CMSC 202 Spring 2023
Project 4 – UMBC Bloons

Assignment: Project 4 – UMBC Bloons

Due Date: Thursday, April 20th at 8:59pm

Value: 80 points

1. Overview

In this project, you will:

- Practice creating classes with inheritance,
- Working with pointers as they pertain to classes,
- Using polymorphism to change the behavior of the classes, and
- Working with a large number of classes.

2. Background

Bloons Tower Defense (also known as Bloons TD or BTD) is a series of tower defense games under the Bloons series created and produced by Ninja Kiwi. The game was initially developed as a browser game, built upon the (now unusable) Adobe Flash platform and released in mid-2007. Later games in the series expanded to support various mobile platforms, including Android, iOS, Windows Phone, PlayStation Portable, Nintendo DSi, Windows, Linux and MacOS. Games in the Bloons series older than Bloons TD 6 are available through the Ninja Kiwi Archive on Steam.

The main objective of Bloons TD is to prevent Bloons (in-game name for balloons) from reaching the end of a defined track on a map that consists of an entrance and exit for the bloons. The game is a tower defense game and thus the player can choose various types of towers to place around the track to defend against the bloons, gaining 1 in-game dollar for every layer of bloon popped. If a bloon reaches the end of a path, the player loses lives. Once these lives are all depleted, the game ends. The bloons always follow the path on the track until they either reach the exit or are popped.

As this project is focused on inheritance and polymorphism, there is an inheritance chain as defined in the figure below.

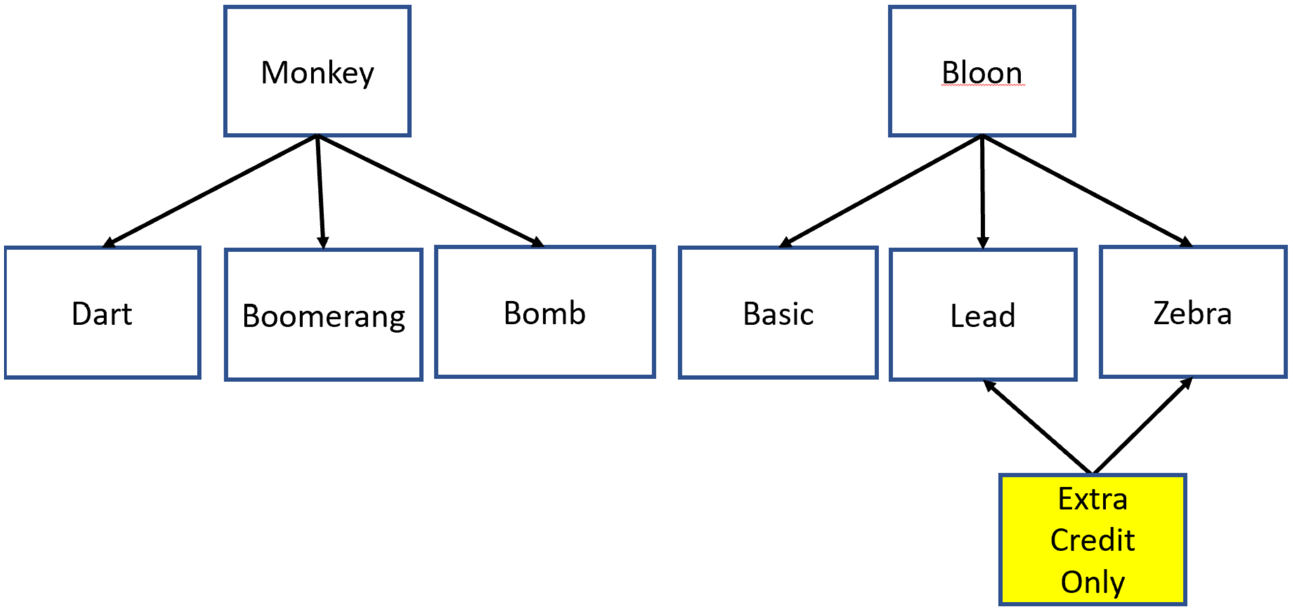

Figure 1. Inheritance Hierarchy for UMBC Bloons.

