Project 6

● Graded

Student
Devin Chen

Total Points
95 / 100 pts

Autograder Score
80.0 / 80.0

Passed Tests
Test compiles (5/5)
Testing Attact Struct (3/3)
Tests Dragon, Ghoul, Mindflayer addAttack, getAttackQueue, clearAttackQueue (21/21)
Tests Dragon, Ghoul, Mindflayer displayAttacks() (6/6)
Tests Cavern's get, initialize, and clear stacks (15/15)
setAttacks() (30/30)

Question 2

Style & Documentation

15 / 20 pts

- ✔ **+ 5 pts** Style
  - **+ 5 pts** Indicates name and date in comment preamble
- ✔ **+ 5 pts** Has inline comments where appropriate
- ✔ **+ 5 pts** Has function preambles with @pre, @post, @param, @return where appropriate
  - **+ 20 pts** No-Compile Adjustment
  - **+ 4 pts** Partial No-Compile: 1/5
  - **+ 8 pts** Partial No-Compile: 2/5
  - **+ 12 pts** Partial No-Compile: 3/5
  - **+ 16 pts** Partial No-Compile: 4/5
  - **+ 0 pts** Insufficient submission

Autograder Results

Test compiles (5/5)

Your program compiles!

Testing Attact Struct (3/3)

Your program passed this test.

Tests Dragon, Ghoul, Mindflayer addAttack, getAttackQueue, clearAttackQueue (21/21)

Your program passed this test.

Tests Dragon, Ghoul, Mindflayer displayAttacks() (6/6)

Your program passed this test.

Tests Cavern's get, initialize, and clear stacks (15/15)

Your program passed this test.

setAttacks() (30/30)

Your program passed this test.

Submitted Files

▾ .gitignore Download

```
1 .DS_Store
2 .vscode
3 *.log
4
5
```

ArrayBag.cpp

Download

```cpp
/*
ArrayBag interface for term project
CSCI 235 Spring 2024
*/


#include "ArrayBag.hpp"

/** default constructor**/
template<class ItemType>
ArrayBag<ItemType>::ArrayBag(): item_count_(0)
{
} // end default constructor

/**
 @return item_count_ : the current size of the bag
 **/
template<class ItemType>
int ArrayBag<ItemType>::getCurrentSize() const
{
	return item_count_;
} // end getCurrentSize

/**
 @return true if item_count_ == 0, false otherwise
 **/
template<class ItemType>
bool ArrayBag<ItemType>::isEmpty() const
{
	return item_count_ == 0;
} // end isEmpty

/**
 @return true if new_entry was successfully added to items_, false otherwise
 **/
template<class ItemType>
bool ArrayBag<ItemType>::add(const ItemType& new_entry)
{
	bool has_room = (item_count_ < DEFAULT_CAPACITY);
	if (has_room)
	{
		items_[item_count_] = new_entry;
		item_count_++;
	return true;
	} // end if

	return false;
} // end add

```

```
/**
 @return true if an_entry was successfully removed from items_, false otherwise
 **/
template<class ItemType>
bool ArrayBag<ItemType>::remove(const ItemType& an_entry)
{
  int found_index = getIndexOf(an_entry);
      bool can_remove = !isEmpty() && (found_index > -1);
      if (can_remove)
      {
            item_count_--;
            items_[found_index] = items_[item_count_];
      }  // end if
      return can_remove;
}  // end remove

/**
 @post item_count_ == 0
 **/
template<class ItemType>
void ArrayBag<ItemType>::clear()
{
      item_count_ = 0;
}  // end clear

/**
 @return the number of times an_entry is found in items_
 **/
template<class ItemType>
int ArrayBag<ItemType>::getFrequencyOf(const ItemType& an_entry) const
{
  int frequency = 0;
  int curr_index = 0;       // Current array index
  while (curr_index < item_count_)
  {
    if (items_[curr_index] == an_entry)
    {
      frequency++;
    }  // end if

    curr_index++;         // Increment to next entry
  }  // end while

  return frequency;
}  // end getFrequencyOf

/**
 @return true if an_entry is found in items_, false otherwise
 **/
template<class ItemType>
bool ArrayBag<ItemType>::contains(const ItemType& an_entry) const
{
```

```
102         return getIndexOf(an_entry) > -1;
103 }  // end contains
104 
105 // ********* PRIVATE METHODS **************//
106 
107 /**
108         @param target to be found in items_
109         @return either the index target in the array items_ or -1,
110         if the array does not containthe target.
111  **/
112 template<class ItemType>
113 int ArrayBag<ItemType>::getIndexOf(const ItemType& target) const
114 {
115         bool found = false;
116   int result = -1;
117   int search_index = 0;
118    // If the bag is empty, item_count_ is zero, so loop is skipped
119    while (!found && (search_index < item_count_))
120    {
121 
122      if (items_[search_index] == target)
123      {
124        found = true;
125        result = search_index;
126      }
127      else
128      {
129        search_index++;
130      }  // end if
131    }  // end while
132 
133    return result;
134 }  // end getIndexOf
135 
136 template<class ItemType>
137 void ArrayBag<ItemType>::operator/=(const ArrayBag<ItemType> &rhs)
138 {
139   int index = 0;
140   int itemsToAdd = rhs.item_count_;
141   while (itemsToAdd > 0)
142   {
143     if (this->item_count_ == DEFAULT_CAPACITY)
144     {
145       break;
146     }
147     if (contains(rhs.items_[index]))
148     {
149       index++;
150       itemsToAdd--;
151       continue;
152     }
153    this->add(rhs.items_[index]);
```

```
154
155     index++;
156     itemsToAdd--;
157   }
158 }
159
160 template<class ItemType>
161 void ArrayBag<ItemType>::operator+=(const ArrayBag<ItemType> &rhs)
162 {
163   int index = 0;
164   int itemsToAdd = rhs.item_count_;
165   while (itemsToAdd > 0)
166   {
167     if (item_count_ == DEFAULT_CAPACITY)
168     {
169       break;
170     }
171     add(rhs.items_[index]);
172     index++;
173     itemsToAdd--;
174   }
175 }
```

ArrayBag.hpp

Download

```
/*
ArrayBag interface for term project
CSCI 235 Spring 2024
*/

#ifndef ARRAY_BAG_
#define ARRAY_BAG_
#include <iostream>
#include <vector>

template <class ItemType>
class ArrayBag
{

  public:
  /** default constructor**/
  ArrayBag();

  /**
     @return item_count_ : the current size of the bag
  **/
  int getCurrentSize() const;

  /**
     @return true if item_count_ == 0, false otherwise
  **/
  bool isEmpty() const;

  /**
     @return true if new_entry was successfully added to items_, false otherwise
  **/
  bool add(const ItemType &new_entry);

  /**
     @return true if an_entry was successfully removed from items_, false otherwise
    **/
  bool remove(const ItemType &an_entry);

  /**
     @post item_count_ == 0
    **/
  void clear();

  /**
     @return true if an_entry is found in items_, false otherwise
    **/
  bool contains(const ItemType &an_entry) const;

  /**
```

```
50        @return the number of times an_entry is found in items_
51      **/
52      int getFrequencyOf(const ItemType &an_entry) const;
53
54      /**
55       * @param:   another ArrayBag object
56       @post:    Combines the contents from both ArrayBag objects, EXCLUDING duplicates.
57       Example: [1, 2, 3] /= [1, 4] will produce [1, 2, 3, 4]
58       */
59       void operator/= (const ArrayBag<ItemType>& a_bag);
60
61
62       /**
63          @param:   another ArrayBag object
64          @post:    Combines the contents from both ArrayBag objects, including duplicates,
65                              adding items from the argument bag as long as there is space.
66                          Example: [1, 2, 3] += [1, 4] will produce [1, 2, 3, 1, 4]
67       */
68       void operator+= (const ArrayBag<ItemType>& a_bag);
69
70      protected:
71      static const int DEFAULT_CAPACITY = 100; //max size of items_ at 100 by default for this project
72      ItemType items_[DEFAULT_CAPACITY];      // Array of bag items
73      int item_count_;                        // Current count of bag items
74
75      /**
76         @param target to be found in items_
77        @return either the index target in the array items_ or -1,
78        if the array does not contain the target.
79        **/
80      int getIndexOf(const ItemType &target) const;
81
82  }; // end ArrayBag
83
84  #include "ArrayBag.cpp"
85  #endif
86
```

