Tyler Forrester Assignment 3

# Design Document

1. Class Hierarchy

Dice Class

asass

private:

int divisor;

public:

Dice();

Dice(int);

void setDivisor(int);

void seed();

int getDivisor();

int roll();

Creature: Abstract Base Class

Main Driver Function

Functions:

void displayMenu(string);

int getChoice(InputValid);

void continueOn(InputValid);

Creature\* pick(int choice, InputValid valid);

bool battle(Creature \* fighter1, Creature \* fighter2, InputValid valid);

Data members:

InputValid valid = InputValid();

Creature \* fighter2

Creature \* fighter1

int choice;

Key

Related by Composition

InputValid

private:

string input;

public:

InputValid();

int validateInt();

double validateDouble();

double validatePrice();

string validateString();

char validateChar();

Derived class

Derived class

Derived class

Creature Derived class

Creature Derived class

Private Base Class

Gollum

private:

int damage(int

def, int attack);

public:

Gollum();

int attack();

void defense(int);

HarryPotter

private:

bool alive;

void life();

int damage(int def, int attack);

public:

HarryPotter();

int attack();

void defense(int);

private:

int mob();

int damage(int def, int attack);

public:

BlueMen();

int attack();

void defense(int);

private:

int damage(int def, int attack);

public:

Medusa();

int attack();

void defense(int);

Reptilee

private:

int damage(int def, int attack);

public:

Reptile();

int attack();

void defense(int);

Medusa

BlueMen

en

private:

int armor;

int strength; Public:

public:

Creature(int, int);

virtual int attack() = 0;

virtual void defense(int) = 0;

int play(int, int);

void setStrength(int);

int getStrength(void);

int getArmor(void);

1. Design Explanation

The idea behind this project was to write a program that would allow two creatures of different (or same) types to fight. Each creature type would have unique properties that were only associated to that creature, but each creature would be restrict to a set of common actions and traits. These common actions and traits would be stored in an abstract class called Creature. The variables strength and armor were common to all creatures so I created these as private in the Creature class, and used accessors for strength and armor and mutator for just strength, armor is immutable once created. Since armor is in a private location with no mutator and Creature cannot be created by composition, armor is securely set. I contemplated making it a constant, but that seemed to be overkill. The functions included in Creature-- aside from the previously noted mutator and accessors-- are a play function and a pure virtual attack and defense functions.

The play function pulls in methods from a previously created Dice Class from Lab B. I decided to make the Dice Class a private base class for Creature. Since the idea behind this entire project was to create a game whose outcomes were based on chance, it seemed natural to include the dice as a base class for the creature class, given that creature class and its associated subclasses interactions are really governed by a type of chance. We can consider the Creature class as a guideline for these random interactions, so Dice should be a base class of creature. Doing this relationship gave me access to changing rolling die function, setting the number of sides functions, and a seeding function. This made implementing some of the future classes features much much easier.

The play function takes two integers are parameters, the number of rolls to be made and the number of sides on the dice. It returns the sum of all the rolls.

The pure virtual attack and defense function are set up to force each of the future subclasses of type Creature to be able to both defend and attack. Since each class does this differently, the methods need to be built individually for each class. However, all creatures must defend and attack, so it’s imperative that the base class force each creature to build a defend and attack function.

The Creature constructor will pass in variables to set both initial strength and permanent armor of each creature.

The first derived class of creature to be built is Reptile. Reptile is the choice, because it has no unique attacks or defends. Reptile constructor passes 18 strength and 7 armor to the creature constructor. I added a damage function to Reptile to make calculating the defense a little bit easier.

For Reptile’s attack, it starts by assigning an integer variable with the return of the Creature play function using 3 die rolls and six sides. It then prints out what the integer was. The print out allows the player to play along with the game as the creatures battle to the death. It then returns the integer on function exit. This allows the other creature’s defense function in the battle to accept this as a parameter.

