Project 4 - *Containers: Reflections*

Design Description

In this project, I will be be implementing a tournament system that extends our work from project 3, which was to creature a fantasy battle simulator. However, unlike in the previous project, we will be having two teams of fantasy creatures fight against one another. When the user first begins the application, they will be presented with a menu that gives them the option to begin the tournament, configure their teams, set the teams' sizes, or quit.

The way the tournament works is that two equally sized teams will be formed and composed of the different creature types created in project 3 (Barbarian, Medusa, Vampire, Blue Men, and Harry Potter). While the sizes of both teams must be the same, the composition of creatures can be totally different (e.g. one team could contain one of each creature type and another could have five vampires). When a user selects to add a particular creature to their roster, they may also assign that creature a given name. And, finally, once a team has its composition set and each creature named, the user will select the order that the creatures will appear in battle, very similar in style to a baseball batting lineup. Once this process has been completed for both teams, the battle may begin!

Once the tournament begins, the first creatures of each team will duke it out, with one being attacker and the other defender before switching roles next turn, over a number of rounds until their is only one team left standing. Once a creature has won a round, it is send to the back of its lineup using a queue-like data structure (again, like baseball after scoring a run) while the loser is send to a stack-like data structure, similar to a heap of dead zombies, where the freshest zombie corpse is sitting at the top of the pile. When a creature is sent to the back of the lineup, only to potentially enter the ring again, it gets some rest and is allowed to recover some strength points.

This back and forth series of rounds continue until one team has had all its creatures defeated. When this happens a screen appears that displays a message on which team won the tournament and how many points they had at the end. An option to display the pile of the dead will be presented, with the name of the most recently defeated showing up on the top.

Initial Design Thoughts

- Because we are going to be using our own linked list type structure with nodes, and we are required to have a name tied to each creature, I am planning on creating a custom node structure that holds a

pointer to the next node, along with a pointer to it's name, and a pointer to a creature.

- I am also planing on allowing teams of up to size fifteen, to help better ensure decent formatting when displaying the team lineup.

- When displaying the results of the round, I am planning on changing it from project 3 which had a very jarring back and forth visual to it when displaying the hit-by-hit report and ensure that the top of the output window remains relatively static so it doesn't look like text is totally shifting, but only content is being added below.

Testing Decisions

For this project, I decided to have my testing available through a menu option so I could continually add and make changes as necessary. This way of very visual unit testing gave me more positive feedback and allowed me to develop test plans much more robust than I normally would have, all things considered. I think I could have gone even further as a majority of my automated testing was only done in terms of the data structures, but it could have been adapted to test for correctness with the classes battling, as opposed to doing all of that by hand, as I had to do.

Class Design Descriptions

Die Class: The Die class represents the object which controls the attack and defense values of the Creatures during a fight. A Die can hold a number of faces (cannot be negative) and can produce a random number between one and the number of faces, or zero if there are no faces, this feature simulates the rolling of a die. A setter and getter are provided for the faces data member of the Die, which is useful if the die needs to change or the values of the Die need to be displayed to the user.

Private data Member(s): unsigned short faces

Public Method(s):
Die( )
Die(unsigned short)
unsigned short roll( ) const
unsigned short getFaces( ) const
void setFaces(unsigned short)

Creature Class: The Creature class is the central object of the entire project. Creature is an abstract class that is to be inherited into specialized Creature sub-types which have specialized abilities. Each creature has a number of attack dice, defense dice, a value for its strength and armor, and a pointer to its enemy (essential for being able to alter an enemy's stats directly through a special ability). The Creature class defines default behavior full rolling attack and defense dice along with making an attack or mounting a defense against another Creature. A standard suite of getters and setters are provided where necessary and a static factory method is defined to aid in the creation of specialized Creatures.