Cavern.cpp

Download

```
/*
CSCI235 Spring 2024
Project 4 - MycoMorsels
Georgina Woo
Dec 24 2023
Cavern.cpp declares the Cavern class along with its private and public members
*/
#include "Cavern.hpp"


Cavern::Cavern() : ArrayBag<Creature*>(), level_sum_{0}, tame_count_{0} {
}

/**
   @param  : the name of an input file
   @pre    : Formatting of the csv file is as follows (each numbered item appears separated by comma, only one value for each numbered item):
            1. TYPE: An uppercase string [DRAGON, GHOUL, MINDFLAYER]
            2. NAME: An uppercase string
            3. CATEGORY: An uppercase string [ALIEN, MYSTICAL, UNDEAD]
            4. HITPOINTS: A positive integer
            5. LEVEL: A positive integer
            6. TAME: 0 (False) or 1 (True)
            7. ELEMENT/FACTION: Uppercase string or strings representing the ELEMENT (For Dragons), or FACTION (For Ghouls) of the creature. If the creature is of a different subclass, the value will be NONE
            8. HEADS: A positive integer of the number of heads the Dragon has. If the creature is of a different subclass, the value will be 0
            9. FLIGHT/TRANSFORM/SUMMONING: 0 (False) or 1 (True) representing if the creature can fly (Dragons), transform (Ghouls), or summon a Thoughtspawn (Mindflayers).
            10. DECAY: A non-negative integer representing the level of decay of the Ghoul. If the creature is of a different subclass, the value will be 0
            11. AFFINITIES: Only applicable to Mindflayers. Affinities are of the form [AFFINITY1];[AFFINITY2] where multiple affinities are separated by a semicolon. Th value may be NONE for a Mindflayer with no affinities, or creatures of other subclasses.
            12. PROJECTILES: Only applicable to Mindflayers. PROJECTILES are of the form [PROJECTILE TYPE1]-[QUANTITY];[PROJECTILE TYPE 2]-[QUANTITY] where multiple types of projectiles are separated by a semicolon. The value may be NONE for a Mindflayer with no projectiles, or creatures of other subclasses.
*/
Cavern::Cavern(const std::string& filename)
{
  std::ifstream fin(filename);
  std::string line;
  std::getline(fin, line); //getting junk header
  std::string type, name, str_category, str_hitpoints, str_level, str_tame, str_element, str_heads, str_flight, str_decay, str_affinities, str_projectiles;
  int hitpoints, level, heads, decay;
  bool tame, flight;
```

```
38  Creature::Category category;
39  Dragon::Element element;
40  Ghoul::Faction faction;
41  std::vector<Mindflayer::Variant> affinities;
42  std::vector<Mindflayer::Projectile> projectiles;
43
44  while(std::getline(fin, line))
45  {
46    std::istringstream iss(line);
47    std::getline(iss, type, ',');
48    std::getline(iss, name, ',');
49    std::getline(iss, str_category, ',');
50    std::getline(iss, str_hitpoints, ',');
51    std::getline(iss, str_level, ',');
52    std::getline(iss, str_tame, ',');
53    std::getline(iss, str_element, ',');
54    std::getline(iss, str_heads, ',');
55    std::getline(iss, str_flight, ',');
56    std::getline(iss, str_decay, ',');
57    std::getline(iss, str_affinities, ',');
58    std::getline(iss, str_projectiles, ',');
59
60    std::istringstream(str_hitpoints) >> hitpoints;
61    std::istringstream(str_level) >> level;
62    std::istringstream(str_tame) >> tame;
63    std::istringstream(str_heads) >> heads;
64    std::istringstream(str_flight) >> flight;
65    std::istringstream(str_decay) >> decay;
66
67    // Convert the category to the category enum
68    if(str_category == "ALIEN")
69    {
70      category = Creature::ALIEN;
71    }
72    else if(str_category == "MYSTICAL")
73    {
74      category = Creature::MYSTICAL;
75    }
76    else if(str_category == "UNDEAD")
77    {
78      category = Creature::UNDEAD;
79    }
80    else
81    {
82      category = Creature::UNKNOWN;
83    }
84
85    if(type == "DRAGON")
86    {
87      if(str_element == "FIRE")
88      {
89        element = Dragon::FIRE;
```

```
90        }
91        else if(str_element == "WATER")
92        {
93          element = Dragon::WATER;
94        }
95        else if(str_element == "EARTH")
96        {
97          element = Dragon::EARTH;
98        }
99        else if(str_element == "AIR")
100       {
101         element = Dragon::AIR;
102       }
103       else
104       {
105         element = Dragon::NONE;
106       }
107       Dragon* new_dragon = new Dragon(name, category, hitpoints, level, tame, element, heads,
    flight);
108       enterCavern(new_dragon);
109     }
110     else if(type == "GHOUL")
111     {
112       if(str_element == "FLESHGORGER")
113       {
114         faction = Ghoul::FLESHGORGER;
115       }
116       else if(str_element == "SHADOWSTALKER")
117       {
118         faction = Ghoul::SHADOWSTALKER;
119       }
120       else if(str_element == "PLAGUEWEAVER")
121       {
122         faction = Ghoul::PLAGUEWEAVER;
123       }
124
125       else
126       {
127         faction = Ghoul::NONE;
128       }
129
130       Ghoul* new_ghoul = new Ghoul(name, category, hitpoints, level, tame, decay, faction, flight);
131       enterCavern(new_ghoul);
132     }
133     else if(type == "MINDFLAYER")
134     {
135       //clear the affinities and projectiles vectors
136       affinities.clear();
137       projectiles.clear();
138       // Read the affinities
139       std::string affinity;
140       std::istringstream iss_affinities(str_affinities);
```

```
141       while(std::getline(iss_affinities, affinity, ';'))
142       {
143         // Convert the affinity to the enum
144         if(affinity == "PSIONIC")
145         {
146           affinities.push_back(Mindflayer::PSIONIC);
147         }
148         else if(affinity == "TELEPATHIC")
149         {
150           affinities.push_back(Mindflayer::TELEPATHIC);
151         }
152         else if(affinity == "ILLUSIONARY")
153         {
154           affinities.push_back(Mindflayer::ILLUSIONARY);
155         }
156
157       }
158
159       // Read the projectiles
160       std::string projectile;
161       std::istringstream iss_projectiles(str_projectiles);
162       while(std::getline(iss_projectiles, projectile, ';'))
163       {
164         std::string type;
165         int quantity;
166         Mindflayer::Projectile new_projectile;
167         std::istringstream iss_projectile(projectile);
168         std::getline(iss_projectile, type, '-');
169         std::getline(iss_projectile, str_projectiles, '-');
170         std::istringstream(str_projectiles) >> quantity;
171         if(type == "PSIONIC")
172         {
173           new_projectile.type_ = Mindflayer::PSIONIC;
174         }
175         else if(type == "TELEPATHIC")
176         {
177           new_projectile.type_ = Mindflayer::TELEPATHIC;
178         }
179         else if(type == "ILLUSIONARY")
180         {
181           new_projectile.type_ = Mindflayer::ILLUSIONARY;
182         }
183         new_projectile.quantity_ = quantity;
184         projectiles.push_back(new_projectile);
185       }
186       Mindflayer* new_mindflayer = new Mindflayer(name, category, hitpoints, level, tame, projectiles,
    flight, affinities);
187       enterCavern(new_mindflayer);
188     }
189 }
190 }
191
```

```
192 bool Cavern::enterCavern(Creature* new_creature) {
193   if (getIndexOf(new_creature) == -1) {
194     if (add(new_creature)){
195       level_sum_ += new_creature->getLevel();
196       if (new_creature->isTame()) {
197         tame_count_++;
198       }
199       return true;
200     }
201   }
202   return false;
203 }
204 
205 bool Cavern::exitCavern(Creature* creature_to_remove) {
206   if (remove(creature_to_remove)) {
207     level_sum_ -= creature_to_remove->getLevel();
208     if (creature_to_remove->isTame()) {
209       tame_count_--;
210     }
211     return true;
212   }
213   return false;
214 }
215 
216 int Cavern::getLevelSum() const {
217   return level_sum_;
218 }
219 
220 int Cavern::calculateAvgLevel() const {
221   if (isEmpty()) {
222     return 0;
223   }
224   return round(level_sum_ / getCurrentSize());
225 }
226 
227 int Cavern::getTameCount() const {
228   return tame_count_;
229 }
230 
231 double Cavern::calculateTamePercentage() const {
232   if (isEmpty()) {
233     return 0;
234   }
235   double tame_percent = (tame_count_>0) ?  (double(tame_count_) / item_count_) * 100: 0.0;
236   return std::ceil(tame_percent*100.0) / 100.0; //round up to to decimal places
237 
238 }
239 
240 int Cavern::tallyCategory(const std::string& category) const {
241   if(category != "UNKNOWN" && category != "UNDEAD" && category != "MYSTICAL" && category !=
    "ALIEN") {
242     return 0;
```

```
243   }
244   int count = 0;
245   for (int i = 0; i < getCurrentSize(); i++) {
246     if (items_[i]->getCategory() == category) {
247       count++;
248     }
249   }
250   return count;
251 }
252 
253 int Cavern::releaseCreaturesBelowLevel(int level) {
254   int count = 0;
255   if (level < 0) {
256     return 0;
257   }
258   else if (level == 0) {
259     count = getCurrentSize();
260     clear();
261     return count;
262   }
263   else {
264     int size = getCurrentSize();
265     for (int i = 0; i < size; i++) {
266       if (items_[i]->getLevel() < level) {
267         exitCavern(items_[i]);
268         count++;
269       }
270     }
271     return count;
272   }
273 }
274 
275 int Cavern::releaseCreaturesOfCategory(std::string category) {
276   int count = 0;
277   if (category == "ALL") {
278     count = getCurrentSize();
279     clear();
280     return count;
281   }
282   else if (category != "UNKNOWN" && category != "UNDEAD" && category != "MYSTICAL" && category
    != "ALIEN") {
283     return 0;
284   }
285   else {
286     int size = getCurrentSize();
287     for (int i = 0; i < size; i++) {
288     if (items_[i]->getCategory() == category) {
289       exitCavern(items_[i]);
290       count++;
291     }
292   }
293   return count;
```

```
294   }
295 }
296
297 void Cavern::cavernReport() const {
298   std::cout << "UNKNOWN: " << tallyCategory("UNKNOWN") << std::endl;
299   std::cout << "UNDEAD: " << tallyCategory("UNDEAD") << std::endl;
300   std::cout << "MYSTICAL: " << tallyCategory("MYSTICAL") << std::endl;
301   std::cout << "ALIEN: " << tallyCategory("ALIEN") << std::endl;
302   std::cout << std::endl;
303
304   std::cout << "AVERAGE LEVEL: " << calculateAvgLevel() << std::endl;
305   std::cout << "TAME: " << calculateTamePercentage() << "%" << std::endl;
306 }
307
308 void Cavern::displayCreatures() const {
309   for (int i = 0; i < getCurrentSize(); i++) {
310     items_[i]->display();
311   }
312 }
313
314 void Cavern::displayCategory(const std::string& category) const {
315   if(category != "UNKNOWN" && category != "UNDEAD" && category != "MYSTICAL" && category !=
    "ALIEN") {
316     return;
317   }
318   for (int i = 0; i < getCurrentSize(); i++) {
319     if (items_[i]->getCategory() == category) {
320       items_[i]->display();
321     }
322   }
323 }
324
325 void Cavern::mycoMorselFeast() {
326   int size = getCurrentSize();
327   for (int i = 0; i < size; i++) {
328     if(items_[i]->eatMycoMorsel()) {
329       exitCavern(items_[i]);
330     }
331   }
332 }
333 /**
334   * @post: Stores the ALIEN Creatures of highest level in the cavern's alien stack, in the order in
    which they appear in the Cavern (i.e., starting from index 0 in items_, thus, if the highest level is 5
    and there are 3 ALIEN creatures with level 5, the one with lowest index in items_ is at the bottom of
    the stack and the one with highest index in item_ is at the top of the stack, with a total of 3 ALIEN
    Creatures on the stack)
335
336   *      : Empty the stack before beginning.
337   */
338 void Cavern::initializeAlienStack(){
339   clearAlienStack();
340   int highestlvl = 0;
```

```
341   for(int i = 0; i <getCurrentSize(); i ++){
342     if(items_[i]->getCategory() == "ALIEN"){
343       if(items_[i]->getLevel() > highestlvl){
344         highestlvl = items_[i]->getLevel();
345       }
346     }
347   }
348   for(int j =0; j <getCurrentSize(); j++){
349     if(items_[j]->getLevel() == highestlvl && items_[j]->getCategory() == "ALIEN"){
350       alien_stack_.push(items_[j]);
351     }
352   }
353 }
354 /**
355   * @post: Stores the UNDEAD Creatures of highest level in the cavern's undead stack, in the order
      in which they appear in the Cavern (i.e., starting from index 0 in items_, thus, if the highest level is
      5 and there are 3 UNDEAD creatures with level 5, the one with lowest index in items_ is at the
      bottom of the stack and the one with highest index in item_ is at the top of the stack, with a total of
      3 UNDEAD Creatures on the stack)
356         : Empty the stack before beginning.
357   */
358 void Cavern::initializeUndeadStack(){
359     clearUndeadStack();
360    int highestlvl = 0;
361   for(int i = 0; i <getCurrentSize(); i ++){
362     if(items_[i]->getCategory() == "UNDEAD"){
363       if(items_[i]->getLevel() > highestlvl){
364         highestlvl = items_[i]->getLevel();
365       }
366     }
367   }
368   for(int j =0; j <getCurrentSize(); j++){
369     if(items_[j]->getLevel() == highestlvl && items_[j]->getCategory() == "UNDEAD"){
370       undead_stack_.push(items_[j]);
371     }
372   }
373 }
374 /**
375   * @post: Stores the MYSTICAL Creatures of highest level in the cavern's mystical stack, in the order
      in which they appear in the Cavern (i.e., starting from index 0 in items_, thus, if the highest level is
      5 and there are 3 MYSTICAL creatures with level 5, the one with lowest index in items_ is at the
      bottom of the stack and the one with highest index in item_ is at the top of the stack, with a total of
      3 MYSTICAL Creatures on the stack)
376   *       : Empty the stack before beginning.
377   */
378 void Cavern::initializeMysticalStack(){
379   clearMysticalStack();
380    int highestlvl = 0;
381   for(int i = 0; i <getCurrentSize(); i ++){
382     if(items_[i]->getCategory() == "MYSTICAL"){
383       if(items_[i]->getLevel() > highestlvl){
384         highestlvl = items_[i]->getLevel();
```

```
385       }
386     }
387   }
388   for(int j =0; j <getCurrentSize(); j++){
389     if(items_[j]->getLevel() == highestlvl && items_[j]->getCategory() == "MYSTICAL"){
390       mystical_stack_.push(items_[j]);
391     }
392   }
393 }
394 /**
395   * @post: Stores the UNKNOWN Creatures of highest level in the cavern's unknown stack, in the order in which they appear in the Cavern (i.e., starting from index 0 in items_, thus, if the highest level is 5 and there are 3 UNKNOWN creatures with level 5, the one with lowest index in items_ is at the bottom of the stack and the one with highest index in item_ is at the top of the stack, with a total of 3 UNKNOWN Creatures on the stack)
396   *      : Empty the stack before beginning.
397   */
398 void Cavern::initializeUnknownStack(){
399    clearUnknownStack();
400    int highestlvl = 0;
401   for(int i = 0; i <getCurrentSize(); i ++){
402     if(items_[i]->getCategory() == "UNKNOWN"){
403       if(items_[i]->getLevel() > highestlvl){
404         highestlvl = items_[i]->getLevel();
405       }
406     }
407   }
408   for(int j =0; j <getCurrentSize(); j++){
409     if(items_[j]->getLevel() == highestlvl && items_[j]->getCategory() == "UNKNOWN"){
410       unknown_stack_.push(items_[j]);
411     }
412   }
413 }
414 /**
415   * @return: A copy of the stack of highest level Aliens in the cavern
416   */
417 std::stack<Creature*> Cavern::getAlienStack() const{
418   return alien_stack_;
419 }
420 /**
421   * @return: A copy of the stack of highest level Undeads in the cavern
422   */
423 std::stack<Creature*> Cavern::getUndeadStack() const{
424   return undead_stack_;
425 }
426 /**
427   * @return: A copy of the stack of highest level Mysticals in the cavern
428   */
429 std::stack<Creature*> Cavern::getMysticalStack() const{
430   return mystical_stack_;
431 }
432 /**
```

```
433   * @return: A copy of the stack of highest level Unknowns in the cavern
434   */
435 std::stack<Creature*> Cavern::getUnknownStack() const{
436   return unknown_stack_;
437 }
438 /**
439   * @post: clears the stack of highest level Aliens in the cavern
440   */
441 void Cavern::clearAlienStack(){
442   while(!(alien_stack_.empty())){
443     alien_stack_.pop();
444   }
445 }
446 /**
447   * @post: clears the stack of highest level Undeads in the cavern
448   */
449 void Cavern::clearUndeadStack(){
450   while(!(undead_stack_.empty())){
451     undead_stack_.pop();
452   }
453 }
454 /**
455   * @post: clears the stack of highest level Mysticals in the cavern
456   */
457 void Cavern::clearMysticalStack(){
458     while(!(mystical_stack_.empty())){
459     mystical_stack_.pop();
460   }
461 }
462 /**
463   * @post: clears the stack of highest level Unknowns in the cavern
464   */
465 void Cavern::clearUnknownStack(){
466     while(!(unknown_stack_.empty())){
467     unknown_stack_.pop();
468   }
469 }
470
471 /**
472   * @param: A stack of creature pointers
473   * @pre:   All the creature on the input stack are of same category and same (highest) level
474   * @post:  For each creature in the stack, rebuild the Cavern's appropriate stack. (For example, if
      the creatures given are of category ALIEN, this function should build the Cavern's Alien stack.)
475   *        Clear the Cavern's stack of the given category before adding the creatures to the stack.
476   *        Before adding each creature to the Cavern's stack, prompt the user to select 2 attacks for the
      creature.
477   *        Preserve the order of the creatures in the stack given. (E.g. The creature at the top of the
      given stack should also become the creature at the top of the Cavern's stack)
478   *        If the input is invalid (valid inputs will be 1,2 or 3 only), keep prompting for a non-negative
      number that is within range, by printing "INVALID INPUT. TRY AGAIN.\n" and prompt for input
      again.
479   *        When a valid action is read, it is passed to the creature's addAttack function to add the
```

```
corresponding attack to the creature's attack queue.
480  *       Prompting for attacks should be done in the following form (hint: use the creature's
displayAttacks function):
481  *       SELECT 2 ATTACKS FOR [CREATURE NAME]
482           [[CREATURE TYPE]] Choose an attack (1-3):
483  *       1: [ATTACK 1 NAME]\t\t2: [ATTACK 2 NAME]\t\t3: [ATTACK 3 NAME]
484           [user input]
485  *       [[CREATURE TYPE]] Choose an attack (1-3):
486  *       1: [ATTACK 1 NAME]\t\t2: [ATTACK 2 NAME]\t\t3: [ATTACK 3 NAME]
487  */
488 void Cavern::setAttacks(std::stack<Creature*> attack){
489   if(attack.size()!= 0){// Run if attack is not size 0 since can't dereference nullptr Segfault
490     std::stack<Creature*> TEMP;// Temp creature to add to the stack later
491     std::string category = attack.top()->getCategory();// Category of the stack
492       if(category == "ALIEN"){
493         clearAlienStack();
494       }
495       else if(category == "UNDEAD"){
496           clearUndeadStack();
497       }
498       else if(category == "MYSTICAL"){    //CLEAR STACK BASED ON THE CATEGORY OF THE STACK using
category for compariosn
499         clearMysticalStack();
500       }
501       else if(category == "UNKNOWN"){
502         clearUnknownStack();
503       }// For Clearing purposes
504
505   while(!(attack.empty())){// While the stack isnt empty
506     int num1;
507     std::cout << "SELECT 2 ATTACKS FOR " << attack.top()->getName() << "\n"; // ASK FOR 2 NUMBERS
508     attack.top()->displayAttacks();
509     std::cin >> num1;
510     while(num1 < 1 || num1 > 3){
511       std::cout <<"INVALID INPUT. TRY AGAIN.\n";
512       std::cin >> num1;
513     }
514     int num2;                                          //Repeatly ask if they give invaild number
515     attack.top()->displayAttacks();
516     std::cin >> num2;
517     while(num2 < 1 || num2 > 3){
518       std::cout <<"INVALID INPUT. TRY AGAIN.\n";
519       std::cin >> num2;
520     }
521       attack.top()->addAttack(num1);    // call addattack for these two numbers given
522       attack.top()->addAttack(num2);
523       TEMP.push(attack.top());  // push them to TEMP and pop out
524       attack.pop();
525       if(attack.empty()){
526         break; // stop the loop when attack is empty
527       }
528   }
```

```
529     while(!(TEMP.empty())){ // add the creatures to TEMP to their respective stack and order is kept
530       if(category== "ALIEN"){
531         alien_stack_.push(TEMP.top());
532       }
533       if(category == "UNDEAD"){
534         undead_stack_.push(TEMP.top());
535       }
536       if(category == "MYSTICAL"){
537         mystical_stack_.push(TEMP.top());
538       }
539       if(category == "UNKNOWN"){
540         unknown_stack_.push(TEMP.top());
541       }
542       TEMP.pop();//POP after adding to the stack
543     }
544   }
545 }
```