For the Reptiles defense, it again calls the play function to populate a defense variable (using 1 six sided die roll). It prints out the defence number again so the user can play along with the fight. Once that happens a health variable is populated by calling getStrength() and subtracting the result of the damage function. We’ll talk more about this in the next paragraph. The total damage is then printed out and the setStrength() is passed the health variable, setting the creatures health to its new level.

The damage function does the basic arithmetic of the game, subtracting armor and defense from attack. If this is a negative number, it returns zero, otherwise it returns the number.

\*\*\*Since the other four classes have many of the same attributes for attack, defense and damage, I am only going to discuss what I changed in these classes. All print screens are changed to reflect the derived class names. \*\*\*

The next class was Medusa which is initialized to 8 strength with 3 armor and rolls 2 six-sided die for attack and 1 one-side die for defense. The key difference with Medusa is that this class has a special ability called “Glare” in which if Medusa rolls a 12 on attack, she automatically kills the other creature. Using an if statement for an attack of 12 in Medusa’s attack, I return some text saying how the special ability was used and a monstrously large attack number (20000). Since the most strength that any creature has currently is 33, this assures death at least until resurrection. I wanted to make the number big enough that Medusa would still kill future creatures. I feel comfortable that if Medusa is still a creature in the game (with a health of 8) that this number would be large enough to kill any future villains. Harry Potter will still be able to be resurrected when Medusa glare strikes. If Harry is strong enough to face Voldemort, he’s strong enough to face Medusa.

Gollum is initialized to 8 strength and 3 armor and normally rolls 1 six sided die in defence and attack. He has a special ability where 5% of the time, he gets to roll 3 six sided die in attack as opposed to one. This is called “Ring”. I chose to use the play function as the randomizer, by passing 1 20-sided die into play and creating an if clause that said if the twenty side die equals one, then call the play function with 3 dice. One number on a twenty side die is equal to a 5% chance. Gollum is very easy to implement, when using the play function.

Blue Men is initialized to 12 strength with 3 armor and rolls 2 10-sided die in attack and normally 3 6-sided die in defence. Their special ability is that as their health declines, they get to roll fewer dice. At 8 health, Bluemen roll two dice and at 4 health, they roll one dice. I accomplished this by creating a private function that returned the number of dice that should be rolled by the play function. It was possible to do it defense, but it seemed cleaner to keep the logic isolated (and possibly reuse for in another situation). It’s a private function because its only for the internal calculations of the Blue Men class. The 10-sided die was easily accomplished by passing 10- sides to the play function.

Harry Potter is initialized with 10 strength and 0 armor and rolls 2 6-sided die in attack and 2 6-sided die in defence. His special ability is the ability to be resurrected once. Since his resurrection can only occur once, I added a private boolean called alive, which the Harry Potter constructor sets to true. Once Harry strength drops below 1, a logical tests looks to see if Harry’s alive is still true. If it is, it calls a private function called “life” to set Harry’s strength back to 10 and sets alive equal to false. I decided to use a private function so it was clearer to the reader what happened when Harry was resurrected.

Now that we have covered all the derived classes of Creature, and Creature itself, let’s turn our attention to the game play. The game is played using a menu that asks the user to select a creature to fight. It’s a two player game, so Player 1 selects his creature and then Player 2 selects his creature. This is done through a menu selection. The program tells the player which creature that he has selected and then pauses until the user presses ‘c’. This gives both players the time to read the screen. Both Players are allowed to select the same sort of creature, although this does create some confusion as the program calls both creatures, “Creature” as opposed to “Creature 1” and “Creature 2”. The user is allowed to exit the game on either creature selected. This gives the reader a second chance to quit before game play commences. The program then pauses and asks the players to press c to start the battle. In the battle, a printout shows both players health before the round. This makes the action more visceral. Player 1 always goes first. If Player 1 attack sets Player 2’s strength to zero, then Player 1 wins before Player 2 gets a turn. This reflects reality. Dead players rarely rise from the grave to kill their murderers. At the end of the round, if both players are still alive then their health is displayed. Otherwise a printout informs the user which Player has won, the memory for the pointers is deallocated and then program prompts the user to continue. The player can savor his victory and other can weep in defeat. The program moves back to the main menu so that players can either play again or quit.