Here is a diagram showing an example of a path with three spots. While each path could have a completely indeterminate length (minimum 2 – no maximum), this example has a size of 3.

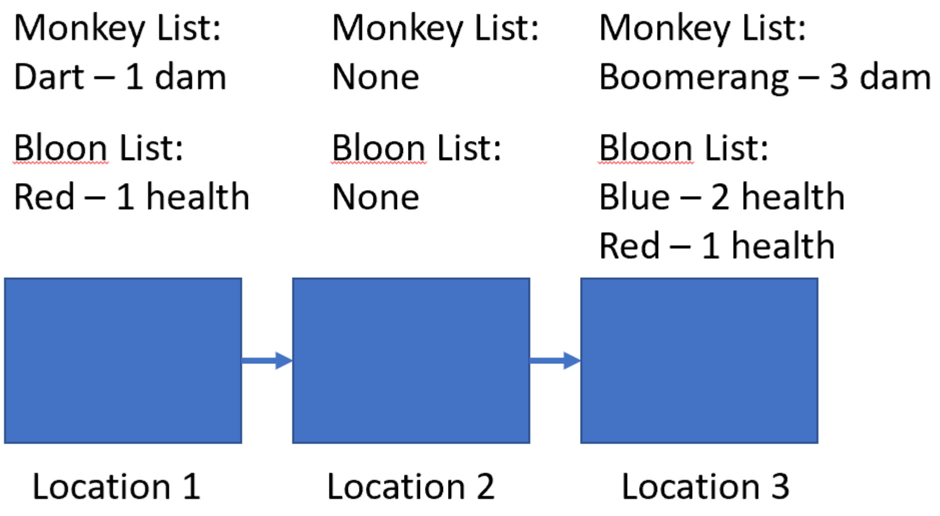

Figure 2. Example of Path with Monkeys and Bloons.

Bloons always move from the starting location down the path towards the maximum location (defined by **`PATH_LENGTH`**). For example, if **`PATH_LENGTH`** was 3 then the bloons would start in location -1 then move when the game starts. Each round they would move exactly 1 space closer to the end (**`PATH_LENGTH`**). When the bloon reaches the end of the path, it hurts the player (reduces **`m_curLife`**) until **`m_curLife`** <= 0 when the player loses.

3. Requirements:

This is a list of the requirements of this application. For this project, it is up to you exactly how you want to implement it. For you to earn all the points, however, you will need to meet all the defined requirements.

- You must follow the coding standard as defined in the CMSC 202 coding standard (found on Blackboard under course materials). This includes comments as required.
- The project must be turned in on time by the deadline listed above.
- The project must be completed in C++. You may not use any libraries or data structures that we have not learned in class. Libraries we have learned include **`<iostream>, <fstream>, <iomanip>, <cmath>, <cstdlib>, <vector>, <ctime.h>`**, and **`<string>`**. You should only use **`namespace std`**.
- You will need to the iterator function **`begin()`** in the **`vector`** library.
- Using the provided files, **`Bloon.h`**, **`Basic.h, Monkey.h, Dart.h, Boomerang.h, Bomb.h, Game.h`** and **`proj4.cpp`**, create a text-based Bloons TD game. You can copy the files from my directory in **`/afs/umbc.edu/users/j/d/jdixon/pub/cs202/proj4`**. To copy them, navigate to your project 4 folder and type:

  ```
  cp /afs/umbc.edu/users/j/d/jdixon/pub/cs202/proj4/* .
  ```
- You may **NOT** modify the headers files.
- There are seven required classes in this project: Game, Bloon, Basic, Monkey, Dart, Boomerang, and Bomb. All functions listed in their class files must be implemented completely (even if you do not use them).
- All entities (and their subclasses) and areas must be dynamically allocated (and deallocated).
- All user input must be validated. For example, if a menu allows for 1, 2, or 3 to be entered and the user enters a 4, it will re-prompt the user. However, the user is expected to always enter the correct data type. i.e. If the user is asked to enter an integer, they will. If they are asked to

enter a character, they will. You do not need to worry about checking for correct data types.