Private data Member(s): vector<Die> attackDice
vector<Die> defenseDice
unsigned short armor
unsigned short strength
Creature* enemy

Public Method(s): Creature( )
Creature(unsigned shortx6)
virtual ~Creature( )
static Creature* CreatureFactory(CreatureType)
unsigned attackRoll( ) const
unsigned defenseRoll( ) const
virtual unsigned attack( )
virtual void defend(unsigned)
virtual CreatureType getType( ) const = 0
virtual string getTypeAsString( ) const = 0
unsigned short getArmor( ) const
unsigned short getStrength( ) const
virtual void setStrength(unsigned short)
void setEnemy( )
virtual void printCurrentStats( ) const

Barbarian Class: The Barbarian represents the most basic of all the Creature subtypes. The Barbarian only implements a specialized version of the getType and getTypeAsString, however, this could have been implemented in the Creature class itself on a redesign. This subtype has no specialized abilities.

Public Method(s): Barbarian( )

virtual CreatureType getType( ) const override
virtual string getTypeAsString( ) const override

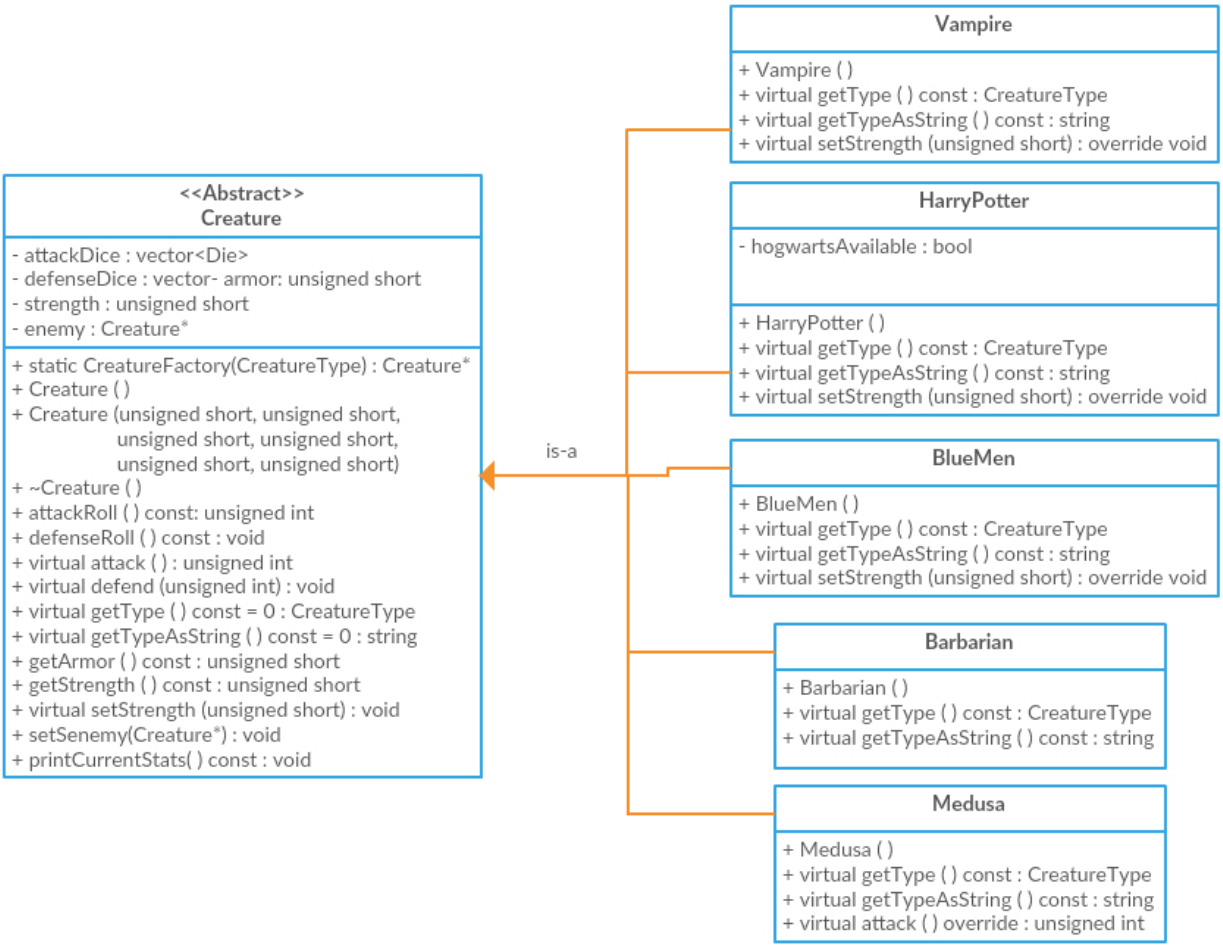
Vampire Class: The Vampire is a subtype of Creature that has implements a specialized defend method that gives the Vampire a 50% to charm his opponent, which allows the Vampire to take no damage, regardless of what the opponent rolls on their attack.