Cavern.hpp Download

```
/*
CSCI235 Spring 2024
Project 4 - MycoMorsels
Georgina Woo
Dec 24 2023
Cavern.hpp declares the Cavern class along with its private and public members
*/
#ifndef CAVERN_HPP
#define CAVERN_HPP

#include "ArrayBag.hpp"
#include "Creature.hpp"
#include "Dragon.hpp"
#include "Ghoul.hpp"
#include "Mindflayer.hpp"

#include <vector>
#include <iostream>
#include <cmath>
#include <iomanip>
#include <fstream>
#include <sstream>
#include <string>


class Cavern : public ArrayBag<Creature*>{
  public:
    /**
      Default constructor.
      Default-initializes all private members.
    */
    Cavern();

    /**
        @param  : the name of an input file
        @pre    : Formatting of the csv file is as follows (each numbered item appears separated by comma, only one value for each numbered item):
                1. TYPE: An uppercase string [DRAGON, GHOUL, MINDFLAYER]
                2. NAME: An uppercase string
                3. CATEGORY: An uppercase string [ALIEN, MYSTICAL, UNDEAD]
                4. HITPOINTS: A positive integer
                5. LEVEL: A positive integer
                6. TAME: 0 (False) or 1 (True)
                7. ELEMENT/FACTION: Uppercase string or strings representing the ELEMENT (For Dragons), or FACTION (For Ghouls) of the creature. If the creature is of a different subclass, the value will be NONE
                8. HEADS: A positive integer of the number of heads the Dragon has. If the creature is of a different subclass, the value will be 0
                9. FLIGHT/TRANSFORM/SUMMONING: 0 (False) or 1 (True) representing if the creature can
```

```
fly (Dragons), transform (Ghouls), or summon a Thoughtspawn (Mindflayers).
46              10. DECAY: A non-negative integer representing the level of decay of the Ghoul. If the creature is of a different subclass, the value will be 0
47              11. AFFINITIES: Only applicable to Mindflayers. Affinities are of the form [AFFINITY1];[AFFINITY2] where multiple affinities are separated by a semicolon. Th value may be NONE for a Mindflayer with no affinities, or creatures of other subclasses.
48              12. PROJECTILES: Only applicable to Mindflayers. PROJECTILES are of the form [PROJECTILE TYPE1]-[QUANTITY];[PROJECTILE TYPE 2]-[QUANTITY] where multiple types of projectiles are separated by a semicolon. The value may be NONE for a Mindflayer with no projectiles, or creatures of other subclasses.
49      */
50      Cavern(const std::string& filename);
51
52      /**
53       * @param   :   A Creature entering the Cavern
54       * @post    :   If the given Creature is not already in the Cavern, add Creature to the Cavern and updates the level sum and the tame Creature count if the creature is tame.
55       * @return  :   returns true if a Creature was successfully added to the Cavern (i.e. items_), false otherwise
56                  :   Hint: Use the above definition of equality will help determine if a Creature is already in the Cavern
57      **/
58      bool enterCavern(Creature* new_creature);
59
60
61      /**
62       * @param   :   A Creature leaving the Cavern
63       * @return  :  returns true if a creature was successfully removed from the Cavern (i.e. items_), false otherwise
64       * @post    :    removes the creature from the Cavern and updates the level sum and the tame count if the creature is tame.
65      **/
66      bool exitCavern(Creature* creature_to_remove);
67
68      /**
69      * @return   :  The integer level count of all the creatures currently in the Cavern
70      **/
71      int getLevelSum() const;
72
73      /**
74      * @return   :  The average level of all the creatures in the Cavern
75      * @post     :    Computes the average level of the Cavern rounded to the NEAREST integer.
76      **/
77      int calculateAvgLevel() const;
78
79      /**
80      * @return   :  The integer count of tame Creatures in the Cavern
81      **/
82      int getTameCount() const;
83
84      /**
85      * @return   :  The percentage (double) of all the tame creatures in the Cavern
```

```
86      * @post    :   Computes the percentage of tame creatures in the Cavern rounded up to 2 decimal places.
87      **/
88      double calculateTamePercentage() const;
89
90      /**
91       * @param  :   A string representing a creature Category with value in
92                  ["UNKNOWN", "UNDEAD", "MYSTICAL", "ALIEN"]
93       * @return  :  An integer tally of the number of creatures in the Cavern of the given category.
94                  If the argument string does not match one of the expected category values,
95                  the tally is zero.
96                  NOTE: no pre-processing of the input string necessary, only uppercase input will match.
97      **/
98      int tallyCategory(const std::string& category) const;
99
100     /**
101        @param  :   An integer representing the level treshold of the creatures to be removed from the Cavern, with default value 0
102        @post   :   Removes all creatures from the Cavern whose level is less than the given level. If no level is given, removes all creatures from the Cavern. Ignore negative input.
103        @return :   The number of creatures removed from the Cavern
104     */
105     int releaseCreaturesBelowLevel(int level = 0);
106
107     /**
108        @param  : A string representing a creature Category with a value in ["UNKNOWN", "UNDEAD", "MYSTICAL", "ALIEN"], or default value "ALL" if no category is given
109        @post   : Removes all creatures from the Cavern whose category matches the given category. If no category is given, removes all creatures from the Cavern.
110        @return : The number of creatures removed from the Cavern
111                  NOTE: no pre-processing of the input string necessary, only uppercase input will match. If the input string does not match one of the expected category values, do not remove any creatures.
112     */
113     int releaseCreaturesOfCategory(std::string category = "ALL");
114
115     /**
116      * @post    : Outputs a report of the creatures currently in the Cavern in the form:
117                  "UNKNOWN: [x]\nUNDEAD: [x]\nMYSTICAL: [x]\nALIEN: [x]\n\nThe average level is: [x]\n[x]% are tame.\n"
118                  Note that the average level should be rounded to the NEAREST integer, and the percentage of tame creatures in the Cavern should be rounded to 2 decimal places.
119
120                 Example output:
121                 UNKNOWN: 3
122                 UNDEAD: 5
123                 MYSTICAL: 8
124                 ALIEN: 6
125
126                 AVERAGE LEVEL: 7
127                 TAME: 46.67%
128     */
```

```
129     void cavernReport() const;
130
131
132     /**
133         @post: For every creature in the cavern, displays each creature's information
134     */
135     void displayCreatures() const;
136
137     /**
138         @param: a string reference to a category
139         @post: For every creature in the cavern of the given category (only exact matches to the input
    string), displays each creature's information
140
141     */
142     void displayCategory(const std::string& category) const;
143
144     /**
145         @post: Every creature in the cavern eats a MycoMorsel.
146     */
147     void mycoMorselFeast();
148 /**
149   * @post: Stores the ALIEN Creatures of highest level in the cavern's alien stack, in the order in
    which they appear in the Cavern (i.e., starting from index 0 in items_, thus, if the highest level is 5
    and there are 3 ALIEN creatures with level 5, the one with lowest index in items_ is at the bottom of
    the stack and the one with highest index in item_ is at the top of the stack, with a total of 3 ALIEN
    Creatures on the stack)
150
151   *      : Empty the stack before beginning.
152   */
153     void initializeAlienStack();
154 /**
155   * @post: Stores the UNDEAD Creatures of highest level in the cavern's undead stack, in the order
    in which they appear in the Cavern (i.e., starting from index 0 in items_, thus, if the highest level is
    5 and there are 3 UNDEAD creatures with level 5, the one with lowest index in items_ is at the
    bottom of the stack and the one with highest index in item_ is at the top of the stack, with a total of
    3 UNDEAD Creatures on the stack)
156         : Empty the stack before beginning.
157   */
158     void initializeUndeadStack();
159 /**
160   * @post: Stores the MYSTICAL Creatures of highest level in the cavern's mystical stack, in the order
    in which they appear in the Cavern (i.e., starting from index 0 in items_, thus, if the highest level is
    5 and there are 3 MYSTICAL creatures with level 5, the one with lowest index in items_ is at the
    bottom of the stack and the one with highest index in item_ is at the top of the stack, with a total of
    3 MYSTICAL Creatures on the stack)
161   *      : Empty the stack before beginning.
162   */
163     void initializeMysticalStack();
164 /**
165   * @post: Stores the UNKNOWN Creatures of highest level in the cavern's unknown stack, in the
    order in which they appear in the Cavern (i.e., starting from index 0 in items_, thus, if the highest
    level is 5 and there are 3 UNKNOWN creatures with level 5, the one with lowest index in items_ is at
```

```
the bottom of the stack and the one with highest index in item_ is at the top of the stack, with a total of 3 UNKNOWN Creatures on the stack)
166   *      : Empty the stack before beginning.
167   */
168     void initializeUnknownStack();
169 /**
170   * @return: A copy of the stack of highest level Aliens in the cavern
171   */
172     std::stack<Creature*> getAlienStack() const;
173 /**
174   * @return: A copy of the stack of highest level Undeads in the cavern
175   */
176     std::stack<Creature*> getUndeadStack() const;
177 /**
178   * @return: A copy of the stack of highest level Mysticals in the cavern
179   */
180     std::stack<Creature*> getMysticalStack() const;
181 /**
182   * @return: A copy of the stack of highest level Unknowns in the cavern
183   */
184     std::stack<Creature*> getUnknownStack() const;
185 /**
186   * @post: clears the stack of highest level Aliens in the cavern
187   */
188     void clearAlienStack();
189 /**
190   * @post: clears the stack of highest level Undeads in the cavern
191   */
192     void clearUndeadStack();
193 /**
194   * @post: clears the stack of highest level Mysticals in the cavern
195   */
196     void clearMysticalStack();
197 /**
198   * @post: clears the stack of highest level Unknowns in the cavern
199   */
200     void clearUnknownStack();
201 /**
202   * @param: A stack of creature pointers
203   * @pre:   All the creature on the input stack are of same category and same (highest) level
204   * @post:  For each creature in the stack, rebuild the Cavern's appropriate stack. (For example, if the creatures given are of category ALIEN, this function should build the Cavern's Alien stack.)
205   *       Clear the Cavern's stack of the given category before adding the creatures to the stack.
206   *       Before adding each creature to the Cavern's stack, prompt the user to select 2 attacks for the creature.
207   *       Preserve the order of the creatures in the stack given. (E.g. The creature at the top of the given stack should also become the creature at the top of the Cavern's stack)
208   *       If the input is invalid (valid inputs will be 1,2 or 3 only), keep prompting for a non-negative number that is within range, by printing "INVALID INPUT. TRY AGAIN.\n" and prompt for input again.
209   *       When a valid action is read, it is passed to the creature's addAttack function to add the corresponding attack to the creature's attack queue.
```

```
210   *       Prompting for attacks should be done in the following form (hint: use the creature's
      displayAttacks function):
211   *       SELECT 2 ATTACKS FOR [CREATURE NAME]
212           [[CREATURE TYPE]] Choose an attack (1-3):
213   *       1: [ATTACK 1 NAME]\t\t2: [ATTACK 2 NAME]\t\t3: [ATTACK 3 NAME]
214           [user input]
215   *       [[CREATURE TYPE]] Choose an attack (1-3):
216   *       1: [ATTACK 1 NAME]\t\t2: [ATTACK 2 NAME]\t\t3: [ATTACK 3 NAME]
217   */
218     void setAttacks(std::stack<Creature*> attack);
219
220
221
222
223   private:
224     int level_sum_;     // sum of all the levels of the creatures in the cavern
225     int tame_count_;    // number of tame creatures in the cavern
226     std::stack<Creature*> alien_stack_;
227     std::stack<Creature*> undead_stack_;
228     std::stack<Creature*> mystical_stack_;
229     std::stack<Creature*> unknown_stack_;
230
231
232 };
233 #endif
234
```

Cavern.o Download

```
1   Large file hidden. You can download it using the button above.
```