- Recommendation for order to code everything: Start with the **`bloon`** classes (**`Bloon.cpp and Basic.cpp`**) because they need to be passed to monkey for the attack. Then work on the base class **`Monkey.h/.cpp`** and then add it to your **`makefile`**. Compile just **`Monkey.o`** first! Then work your way down to the child (**`Dart.cpp, Boomerang.cpp,`** and **`Bomb.cpp`**) and compile just one at a time. Once that works, start on **`Game.cpp`**. Start at the top of the main menu. Remember, incrementally code your classes, and compile them as you go.

3.1. Class Requirements

- Monkey – Use **`Monkey.h`** to write the **`Monkey.cpp`** file for this class. Monkeys are the base class for the three children: dart, boomerang, and bomb. They have three base characteristics: damage, type, and location. Damage is how many health the bloon is reduced for each attack. Type is the string equivalent for the subtype (so, dart has a type of dart). Location is where on the path the monkey exists.
- Children of Monkey (Dart, Boomerang, and Bomb) – Dart monkeys are the weakest and cheapest of the monkeys. They attack the first bloon in their location only. Boomerang monkeys are more expensive and hit every bloon in their location twice (once when the boomerang goes out and once when it comes back). Bomb monkeys launch a bomb and every bloon in their location is damaged. These monkeys only damage bloons that have a health greater than 0.
- Bloons – Use **`Bloon.h`** to write the **`Bloon.cpp`** file for this class. The parent class for all bloons, a bloon is the game's version of a balloon. A bloon always starts at position 0 of the path. They will either be popped by a monkey and removed from the path or they will progress to the end of the path where they will damage the player. For the base project, there is only one child class required, basic.
- Children of Bloon (Basic (Extra credit – Lead, and Zebra)) – Basic bloons are simple bloons that have a color. A bloons color is directly related to how much health it has. So, a bloon with one health, is red. A bloon with two health is blue, three is green, four is yellow, five is pink, and anything greater than or equal to six is black. If a green bloon is hit

for two damage then it becomes red. If a black bloon with a health of eight is hit for two damage, it remains black. If a bloon has a health of less than one, it is destroyed.

See Extra Credit below for additional information about the Lead and Zebra bloons.

- Game – Use **`Game.h`** to write the **`Game.cpp`** file for this class. This is the longest and most complicated of the classes as it manages the entire game. Once Bloon (and Basic) and Monkeys (and Dart, Boomerang, and Bomb) have been programmed, you should start on Game.cpp.

  First, stub out the entire **`Game.cpp`** file. Then implement the constructor (which is simple). Then you can code **`StartGame`** and **`MainMenu`**. For now, just make the main menu display the user's choice. You can then code PrintMap which initially will print each location with monkeys and bloons with "none." Test each function as you go. You can then progress down the rest of the main menu. The last set of functions will be related to PlayRound. This function is really three functions(PopulateBloons, ResolveBattle, and MoveBloons). These should be coded last.

  One of the more complicated things is removing bloons from m_bloons that have a health of less than 1. You need to deallocate (delete) them and erase them from the vector. Vectors have a function called erase that can remove something from the middle of a vector but it uses an iterator. The syntax is: **`m_bloons.erase(m_bloons.begin() + counter)`**

4. Information about Erase in Vectors

We can erase information from a vector using the function **`erase`**. However, when we erase something from a vector, the vector will reduce the container size by one. As such, if you are using the erase function in a for loop then it will cause problems.

