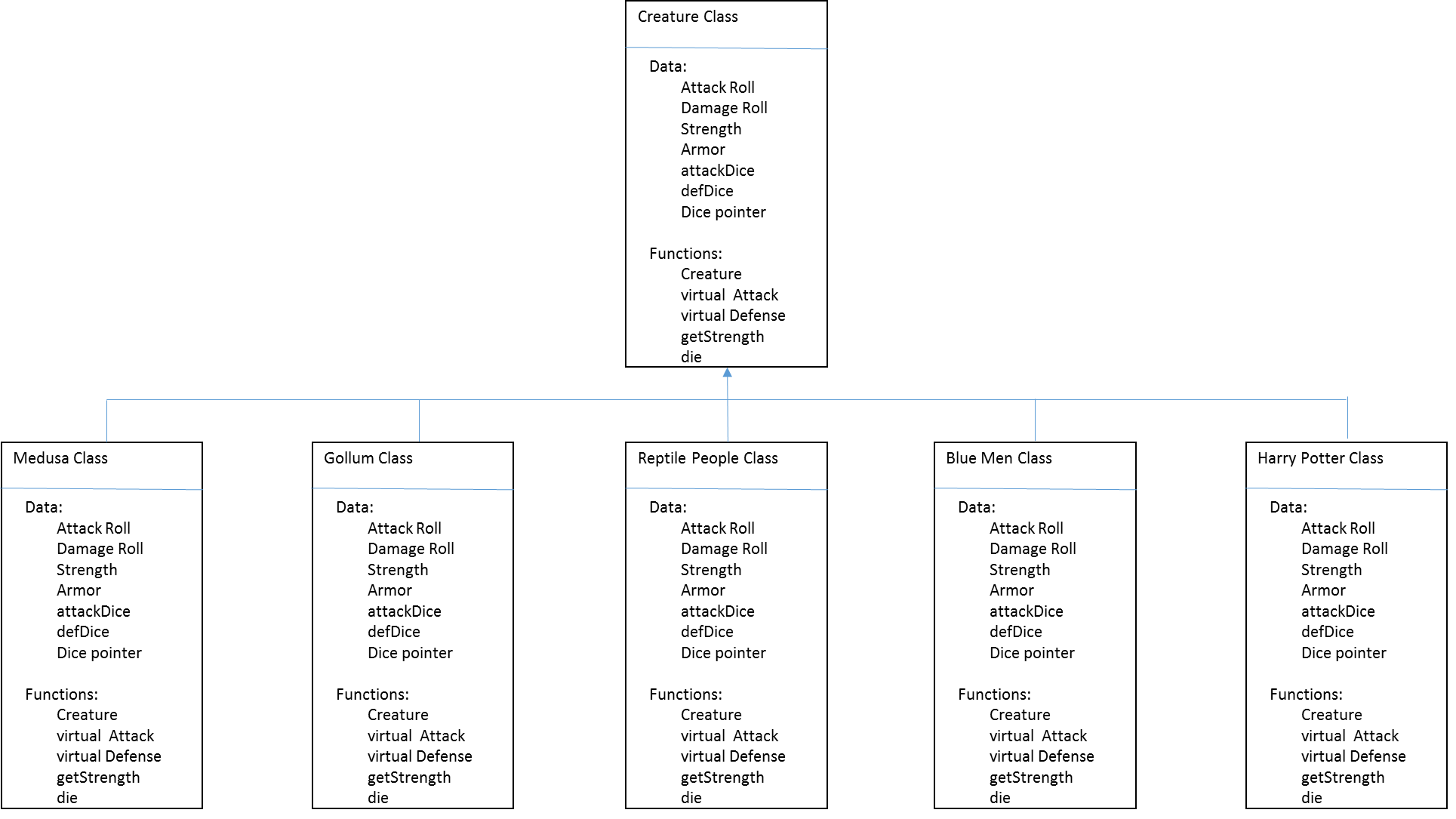
**Objective:** You will create a simple class hierarchy as the basis for a fantasy combat game. The “universe” contains Goblins, barbarians, Reptile People, Blue Men and possibly others. Each will have characteristics for attack, defense, armor, and strength points.

**Breaking Down the Code:**

|  |  |
| --- | --- |
| Data Needed | Actions Needed |
| Creatures class (abstract) | Attack move |
| Medusa subclass | Defense move |
| Gollum subclass | Armor move |
| Reptile People subclass | Strengthen move |
| Blue Men subclass | Roll Dice (with random num generator) |
| Harry Potter subclass | Creature constructor |
| Strength Points |  |
| Dice class |  |
| Armor value(?) |  |
| 2 Pointers to creature for combat |  |

**Class Hierarchy:**



**Design/Implementation Part 1:**

Creating the Creature Class and Generic Reptile People class (no special powers).