Creature.cpp Download

```cpp
/*
CSCI235 Spring 2024
Project 4 - MycoMorsels
Georgina Woo
Dec 24 2023
Creature.hpp declares the Creature class along with its private and public members
*/

#include "Creature.hpp"

/**
   Default constructor.
   Default-initializes all private members.
   Default creature name: "NAMELESS".
   Booleans are default-initialized to False.
   Default enum value: UNKNOWN
   Default Hitpoints and Level: 1.
*/
Creature::Creature(): name_{"NAMELESS"},  category_{UNKNOWN}, hitpoints_{1}, level_{1}, tame_{false}
{

}

/**
   Parameterized constructor.
   @param     : A reference to the name of the creature (a string). Set the creature's name to NAMELESS if the provided string contains non-alphabetic characters.
   @param     : The category of the creature (a Category enum) with default value UNKNOWN
   @param     : The creature's hitpoints (an integer) , with default value 1 if not provided, or if the value provided is 0 or negative
   @param     : The creature's level (an integer), with default value 1 if not provided, or if the value provided is 0 or negative
   @param     : A flag indicating whether the creature is tame, with default value False
   @post      : The private members are set to the values of the corresponding parameters.
   Hint: Notice the default arguments in the parameterized constructor.
*/
Creature::Creature(const std::string& name, Category category, int hitpoints, int level, bool tame): category_{category}
{
   if(!setName(name))
   {
      name_ = "NAMELESS";
   }

   if(!setHitpoints(hitpoints))
   {
      hitpoints_ = 1;
   }
```

```
45      if(!setLevel(level))
46      {
47          level_ = 1;
48      }
49      tame_ = tame;
50
51  }
52
53  /**
54      @param : the name of the Creature, a reference to string
55      @post  : sets the Creature's name to the value of the parameter in UPPERCASE.
56               (convert any lowercase character to uppercase)
57               Only alphabetical characters are allowed.
58             : If the input contains non-alphabetic characters, do nothing.
59      @return : true if the name was set, false otherwise
60  */
61  bool Creature::setName(const std::string& name)
62  {
63      if (name.length() == 0)
64      {
65          return false;
66      }
67      else
68      {
69          std::string nameUpper = name;
70          for (int i = 0; i < name.length(); i++)
71          {
72              if (!isalpha(name[i]))
73              {
74                  return false;
75              }
76              else
77              {
78                  nameUpper[i] = toupper(name[i]);
79              }
80          }
81          name_ = nameUpper;
82          return true;
83      }
84  }
85
86  /**
87       @return : the name of the Creature
88  */
89  std::string Creature::getName() const
90  {
91      return name_;
92  }
93
94
95  /**
96      @param  : the category of the Creature (an enum)
```

```
97      @post   : sets the Creature's category to the value of the parameter
98  */
99  void Creature::setCategory(const Category& category)
100 {
101     category_ = category;
102 }
103
104
105 /**
106      @return : the category of the Creature (in string form)
107 */
108 std::string Creature::getCategory() const
109 {
110     switch(category_)
111     {
112         case UNDEAD:
113             return "UNDEAD";
114         case MYSTICAL:
115             return "MYSTICAL";
116         case ALIEN:
117             return "ALIEN";
118         default:
119             return "UNKNOWN";
120     }
121 }
122
123 /**
124     @param  : an integer that represents the creature's hitpoints
125     @pre    : hitpoints > 0 : Creatures cannot have 0 or negative hitpoints (do nothing for invalid
    input)
126     @post   : sets the hitpoints private member to the value of the parameter
127     @return : true if the hitpoints were set, false otherwise
128 */
129 bool Creature::setHitpoints(const int& hitpoints)
130 {
131     if (hitpoints > 0)
132     {
133         hitpoints_ = hitpoints;
134         return true;
135     }
136     else
137     {
138         return false;
139     }
140 }
141
142
143 /**
144      @return : the value stored in hitpoints_
145 */
146 int Creature::getHitpoints() const
147 {
```

```
148     return hitpoints_;
149 }
150
151 /**
152     @param  : an integer level
153     @pre    : level >  0 : Characters cannot have 0 or negative levels (do nothing for invalid input)
154     @post   : sets the level private member to the value of the parameter
155     @return : true if the level was set, false otherwise
156 */
157 bool Creature::setLevel(const int& level)
158 {
159     if (level > 0)
160     {
161         level_ = level;
162         return true;
163     }
164     else
165     {
166         return false;
167     }
168 }
169
170
171 /**
172      @return : the value stored in level_
173 */
174 int Creature::getLevel() const
175 {
176     return level_;
177 }
178
179
180 /**
181     @param  : a boolean value
182     @post   : sets the tame flag to the value of the parameter
183 */
184 void Creature::setTame(const bool& tame)
185 {
186     tame_ = tame;
187 }
188
189
190 /**
191     @return true if the creature is tame, false otherwise
192     Note: this is an accessor function and must follow the same convention as all accessor functions
    even if it is not called getTame
193 */
194 bool Creature::isTame() const
195 {
196     return tame_;
197 }
198
```

```cpp
/**
   @post     : displays Creature data in the form:
   "[NAME]\n
   Category: [CATEGORY]\n
   Level: [LEVEL]\n
   Hitpoints: [Hitpoints]\n
   Tame: [TRUE/FALSE]"
*/
// void Creature::display() const
// {
//     std::cout << name_ << std::endl;
//     std::cout << "Category: " << getCategory() << std::endl;
//     std::cout << "Level: " << level_ << std::endl;
//     std::cout << "Hitpoints: " << hitpoints_ << std::endl;
//     std::cout << "Tame: " << (tame_ ? "TRUE" : "FALSE") << std::endl;
// }

bool Creature::operator==(const Creature& other_creature) const
{
    return (name_ == other_creature.name_ && category_ == other_creature.category_ && level_ == other_creature.level_ && tame_ == other_creature.tame_);
}

bool Creature::operator!=(const Creature& other_creature) const
{
    return !(*this == other_creature);
}

/**
  * @return a copy of the attackQueue
*/
std::queue<Attack> Creature::getAttackQueue() const{
    return attack_queue_;
}

/**
  * @param: A const reference to int indicating the attack to be added to the queue.
  * Pure virtual function to be implemented by the derived classes
  * virtual void addAttack(const int &attack) =0;
 */

void Creature::addAttack(const Attack &attack){
    attack_queue_.push(attack);
}
/**
  * @post: the attackQueue is emptied
*/
void Creature::clearAttackQueue(){
    while(!attack_queue_.empty()){
        attack_queue_.pop();
    }
}
```

Creature.hpp Download

```
/*
CSCI235 Spring 2024
Project 4 - MycoMorsels
Georgina Woo
Dec 24 2023
Creature.hpp declares the Creature class along with its private and public members
*/
#ifndef CREATURE_HPP_
#define CREATURE_HPP_
#include <iostream>
#include <string>
#include <cctype>
#include <vector>
#include <stack>
#include <queue>


struct Attack
{
   std::string name_;
   std::vector<std::string> type_;
   std::vector<int> damage_;
};

class Creature
{
   public:
      enum Category {UNKNOWN, UNDEAD, MYSTICAL, ALIEN};
      /**
         Default constructor.
         Default-initializes all private members.
         Default creature name: "NAMELESS".
         Booleans are default-initialized to False.
         Default enum value: UNKNOWN
         Default Hitpoints and Level: 1.
      */
      Creature();

      /**
         Parameterized constructor.
         @param     : The name of the creature (a string)
         @param     : The category of the creature (a Category enum) with default value UNKNOWN
         @param     : The creature's hitpoints (an integer), with default value 1 if not provided, or if the value provided is 0 or negative
         @param     : The creature's level (an integer), with default value 1 if not provided, or if the value provided is 0 or negative
         @param     : A flag indicating whether the creature is tame, with default value False
         @post      : The private members are set to the values of the corresponding parameters.
         Hint: Notice the default arguments in the parameterized constructor.
```

```
48      */
49      Creature(const std::string& name, Category category = UNKNOWN, int hitpoints = 1, int level =
        1, bool tame = false);
50
51      /**
52          @param : the name of the Creature, a string
53          @post  : sets the Creature's name to the value of the parameter in UPPERCASE (convert any
        lowercase character to upppercase
54                   Only alphabetical characters are allowed.
55                 : If the input contains non-alphabetic characters, do nothing.
56          @return : true if the name was set, false otherwise
57      */
58      bool setName(const std::string& name);
59
60      /**
61          @return : the name of the Creature
62      */
63      std::string getName() const;
64
65
66      /**
67          @param  : the category of the Creature (an enum)
68          @post   : sets the Creature's category to the value of the parameter
69      */
70      void setCategory(const Category& category);
71
72
73      /**
74          @return : the race of the Creature (in string form)
75      */
76      std::string getCategory() const;
77
78      /**
79          @param  : an integer that represents the creature's hitpoints
80          @pre    : hitpoints > 0 : Creatures cannot have 0 or negative hitpoints (do nothing for invalid
        input)
81          @post   : sets the hitpoints private member to the value of the parameter
82          @return : true if the hitpoints were set, false otherwise
83      */
84      bool setHitpoints(const int& hitpoints);
85
86
87      /**
88          @return : the value stored in hitpoints_
89      */
90      int getHitpoints() const;
91
92      /**
93          @param  : an integer level
94          @pre    : level > 0 : Creatures cannot have 0 or negative levels (do nothing for invalid input)
95          @post   : sets the level private member to the value of the parameter
96          @return : true if the level was set, false otherwise
```

```
97        */
98        bool setLevel(const int& level);
99
100
101       /**
102           @return : the value stored in level_
103       */
104       int getLevel() const;
105
106
107       /**
108          @param  : a boolean value
109          @post   : sets the tame flag to the value of the parameter
110       */
111       void setTame(const bool& tame);
112
113
114       /**
115          @return true if the Creature is tame, false otherwise
116          Note: this is an accessor function and must follow the same convention as all accessor
      functions even if it is not called getTame
117       */
118       bool isTame() const;
119
120        /**
121          @post     : displays Creature data in the form:
122          "[NAME]\n
123          Category: [CATEGORY]\n
124          Level: [LEVEL]\n
125          Hitpoints: [Hitpoints]\n
126          Tame: [TRUE/FALSE]"
127       */
128       virtual void display() const = 0;
129
130
131       /**
132        @param        : A const reference to the right hand side of the == operator.
133       @return     : Returns true if the right hand side creature is "equal", false otherwise.
134                              Two creatures are equal if they have the same name, same category, same
      level, and if they're tame or not
135                                 NOTE: By this definition, only the aforementioned subset of the creature's
      attributes must be equal for two creatures to be deemed "equal".
136
137       Example: In order for creature1 to be == to creature2 we only need:
138       - The same name
139       - The same category
140       - The same level
141       - They must either be both tame or not
142       */
143       bool operator==(const Creature& rhs) const;
144
145       /**
```

```
146         @param      : A const reference to the right hand side of the != operator.
147         @return     : Returns true if the right hand side creature is NOT "equal" (!=), false
148                               otherwise. Two creatures are NOT equal if any of their name, category or
    level are
149                               not equal, or if one is tame and the other is not.
150                               NOTE: By this definition, one or more of the aforementioned subset of the
    creature's attributes only must be different for two creatures to be
151                               deemed "NOT equal".
152         */
153         bool operator!=(const Creature& rhs) const;
154
155
156         /**
157           @post     : Modifies the creature's private member variables (the exact modifications will be
    subclass specific)
158           @return   : true if the creature leaves the Cavern, false otherwise
159         */
160         virtual bool eatMycoMorsel() = 0;
161         /**
162          * @return a copy of the attackQueue
163         */
164         std::queue<Attack> getAttackQueue() const;
165         /**
166          * @param: A const reference to int indicating the attack to be added to the queue.
167          * Pure virtual function to be implemented by the derived classes
168          */
169         virtual void addAttack(const int &attack) = 0;
170         /**
171          * @param: A const reference to attack to be added to the queue.
172          * @post: The attack is added to the queue
173          */
174         void addAttack(const Attack &attack);
175         /**
176          * @post: the attackQueue is emptied
177         */
178         void clearAttackQueue();
179         /**
180          * @post: Displays the options for the attacks
181          * Pure virtual function to be implemented by the derived classes
182         */
183         virtual void displayAttacks() = 0;
184
185       private:
186         // The name of the creature (a string in UPPERCASE)
187         std::string name_;
188         // The category of the creature (an enum)
189         Category category_;
190         // The creature's hitpoints (a non-zero, non-negative integer)
191         int hitpoints_;
192         // The creature's level (a non-zero, non-negative integer)
193         int level_;
194         // A flag indicating whether the creature is tame
```

```
195         bool tame_;
196 
197         std::queue<Attack> attack_queue_;
198 
199 
200 };
201 #endif
```

Creature.o Download

```
1   Large file hidden. You can download it using the button above.
```

