Bryan Arnold

Assignment 6

CSE 2100

11/13/16

Description: This project was to simulate the scenario of the three goats and three trolls game. The end goal is to show the trend of each colored goat's numbers, as well as the total coins of each troll at its bridge over time. My program is designed by using the HeapAdaptablePriorityQueue class from the book, then implement this method in terms of the scenario. My program creates objects for goats and trolls, assigns the goats and trolls values and assignments to each other, then executes the simulation of the goats crossing bridges until 10000 time units. Other classes were implemented from the code from the book to help utilizes the code from the book but centered around goats (not used due to incompletion).

Tradeoffs: I considered whether to fully follow the skeleton structure put up on piazza, or to use my own version of the project. I found most of the structure extremely helpful, but a few methods I found unnecessary which helped me cut down on code. Another tradeoff I thought about was how to fully implement the HeapAdaptablePriorityQueue from the book, or use a series of ArrayLists() instead. The code from the book is already presented and no code writing was needed, but the concepts on how to use them were complicated. On the other hand, using ArrayLists() instead of the queues would be much easier to understand, but the code required to write it was very complicated. I initially went for trying to understand the queue code, which after weeks of trying to grasp got no success (due to multiple errors). So, I ended up trying to do it with only ArrayLists(), but the code was pretty complex and I couldn't get it all done in time, which is why this project is incomplete. I shifted between the two ideas, neither being successful. I definitely could've gotten the ArrayList() only to work, if I had swapped to them sooner. Lastly, I thought of whether include GoatPriority class in the final implementation attempt of my project. This class was only really useful in the queue implementation attempt, but could have some uses in the ArrayLiss() attempt. I determined only one method was useful (update method), and it could be put into the Goat, so I did just that.

Extensions: The obvious improvements to my program are a working program, either ArrayList() or queue based. Having a working simulation would accurately find how this scenario would play out and would be a fully functioning program. Another improvement would be figuring out how to utilize the queue classes. If these were used correctly, the amount of code would be very small compared to the ArrayList() version, and would work, and would teach me the concepts of the queue and heap. To achieve these, I should've sat down and collaborated more with other students and sat down and did this without other distractions. The skeleton structure and piazza were helpful, but understanding the concepts of what to do were more important to know.

Test Cases: For my goats' payment plans, the grey goats had their own plan and the black and white goats had their own shared plan (this is all in the strategy method in the goat class). For the grey goats, they only want to pay when they must. This means they don't have to pay at their own colored bridge, the grey bridge, but must pay at the other two bridges. So, this can simply be planned that they pay only at the white and black troll bridges and that's it. As for white and black goats, their plan is more complicated. The goal of these goats is to make their populations as big as possible, so they want to go around as fast as possible. Since they have no choice on whether to pay at other colored bridges, they simply must pay at the bridges that aren't their color. At their own bridge is another story. If a black or white goat is at its own bridge, and no other goats of other colors are ahead of it in the queue, it won't pay to gain priority. This is because there is no need to pay for priority if you won't gain any priority by doing so. A goat will only pay priority when goats of other color are ahead of it, or when other goats of the same color are ahead of it. This is because it doesn't want to waste money if it doesn't have to. Lastly, these goats won't pay for priority if their coin counts would drop below 1, as they would die. So, they wait in the queue in hopes to get past the bridge. Since my simulation was incomplete, I can't show the test of cases/solution graphs of the project. I can although, theorize through reasoning of what the numbers should looks like. The grey troll will have the least coins, since it doesn't receive coins from the grey goat while it's in the queue, and the white and black goats will have a higher coin count, due to priority payment. One of these trolls counts will be lower eventually, dipping below the grey troll's coins, due to its goats being wiped out by the other competing goat. As for the goat totals, any goat will never go over 100 of its kind, and two of the type of goats will die out. The grey goat will never die out, since it will lose the least amount of money from the rotations of bridges, while the white and black goat counts will initially be higher. Once there are more goats in the mix, the competition for priority and the grey bridge in the mix will cause one of the goats to start to decline rapidly, as well as the grey goat's since they are going to be low priority constantly in other bridges. So, the grey goats and either the white or black goats will die out eventually. I believe the black goats will survive, since they go through the lower priority queue of grey first, meaning they get back to black bridge without going through many priority advancing queues like the black and white queue. The only test case I have is the following test of my goat class and it speaks for itself. I wish I had more time for this project because it is a fun concept, just hard to grasp the concepts to implement it.

```
361    public static void main(String[] args) { //for testing
362
363        Goat goat = new Goat("Black");
364
365        GoatQueue<GoatPriority, Goat> queue= new GoatQueue<GoatPriority, Goat>();
366
367        queue.setSize(0);
368
369        Troll troll = new Troll("Black", queue);
370
371        System.out.println(goat.coinChecker(goat));
372
373        goat.addCoins(goat);
374
375        System.out.println(goat.coinChecker(goat));
376
377        goat.payToll(troll);
378
379        System.out.println(goat.coinChecker(goat));
380
381        goat.strategy(troll);
382
383        System.out.println(goat.getWillPay());
384
385    } //end method
386
387 } //end class
388
```

Problems @ Javadoc Declaration Console

<terminated> Goat [Java Application] C:\Program Files (x86)\Java\jre1.8.0_91\bin\javaw.exe (Nov 13, 2016, 11:48:00 PM)

```
100
100
100
false
200
200
200
false
Troll coins: 20
180
200
200
false
false
```