**Pseudocode:**

*Creature.hpp file:*

Class Creature {

Protected:

double attackRoll

double damageRoll

double strength

Int armor

Int attackDice;

Int defDice;

Dice \*d1;

Public:

Creature()

double Virtual attack()

double Virtual defense(double)

double getStrength() const

virtual Bool die()

}

*Creature.cpp file:*

Creature Constructor()

SET strength to 10

CREATE new Dice object

SET dice side count to 6

double getStrength() const

RETURN strength

Bool die() const

CREATE bool dead

SET dead = false

IF (strength <= 0)

SET dead = true

RETURN dead

*ReptilePeople.hpp file:*

Class ReptilePeople {

Public:

ReptilePeople()

double Virtual attack()

double Virtual defense(double)

}

*ReptilePeople.cpp file:*

ReptilePeople()

SET strength = 18

SET armor = 7

SET attackDice = 3

SET defDice = 1

SET dice side count = 6

double Virtual attack()

CALL dice setRollCount function

SEND attackDice value

SET attackRoll = 0

CALL roll function

STORE returned values

RETURN attackRoll

double virtual defense(double attackRoll)

IF (attackRoll = 550)

SET strength = -1

DISPLAY “game over”

ELSE

CALL dice’s setRollCount function

SEND defDice value

SET damageRoll= attackRoll

CREATE double roll

SET roll = 0

CALL roll function

STORE returned values in roll

SUBTRACT damageRoll - roll

STORE in damageRoll;

SUBTRACT damageRoll – Armor IF (result is positive)

SET strength = strength – damageRoll

RETURN strength

**Part 1 Test Plan:**

(Note: cout values placed throughout functions for testing have been removed in final files)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case | Input Value | Driver Function | Expected Outcome | Actual Outcome |
| Successful Creation of ReptilePeople objects | ReptilePeople \*rp1;  Rp1 = new ReptilePeople; | Creature & ReptilePeople Constructor and  getStrength() | “rp1 strength: 10”  “rp1 armor: 7” | “rp1 strength is 10”  “rp1 armor: 7” |
| Attack function correctly rolls dice & sums valuesⁱ | Double attackRoll = rp2->attack | Attack() | “roll: \*rndm nbr\*” | “roll: 0” |
| Defense function correctly rolls nbr of dice | lifeStrength = rp1->defense(attackRoll) | Defense() | “rp1 strength: \*damageRoll – armor value\*” | “rp1 strength: \*damageRoll – armor value\*” |

Fixes to Poor Test Outcome:

1. Altered Dice class’s rollDie() function in the following ways:
   1. Removed mean, median, mode analysis equations and all associated variables
   2. Altered return type from double pointer to double
   3. Solved Problem

**Design/Implementation Part 2:**

**Pseudocode:**

*Medusa.hpp file:*

Class Medusa {

Public:

Medusa()

double Virtual attack()

double Virtual defense(double)

}

*Medusa.cpp file:*

Medusa()

SET strength = 8

SET armor = 3

SET attackDice = 2

SET defDice = 1

SET dice side count = 6

double Virtual attack()

CALL dice’s setRollCount function

SEND attackDice value

SET attackRoll = 0

CALL roll function

STORE returned values

IF (attackRoll == 12)

SET attackRoll = 550

RETURN attackRoll

double virtual defense(double attackRoll)

IF (attackRoll = 550)

SET strength = -1

DISPLAY “game over”

ELSE

CALL dice’s setRollCount function

SEND defDice value

SET damageRoll= attackRoll

CREATE double roll

SET roll = 0

CALL roll function

STORE returned values in roll

SUBTRACT damageRoll - roll

STORE in damageRoll;

SUBTRACT damageRoll – Armor IF (result is positive)