Because vectors use an array as their underlying storage, erasing elements in positions other than the vector end causes the container to relocate all the elements after the segment erased to their new positions. It does not necessarily do this in order.

The erase function requires that we use an iterator such as **`begin()`** to indicate where the start of the vector exists. As we continue to erase items from the vector, it leaves gaps in the structure which the compiler automatically fills. For this reason, when you are erasing items, it may make the items out of order. Normal syntax will be something like **`m_bloons.erase(m_bloons.begin() + counter).`**

This is inefficient; however, we will learn later the more efficient way to do this. For now, though, there are two functions that need to be in a loop – **`RemovePopped()`** and **`CheckPath()`**. Just make these functions run up to 5 times each and that will ensure that the effect has propagated throughout the vector.

5. Sample Input and Output

For this project, there are no input files.

All starting files can be downloaded from Prof. Dixon's data folder by navigating to your project 4 folder and typing the following command:

```
cp /afs/umbc.edu/users/j/d/jdixon/pub/cs202/proj4/* .
```

In the sample output below, user input is colored blue for clarity. After compiling and running proj4, the output would look like this:

```
Welcome to UMBC Bloons!
What would you like to do?
1. View Map
2. Buy New Monkey
3. Improve Existing Monkey
4. Progress Round
5. Check Stats
6. Quit
1

**Location 1**
--Monkeys--
```

```
None
<<Bloons>>
None

**Location 2**
--Monkeys--
None
<<Bloons>>
None

**Location 3**
--Monkeys--
None
<<Bloons>>
None
What would you like to do?
1. View Map
2. Buy New Monkey
3. Improve Existing Monkey
4. Progress Round
5. Check Stats
6. Quit
6
Thanks for playing!
```

Now, we can buy a monkey and check out stats.

```
Welcome to UMBC Bloons!
What would you like to do?
1. View Map
```

```
2. Buy New Monkey
3. Improve Existing Monkey
4. Progress Round
5. Check Stats
6. Quit
2
What type of monkey would you like to buy?
1. Dart Monkey
2. Boomerang Monkey
3. Bomb Monkey
4. Cancel
2
Where would you like to place the new monkey? (1-3)
2
New Boomerang monkey placed in location 2
What would you like to do?
1. View Map
2. Buy New Monkey
3. Improve Existing Monkey
4. Progress Round
5. Check Stats
6. Quit
5
**CURRENT STATS**
Current Round: 1
Monkeys Working: 1
Current Money: 6
Current Life: 100
```

```
What would you like to do?
1. View Map
2. Buy New Monkey
3. Improve Existing Monkey
4. Progress Round
5. Check Stats
6. Quit
6
Thanks for playing!
```

Here is an example of some input validation.

```
Welcome to UMBC Bloons!
What would you like to do?
1. View Map
2. Buy New Monkey
3. Improve Existing Monkey
4. Progress Round
5. Check Stats
6. Quit
7
0
What would you like to do?
1. View Map
2. Buy New Monkey
3. Improve Existing Monkey
4. Progress Round
5. Check Stats
6. Quit
2
```

```
What type of monkey would you like to buy?
1. Dart Monkey
2. Boomerang Monkey
3. Bomb Monkey
4. Cancel
0
What type of monkey would you like to buy?
1. Dart Monkey
2. Boomerang Monkey
3. Bomb Monkey
4. Cancel
2
Where would you like to place the new monkey? (1-3)
4
Where would you like to place the new monkey? (1-3)
2
New Boomerang monkey placed in location 2
What would you like to do?
1. View Map
2. Buy New Monkey
3. Improve Existing Monkey
4. Progress Round
5. Check Stats
6. Quit
7
What would you like to do?
1. View Map
2. Buy New Monkey
```

```
3. Improve Existing Monkey
4. Progress Round
5. Check Stats
6. Quit
6
Thanks for playing!
```