To validate the number and char inputs, I used an InputValid class which I created for the first Lab. I’ve been using the code to make the cin operations more user friendly. Only minimal testing for this is included, because it’s been tested on the last 8 assignments.

# Test Cases

I handled the randomness in this program, by using already verified random methods from Lab B. This gave me certainty in the how my dice rolls would function and left me to only worry about whether or not the creatures were appropriately coded. To do this I used fixed numbers for attack and defense to check my programs arithmetic, medusa’s glare, and the BlueMen’s dice. Gollum was tested by making sure the special text displayed. Again since the dice had already been validated there was no need to worry about the random number generator. My test cases will deal with the new code in this program rather than previous assignments code.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Cases | Input Values | Driver Functions | Expected Outcomes | Observed Outcomes |
| Selects Reptile | Input = 1 | getChoice() pick() | Creates Creature \* Pointing to Reptile | Creates Creature \* Pointing to Reptile |
| Selects Medusa | Input = 2 | getChoice() pick() | Creates Creature \* Pointing to Medusa | Creates Creature \* Pointing to Medusa |
| Selects Gollum | Input = 3 | getChoice() pick() | Creates Creature \* Pointing to Gollum | Creates Creature \* Pointing to Gollumj |
| Selects BlueMen | Input = 4 | getChoice() pick() | Creates Creature \* Pointing to BlueMen | Creates Creature \* Pointing to BlueMen |
| Selects  HarryPotter | Input = 5 | getChoice() pick() | Creates Creature \* Pointing to HarryPotter | Creates Creature \* Pointing to Harry Potter |
| Quits Program | Input = 6 | Main() | Program exited | Program  existed |
| Reptile is initialized to 7 armor and 8 strength | Input = 1 | pick() Reptile() Creature() | Reptile is initialized to 7 armor and 8 strength | Reptile is initialized to 7 armor and 8 strength |
| Reptile Damage is appropriately calculated | 13 Attack 4 Defense | attack() def() damage() | 2 Damage | 2 Damage |
| Strength is appropriate | 13 Attack 4 Defense | attack()def() damage()getStrength()setStrength() | 6 Strength | 6 Strength |
| Roll Dice 10 Times | Dice1 | roll() | Numbers between 1 and 6 | Numbers between 1 and 6 |
| Roll 3 Dice together 10 times | 6 sides | 3x \* roll() | Numbers between 3 and 18 | Numbers between 3 and 18 |
| Play until one creature is dead then correctly delete | Valgrind Select two Reptiles | main()battle()attack()def() damage()getStrength()setStrength()main() | No Memory Leaks, program returns to main menu | No Memory Leaks, program returns to main menu |
| Medusa’s Stoney Glare | Medusa rolls a 12 attack | Battle()Medusa::attack() | Medusa has a rolled a 12 in Attack.  Medusa's Stony Glare Has Killed the Other Player!! | Medusa has a rolled a 12 in Attack.  Medusa's Stony Glare Has Killed the Other Player!! |
| Medusa Rolls 1 Defense Dice | Play(1,6) | Play(1,6) | Rolls 1 6-sided die for outputs between 1-6 | Rolls 1 6-sided die for outputs between 1-6 |
| Medusa Rolls 2 Attack Dice | Play(2,6) | Play(2,6) | Rolls 2 6-sided die for outputs between 2-12 | Rolls 2 6-sided die for outputs between 2-12 |
| Medusa Initialized 3 armor and 8 Strength | 13 Attack 4 Defense | attack() def() damage() | 6 Damage  2 Health | 6 Damage  2 Health |
| Gollum’s Ring | 5% chance of 3 die | Battle()Gollum::attack() | Gollum wears the RING and is INVISIBLE! Gollum rolls 3 attack die.  Gollum has rolled a 13 in a Special Attack. | Gollum wears the RING and is INVISIBLE! Gollum rolls 3 attack die.  Gollum has rolled a 15 in a Special Attack. |