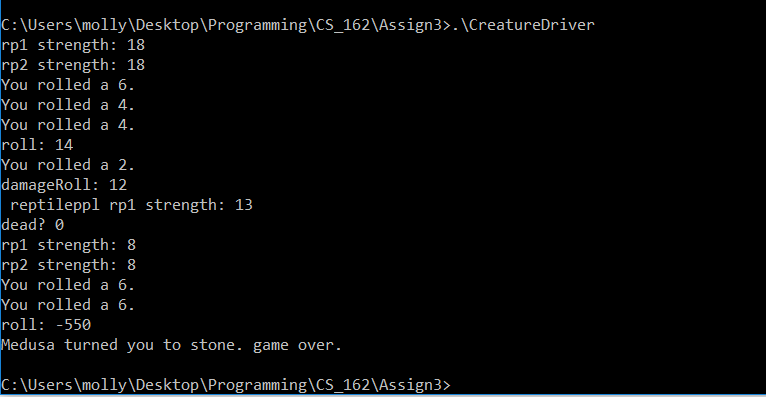
SET strength = strength – damageRoll

RETURN strength

**Part 2 Test Plan:**

(Note: cout values placed throughout functions for testing have been removed in final files)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case | Input Value | Driver Function | Expected Outcome | Actual Outcome |
| Successful Creation of Medusa objects | Rp1 = new Medusa; | Medusa Constructor and  getStrength() | “rp1 strength: 8” | “rp1 strength: 8” |
| Attack function correctly rolls dice & sums valuesⁱ | attackRoll = rp2->attack() | Attack() | “roll: \*rndm nbr\*” | “roll: \*rndm nbr” |
| Defense function correctly rolls nbr of dice | lifeStrength = rp1->defense(attackRoll) | Defense() | “rp1 strength: \*damageRoll – armor value\*” | “rp1 strength: \*damageRoll – armor value\*” |
| Defense function correctly id’s roll of 12. | attackRoll = rp2->attack() | Defense() | “roll: -550”  “Medusa turned you to stone. Game over.” | “roll: -550”  “Medusa turned you to stone. Game over.” |



**Design/Implementation Part 3:**

**Pseudocode:**

*Gollum.hpp file*:

Class Gollum : public Creature {

Public:

Gollum()

double Virtual attack()

double Virtual defense(double)

}

*Gollum.cpp file*:

Gollum()

SET strength = 8

SET armor = 3

SET attackDice = 1

SET defDice = 1

SET dice side count = 6

Double attack()

SET attackRoll = 0

CREATE ring variable

SET ring = 13

CREATE ringRoll variable

GENERATE random number between 1 – 20

SET ringRoll to value

IF (ringRoll == ring)

SET attackDice = 3

CALL dice’s setRollCount function

SEND attacDice

CALL roll function

STORE returned values

RETURN attackRoll

Double defense(double attackRoll)

IF (attackRoll = 550)

SET strength = -1

DISPLAY “game over”

ELSE

CALL dice’s setRollCount function

SEND defDice value

SET damageRoll= attackRoll

CREATE double roll

SET roll = 0

CALL roll function

STORE returned values in roll

SUBTRACT damageRoll - roll

STORE in damageRoll;

SUBTRACT damageRoll – Armor IF (result is positive)

SET strength = strength – damageRoll

RETURN strength

**Part 3 Test Plan:**

(Note: cout values placed throughout functions for testing have been removed in final files)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case | Input Value | Driver Function | Expected Outcome | Actual Outcome |
| Successful Creation of Gollum objects | Rp1 = new Gollum; | Gollum Constructor and  getStrength() | “rp1 strength: 8” | “rp1 strength: 8” |
| Attack function correctly rolls dice & sums values | attackRoll = rp2->attack() | Attack() | “roll: \*rndm nbr\*” | “roll: \*rndm nbr” |
| Defense function correctly rolls nbr of dice | lifeStrength = rp1->defense(attackRoll) | Defense() | “rp1 strength: \*damageRoll – armor value\*” | “rp1 strength: \*damageRoll – armor value\*” |
| Attack function correctly uses 3 dice if number 13 is randomly generated | ringRoll = 13 | Attack() | 3 rolls appear in output screen for Gollum attack | 3 rolls appear in output screen for Gollum attack |

**Design/Implementation Part 4:**

**Pseudocode:**

*BlueMen.hpp file:*

Class BlueMen : public Creature {

Private:

Dice \*d2

Public:

BlueMen()

double Virtual attack()

double Virtual defense(double)

}

*BlueMen.cpp file*:

BlueMen()

SET strength = 12

SET armor = 3

SET attackDice = 2

SET defDice = 3

SET d1 dice side count = 6

DECLARE new d2 Dice

SET d2 dice side count = 10

Double attack()

CALL d2 dice’s setRollCount function

SEND attackDice

SET attackRoll = 0

CALL d2 roll function

STORE returned values

RETURN attackRoll

Double defense(double attackRoll)

IF (attackRoll = 550)

SET strength = -1

DISPLAY “game over”