Here is a longer run where you can see a couple of rounds of progress.

```
Welcome to UMBC Bloons!
What would you like to do?
1. View Map
2. Buy New Monkey
3. Improve Existing Monkey
4. Progress Round
5. Check Stats
6. Quit
1

**Location 1**
--Monkeys--
None
<<Bloons>>
None

**Location 2**
--Monkeys--
None
<<Bloons>>
None
```

```
**Location 3**
--Monkeys--
None
<<Bloons>>
None
What would you like to do?
1. View Map
2. Buy New Monkey
3. Improve Existing Monkey
4. Progress Round
5. Check Stats
6. Quit
2
What type of monkey would you like to buy?
1. Dart Monkey
2. Boomerang Monkey
3. Bomb Monkey
4. Cancel
2
Where would you like to place the new monkey? (1-3)
2
New Boomerang monkey placed in location 2
What would you like to do?
1. View Map
2. Buy New Monkey
3. Improve Existing Monkey
4. Progress Round
```

```
5. Check Stats
6. Quit
1

**Location 1**
--Monkeys--
None
<<Bloons>>
None

**Location 2**
--Monkeys--
1. Boomerang Monkey: Location 2 Damage: 1
<<Bloons>>
None

**Location 3**
--Monkeys--
None
<<Bloons>>
None
What would you like to do?
1. View Map
2. Buy New Monkey
3. Improve Existing Monkey
4. Progress Round
5. Check Stats
6. Quit
```

```
3
Which monkey would you like to improve (1-1)
1. Boomerang Monkey: Location 2 Damage: 1
1
Boomerang in position 1 improved!
What would you like to do?
1. View Map
2. Buy New Monkey
3. Improve Existing Monkey
4. Progress Round
5. Check Stats
6. Quit
4
Starting Round 1
A new red bloon appears!
The bloons move along the path
Round 1 Completed
What would you like to do?
1. View Map
2. Buy New Monkey
3. Improve Existing Monkey
4. Progress Round
5. Check Stats
6. Quit
4
Starting Round 2
A new red bloon appears!
A new blue bloon appears!
```

```
The bloons move along the path
Round 2 Completed
What would you like to do?
1. View Map
2. Buy New Monkey
3. Improve Existing Monkey
4. Progress Round
5. Check Stats
6. Quit
1

**Location 1**
--Monkeys--
None
<<Bloons>>
Bloon: red Health: 1
Bloon: blue Health: 2

**Location 2**
--Monkeys--
1. Boomerang Monkey: Location 2 Damage: 3
<<Bloons>>
Bloon: red Health: 1

**Location 3**
--Monkeys--
None
<<Bloons>>
```

```
None
What would you like to do?
1. View Map
2. Buy New Monkey
3. Improve Existing Monkey
4. Progress Round
5. Check Stats
6. Quit
4
Starting Round 3
A new red bloon appears!
A new blue bloon appears!
A new green bloon appears!
The boomerang monkey throws a boomerang!
The red bloon pops!
The bloon is now gone!
The bloons move along the path
Round 3 Completed
What would you like to do?
1. View Map
2. Buy New Monkey
3. Improve Existing Monkey
4. Progress Round
5. Check Stats
6. Quit
1

**Location 1**
```

```
--Monkeys--
None
<<Bloons>>
Bloon: red Health: 1
Bloon: blue Health: 2
Bloon: green Health: 3

**Location 2**
--Monkeys--
1. Boomerang Monkey: Location 2 Damage: 3
<<Bloons>>
Bloon: red Health: 1
Bloon: blue Health: 2

**Location 3**
--Monkeys--
None
<<Bloons>>
None
What would you like to do?
1. View Map
2. Buy New Monkey
3. Improve Existing Monkey
4. Progress Round
5. Check Stats
6. Quit
4
Starting Round 4
```

```
A new red bloon appears!
A new blue bloon appears!
A new green bloon appears!
A new yellow bloon appears!
The boomerang monkey throws a boomerang!
The red bloon pops!
The bloon is now gone!
The boomerang monkey throws a boomerang!
The blue bloon pops!
The bloon is now gone!
The bloons move along the path
Round 4 Completed
What would you like to do?
1. View Map
2. Buy New Monkey
3. Improve Existing Monkey
4. Progress Round
5. Check Stats
6. Quit
1

**Location 1**
--Monkeys--
None
<<Bloons>>
Bloon: red Health: 1
Bloon: blue Health: 2
Bloon: green Health: 3
```

```
Bloon: yellow Health: 4

**Location 2**
--Monkeys--
1. Boomerang Monkey: Location 2 Damage: 3
<<Bloons>>
Bloon: red Health: 1
Bloon: blue Health: 2
Bloon: green Health: 3

**Location 3**
--Monkeys--
None
<<Bloons>>
None
What would you like to do?
1. View Map
2. Buy New Monkey
3. Improve Existing Monkey
4. Progress Round
5. Check Stats
6. Quit
4
Starting Round 5
A new red bloon appears!
A new blue bloon appears!
A new green bloon appears!
A new yellow bloon appears!
```

```
A new pink bloon appears!
The boomerang monkey throws a boomerang!
The red bloon pops!
The bloon is now gone!
The boomerang monkey throws a boomerang!
The blue bloon pops!
The bloon is now gone!
The boomerang monkey throws a boomerang!
The green bloon pops!
The bloon is now gone!
The bloons move along the path
Round 5 Completed
What would you like to do?
1. View Map
2. Buy New Monkey
3. Improve Existing Monkey
4. Progress Round
5. Check Stats
6. Quit
5
**CURRENT STATS**
Current Round: 6
Monkeys Working: 1
Current Money: 14
Current Life: 100
What would you like to do?
1. View Map
2. Buy New Monkey
```

```
3. Improve Existing Monkey
4. Progress Round
5. Check Stats
6. Quit
6
Thanks for playing!
```

