Reflections
Holly Buteau
CS 162
Assignment 4

Design:

<u>Creature:</u>
<u>protected:</u>
```
        int armor;
        int strength;
        int aRoll1;
        int aRoll2;
        int dRoll1;
        int dRoll2;
        int dRoll3;
        int aTotal;
        int dTotal;
        bool defeated;
        string name;
```

<u>public:</u>
```
    Creature();
     virtual void Battle(Creature *c1) = 0;
     virtual int Attack() = 0;
     virtual int Defense() = 0;
     virtual string getName() = 0;
     virtual int getArmor() = 0;
     virtual int getStrength() = 0;
     virtual bool getDefeated() = 0;
     virtual void damage(int aTotal, int dTotal) = 0;
     virtual void setDefeated() = 0;
     virtual void setName(string n) = 0;
     virtual void heal() = 0;
     virtual void scoreIncrement() = 0;
```

Barbarian:
<u>public:</u>
```
        Barbarian();
        virtual int Attack();
        virtual int Defense();
        virtual void Battle(Creature *c1);
        virtual string getName();
        virtual int getArmor();
        virtual int getStrength();
        virtual bool getDefeated();
        virtual void damage(int aTotal, int dTotal);
```

Vampire:

public:

```
    Vampire();
    virtual int Attack();
    virtual int Defense();
    virtual void Battle(Creature *c1);
    virtual string getName();
    virtual int getArmor();
    virtual int getStrength();
    virtual bool getDefeated();
    virtual void damage(int aTotal, int dTotal);
    virtual void setDefeated();
    virtual void setName(string n);
    virtual void heal();BlueMen:
```

Medusa:

public:

```
    Medusa();
    virtual int Attack();
    virtual int Defense();
    virtual void Battle(Creature *c1);
    virtual string getName();
    virtual int getArmor();
    virtual int getStrength();
    virtual bool getDefeated();
    virtual void damage(int aTotal, int dTotal);
    virtual void setDefeated();
    virtual void setName(string n);
    virtual void heal();BlueMen:
```

```
virtual int Attack(conditional);
virtual int Defense();

HarryPotter:
public:
        HarryPotter();
        virtual int Attack();
        virtual int Defense();
        virtual void Battle(Creature *c1);
        virtual string getName();
        virtual int getArmor();
        virtual int getStrength();
        virtual bool getDefeated();
        virtual void damage(int aTotal, int dTotal);
        virtual void setDefeated();
        virtual void setName(string n);
        virtual void heal();BlueMen:

BlueMen();
public:
        BlueMen ();
        virtual int Attack();
        virtual int Defense();
        virtual void Battle(Creature *c1);
        virtual string getName();
        virtual int getArmor();
        virtual int getStrength();
        virtual bool getDefeated();
        virtual void damage(int aTotal, int dTotal);
        virtual void setDefeated();
        virtual void setName(string n);
        virtual void heal();

CreatureQueue:
protected:
struct CreatureNode

{

        CreatureNode *next;

        Creature *critter;

};

        CreatureNode *tail;
        CreatureNode *head;
```

public:
```
	CreatureQueue();
	~CreatureQueue();
	void add(Creature *thing);
	void addBack(Creature *thing);
	void remove();
	void displayQueue();
	Creature* getFirst();
```

Design cont:
I will explain the design of only the newest methods and members. New Members: string name. Since the user was going to be allowed the enter the name for the characters, I needed to accept a string for the name. New Methods: I needed a setName function so that I could change the name of the creatures. I accepted a string argument for this. I also added a heal method so that I could call that if the creatures lived through the battle and they would be randomly healed. I will also explain the members and methods of the new CreatureQueue class. Members: A struct called QueueNode that will hold all the creature pointers in the line ups. A next pointer to determine the next creature in line. A critter pointer to point to the creature. A tail and a head pointer to point to the front and back of the queues. Methods: add will take a pointer to a Creature object and add it the struct. AddBack will also take a pointer to a Creature object and add it to the back to the queue. Remove will remove the top node from the struct. displayQueue will display the contents of the called queue. getFirst will return the first creature in the queue.