Dragon.cpp Download

```cpp
/*
CSCI235 Spring 2024
Project 4 - MycoMorsels
Georgina Woo
Dec 24 2023
Dragon.cpp implements the constructors and private and public functions of the Dragon class
*/

#include "Dragon.hpp"

Dragon::Dragon() : element_{NONE}, number_of_heads_{1}, flight_{false}
{
    setCategory(MYSTICAL);

}


Dragon::Dragon(const std::string& name, Category category, int hitpoints, int level, bool tame, Element element, int number_of_heads, bool flight) : Creature(name, category, hitpoints, level, tame)
{
    element_ = element;
    if(!setNumberOfHeads(number_of_heads))
    {
        number_of_heads_ = 1;
    }
    flight_ = flight;
}

std::string Dragon::getElement() const
{
    switch(element_)
    {
        case FIRE:
            return "FIRE";
        case WATER:
            return "WATER";
        case EARTH:
            return "EARTH";
        case AIR:
            return "AIR";
        default:
            return "NONE";
    }
}

void Dragon::setElement(const Element& element)
{
    element_ = element;
```

```
48  }
49
50  int Dragon::getNumberOfHeads() const
51  {
52      return number_of_heads_;
53  }
54
55  bool Dragon::setNumberOfHeads(const int& number_of_heads)
56  {
57      if(number_of_heads > 0)
58      {
59          number_of_heads_ = number_of_heads;
60          return true;
61      }
62      else
63      {
64          return false;
65      }
66  }
67
68  bool Dragon::getFlight() const
69  {
70      return flight_;
71  }
72
73  void Dragon::setFlight(const bool& flight)
74  {
75      flight_ = flight;
76  }
77
78
79  void Dragon::display() const
80  {
81      std::cout << "DRAGON - " << getName() << std::endl;
82      std::cout << "CATEGORY: " << getCategory() << std::endl;
83      std::cout << "HP: " << getHitpoints() << std::endl;
84      std::cout << "LVL: " << getLevel() << std::endl;
85      std::cout << "TAME: " << (isTame() ? "TRUE" : "FALSE") << std::endl;
86      std::cout << "ELEMENT: " << getElement() << std::endl;
87      std::cout << "HEADS: " << getNumberOfHeads() << std::endl;
88      std::cout << "IT " << (getFlight() ? "CAN" : "CANNOT") << " FLY" << std::endl;
89  }
90
91
92
93  bool Dragon::eatMycoMorsel()
94  {
95      if(getCategory() == "UNDEAD")
96      {
97          setTame(true);
98          setHitpoints(getHitpoints() + 1);
99          return false;
```

```
100     }
101     else if(getCategory() == "ALIEN")
102     {
103         setHitpoints(getHitpoints() + 1);
104         return false;
105     }
106     else if(getCategory() == "MYSTICAL")
107     {
108         if(getElement() == "FIRE" || getElement() == "EARTH")
109         {
110             setHitpoints(getHitpoints() + 1);
111             return false;
112         }
113         else if(getHitpoints() == 1)
114         {
115             return true;
116         }
117         else
118         {
119             setHitpoints(getHitpoints() - 1);
120             setTame(false);
121             return false;
122         }
123     }
124     else
125     {
126         return false;
127     }
128 }
129 
130 /**
131  * @param: A const reference to int corresponding to the attack to be added to the attack queue.
132  * @post: Adds an attack to the attack queue based on the int parameter.
133  * Here are the attacks for the Dragon:
134 
135 * 1: Attack name: BITE
136 *      if ALIEN: 4 PHYSICAL
137 *      if MYSTICAL: 2 PHYSICAL, and additional damage of 1 [FIRE/WATER/EARTH/AIR] if the Dragon has an elemental affinity of FIRE, WATER, EARTH, or AIR)
138 *      if UNDEAD: 2 PHYSICAL, 1 POISON
139 *      if UNKNOWN: 2 PHYSICAL
140 
141 * 2: Attack name: STOMP
142 *      if ALIEN: 2 PHYSICAL
143 *      if MYSTICAL: 1 PHYSICAL, and additional damage of 1 [FIRE/WATER/EARTH/AIR] if the Dragon has an elemental affinity of FIRE, WATER, EARTH, or AIR)
144 *      if UNDEAD: 1 PHYSICAL, 1 POISON
145 *      if UNKNOWN: 1 PHYSICAL
146 
147 * 3: Attack name: [ELEMENTAL BREATH/BAD BREATH], where the name is ELEMENTAL BREATH if the Dragon has an elemental affinity, otherwise it is BAD BREATH
148 *      if ALIEN: 6 [POISON/FIRE/WATER/EARTH/AIR], where the damage type is the Dragon's
```

```
elemental affinity if it has one, otherwise it is POISON
149 *      if MYSTICAL: 3 [POISON/FIRE/WATER/EARTH/AIR], where the damage type is the Dragon's
elemental affinity if it has one, otherwise it is POISON
150 *      if UNDEAD: 3 [POISON/FIRE/WATER/EARTH/AIR], 1 POISON. The damage types and amount are
added to the vector as two separate entries, even if both entries are POISON type.
151 *      if UNKNOWN: 3 [POISON/FIRE/WATER/EARTH/AIR] where the damage type is the Dragon's
elemental affinity if it has one, otherwise it is POISON
152 */
153 void Dragon::addAttack(const int &attack){
154     Attack Temp;
155     if(attack == 1){// if attack is 1
156         Temp.name_ = "BITE";// name default bite
157         if(getCategory() == "ALIEN"){//If Category is ALIEN then attack = 4 PHYSICAL
158             Temp.damage_.push_back(4);
159             Temp.type_.push_back("PHYSICAL");
160         }
161 
162         if(getCategory() == "MYSTICAL"){// If Category is MYSTICAL base Attack = 2 PHYSICAL
163             Temp.damage_.push_back(2);
164             Temp.type_.push_back("PHYSICAL");
165             if(element_ != NONE){//If Element is Not NONE add 1 and the element name
166                 Temp.damage_.push_back(1);
167                 Temp.type_.push_back(getElement());
168             }
169         }
170         if(getCategory() == "UNDEAD"){ // undead 2Physical and 1 poison
171             Temp.damage_.push_back(2);
172             Temp.type_.push_back("PHYSICAL");
173             Temp.damage_.push_back(1);
174             Temp.type_.push_back("POISON");
175         }
176         if(getCategory() == "UNKNOWN"){// if unknown 2 physcail
177             Temp.damage_.push_back(2);
178             Temp.type_.push_back("PHYSICAL");
179         }
180     }
181     if(attack == 2){// runs if attack == 2
182         Temp.name_ = "STOMP";// default name STOMP
183         if(getCategory() == "ALIEN"){// If aleiane default 2 PHYSICL
184             Temp.damage_.push_back(2);
185             Temp.type_.push_back("PHYSICAL");
186         }
187         if(getCategory() == "MYSTICAL"){//if mystical default 1 physcial
188             Temp.damage_.push_back(1);
189             Temp.type_.push_back("PHYSICAL");
190 
191             if(element_ != NONE){//If Element is Not NONE add 1 and the element name
192                 Temp.damage_.push_back(1);
193                 Temp.type_.push_back(getElement());
194             }
195         }
196         if(getCategory() == "UNDEAD"){ // if undead deafult 1physcail and 1 posion
```

```
197             Temp.damage_.push_back(1);
198             Temp.type_.push_back("PHYSICAL");
199             Temp.damage_.push_back(1);
200             Temp.type_.push_back("POISON");
201         }
202         if(getCategory() == "UNKNOWN"){ // if unknown 1 physcial
203             Temp.damage_.push_back(1);
204             Temp.type_.push_back("PHYSICAL");
205         }
206     }
207     if(attack == 3){// run if attack == 3
208         if(element_ == NONE){ // check the element of the creature. If NONE then the attack is named
    Bad breath
209             Temp.name_ = "BAD BREATH";
210         }
211         else{
212             Temp.name_ = "ELEMENTAL BREATH";// it not true then we know it has a elmenet so it is now
    ELEMENTAL breath
213         }
214         if(getCategory() == "ALIEN"){// If aleian defualt attack 6 type yet to be decalred
215             Temp.damage_.push_back(6);
216             if(getElement() == "NONE"){// check if there is a element if true then it is the element name
    else poison. WOuld be easier just checking the name of Temp.name
217                 Temp.type_.push_back("POISON");
218             }
219             else{
220                 Temp.type_.push_back(getElement());
221             }
222         }
223         if(getCategory() == "MYSTICAL"){// if mystical defualt 3 type undecalred
224             Temp.damage_.push_back(3);
225             if(getElement() == "NONE"){// check if there is a element if true then it is the element name
    else poison.
226                 Temp.type_.push_back("POISON");
227             }
228             else{
229             Temp.type_.push_back(getElement());
230             }
231         }
232         if(getCategory() == "UNDEAD"){//default 3 type undecalred same as previous
233             Temp.damage_.push_back(3);
234             if(getElement() == "NONE"){
235                 Temp.type_.push_back("POISON");
236             }
237             else
238             {
239                 Temp.type_.push_back(getElement());
240             }
241             Temp.damage_.push_back(1); // addiational attack
242             Temp.type_.push_back("POISON");
243         }
244         if(getCategory() == "UNKNOWN"){
```

```
245             Temp.damage_.push_back(3);
246             if(getElement() == "NONE"){
247                 Temp.type_.push_back("POISON");
248             }
249             else
250             {
251                 Temp.type_.push_back(getElement());
252             }
253         }
254     }
255     Creature::addAttack(Temp); // add these attack to the queue
256 }
257 
258 /**
259  * @post:  displays the attacks of the Dragon in the form:
260  *       [DRAGON] Choose an attack (1-3):\n1: BITE\t\t2: STOMP\t\t3: ELEMENTAL BREATH\n
261  */
262 void Dragon::displayAttacks(){
263     std::cout << "[DRAGON] Choose an attack (1-3):\n1: BITE\t\t2: STOMP\t\t3: ELEMENTAL BREATH\n";
264 }
```

Dragon.hpp Download

```
1  /*
2  CSCI235 Spring 2024
3  Project 4 - MycoMorsels
4  Georgina Woo
5  Dec 24 2023
6  Dragon.hpp defines the constructors and private and public functions of the Dragon class
7  */
8
9  #ifndef DRAGON_HPP
10 #define DRAGON_HPP
11
12 #include "Creature.hpp"
13
14
15
16 class Dragon : public Creature
17 {
18
19    public:
20
21       enum Element {NONE, FIRE, WATER, EARTH, AIR};
22
23       /**
24          Default constructor.
25          Default-initializes all private members.
26          Default Category: MYSTICAL
27          Default element: NONE
28          Default number of head(s): 1
29          Booleans are default-initialized to False.
30       */
31       Dragon();
32
33       /**
34          Parameterized constructor.
35          @param     : The name of the Dragon (a reference to string)
36          @param     : The category of the Dragon (a Category enum) with default value MYSTICAL
37          @param     : The Dragon's hitpoints (an integer), with default value 1 if not provided, or if
   the value provided is 0 or negative
38          @param     : The Dragon's level (an integer), with default value 1 if not provided, or if the
   value provided is 0 or negative
39          @param     : A flag indicating whether the Dragon is tame, with default value False
40          @param     : The element (an Element enum), with default value NONE if not provided
41          @param     : The number of heads (an integer), with default value 1 if not provided, or if the
   value provided is 0 or negative
42          @param     : A flag indicating whether the Dragon can fly, with default value False
43          @post      : The private members are set to the values of the corresponding parameters.
44          Hint: Notice the default arguments in the parameterized constructor.
45       */
46       Dragon(const std::string& name, Category category = MYSTICAL, int hitpoints = 1, int level = 1,
```

```
    bool tame = false, Element element = NONE, int number_of_heads = 1, bool flight = false);

     /**
        Getter for the element.
        @return     : The element (a string representation of the Element enum)
     */
     std::string getElement() const;

     /**
        Setter for the element.
        @param      : A reference to the element (an Element enum)
        @post       : The element is set to the value of the parameter.
     */
     void setElement(const Element& element);

     /**
        Getter for the number of heads.
        @return     : The number of heads (an integer)
     */
     int getNumberOfHeads() const;

     /**
        Setter for the number of heads.
        @param      : A reference to the number of heads (an integer)
        @pre        : The number of heads is > 0. Do nothing for invalid values.
        @post       : The number of heads is set to the value of the parameter.
        @return     : True if the number of heads is set, false otherwise.
     */
     bool setNumberOfHeads(const int& number_of_heads);

     /**
        Getter for the flight_ flag.
        @return     : The flight_ flag (a boolean)
     */
     bool getFlight() const;

     /**
        Setter for the flight_ flag.
        @param      : A reference to the flight flag (a boolean)
        @post       : The flight_ flag is set to the value of the parameter.
     */
     void setFlight(const bool& flight);



     // override the Creature display function
     /**
        @post     : displays Dragon data in the form:
        "DRAGON - [NAME]\n
        CATEGORY: [CATEGORY]\n
        HP: [HITPOINTS]\n
        LVL: [LEVEL]\n
```

```
98            TAME: [TAME]\n
99            ELEMENT: [ELEMENT]\n
100           HEADS: [NUMBER OF HEADS]\n
101           IT [CAN/CANNOT] FLY\n"
102
103           Example:
104
105       */
106       void display() const override;
107
108       /**
109           @post   : If the creature is UNDEAD, it becomes tame if not already, as it appreciates the gesture, even though as an UNDEAD it does not really eat.
110                     It gains 1 hitpoint from the mysterious powers it receives by wearing the mushroom as a hat. Nothing else happens.
111                     If the creature is an ALIEN, it consumes the mushroom and gains 1 additional hitpoint. Nothing else happens.
112                     If the creature is MYSTICAL, it consumes the mushroom and gains 1 additional hitpoint if it has FIRE or EARTH affinity (Either by cooking the mushroom first, or by being fungi-tolerant). Nothing else happens.
113                     But if it is MYSTICAL and has WATER, AIR, or no affinity, if it only has 1 hitpoint left, it doesn't want to risk it and leaves the Cavern. If it has more than 1 hitpoint, it loses 1 hitpoint and becomes untamed if it was tame. Nothing else happens.
114           @return : true if the creature leaves the Cavern, false otherwise
115       */
116       bool eatMycoMorsel() override;
117
118       /**
119        * @param: A const reference to int corresponding to the attack to be added to the attack queue.
120        * @post: Adds an attack to the attack queue based on the int parameter.
121        * Here are the attacks for the Dragon:
122
123       * 1: Attack name: BITE
124       *     if ALIEN: 4 PHYSICAL
125       *     if MYSTICAL: 2 PHYSICAL, and additional damage of 1 [FIRE/WATER/EARTH/AIR] if the Dragon has an elemental affinity of FIRE, WATER, EARTH, or AIR)
126       *     if UNDEAD: 2 PHYSICAL, 1 POISON
127       *     if UNKNOWN: 2 PHYSICAL
128
129       * 2: Attack name: STOMP
130       *     if ALIEN: 2 PHYSICAL
131       *     if MYSTICAL: 1 PHYSICAL, and additional damage of 1 [FIRE/WATER/EARTH/AIR] if the Dragon has an elemental affinity of FIRE, WATER, EARTH, or AIR)
132       *     if UNDEAD: 1 PHYSICAL, 1 POISON
133       *     if UNKNOWN: 1 PHYSICAL
134
135       * 3: Attack name: [ELEMENTAL BREATH/BAD BREATH], where the name is ELEMENTAL BREATH if the Dragon has an elemental affinity, otherwise it is BAD BREATH
136       *     if ALIEN: 6 [POISON/FIRE/WATER/EARTH/AIR], where the damage type is the Dragon's elemental affinity if it has one, otherwise it is POISON
137       *     if MYSTICAL: 3 [POISON/FIRE/WATER/EARTH/AIR], where the damage type is the Dragon's
```

```
elemental affinity if it has one, otherwise it is POISON
138         *      if UNDEAD: 3 [POISON/FIRE/WATER/EARTH/AIR], 1 POISON. The damage types and amount are added to the vector as two separate entries, even if both entries are POISON type.
139         *      if UNKNOWN: 3 [POISON/FIRE/WATER/EARTH/AIR] where the damage type is the Dragon's elemental affinity if it has one, otherwise it is POISON
140         */
141         void addAttack(const int &attack) override;
142
143         /**
144          * @post:  displays the attacks of the Dragon in the form:
145          *        [DRAGON] Choose an attack (1-3):\n1: BITE\t\t2: STOMP\t\t3: ELEMENTAL BREATH\n
146          */
147         void displayAttacks() override;
148
149         private:
150             Element element_;
151             int number_of_heads_;
152             bool flight_;
153
154 };
155
156 #endif // DRAGON_HPP
```