| Gollum Rolls 1 Defense Dice | Play(1,6) | Play(1,6) | Rolls 1 6-sided die for outputs between 1-6 | Rolls 1 6-sided die for outputs between 1-6 |
| Gollum Rolls 2 Attack Dice | Play(2,6) | Play(2,6) | Rolls 2 6-sided die for outputs between 2-12 | Rolls 2 6-sided die for outputs between 2-12 |
| Gollum Initialized 3 armor and 8 Strength | 13 Attack 4 Defense | attack() def() damage() | 6 Damage  2 Health | 6 Damage  2 Health |
| BlueMen Defense | BlueMen vs BlueMen Battle | Battle() BlueMen::defense() | Bluemen has rolled 3 die(s) when health greater than 8; 2 die(s) when health greater than 4; 1 die when health less than 5. | Bluemen has rolled 3 die(s) when health greater than 8; 2 die(s) when health greater than 4; 1 die when health less than 5. |
| BlueMen Rolls 2 10 sided Attack Dice | Play(2,10) | Play(2,10) | Rolls 2 10-sided die for outputs between 2-20 | Rolls 2 10-sided die for outputs between 2-20 |
| Bluemen Initialized 3 armor and 12 Strength | 13 Attack 4 Defense | attack() def() damage() | 6 Damage  6 Health | 6 Damage  6 Health |
| Harry Potter Resurrected | Play through game until Harry dies twice. | Battle() HarryPotter::defense | Harry Potter resurrects to 10 health the first time and dies the second time. | Harry Potter resurrects to 10 health the first time and dies the second time. |
| Harry Potter Defense | Play(2, 6) | Battle() HarryPotter::defense() | Rolls 2 6-sided die for outputs between 2-12 | Rolls 2 6-sided die for outputs between 2-12 |
| Harry PotterRolls 2 10 sided Attack Dice | Play(2,6) | Play(2,6) | Rolls 2 6-sided die for outputs between 2-12 | Rolls 2 6-sided die for outputs between 2-12 |
| Harry Potter Initialized 0 armor and 10 Strength | 13 Attack 4 Defense | attack() def() damage() | 9 Damage  1 Health | 9 Damage  1 Health |
| Harry Potter vs Reptile (check polymorphism)\_ | Select Harry Potter vs Reptile | Pick() battle() | Successfully fought | Successfully fought |
| Harry Potter vs BlueMen | Select Harry Potter vs BlueMen | Pick() battle() | Successfully fought | Successfully fought |
| Harry Potter vs Medusa | Select Harry Potter vs Medusa | Pick() battle() | Successfully fought | Successfully fought |
| Harry Potter vs Gollum | Select Harry Potter vs Gollum | Pick() battle() | Successfully fought | Successfully fought |
| Reptile vs BlueMen | Select Reptile vs BlueMen | Pick() battle() | Successfully fought | Successfully fought |
| Reptile vs Medusa | Select Reptile vs Medusa | Pick() battle() | Successfully fought | Successfully fought |
| Reptile vs Gollum | Select Reptile vs Gollum | Pick() battle() | Successfully fought | Successfully fought |
| Bluemen vs Medusa | Select BlueMen vs Medusa | Pick() battle() | Successfully fought | Successfully fought |
| BlueMen vs Gollum | Select BlueMen vs Gollum | Pick() battle() | Successfully fought | Successfully fought |
| Medusa vs Gollum | Select Medusa vs Gollum | Pick() battle() | Successfully fought | Successfully fought |

# Reflections

As you probably saw in my design document, the damage function was in each derived class and never modified. In the final version the damage function is now in the Creature class. Coding of this assignment was straightforward, letting the creature class inherit from the dice class made most of the function writing very straight forward. If was to rewrite the program, I would spend more time identifying which functions should belong in the creature class. It was a very tedious process removing the function from 10 files and placing it in one. It seems far easier to do this once. I’d also improve the UI. Its currently very clunky. The rest of this program seems to have the basis for a fun video game.