Changes:
Data Members:
I maintained all the same data members and did not have to make changes here.

Methods:
My lack of knowledge of memory allocation lead me to multiple errors with memory leaks. After attending a q&a session and talking to my peers, I realized that I needed a destructor for all of my classes. This was something I had not included before. I also added a scoreIncrement method so that I could keep track of how often a creature was winning for the winners line up. In the CreatureQueue class, I added an isEmpty method to check for empty structs before attempting to use them. I also took out the add method as having two add methods was unnecessary.

Test Plan: Test Case	Input Values	Expected Outcome	Observed Outcome
Barbarian attack rolls are random	Run program, pick barbarian	Dice rolls outputted are random in the range of 2-12	Dice rolls outputted are random in the range of 2-12
Medusa attack rolls are random	Run program, pick medusa	Dice rolls outputted are random in the range of 2-12	Dice rolls outputted are random in the range of 2-12
Vampire attack rolls are random	Run program, pick vampire	Dice rolls outputted are random in the range of 1-12	Dice rolls outputted are random in the range of 1-12
Blue Men attack rolls are random	Run program, pick blue men	Dice rolls outputted are random in the range of 2-20	Dice rolls outputted are random in the range of 2-20
Harry Potter attack rolls are random	Run program, pick Harry Potter	Dice rolls outputted are random in the range of 2-12	Dice rolls outputted are random in the range of 2-12
Barbarian defense rolls are random	Run program, pick barbarian	Dice rolls outputted are random in the range of 2-12	Dice rolls outputted are random in the range of 2-12
Medusa defense rolls are random	Run program, pick medusa	Dice rolls outputted are random in the range of 1-6	Dice rolls outputted are random in the range of 1-6
Vampire defense rolls are random	Run program, pick vampire	Dice rolls outputted are random in the range of 1-6	Dice rolls outputted are random in the range of 1-6
Harry Potter defense rolls are random	Run program, pick Harry Potter	Dice rolls outputted are random in the range of 2-12	Dice rolls outputted are random in the range of 2-12
Blue Men defense rolls are random	Run program, pick blue men	Depending on health, dice rolls are random in rages of 1-6, 2-12, or 3-18	Depending on health, dice rolls are random in rages of 1-6, 2-12, or 3-18
Vampire conditionals are applied	Run program, pick vampire	When charm works, vampire takes no damage	When charm works, vampire takes no damage
Medusa conditionals are applied	Run program, pick medusa	When glare works, Medusa automatically wins	When glare works, Medusa automatically wins
Harry Potter conditionals are applied	Run program, pick Harry Potter	When health is <= 0, Harry resurrects once with health of 20	When health is <= 0, Harry resurrects once with health of 20
Blue Men conditionals are applied	Run program, pick blue men	When health is between 8 and 4, only 2 defense	When health is between 8 and 4, only 2 defense

Test Plan: Test Case	Input Values	Expected Outcome	Observed Outcome
Lineup correctly Lists players in order	Run program, picking several creatures	Creatures are listed as they were entered	Creatures are listed as they were entered
Lineup changes between rounds to reflect deleted creatures	Run program, pick several creatures	Lineup reflects removed creatures,	Core dumps: memory leaks dealing with remove function
Loser lineup reflects creatures who lost	Run program, pick several creatures	Losers accurately portrays creatures who lost.	Core dumps: memory leaks dealing with display function
Player winner is accurately displayed	Run program, pick several creatures	One player is displayed as winner with score	Core dumps: memory leaks before getting to this point
Lineup of winners is accurately displayed	Run program, pick several creatures	The three top scoring creatures are listed in order of points	Core dumps: memory leaks before getting to this point

Reflections:

This assignment was one that I struggled with incredibly. Every step of the way, I faced memory leaks and core dumps. I am very disappointed that I never was able to fix them. I hate turning in unfinished assignments. I need to make sure that I continue studying memory allocation and work on this issue.