Project 4

● Graded

Student
Devin Chen

Total Points
30 / 100 pts

Autograder Score
5.0 / 80.0

Failed Tests
Test compiles (0/5)
Tests subclasses display() (0/10)
Testing late binding for Dragon, Ghoul, and Mindflayer display() (0/5)
Tests Dragon eatMycoMorsel() (0/5)
Tests Ghoul eatMycoMorsel() (0/5)
Tests Mindflayer eatMycoMorsel() (0/5)
Tests Cavern parameterized constructor with input csv file and displayCreatures() (0/25)
Tests Cavern displayCategory() (0/5)
Tests if Creature is an abstract class (0/10)

Passed Tests
Tests that Cavern now holds pointers to Creatures (5/5)

Question 2

Style & Documentation

25 / 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
  - **+ 5 pts** Partial No-Compile: Implements new Cavern methods
  - **+ 5 pts** Partial No-Compile: Implements new Dragon methods
  - **+ 5 pts** Partial No-Compile: Implements new Ghoul methods
  - **+ 5 pts** Partial No-Compile: Implements new Mindflayer methods
  - **+ 0 pts** Insufficient submission

Autograder Results

Test compiles (0/5)

Your program does not compile. Please refer to the Project Specification.

Test Failed: 'Does not compile' != ''
- Does not compile
+

Tests subclasses display() (0/10)

Test Failed: [Errno 2] No such file or directory: './main'

Testing late binding for Dragon, Ghoul, and Mindflayer display() (0/5)

Test Failed: [Errno 2] No such file or directory: './main'

Tests Dragon eatMycoMorsel() (0/5)

Test Failed: [Errno 2] No such file or directory: './main'

Tests Ghoul eatMycoMorsel() (0/5)

Test Failed: [Errno 2] No such file or directory: './main'

Tests Mindflayer eatMycoMorsel() (0/5)

Test Failed: [Errno 2] No such file or directory: './main'

Tests Cavern parameterized constructor with input csv file and displayCreatures() (0/25)

Test Failed: [Errno 2] No such file or directory: './main'

Tests Cavern displayCategory() (0/5)

Test Failed: [Errno 2] No such file or directory: './main'

Tests if Creature is an abstract class (0/10)

Test Failed: [Errno 2] No such file or directory: './main'

Tests that Cavern now holds pointers to Creatures (5/5)

Your Cavern class now holds pointers to Creatures.

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

```cpp
/*
CSCI235 Spring 2024
Project 3 - Cavern Class
Georgina Woo
Dec 24 2023
Taven.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} {
}

Cavern::Cavern(std::string& inputfile): ArrayBag<Creature*>(){
std::ifstream cfile;
cfile.open(inputfile);
std::string linenumber;

while(getline(cfile, linenumber)){

  }
}

bool Cavern::enterCavern(Creature* new_creature) {
  if (getIndexOf(new_creature) == -1) {
    if (add(new_creature)){
      level_sum_ += (*new_creature).getLevel();
      if ((*new_creature).isTame()) {
        tame_count_++;
      }
      return true;
    }
  }
  return false;
}

bool Cavern::exitCavern(Creature* creature_to_remove) {
  if (remove(creature_to_remove)) {
    level_sum_ -= (*creature_to_remove).getLevel();
    if ((*creature_to_remove).isTame()) {
      tame_count_--;
    }
    return true;
  }
  return false;
}

int Cavern::getLevelSum() const {
  return level_sum_;
}
```

```
50
51  int Cavern::calculateAvgLevel() const {
52    if (isEmpty()) {
53      return 0;
54    }
55    return round(level_sum_ / getCurrentSize());
56  }
57
58  int Cavern::getTameCount() const {
59    return tame_count_;
60  }
61
62  double Cavern::calculateTamePercentage() const {
63    if (isEmpty()) {
64      return 0;
65    }
66    double tame_percent = (tame_count_>0) ?  (double(tame_count_) / item_count_) * 100: 0.0;
67    // return the tame percentage rounded up to two decimal places
68
69    return std::ceil(tame_percent * 100) / 100;
70
71  }
72
73  int Cavern::tallyCategory(const std::string& category) const {
74    if(category != "UNKNOWN" && category != "UNDEAD" && category != "MYSTICAL" && category !=
   "ALIEN") {
75      return 0;
76    }
77    int count = 0;
78    for (int i = 0; i < getCurrentSize(); i++) {
79      if (items_[i]->getCategory() == category) {
80        count++;
81      }
82    }
83    return count;
84  }
85
86  int Cavern::releaseCreaturesBelowLevel(int level) {
87    int count = 0;
88    if (level < 0) {
89      return 0;
90    }
91    else if (level == 0) {
92      count = getCurrentSize();
93      clear();
94      return count;
95    }
96    else {
97      int size = getCurrentSize();
98      for (int i = 0; i < size; i++) {
99        if (items_[i]->getLevel() < level) {
100         exitCavern(items_[i]);
```

```
        count++;
      }
    }
    return count;
  }
}

int Cavern::releaseCreaturesOfCategory(const std::string& category) {
  int count = 0;
  if (category == "ALL") {
    count = getCurrentSize();
    clear();
    return count;
  }
  else if (category != "UNKNOWN" && category != "UNDEAD" && category != "MYSTICAL" && category
!= "ALIEN") {
    return 0;
  }
  else {
    int size = getCurrentSize();
    for (int i = 0; i < size; i++) {
    if (items_[i]->getCategory() == category) {
      exitCavern(items_[i]);
      count++;
    }
  }
  return count;
  }
}

void Cavern::cavernReport() const {
  std::cout << "UNKNOWN: " << tallyCategory("UNKNOWN") << std::endl;
  std::cout << "UNDEAD: " << tallyCategory("UNDEAD") << std::endl;
  std::cout << "MYSTICAL: " << tallyCategory("MYSTICAL") << std::endl;
  std::cout << "ALIEN: " << tallyCategory("ALIEN") << std::endl;
  std::cout << std::endl;

  std::cout << "AVERAGE LEVEL: " << calculateAvgLevel() << std::endl;
  std::cout << "TAME: " << calculateTamePercentage() << "%" << std::endl;
}
```

