically submit your Design Doc (.pdf file!!!) by the design due date, on Canvas.

Program Code – Due Sunday, 5/22/2022, 11:59pm on TEACH

Additional Implementation Requirements:

- Your user interface must provide clear instructions for the user and information about the data being presented
- Your program must catch all required errors and recover from them.
- You are not allowed to use libraries that are not introduced in class, more specifically, you may not use the <algorithm> library in your program. Any searching or sorting functionality must be implemented "manually" in your implementation.
- Your program should be properly decomposed into tasks and subtasks using functions. To help you with this, use the following:
    - Make each function do one thing and one thing only.
    - No more than 15 lines inside the curly braces of any function, including main(). Whitespace, variable declarations, single curly braces, vertical spacing, comments, and function headers do not count.
    - Functions over 15 lines need justification in comments.
    - Do not put multiple statements into one line.
- No global variables allowed (those declared outside of many or any other function, global constants are allowed).
- No goto function allowed
- You must not have any memory leaks
- You program should not have any runtime error, e.g. segmentation fault
- Make sure you follow the style guidelines, have a program header and function headers with appropriate comments, and be consistent.

Compile and Submit your assignment

When you compile your code, it is acceptable to use C++11 functionality in your program. In order to support this, change your `Makefile` to include the proper flag.
For example, consider the following approach (note the inclusion of **`-std=c++11`**):

```
g++ -std=c++11 <other flags and parameters>
```

In order to submit your homework assignment, you must create a **zip file** that contains your `.h, .cpp,` and `Makefile` files. This zip file will be submitted to TEACH . In order to create the zip file, use the following command:

```
zip assign4.zip <list of all .h and .cpp files> makefile
```

Remember to sign up with a TA to demo your assignment. The deadline of demoing this assignment without penalties is 6/3/2022.