Public Method(s): Vampire( )
virtual CreatureType getType( ) const override
virtual string getTypeAsString( ) const override
virtual void defend(unsigned) override

Medusa Class: The Medusa is a subtype of Creature that has implements a specialized attack called "glare" that allows Medusa to kill an enemy when it rolls a twelve. When this special attack happens, Medusa sets the enemy's health to zero. This was done instead of sending a massive amount of damage because it seemed cleaner and more intuitive, although some complications came up with the Vampire, which will be discussed at the end of the reflections document.

Public Method(s): Medusa( )
virtual CreatureType getType( ) const override
virtual string getTypeAsString( ) const override
virtual unsigned attack(x) override

Creature Class Hierarchy Design



Input Testing Plan and Results

Input Testing Plan: Enter a menu selection (main menu)

Test Case	Input Values	Driver Functions	Expected Outcome	
Boundary/Negative:	“0”	mainMenu.getSelection( )	User re-prompted.	User re-prompted.
Boundary/Negative:	“5”	mainMenu.getSelection( )	User re-prompted.	User re-prompted.
Positive: Within Range	“1”	mainMenu.getSelection( )	Fight occurs if settings are configured.	Fight occurs if settings are configured.
Positive: Within Range	“4”	mainMenu.getSelection( )	Program Exits	Program Exits
Negative: Numbers, symbols, and/or whitespace combinations	“ *4$”	mainMenu.getSelection( )	User re-prompted.	User re-prompted.

Input Testing Plan: Creature Selection Menu

Test Case	Input Values	Driver Functions	Expected Outcome	Actual Outcome
Boundary/Negative: Largest number below range	“0”	creatureSelectionMenu.getSelection()	User re-prompted.	User re-prompted.
Boundary: Smallest number within range	“1”	creatureSelectionMenu.getSelection()	Main menu displays user has selected VAMPIRE	Main menu displays user has selected VAMPIRE
Positive: Number within range	“5”	creatureSelectionMenu.getSelection()	Main menu displays user has selected HARRY POTTER	Main menu displays user has selected HARRY POTTER
Negative: Number below range	“-32”	creatureSelectionMenu.getSelection()	User re-prompted.	User re-prompted.
Negative: Prepending valid value with whitespace	“ 3”	creatureSelectionMenu.getSelection()	User re-prompted.	User re-prompted.

Valgrind Memory Leak Results (No Leaks Possible)

```
==23864==
==23864== HEAP SUMMARY:
==23864==     in use at exit: 0 bytes in 0 blocks
==23864==   total heap usage: 150,029 allocs, 150,029 frees, 3,816,333 bytes all
ocated
==23864==
==23864== All heap blocks were freed -- no leaks are possible
==23864==
==23864== For counts of detected and suppressed errors, rerun with: -v
==23864== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Successfully Tested Creature Combinations
(* indicates incorrect behavior @ some point during development–now fixed)
Vampire - Vampire
Vampire - Barbarian
Vampire - Blue Men*
Vampire - Medusa*
Vampire - Harry Potter
Barbarian - Barbarian
Barbarian - Blue Men*
Barbarian - Medusa
Barbarian - Harry Potter
Blue Men - Blue Men*
Blue Men - Medusa*
Blue Men - Harry Potter*
Medusa - Medusa
Medusa - Harry Potter
Harry Potter - Harry Potter