Cavern.hpp Download

```
1  /*
2  CSCI235 Spring 2024
3  Project 3 - Cavern Class
4  Georgina Woo
5  Dec 24 2023
6  Cavern.hpp declares the Cavern class along with its private and public members
7  */
8  #ifndef CAVERN_HPP
9  #define CAVERN_HPP
10
11 #include "ArrayBag.hpp"
12 #include "Creature.hpp"
13 #include <vector>
14 #include <iostream>
15 #include <cmath>
16 #include <iomanip>
17 #include <fstream>
18
19 class Cavern : public ArrayBag<Creature*>{
20   public:
21     /**
22       Default constructor.
23       Default-initializes all private members.
24     */
25     Cavern();
26
27     Cavern(std::string& inputfile);
28     /**
29      * @param   :   A Creature entering the Cavern
30      * @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.
31      * @return  :   returns true if a Creature was successfully added to the Cavern (i.e. items_), false
   otherwise
32                 :   Hint: Use the above definition of equality will help determine if a Creature is already in
   the Cavern
33     **/
34     bool enterCavern(Creature* new_creature);
35
36
37     /**
38      * @param   :   A Creature leaving the Cavern
39      * @return  :  returns true if a creature was successfully removed from the Cavern (i.e. items_),
   false otherwise
40      * @post    :    removes the creature from the Cavern and updates the level sum and the tame
   count if the creature is tame.
41     **/
42     bool exitCavern(Creature* creature_to_remove);
43
44     /**
```

```
45      * @return   :  The integer level count of all the creatures currently in the Cavern
46      **/
47      int getLevelSum() const;
48
49      /**
50      * @return   :  The average level of all the creatures in the Cavern
51      * @post     :    Computes the average level of the Cavern rounded to the NEAREST integer.
52      **/
53      int calculateAvgLevel() const;
54
55      /**
56      * @return   :  The integer count of tame Creatures in the Cavern
57      **/
58      int getTameCount() const;
59
60      /**
61      * @return   :  The percentage (double) of all the tame creatures in the Cavern
62      * @post     :    Computes the percentage of tame creatures in the Cavern rounded up to 2 decimal
    places.
63      **/
64      double calculateTamePercentage() const;
65
66      /**
67       * @param  :   A string representing a creature Category with value in
68                  ["UNKNOWN", "UNDEAD", "MYSTICAL", "ALIEN"]
69       * @return  :  An integer tally of the number of creatures in the Cavern of the given category.
70                  If the argument string does not match one of the expected category values,
71                  the tally is zero.
72                  NOTE: no pre-processing of the input string necessary, only uppercase input will match.
73      **/
74      int tallyCategory(const std::string& category) const;
75
76      /**
77          @param  :   An integer representing the level treshold of the creatures to be removed from the
    Cavern, with default value 0
78          @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.
79          @return :   The number of creatures removed from the Cavern
80      */
81      int releaseCreaturesBelowLevel(int level = 0);
82
83      /**
84          @param  : A string representing a creature Category with a value in ["UNKNOWN", "UNDEAD",
    "MYSTICAL", "ALIEN"], or default value "ALL" if no category is given
85          @post   : Removes all creatures from the Cavern whose category matches the given category. If
    no category is given, removes all creatures from the Cavern.
86          @return : The number of creatures removed from the Cavern
87                  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.
88      */
89      int releaseCreaturesOfCategory(const std::string& category = "ALL");
```

```
90
91      /**
92       * @post    : Outputs a report of the creatures currently in the Cavern in the form:
93                  "UNKNOWN: [x]\nUNDEAD: [x]\nMYSTICAL: [x]\nALIEN: [x]\n\nThe average level is: [x]
        \n[x]% are tame.\n"
94                  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.
95
96                  Example output:
97                  UNKNOWN: 3
98                  UNDEAD: 5
99                  MYSTICAL: 8
100                 ALIEN: 6
101
102                 AVERAGE LEVEL: 7
103                 TAME: 46.67%
104     */
105     void cavernReport() const;
106
107     void displayCategory(const std::string &Cate);
108
109     void mycoMorselFeast();
110
111   private:
112     int level_sum_;     // sum of all the levels of the creatures in the cavern
113     int tame_count_;    // number of tame creatures in the cavern
114
115 };
116 #endif;
117
```