ELSE

CALL d1 dice’s setRollCount function

SEND defDice value

SET damageRoll= attackRoll

CREATE double roll

SET roll = 0

CALL d2 roll function

STORE returned values in roll

SUBTRACT damageRoll - roll

STORE in damageRoll;

SUBTRACT damageRoll – Armor IF (result is positive)

SET strength = strength – damageRoll

IF (strength has lost < 4 points)

No change

ELSE IF (strength has lost 4-7 ponts)

SET defDice = 2

ELSE

SET defDice = 1

RETURN strength

**Part 4 Test Plan:**

(Note: cout values placed throughout functions for testing have been removed in final files)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| c | Input Value | Driver Function | Expected Outcome | Actual Outcome |
| Successful Creation of BlueMen objects | Rp1 = new BlueMen; | BlueMen Constructor and  getStrength() | “rp1 strength: 12” | “rp1 strength: 12” |
| Attack function correctly rolls dice & sums values | attackRoll = rp2->attack() | Attack() | “roll: \*rndm nbr\*” | “roll: \*rndm nbr” |
| Defense function correctly rolls nbr of dice | lifeStrength = rp1->defense(attackRoll) | Defense() | “rp1 strength: \*damageRoll – armor value\*” | “rp1 strength: \*damageRoll – armor value\*” |
| Defense function correctly removes die in strength increments of 4 | Strength = 8 | Defense() | 2 rolls appear in output screen for BlueMen Defense | 2 rolls appear in output screen for BlueMen Defense |

**Design/Implementation Part 5:**

**Pseudocode:**

HarryPotter.hpp file:

Class HarryPotter : public Creature {

Private:

Int catLife

Public:

HarryPotter()

double Virtual attack()

double Virtual defense(double)

virtual bool die();

}

*HarryPotter.cpp file*:

HarryPotter()

SET strength = 10

SET armor = 0

SET attackDice = 2

SET defDice = 2

SET d1 dice side count = 6

SET catLife = 0;

Double attack()

CALL dice’s setRollCount function

SEND attackDice

SET attackRoll = 0

CALL roll function

STORE returned values

RETURN attackRoll

Double defense(double attackRoll)

IF (attackRoll = 550)

SET strength = -1

DISPLAY “game over”

ELSE

CALL d1 dice’s setRollCount function

SEND defDice value

SET damageRoll= attackRoll

CREATE double roll

SET roll = 0

CALL d2 roll function

STORE returned values in roll

SUBTRACT damageRoll - roll

STORE in damageRoll;

SUBTRACT damageRoll – Armor IF (result is positive)

SET strength = strength – damageRoll

RETURN strength

Bool die() const

CREATE bool dead

SET dead = false

IF (strength <= 0)

SET dead = true

IF (dead = true)

IF (catLife = 0)

SET dead = false

CALL resetStrength function

SET catLife = 1

RETURN dead

**Part 5 Test Plan:**

(Note: cout values placed throughout functions for testing have been removed in final files)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| c | Input Value | Driver Function | Expected Outcome | Actual Outcome |
| Successful Creation of HarryPotter objects | Rp1 = new HarryPotter; | HarryPotter Constructor and  getStrength() | “rp1 strength: 10” | “rp1 strength: 10” |
| Attack function correctly rolls dice & sums values | attackRoll = rp2->attack() | Attack() | “roll: \*rndm nbr\*” | “roll: \*rndm nbr” |
| Defense function correctly rolls nbr of dice | lifeStrength = rp1->defense(attackRoll) | Defense() | “rp1 strength: \*damageRoll – armor value\*” | “rp1 strength: \*damageRoll – armor value\*” |
| Die function correctly resets if catLife == 0ⁱ | catLife == 0 | Die() | HarryPotter’s strength = 10 next turn | HarryPotter’s strength = 10 next turn |

**Reflections:**

After hard-coding data into a driver file to confirm that each function was appropriately working, the code above was modified slightly in order to be more compatible with a looping environment. The Creature class was extended to include a name member variable with setName and getName functions in order to allow the user to see the name of the Creature during battles. A strengthDefault member and resetStrength function were also included in the Creature class in order to reset character strengths automatically between each battle. One type of each character was placed into an array of pointers. The main method loops through the array in a way to ensure that each character battles each other once.

Because it is not a common feature for all Creatures to have a second chance at life (like Harry Potter), the decision was made to keep the function “resetCatLife” within the HarryPotter class only, and not declare it as a virtual function in the Creature class. Therefore, a dynamic\_cast is needed in the main file between battles in order to gain access to this function through a Creature pointer variable. Access is needed in order to reset the trigger variable that allows HarryPotter objects to regenerate once before the next battle.