Dragon.o Download

```
1   Large file hidden. You can download it using the button above.
```

Ghoul.cpp Download

```cpp
/*
CSCI235 Spring 2024
Project 4 - MycoMorsels
Georgina Woo
Dec 24 2023
Ghoul.cpp implements the constructors and private and public functions of the Ghoul class
*/

#include "Ghoul.hpp"

Ghoul::Ghoul() :  decay_{0}, faction_{NONE}, transformation_{false}
{
    setCategory(UNDEAD);
}

Ghoul::Ghoul(const std::string& name, Category category, int hitpoints, int level, bool tame, int decay, Faction faction, bool transformation) : Creature(name, category, hitpoints, level, tame)
{
    if(!setDecay(decay))
    {
        decay_ = 0;
    }
    faction_ = faction;
    transformation_ = transformation;
}

int Ghoul::getDecay() const
{
    return decay_;
}

bool Ghoul::setDecay(const int& decay)
{
    if(decay >= 0)
    {
        decay_ = decay;
        return true;
    }
    else
    {
        return false;
    }
}

std::string Ghoul::getFaction() const
{
    switch(faction_)
    {
        case FLESHGORGER:
```

```
49            return "FLESHGORGER";
50        case SHADOWSTALKER:
51            return "SHADOWSTALKER";
52        case PLAGUEWEAVER:
53            return "PLAGUEWEAVER";
54        default:
55            return "NONE";
56    }
57 }
58
59 void Ghoul::setFaction(const Faction& faction)
60 {
61    faction_ = faction;
62 }
63
64 bool Ghoul::getTransformation() const
65 {
66    return transformation_;
67 }
68
69 void Ghoul::setTransformation(const bool& transformation)
70 {
71    transformation_ = transformation;
72 }
73
74
75 /**
76    @post     : displays Ghoul data in the form:
77    GHOUL - [NAME]\n
78    CATEGORY: [CATEGORY]\n
79    HP: [HITPOINTS]\n
80    LVL: [LEVEL]\n
81    TAME: [TAME]\n
82    DECAY: [DECAY]\n
83    FACTION: [FACTION]\n
84    IT [CAN/CANNOT] TRANSFORM\n"
85
86    Example:
87
88 */
89 void Ghoul::display() const
90 {
91    std::cout << "GHOUL - " << getName() << std::endl;
92    std::cout << "CATEGORY: " << getCategory() << std::endl;
93    std::cout << "HP: " << getHitpoints() << std::endl;
94    std::cout << "LVL: " << getLevel() << std::endl;
95    std::cout << "TAME: " << (isTame() ? "TRUE" : "FALSE") << std::endl;
96    std::cout << "DECAY: " << getDecay() << std::endl;
97    std::cout << "FACTION: " << getFaction() << std::endl;
98    std::cout << "IT " << (getTransformation() ? "CAN" : "CANNOT") << " TRANSFORM" << std::endl;
99 }
100
```

```
101
102 bool Ghoul::eatMycoMorsel()
103 {
104     if(getCategory() == "UNDEAD")
105     {
106         setTame(true);
107         setHitpoints(getHitpoints() + 1);
108         return false;
109     }
110     else if(getFaction() == "FLESHGORGER")
111     {
112         if(isTame())
113         {
114             setTame(false);
115             return false;
116         }
117         else
118         {
119             return true;
120         }
121     }
122     else if(getFaction() == "SHADOWSTALKER")
123     {
124         if(isTame())
125         {
126             if(getHitpoints() == 1)
127             {
128                 setTame(false);
129             }
130             else
131             {
132                 setHitpoints(getHitpoints() - 1);
133             }
134             return false;
135         }
136         else
137         {
138             return false;
139         }
140     }
141     else
142     {
143         return false;
144     }
145 }
146
147 /**
148 * @param: A const reference to int corresponding to the attack to be added to the attack queue.
149 * @post: Adds an attack to the attack queue based on the int parameter.
150 * Here are the attacks for the Ghoul:
151 *
152 * 1: Attack name: BITE
```

```
*   if ALIEN: 4 PHYSICAL
*   if MYSTICAL:
*      if FLESHGORGER: 2 PHYSICAL
*      if SHADOWSTALKER: 2 PHYSICAL, 1 VOID
*      if PLAGUEWEAVER: 2 PHYSICAL, 1 POISON
*      if NONE: 2 PHYSICAL
*  if UNDEAD: 2 PHYSICAL, 1 POISON
*  if UNKNOWN: 2 PHYSICAL
*
* 2:
*  if FLESHGORGER/NONE:
*      Attack name: CLAW
*      2 PHYSICAL
*  if SHADOWSTALKER:
*     Attack name: SLASH
*     2 PHYSICAL, 1 VOID
* if PLAGUEWEAVER:
*    Attack name: INFECT
*    2 PHYSICAL, 1 POISON
*
* 3:
*  if FLESHGORGER/NONE:
*     Attack name: FLY SWARM
*     3 PHYSICAL
* if SHADOWSTALKER:
*     Attack name: SHROUD OF DARKNESS
*     2 PHYSICAL, 1 VOID
* if PLAGUEWEAVER:
*     Attack name: PLAGUE CLOUD
*     2 PHYSICAL, 1 POISON
*
*/
void Ghoul::addAttack(const int &attack){
    Attack Temp;
    if(attack == 1){
        Temp.name_ = "BITE";
        if(getCategory() == "ALIEN"){// push 4phycail if alein
            Temp.damage_.push_back(4);
            Temp.type_.push_back("PHYSICAL");
        }
        else if(getCategory() == "MYSTICAL"){
            Temp.damage_.push_back(2);
            Temp.type_.push_back("PHYSICAL");// default 2physcial
            if(getFaction() == "SHADOWSTALKER"){
                Temp.damage_.push_back(1);
                Temp.type_.push_back("VOID");// check faction and push attack based on the faction
            }
            else if(getFaction() == "PLAGUEWEAVER"){
                Temp.damage_.push_back(1);
                Temp.type_.push_back("POISON");
            }
        }
```

```
205         else if(getCategory() == "UNDEAD"){
206             Temp.damage_.push_back(2);
207             Temp.type_.push_back("PHYSICAL");// default
208             Temp.damage_.push_back(1);
209             Temp.type_.push_back("POISON");
210         }
211         else if (getCategory() == "UNKNOWN"){
212             Temp.damage_.push_back(2);//default
213             Temp.type_.push_back("PHYSICAL");
214         }
215     }
216     if(attack == 2){
217         Temp.name_ = "CLAW";
218         Temp.damage_.push_back(2);//default
219         Temp.type_.push_back("PHYSICAL");
220         if(getFaction() == "SHADOWSTALKER"){
221             Temp.name_ = "SLASH";
222             Temp.damage_.push_back(1);// check faaction and rename temp.name for certain factions
    also adding additonal attack
223             Temp.type_.push_back("VOID");
224         }
225         else if(getFaction() == "PLAGUEWEAVER"){
226             Temp.name_ = "INFECT";
227             Temp.damage_.push_back(1);
228             Temp.type_.push_back("POISON");
229         }
230     }
231     if(attack == 3){// no default
232         if(getFaction() == "NONE"){
233             Temp.name_ = "FLY SWARM";// if NONE FlY swarm 3 physcail
234             Temp.damage_.push_back(3);
235             Temp.type_.push_back("PHYSICAL");
236         }
237 
238         if(getFaction() == "FLESHGORGER"){
239             Temp.name_ = "FLY SWARM";// FLYSwarm 3 physcail if fleshgorger faction
240             Temp.damage_.push_back(3);
241             Temp.type_.push_back("PHYSICAL");
242         }
243 
244         if(getFaction() == "SHADOWSTALKER"){
245             Temp.name_ = "SHROUD OF DARKNESS";// SHROUDED OF DARKNESS 2 PHYSCIAL AND 1 VOID
    if SHADOWSTALKER FACTION
246             Temp.damage_.push_back(2);
247             Temp.type_.push_back("PHYSICAL");
248             Temp.damage_.push_back(1);
249             Temp.type_.push_back("VOID");
250         }
251         else if(getFaction() == "PLAGUEWEAVER"){
252             Temp.name_ = "PLAGUE CLOUD";
253             Temp.damage_.push_back(2);// PLAUGUE CLOUD 2 PHYSciAL and 1 POSIN if PLAUGUEWEAVER
    FACTION
```

```
254             Temp.type_.push_back("PHYSICAL");
255             Temp.damage_.push_back(1);
256             Temp.type_.push_back("POISON");
257         }
258     }
259     Creature::addAttack(Temp); // push into the creatures attackqueue
260 }
261 /**
262   @post     : displays the attacks of the Ghoul in the form:
263   [GHOUL] Choose an attack (1-3):\n1: BITE\t\t2: CLAW\t\t3: CLOUD OF DOOM\n
264 */
265 void Ghoul::displayAttacks(){
266 std::cout << "[GHOUL] Choose an attack (1-3):\n1: BITE\t\t2: CLAW\t\t3: CLOUD OF DOOM\n";
267 }
```

Ghoul.hpp Download

```
/*
CSCI235 Spring 2024
Project 4 - MycoMorsels
Georgina Woo
Dec 24 2023
Ghoul.hpp defines the constructors and private and public functions of the Ghoul class
*/

#ifndef GHOUL_HPP
#define GHOUL_HPP

#include "Creature.hpp"



class Ghoul : public Creature
{
   public:

      enum Faction {NONE, FLESHGORGER, SHADOWSTALKER, PLAGUEWEAVER};

      /**
         Default constructor.
         Default-initializes all private members.
         Default Category: UNDEAD
         Default decay: 0
         Default faction: NONE
         Booleans are default-initialized to False.
      */
      Ghoul();

      /**
         Parameterized constructor.
         @param     : The name of the Ghoul (a reference string)
         @param     : The category of the Ghoul (a Category enum) with default value UNDEAD
         @param     : The Ghoul's hitpoints (an integer), with default value 1 if not provided, or if the value provided is 0 or negative
         @param     : The Ghoul's level (an integer), with default value 1 if not provided, or if the value provided is 0 or negative
         @param     : A flag indicating whether the Ghoul is tame, with default value False
         @param     : The level of decay (an integer), with default value 0 if not provided, or if the value provided is negative
         @param     : The faction (a Faction enum), with default value NONE if not provided
         @param     : A flag indicating whether the Ghoul can transform, with default value False
         @post      : The private members are set to the values of the corresponding parameters.
         Hint: Notice the default arguments in the parameterized constructor.
      */
      Ghoul(const std::string& name, Category category = UNDEAD, int hitpoints = 1, int level = 1, bool tame = false, int decay = 0, Faction faction = NONE, bool transformation = false);
```

```
46
47      /**
48         Getter for the level of decay.
49         @return     : The level of decay (an integer)
50      */
51      int getDecay() const;
52
53      /**
54         Setter for the level of decay.
55         @param      : A reference to the level of decay (an integer)
56         @pre        : The level of decay must be >= 0 (do nothing otherwise)
57         @post       : The level of decay is set to the value of the parameter.
58         @return     : true if the level of decay was set, false otherwise
59      */
60      bool setDecay(const int& decay);
61
62      /**
63         Getter for the faction.
64         @return     : The faction (a string representation of the Faction enum)
65      */
66      std::string getFaction() const;
67
68      /**
69         Setter for the faction.
70         @param      : A reference to the faction (a Faction enum)
71         @post       : The faction is set to the value of the parameter.
72      */
73      void setFaction(const Faction& faction);
74
75      /**
76         Getter for the transformation.
77         @return     : The transformation (a boolean)
78      */
79      bool getTransformation() const;
80
81      /**
82         Setter for the transformation.
83         @param      : A reference to the transformation (a boolean)
84         @post       : The transformation is set to the value of the parameter.
85      */
86      void setTransformation(const bool& transformation);
87
88
89          /**
90         @post     : displays Ghoul data in the form:
91         GHOUL - [NAME]\n
92         CATEGORY: [CATEGORY]\n
93         HP: [HITPOINTS]\n
94         LVL: [LEVEL]\n
95         TAME: [TAME]\n
96         DECAY: [DECAY]\n
97         FACTION: [FACTION]\n
```

```
98              IT [CAN/CANNOT] TRANSFORM\n"
99
100             Example:
101
102         */
103         void display() const override;
104
105         /**
106             @post   : If the creature is UNDEAD, it becomes tame if not already, as it appreciates the gesture, even though as an UNDEAD it does not really eat. It gains 1 hitpoint from the mysterious powers it receives by wearing the mushroom as a hat. Nothing else happens.
107                       If the creature is of Faction FLESHGORGER, it becomes so offended by being offered a mushroom that it becomes untamed if it was tame. If it was already untamed, it leaves the Cavern. Nothing else happens.
108                       If the creature was of Faction SHADOWSTALKER, if it was untame, it hides from the mushroom and nothing happens. If it were tame, it eats the mushroom and loses 1 hitpoint, unless it only had 1 hitpoint left in which case it stays at 1 hitpoint but becomes untame. Nothing else happens.
109             @return   : true if the creature leaves the Cavern, false otherwise
110         */
111         bool eatMycoMorsel() override;
112 /**
113 * @param: A const reference to int corresponding to the attack to be added to the attack queue.
114 * @post: Adds an attack to the attack queue based on the int parameter.
115 * Here are the attacks for the Ghoul:
116 *
117 * 1: Attack name: BITE
118 *   if ALIEN: 4 PHYSICAL
119 *   if MYSTICAL:
120 *      if FLESHGORGER: 2 PHYSICAL
121 *      if SHADOWSTALKER: 2 PHYSICAL, 1 VOID
122 *      if PLAGUEWEAVER: 2 PHYSICAL, 1 POISON
123 *      if NONE: 2 PHYSICAL
124 *  if UNDEAD: 2 PHYSICAL, 1 POISON
125 *  if UNKNOWN: 2 PHYSICAL
126 *
127 * 2:
128 *  if FLESHGORGER/NONE:
129 *      Attack name: CLAW
130 *      2 PHYSICAL
131 *  if SHADOWSTALKER:
132 *     Attack name: SLASH
133 *     2 PHYSICAL, 1 VOID
134 * if PLAGUEWEAVER:
135 *    Attack name: INFECT
136 *    2 PHYSICAL, 1 POISON
137 *
138 * 3:
139 *  if FLESHGORGER/NONE:
140 *     Attack name: FLY SWARM
141 *     3 PHYSICAL
142 * if SHADOWSTALKER:
```

```
143 *     Attack name: SHROUD OF DARKNESS
144 *     2 PHYSICAL, 1 VOID
145 * if PLAGUEWEAVER:
146 *     Attack name: PLAGUE CLOUD
147 *     2 PHYSICAL, 1 POISON
148 *
149 */
150         void addAttack(const int &attack) override;
151 /**
152   @post     : displays the attacks of the Ghoul in the form:
153   [GHOUL] Choose an attack (1-3):\n1: BITE\t\t2: CLAW\t\t3: CLOUD OF DOOM\n
154 */
155         void displayAttacks() override;
156
157
158
159     private:
160         int decay_;
161         Faction faction_;
162         bool transformation_;
163 };
164
165 #endif // GHOUL_HPP
```