An additional **`proj4_sample1.txt`** was included to see a longer run.

6. Compiling and Running

You need to write a **`makefile`** for this project as one will not be provided for you in order to compile and test. Make sure to create any macros to help!

Once you have compiled using your **`makefile`**, enter the command **`make run`** or **`./proj4`** to run your program. Make sure you have implemented that as part of your make file. If your executable is not proj4, you will lose points. It should look similar to the sample output provided above. Some of the simulation is based on randomness so the output will vary.

Because we are using a significant amount of dynamic memory for this project, you are required to manage any memory leaks that might be created. Remember, in general, for each item that is dynamically created, it should be deleted using a destructor.

One way to test to make sure that you have successfully removed any of the memory leaks is to use the **`valgrind`** command.

Since this project makes extensive use of dynamic memory, it is important that you test your program for memory leaks using **`valgrind`**:

```
valgrind ./proj4
```

Note: If you accidently use **`valgrind make run1`**, you may end up with some memory that is still reachable. Do not test this – test using the command above where you include the input file. The makefile should include **`make val1`** (which is ok).

If you have no memory leaks, you should see output like the following:

```
==5606==
==5606== HEAP SUMMARY:
==5606==     in use at exit: 0 bytes in 0 blocks
==5606==   total heap usage: 87 allocs, 87 frees, 10,684 bytes
allocated
==5606==
==5606== All heap blocks were freed -- no leaks are possible
==5606==
==5606== For counts of detected and suppressed errors, rerun
with: -v
==5606== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 6
from 6)
```

The important part is “in use at exit: 0 bytes 0 blocks,” which tells me all the dynamic memory was deleted before the program exited. If you see anything other than "0 bytes 0 blocks" there is probably an error in one of your destructors. We will evaluate this as part of the grading for this project.

Additional information on `valgrind` can be found here: http://valgrind.org/docs/manual/quick-start.html

Once you have compiled using the provided `makefile`, enter the commands `make run` or `./proj4` to run your program. If your executable is not proj4, you will lose points. It should look like the sample output provided above.