Creature.cpp Download

```cpp
/*
CSCI235 Spring 2024
Project 3 - Cavern
Georgina Woo
Nov 13 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
```

```
199 /**
200     @post     : displays Creature data in the form:
201     "[NAME]\n
202     Category: [CATEGORY]\n
203     Level: [LEVEL]\n
204     Hitpoints: [Hitpoints]\n
205     Tame: [TRUE/FALSE]"
206 */
207
208
209 bool Creature::operator==(const Creature& other_creature) const
210 {
211     return (name_ == other_creature.name_ && category_ == other_creature.category_ && level_ ==
    other_creature.level_ && tame_ == other_creature.tame_);
212 }
213
214 bool Creature::operator!=(const Creature& other_creature) const
215 {
216     return !(*this == other_creature);
217 }
```

Creature.hpp

Download

```
/*
CSCI235 Spring 2024
Project 3 - Cavern
Georgina Woo
Nov 13 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>


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.
      */
      Creature(const std::string& name, Category category = UNKNOWN, int hitpoints = 1, int level = 1, bool tame = false);

      /**
          @param : the name of the Creature, a string
          @post  : sets the Creature's name to the value of the parameter in UPPERCASE (convert any lowercase character to upppercase
                   Only alphabetical characters are allowed.
                 : If the input contains non-alphabetic characters, do nothing.
```

```
46          @return : true if the name was set, false otherwise
47      */
48      bool setName(const std::string& name);
49
50      /**
51          @return : the name of the Creature
52      */
53      std::string getName() const;
54
55
56      /**
57          @param  : the category of the Creature (an enum)
58          @post   : sets the Creature's category to the value of the parameter
59      */
60      void setCategory(const Category& category);
61
62
63      /**
64          @return : the race of the Creature (in string form)
65      */
66      std::string getCategory() const;
67
68      /**
69          @param  : an integer that represents the creature's hitpoints
70          @pre    : hitpoints > 0 : Creatures cannot have 0 or negative hitpoints (do nothing for invalid
    input)
71          @post   : sets the hitpoints private member to the value of the parameter
72          @return : true if the hitpoints were set, false otherwise
73      */
74      bool setHitpoints(const int& hitpoints);
75
76
77      /**
78          @return : the value stored in hitpoints_
79      */
80      int getHitpoints() const;
81
82      /**
83          @param  : an integer level
84          @pre    : level > 0 : Creatures cannot have 0 or negative levels (do nothing for invalid input)
85          @post   : sets the level private member to the value of the parameter
86          @return : true if the level was set, false otherwise
87      */
88      bool setLevel(const int& level);
89
90
91      /**
92          @return : the value stored in level_
93      */
94      int getLevel() const;
95
96
```