Ghoul.o Download

```
1 Large file hidden. You can download it using the button above.
```

Makefile Download

```
1  CXX = g++
2  CXXFLAGS = -std=c++17 -g -Wall -O2
3
4  PROG ?= main
5  OBJS = Creature.o Cavern.o main.o Dragon.o Ghoul.o Mindflayer.o
6
7  all: $(PROG)
8
9  .cpp.o:
10         $(CXX) $(CXXFLAGS) -c -o $@ $<
11
12 $(PROG): $(OBJS)
13         $(CXX) $(CXXFLAGS) -o $@ $(OBJS)
14
15 clean:
16         rm -rf $(EXEC) *.o *.out main
17
18 rebuild: clean all
19
```

Mindflayer.cpp

Download

```cpp
/*
CSCI235 Spring 2024
Project 4 - MycoMorsels
Georgina Woo
Dec 24 2023
Mindflayer.cpp implements the constructors and private and public functions of the Mindflayer class
*/

#include "Mindflayer.hpp"

Mindflayer::Mindflayer() : affinities_{}, summoning_{false}, projectiles_{}
{
    setCategory(ALIEN);
}

Mindflayer::Mindflayer(const std::string& name, Category category, int hitpoints, int level, bool tame, std::vector<Projectile> projectiles, bool summoning, std::vector<Variant> affinities) : Creature(name, category, hitpoints, level, tame)
{
    setProjectiles(projectiles);
    summoning_ = summoning;
    setAffinities(affinities);
}

std::vector<Mindflayer::Projectile> Mindflayer::getProjectiles() const
{
    return projectiles_;
}

void Mindflayer::setProjectiles(const std::vector<Projectile>& projectiles)
{
    std::vector<Projectile> temp;
    for(int i = 0; i < projectiles.size(); i++)
    {
        bool found = false;
        for(int j = 0; j < temp.size(); j++)
        {
            if(projectiles[i].type_ == temp[j].type_)
            {
                if(projectiles[i].quantity_ > 0)
                {
                    temp[j].quantity_ += projectiles[i].quantity_;
                    found = true;
                }
            }
        }
        if(!found)
        {
```

```
            if(projectiles[i].quantity_ > 0)
            {
                temp.push_back(projectiles[i]);
            }
        }
    }
    projectiles_ = temp;
}

void Mindflayer::setSummoning(const bool& summoning)
{
    summoning_ = summoning;
}

bool Mindflayer::getSummoning() const
{
    return summoning_;
}

std::vector<Mindflayer::Variant> Mindflayer::getAffinities() const
{
    return affinities_;
}

void Mindflayer::setAffinities(const std::vector<Variant>& affinities)
{
    std::vector<Variant> temp;
    for(int i = 0; i < affinities.size(); i++)
    {
        bool found = false;
        for(int j = 0; j < temp.size(); j++)
        {
            if(affinities[i] == temp[j])
            {
                found = true;
            }
        }
        if(!found)
        {
            temp.push_back(affinities[i]);
        }
    }
    affinities_ = temp;
}

std::string Mindflayer::variantToString(const Variant& variant) const
{
    switch(variant)
    {
        case PSIONIC:
            return "PSIONIC";
        case TELEPATHIC:
```

```
99            return "TELEPATHIC";
100         case ILLUSIONARY:
101            return "ILLUSIONARY";
102         default:
103            return "NONE";
104      }
105   }
106
107
108
109   void Mindflayer::display() const{
110      std::cout << "MINDFLAYER - " << getName() << std::endl;
111      std::cout << "CATEGORY: " << getCategory() << std::endl;
112      std::cout << "HP: " << getHitpoints() << std::endl;
113      std::cout << "LVL: " << getLevel() << std::endl;
114      std::cout << "TAME: " << (isTame() ? "TRUE" : "FALSE") << std::endl;
115      std::cout << "SUMMONING: " << (getSummoning() ? "TRUE" : "FALSE") << std::endl;
116      for(int i = 0; i < projectiles_.size(); i++)
117      {
118         std::cout << variantToString(projectiles_[i].type_) << ": " << projectiles_[i].quantity_ << std::endl;
119      }
120      if(affinities_.size() > 0)
121      {
122         std::cout << "AFFINITIES: " << std::endl;
123         for(int i = 0; i < affinities_.size(); i++)
124         {
125            std::cout << variantToString(affinities_[i]) << std::endl;
126         }
127      }
128   }
129
130   /**
131      @post   : If the creature is UNDEAD, it becomes tame if not already, as it appreciates the gesture, even though as an UNDEAD it does not really eat. It gains 1 hitpoint from the mysterious powers it receives by wearing the mushroom as a hat. Nothing else happens.
132               If the creature is MYSTICAL, if it can summon a Thoughtspawn, it gives the mushroom to the Thoughtspawn instead. Nothing else happens.
133               If it cannot summon a Thoughtspawn and is tame, it eats the mushroom to be polite. If it only had 1 hitpoint left, it remains at 1 hitpoint and becomes untame, else it loses 1 hitpoint. Nothing else happens.
134               If it cannot summon a Thoughtspawn and is untame, it decides it doesn't have to deal with this and it leaves the Cavern. Nothing else happens.
135               If the creature is an ALIEN, if it has Telepathic affinity, it convinces Selfa Ensert to eat the mushroom instead, and gains 1 hitpoint from laughing at their resulting illness (as laughter is good for the soul).
136               If it is an ALIEN and does not have Telepathic affinity, but has a Telepathic projectile, it uses one of its Telepathic projectiles to achieve the same effect. (Remember to remove the projectile from the vector if it only had 1 left)
137               If it is an ALIEN and does not have Telepathic affinity or a Telepathic projectile, it eats the mushroom and gains 2 hitpoints. As it turns out, the mushroom was good for it. It becomes tame if it was not already. Nothing else happens.
138      @return   : true if the creature leaves the Cavern, false otherwise
```

```
*/
bool Mindflayer::eatMycoMorsel(){
    if(getCategory() == "UNDEAD")
    {
        setTame(true);
        setHitpoints(getHitpoints() + 1);
        return false;
    }
    else if(getCategory() == "MYSTICAL")
    {
        if(getSummoning())
        {
            return false;
        }
        else if(isTame())
        {
            if(getHitpoints() == 1)
            {
                setTame(false);
                return false;
            }
            else
            {
                setHitpoints(getHitpoints() - 1);
                return false;
            }
        }
        else
        {
            return true;
        }
    }
    else if(getCategory() == "ALIEN")
    {
        bool telepathic = false;
        for(int i = 0; i < affinities_.size(); i++)
        {
            if(affinities_[i] == TELEPATHIC)
            {
                telepathic = true;
            }
        }
        bool telepathicProjectile = false;
        for(int i = 0; i < projectiles_.size(); i++)
        {
            if(projectiles_[i].type_ == TELEPATHIC)
            {
                telepathicProjectile = true;
                if(projectiles_[i].quantity_ == 1)
                {
                    projectiles_.erase(projectiles_.begin() + i);
                }
```

```
            else
            {
                projectiles_[i].quantity_--;
            }
        }
    }
    if(telepathic || telepathicProjectile)
    {
        setHitpoints(getHitpoints() + 1);
        return false;
    }
    else
    {
        setTame(true);
        setHitpoints(getHitpoints() + 2);
        return false;
    }
  }
  else
  {
    return false;
  }

}

/**
* @param: A const reference to int corresponding to the attack to be added to the attack queue.
* @post: Adds an attack to the attack queue based on the int parameter.
* Here are the attacks for the Mindflayer:
*
* 1: PSIONIC BOLT/TENTACLE SLAP
*  If the Mindflayer has a PSIONIC projectile:
*     Attack name: PSIONIC BOLT
*     If the Mindflayer has a PSIONIC affinity:
*        Damage: 3 PSIONIC
*     Else:
*        Damage: 2 PSIONIC
* If the Mindflayer does not have a PSIONIC projectile:
*    Attack name: TENTACLE SLAP
*    Damage: 1 PHYSICAL, 1 EMOTIONAL
*
* 2: TELEPATHIC BOLT/TENTACLE SLAP
* If the Mindflayer has a TELEPATHIC projectile:
*     Attack name: TELEPATHIC BOLT
*     If the Mindflayer has a TELEPATHIC affinity:
*     Damage: 3 TELEPATHIC
*     Else:
*        Damage: 2 TELEPATHIC
* If the Mindflayer does not have a TELEPATHIC projectile:
*     Attack name: TENTACLE SLAP
*     Damage: 1 PHYSICAL, 1 EMOTIONAL
*
```

```
* 3: ILLUSIONARY BOLT/TENTACLE SLAP
* If the Mindflayer has an ILLUSIONARY projectile:
*      Attack name: ILLUSIONARY BOLT
*      If the Mindflayer has an ILLUSIONARY affinity:
*          Damage: 3 ILLUSIONARY
*      Else:
*          Damage: 2 ILLUSIONARY
* If the Mindflayer does not have an ILLUSIONARY projectile:
*      Attack name: TENTACLE SLAP
*      Damage: 1 PHYSICAL, 1 EMOTIONAL
*
*/
void Mindflayer::addAttack(const int &attack){
   Attack Temp;
   if(attack == 1){
      bool proj = false;// default false for proj
      for(int i=0;i < projectiles_.size(); i++){// check if proj pisonic is there. if there set bool proj to true and stop running
         if(projectiles_[i].type_ == PSIONIC){
            proj = true;
            break;
         }
      }
      if(proj){//Run if proj is true
         Temp.name_ = "PSIONIC BOLT";// name PISONIC BOLT
         bool aff = false; // aff deafult false
         for(int j =0; j<affinities_.size(); j++){
            if(affinities_[j] == PSIONIC){// check if Affinity PSIONIC is there break if it is there
               aff = true;
               break;
            }
         }
         if(aff){// If aff is true push 3 PISONIC
            Temp.damage_.push_back(3);
            Temp.type_.push_back("PSIONIC");
         }
         else{// else push 2 PISONIC
            Temp.damage_.push_back(2);
            Temp.type_.push_back("PSIONIC");
         }
      }
      else{// run if proj isnt true
         Temp.name_= "TENTACLE SLAP";
         Temp.damage_.push_back(1);
         Temp.type_.push_back("PHYSICAL");
         Temp.damage_.push_back(1);
         Temp.type_.push_back("EMOTIONAL");
      }
   }
   if(attack == 2){
      bool proj = false;// rest of the implmenation are the same as the previous one. But checking for different Variant
```

```
293         for(int i=0;i < projectiles_.size(); i++){
294             if(projectiles_[i].type_ == TELEPATHIC){
295                 proj = true;
296                 break;
297             }
298         }
299         if(proj){
300             Temp.name_ = "TELEPATHIC BOLT";
301             bool aff = false;
302             for(int j =0; j<affinities_.size(); j++){
303                 if(affinities_[j] == TELEPATHIC){
304                     aff = true;
305                     break;
306                 }
307             }
308             if(aff){
309                 Temp.damage_.push_back(3);
310                 Temp.type_.push_back("TELEPATHIC");
311             }
312             else{
313                 Temp.damage_.push_back(2);
314                 Temp.type_.push_back("TELEPATHIC");
315             }
316         }
317         else{
318             Temp.name_= "TENTACLE SLAP";
319             Temp.damage_.push_back(1);
320             Temp.type_.push_back("PHYSICAL");
321             Temp.damage_.push_back(1);
322             Temp.type_.push_back("EMOTIONAL");
323         }
324     }
325     if(attack == 3){
326         bool proj = false;
327         for(int i=0;i < projectiles_.size(); i++){
328             if(projectiles_[i].type_ == ILLUSIONARY){
329                 proj = true;
330                 break;
331             }
332         }
333         if(proj){
334             Temp.name_ = "ILLUSIONARY BOLT";
335             bool aff = false;
336             for(int j =0; j<affinities_.size(); j++){
337                 if(affinities_[j] == ILLUSIONARY){
338                     aff = true;
339                     break;
340                 }
341             }
342             if(aff){
343                 Temp.damage_.push_back(3);
344                 Temp.type_.push_back("ILLUSIONARY");
```

```
345             }
346             else{
347                 Temp.damage_.push_back(2);
348                 Temp.type_.push_back("ILLUSIONARY");
349             }
350         }
351         else{
352             Temp.name_= "TENTACLE SLAP";
353             Temp.damage_.push_back(1);
354             Temp.type_.push_back("PHYSICAL");
355             Temp.damage_.push_back(1);
356             Temp.type_.push_back("EMOTIONAL");
357         }
358     }
359     Creature::addAttack(Temp);
360 }
361 /**
362   @post     : displays the attacks of the Mindflayer in the form:
363   [MINDFLAYER] Choose an attack (1-3):\n1: PSIONIC BOLT\t\t2: TELEPATHIC BOLT\t\t3: ILLUSIONARY
    BOLT\n
364 */
365 void Mindflayer::displayAttacks() {
366 std::cout <<"[MINDFLAYER] Choose an attack (1-3):\n1: PSIONIC BOLT\t\t2: TELEPATHIC BOLT\t\t3:
    ILLUSIONARY BOLT\n";
367 }
```