7. Completing your Project

When you have completed your project, you can copy it into the submission folder. You can copy your files into the submission folder as many times as you like (before the due date). We will only grade what is in your submission folder.

For this project, you should submit **everything but the makefile and sample files** (no `makefile or proj4_sample1.txt`) files to the `proj4` subdirectory. As you should have already set up your symbolic link for this class, you can just copy your files listed above to the submission folder.

a. cd to your project folder. An example might be cd `~/202/projects/proj4`

b.
```
cp Bloon.h Bloon.cpp Basic.h Basic.cpp
Monkey.h Monkey.cpp Dart.h Dart.cpp
Boomerang.h Boomerang.cpp Bomb.h Bomb.cpp
Game.h Game.cpp proj4.cpp ~/cs202proj/proj4
```

You can check to make sure that your files were successfully copied over to the submission directory by entering the command

```
ls ~/cs202proj/proj4
```

You can check that your program compiles and runs in the `proj4` directory, but please clean up any `.o` and executable files. Again, do not develop your code in this directory and you should not have the only copy of your program here. Uploading of any `.gch` files will result in a severe penalty.

For additional information about project submissions, there is a more complete document available in Blackboard under “Course Materials” and “Project Submission.”

IMPORTANT: If you want to submit the project late (after the due date), you will need to copy your files to the appropriate late folder. If you can no longer copy the files into the proj4 folder, it is because the due date has passed. You should be able to see your proj4 files but you can no longer edit or copy the files in to your proj4 folder. (They will be read only)

- If it is 0-24 hours late, copy your files to `~/cs202proj/proj4-late1`
- If it is 24-48 hours late, copy your files to `~/cs202proj/proj4-late2`
- If it is after 48 hours late, it is too late to be submitted.

8. Extra Credit (Optional – up to 16pts)

If you have completed the base project and you have additional time, you can optionally complete a few additional features for extra credits.

****STOP**** - Turn in your base project first. You must turn in two sets of files. One for the base project (worth 80 points) in the normal submission directory. Second for the extra credit (worth up to 16 points) in a new folder called `extra_credit` in your normal submission directory.

DO NOT TURN IN YOUR EXTRA CREDIT AS YOUR BASE PROJECT. YOU WILL GET A ZERO FOR BOTH.

You can only attempt this after you have completed the base project and turned the base project and the extra credit in on time (no late submission for either). Additionally, office hours cannot be used to help you complete the extra credit – you can only ask clarifying questions.

To earn the extra credit, you must add the functionality to the base project to add two additional bloon children classes named, "lead" and "zebra" bloons. To complete this, you will need to add two new classes that are children of the Bloon class. You will create **`Lead.h, Lead.cpp, Zebra.h, and Zebra.cpp`**. Additionally, you will need to edit Game (you can edit both **`Game.h`** and **`Game.cpp`** if necessary). Each successful class implementation will be worth up to 8 points if it works.

In our version, lead bloons can only be hurt initially by bomb monkeys. Once they are damaged by a bomb monkey, they turn into a basic bloon (black with health of 10). Zebra bloons split into two basic bloons (black with health of 10 each). Add some logic to PopulateBloons that adds a lead bloon after round 10 and a zebra bloon after round 12.

You cannot turn in the extra credit after the original due date. You cannot earn extra credit if you do not turn in the base project.

To turn in the extra credit, you should create a new folder in your submission folder named "**`extra_credit`**". Then copy your extra credit files to that directory

```
mkdir ~/cs202proj/proj4/extra_credit
```

Copy all of your files to the extra_credit directory.

MAKE SURE THAT YOU HAVE SUBMITTED TWO VERSION: BASE VERSION in **`~/cs202proj/proj4`** AND THE EXTRA CREDIT in **`~/cs202proj/proj4/extra_credit`**