Mindflayer.hpp

Download

```
/*
CSCI235 Spring 2024
Project 4 - MycoMorsels
Georgina Woo
Dec 24 2023
Mindflayer.hpp defines the constructors and private and public functions of the Mindflayer class
*/

#ifndef MINDFLAYER_HPP
#define MINDFLAYER_HPP

#include "Creature.hpp"
#include <vector>



class Mindflayer : public Creature{
   public:

      enum Variant {PSIONIC, TELEPATHIC, ILLUSIONARY};

struct Projectile{
   Variant type_;
   int quantity_;
};
      /**
         Default constructor.
         Default-initializes all private members.
         Default Category: ALIEN
         Default summoning: False
      */
      Mindflayer();

      /**
         Parameterized constructor.
         @param     : A reference to the name of the Mindflayer (a string)
         @param     : The category of the Mindflayer (a Category enum) with default value ALIEN
         @param     : The Mindflayer's hitpoints (an integer), with default value 1 if not provided, or if the value provided is 0 or negative
         @param     : The Mindflayer's level (an integer), with default value 1 if not provided, or if the value provided is 0 or negative
         @param     : A flag indicating whether the Mindflayer is tame, with default value False
         @param     : The projectiles (a vector of Projectile structs), with default value an empty vector if not provided
         @param     : A flag indicating whether the Mindflayer can summon a Thoughtspawn, with default value False
         @param     : The affinities (a vector of Variant enums), with default value an empty vector if not provided
         @post      : The private members are set to the values of the corresponding parameters.
```

```
45          Hint: Notice the default arguments in the parameterized constructor.
46      */
47      Mindflayer(const std::string& name, Category category = ALIEN, int hitpoints = 1, int level = 1,
    bool tame = false, std::vector<Projectile> projectiles = {}, bool summoning = false,
    std::vector<Variant> affinities = {});
48
49      /**
50          Getter for the projectiles.
51          @return     : The projectiles (a vector of Projectile structs)
52      */
53      std::vector<Projectile> getProjectiles() const;
54
55      /**
56          Setter for the projectiles.
57          @param      : A reference to the projectiles (a vector of Projectile structs)
58          @post       : The projectiles are set to the value of the parameter. There should not be any
    duplicate projectiles in Mindflayer's projectiles vector.
59                      : For example, if the vector in the given parameter contains the following Projectiles:
    {{PSIONIC, 2}, {TELEPATHIC, 1}, {PSIONIC, 1}, {ILLUSIONARY, 3}},
60                      : the projectiles vector should be set to contain the following Projectiles: {{PSIONIC,
    3}, {TELEPATHIC, 1}, {ILLUSIONARY, 3}}.
61                      : If the quantity of a projectile is 0 or negative, it should not be added to the
    projectiles vector.
62                      : Note the order of the projectiles in the vector.
63      */
64      void setProjectiles(const std::vector<Projectile>& projectiles);
65
66      /**
67          Getter for the summoning.
68          @return     : The summoning (a boolean)
69      */
70      bool getSummoning() const;
71
72      /**
73          Setter for the summoning.
74          @param      : A reference to the summoning (a boolean)
75          @post       : The summoning is set to the value of the parameter.
76      */
77      void setSummoning(const bool& summoning);
78
79      /**
80          Getter for the affinities.
81          @return     : The affinities (a vector of Variant enums)
82      */
83      std::vector<Variant> getAffinities() const;
84
85      /**
86          Setter for the affinities.
87          @param      : A reference to the affinities (a vector of Variant enums)
88          @post       : The affinities are set to the value of the parameter.
89                      : There should not be any duplicate affinities in Mindflayer's affinities vector.
90                      : For example, if the vector in the given parameter contains the following affinities:
```

```
{PSIONIC, TELEPATHIC, PSIONIC, ILLUSIONARY},
                    : the affinities vector should be set to contain the following affinities: {PSIONIC, TELEPATHIC, ILLUSIONARY}.
                    : Note the order of the affinities in the vector.
    */
    void setAffinities(const std::vector<Variant>& affinities);

    /**
       @param      : A reference to the Variant
       @return     : The string representation of the variant
    */
    std::string variantToString(const Variant& variant) const;



        /**
       @post    : displays Mindflayer data in the form:
       "MINDFLAYER - [NAME]\n
       CATEGORY: [CATEGORY]\n
       HP: [HITPOINTS]\n
       LVL: [LEVEL]\n
       TAME: [TAME]\n
       SUMMONING: [SUMMONING]\n
       [PROJECTILE TYPE 1]: [QUANTITY 1]\n
       [PROJECTILE TYPE 2]: [QUANTITY 2]\n
       AFFINITIES: \n
       [AFFINITY 1]\n
       [AFFINITY 2]\n"

       NOTE: For the Projectiles, print out the type and quantity of each projectile in the format:
       [TYPE]: [QUANTITY] for each projectile in the vector, where the type is the string equivalent of the Variant (eg. "PSIONIC"/"TELEPATHIC"/"ILLUSIONARY"). If there are no projectiles, don't print anything.

       For the Affinities, print out each affinity in the format: [AFFINITY 1]\n[AFFINITY 2]\n for each Affinity in the vector, where the Affinity is the string equivalent of the Variant (eg. "PSIONIC"/"TELEPATHIC"/"ILLUSIONARY"). If there are no affinities, don't print anything, including the label "AFFINITIES:".

       Example:

    */
    void display() const override;

    /**
       @post   : If the creature is UNDEAD, it becomes tame if not already, as it appreciates the gesture, even though as an UNDEAD it does not really eat. It gains 1 hitpoint from the mysterious powers it receives by wearing the mushroom as a hat. Nothing else happens.
                 If the creature is MYSTICAL, if it can summon a Thoughtspawn, it gives the mushroom to the Thoughtspawn instead. If it cannot summon a Thoughtspawn and is tame, it eats the mushroom to be polite. If it only had 1 hitpoint left, it remains at 1 hitpoint and becomes untame,
```

```
    else it loses 1 hitpoint. If it cannot summon a Thoughtspawn and is untame, it decides it doesn't
    have to deal with this and it leaves the Cavern. Nothing else happens.
131                 If the creature is an ALIEN, if it has Telepathic affinity, it convinces Selfa Ensert to eat
    the mushroom instead, and gains 1 hitpoint from laughing at their resulting illness (as laughter is
    the best medicine).
132                 If it is an ALIEN and does not have Telepathic affinity, but has a Telepathic projectile, it
    uses one of such projectile to achieve the same effect. (Remember to remove the projectile from the
    vector if it only had 1 left)
133                 If it is an ALIEN and does not have Telepathic affinity or a Telepathic projectile, it eats
    the mushroom and gains 2 hitpoints. As it turns out, the mushroom was good for it. It becomes
    tame if it was not already. Nothing else happens.
134         @return   : true if the creature leaves the Cavern, false otherwise
135       */
136       bool eatMycoMorsel() override;
137             /**
138       * @param: A const reference to int corresponding to the attack to be added to the attack
    queue.
139       * @post: Adds an attack to the attack queue based on the int parameter.
140       * Here are the attacks for the Mindflayer:
141       *
142       * 1: PSIONIC BOLT/TENTACLE SLAP
143       *  If the Mindflayer has a PSIONIC projectile:
144       *      Attack name: PSIONIC BOLT
145       *      If the Mindflayer has a PSIONIC affinity:
146       *          Damage: 3 PSIONIC
147       *      Else:
148       *          Damage: 2 PSIONIC
149       * If the Mindflayer does not have a PSIONIC projectile:
150       *     Attack name: TENTACLE SLAP
151       *     Damage: 1 PHYSICAL, 1 EMOTIONAL
152       *
153       * 2: TELEPATHIC BOLT/TENTACLE SLAP
154       * If the Mindflayer has a TELEPATHIC projectile:
155       *      Attack name: TELEPATHIC BOLT
156       *      If the Mindflayer has a TELEPATHIC affinity:
157       *      Damage: 3 TELEPATHIC
158       *      Else:
159       *          Damage: 2 TELEPATHIC
160       * If the Mindflayer does not have a TELEPATHIC projectile:
161       *      Attack name: TENTACLE SLAP
162       *      Damage: 1 PHYSICAL, 1 EMOTIONAL
163       *
164       * 3: ILLUSIONARY BOLT/TENTACLE SLAP
165       * If the Mindflayer has an ILLUSIONARY projectile:
166       *      Attack name: ILLUSIONARY BOLT
167       *      If the Mindflayer has an ILLUSIONARY affinity:
168       *          Damage: 3 ILLUSIONARY
169       *      Else:
170       *          Damage: 2 ILLUSIONARY
171       * If the Mindflayer does not have an ILLUSIONARY projectile:
172       *      Attack name: TENTACLE SLAP
173       *      Damage: 1 PHYSICAL, 1 EMOTIONAL
```

```
174         *
175         */
176         void addAttack(const int &attack) override;
177 
178         /**
179          @post     : displays the attacks of the Mindflayer in the form:
180         [MINDFLAYER] Choose an attack (1-3):\n1: PSIONIC BOLT\t\t2: TELEPATHIC BOLT\t\t3:
    ILLUSIONARY BOLT\n
181         */
182         void displayAttacks() override;
183 
184 
185 
186     private:
187         std::vector<Projectile> projectiles_;
188         bool summoning_;
189         std::vector<Variant> affinities_;
190 
191 };
192 #endif // MINDFLAYER_HPP
```

Mindflayer.o Download

```
1  Large file hidden. You can download it using the button above.
```

README.md Download

```
1  [![Review Assignment Due Date](https://classroom.github.com/assets/deadline-readme-button-
   24ddc0f5d75046c5622901739e7c5dd533143b0c8e959d652212380cedb1ea36.svg)]
   (https://classroom.github.com/a/Q3pmTV4v)
2   # Project6
3 
4 
5 
6  The project specification is available on Blackboard
```

creatures.csv Download

```
TYPE,NAME,CATEGORY,HITPOINTS,LEVEL,TAME,ELEMENT/FACTION,HEADS,FLIGHT/TRANSFORM/SUMM
DRAGON,DRIFON,ALIEN,17,7,1,FIRE,5,0,0,NONE,NONE
DRAGON,JHARY,UNDEAD,15,11,0,WATER,3,1,0,NONE,NONE
GHOUL,ZYRAJA,MYSTICAL,20,6,1,FLESHGORGER,1,0,1,NONE,NONE
MINDFLAYER,NYLTHOR,ALIEN,3,8,1,NONE,1,0,0,PSIONIC,PSIONIC-2
DRAGON,QUIVARA,UNDEAD,14,5,1,EARTH,3,0,0,NONE,NONE
GHOUL,LYTHARA,ALIEN,7,9,1,PLAGUEWEAVER,1,0,1,NONE,NONE
GHOUL,ZEPHYX,MYSTICAL,12,9,1,SHADOWSTALKER,1,1,1,NONE,NONE
MINDFLAYER,FAELAN,MYSTICAL,16,12,0,NONE,1,1,0,TELEPATHIC,ILLUSIONARY-3
DRAGON,VYNTHOR,UNDEAD,20,9,1,AIR,9,0,0,NONE,NONE
MINDFLAYER,QUIXARA,ALIEN,17,4,0,NONE,1,1,0,NONE,NONE
GHOUL,THALYN,MYSTICAL,16,10,0,FLESHGORGER,1,1,0,NONE,NONE
MINDFLAYER,XYLIX,UNDEAD,20,11,1,NONE,1,0,0,ILLUSIONARY,TELEPATHIC-1;ILLUSIONARY-2
DRAGON,ZEPHYRA,UNDEAD,15,4,1,WATER,10,0,0,NONE,NONE
DRAGON,VYLTHOR,ALIEN,7,9,1,EARTH,8,0,0,NONE,NONE
GHOUL,BOB,MYSTICAL,2,12,0,SHADOWSTALKER,1,1,0,NONE,NONE
MINDFLAYER,ZYRANA,ALIEN,16,7,0,NONE,1,1,0,PSIONIC;TELEPATHIC,NONE
GHOUL,MYTHOS,UNDEAD,18,9,0,PLAGUEWEAVER,1,0,0,NONE,NONE
DRAGON,KRYLIX,ALIEN,2,9,0,AIR,6,1,0,NONE,NONE
GHOUL,VORYN,MYSTICAL,17,9,1,FLESHGORGER,1,0,1,NONE,NONE
MINDFLAYER,JHRISMAS,MYSTICAL,5,12,0,NONE,1,1,0,NONE,NONE
MINDFLAYER,QUIZARA,UNKNOWN,11,2,0,NONE,1,1,0,PSIONIC;TELEPATHIC;ILLUSIONARY,PSIONIC-1;T
GHOUL,VYRIX,UNKNOWN,7,7,1,PLAGUEWEAVER,1,0,3,NONE,NONE
DRAGON,DRIFANA,UNKNOWN,11,10,1,WATER,2,1,0,NONE,NONE
GHOUL,XALYN,UNKNOWN,5,10,1,SHADOWSTALKER,1,0,2,NONE,NONE
DRAGON,KRYTHOR,UNKNOWN,3,2,1,AIR,3,0,0,NONE,NONE
MINDFLAYER,JESTOR,UNKNOWN,13,10,0,NONE,1,0,0,NONE,NONE
MINDFLAYER,ZYRIXIS,UNKNOWN,6,4,0,NONE,1,0,0,NONE,NONE
GHOUL,JHERRY,UNDEAD,1,11,1,FLESHGORGER,1,1,5,NONE,NONE
```

main Download

```
Large file hidden. You can download it using the button above.
```

main.cpp Download

```cpp
#include "Creature.hpp"
#include "Cavern.hpp"

int main(){
    Cavern c("creatures.csv");
    c.initializeMysticalStack();
    std::stack<Creature*> a_stack=c.getMysticalStack();
    c.setAttacks(a_stack);

    while (!a_stack.empty())
    {
        Creature* c = a_stack.top();
        a_stack.pop();
        std::cout << c->getName() << " Attacks: " << std::endl;
        std::queue<Attack> q = c->getAttackQueue();
        while (!q.empty())
        {
            Attack a = q.front();
            q.pop();
            std::cout << "Name: " << a.name_ << std::endl;
            std::cout << "Type: ";
            for (int i = 0; i < a.type_.size(); i++)
            {
                std::cout << a.type_[i] << " ";
            }
            std::cout << std::endl;
            std::cout << "Damage: ";
            for (int i = 0; i < a.damage_.size(); i++)
            {
                std::cout << a.damage_[i] << " ";
            }
            std::cout << std::endl;
        }
    }
}

```

main.o Download

Large file hidden. You can download it using the button above.