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

```
144                          not equal, or if one is tame and the other is not.
145                          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
146                          deemed "NOT equal".
147       */
148       bool operator!=(const Creature& rhs) const;
149
150    private:
151       // The name of the creature (a string in UPPERCASE)
152       std::string name_;
153       // The category of the creature (an enum)
154       Category category_;
155       // The creature's hitpoints (a non-zero, non-negative integer)
156       int hitpoints_;
157       // The creature's level (a non-zero, non-negative integer)
158       int level_;
159       // A flag indicating whether the creature is tame
160       bool tame_;
161
162 };
163
164 #endif
```

Dragon.cpp Download

```cpp
/*
CSCI235 Spring 2024
Project 2 - Derived Classes
Georgina Woo
Dec 23 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;
```

```
}

int Dragon::getNumberOfHeads() const
{
    return number_of_heads_;
}

bool Dragon::setNumberOfHeads(const int& number_of_heads)
{
    if(number_of_heads > 0)
    {
        number_of_heads_ = number_of_heads;
        return true;
    }
    else
    {
        return false;
    }
}

bool Dragon::getFlight() const
{
    return flight_;
}

void Dragon::setFlight(const bool& flight)
{
    flight_ = flight;
}

void Dragon::display()const{
    std::cout
    << "DRAGON - " << getName() << "\n"
    << "CATEGORY: " << getCategory() << "\n"
    << "HP: " << getHitpoints() << "\n"
    << "LVL: " << getLevel() << "\n"
    << "TAME: " << (isTame()? "TRUE" : "FALSE") << "\n"
    << "ELEMENT: " << getElement() << "\n"
    << "HEADS: " << getNumberOfHeads() << "\n"
    << "IT " << (getFlight() ? "CAN" : "CANNOT") << " FLY\n";
}

bool Dragon::eatMycoMorsel(){
    if(getCategory() == "UNDEAD"){
        setTame(true);
        setHitpoints(getHitpoints() +1);
    }
    if(getCategory() == "ALIEN"){
        setHitpoints(getHitpoints() +1);
    }
    if(getCategory() == "MYSTICAL"){
        if(getElement() == "FIRE"|| "EARTH"){
```

```
100             setHitpoints(getHitpoints() +1);
101         }
102         else{
103         if(getHitpoints() == 1 ){
104             return true;
105         }
106         else{
107             setHitpoints(getHitpoints() - 1);
108             setTame(false);
109         }
110         }
111     }
112     return false;
113 }
```

Dragon.hpp Download

```
1  /*
2  CSCI235 Spring 2024
3  Project 2 - Derived Classes
4  Georgina Woo
5  Dec 23 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);

     virtual void display() const override;

     virtual bool eatMycoMorsel() override;

     private:
        Element element_;
        int number_of_heads_;
        bool flight_;

```

```
98  };
99
100 #endif // DRAGON_HPP
```

Ghoul.cpp Download

```cpp
/*
CSCI235 Spring 2024
Project 2 - Derived Classes
Georgina Woo
Dec 23 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:
```

```
            return "FLESHGORGER";
        case SHADOWSTALKER:
            return "SHADOWSTALKER";
        case PLAGUEWEAVER:
            return "PLAGUEWEAVER";
        default:
            return "NONE";
    }
}

void Ghoul::setFaction(const Faction& faction)
{
    faction_ = faction;
}

bool Ghoul::getTransformation() const
{
    return transformation_;
}

void Ghoul::setTransformation(const bool& transformation)
{
    transformation_ = transformation;
}

void Ghoul::display() const{
    std::cout
    << "GHOUL - " << getName() << "\n"
    << "CATEGORY: " << getCategory() << "\n"
    << "HP: " << getHitpoints() << "\n"
    << "LVL: " << getLevel() << "\n"
    << "TAME: " << (isTame()? "TRUE" : "FALSE") << "\n"
    << "DECAY: " << getDecay() << "\n"
    << "FACTION: " << getFaction() << "\n"
    << "IT " << (getTransformation() ? "CAN" : "CANNOT") << " TRANSFORM\n";
}

bool Ghoul::eatMycoMorsel(){
    if(getFaction() == "FLESHGORGER"){
        if(isTame()){
            setTame(false);
        }
        if(isTame() == false){
            return true;
        }
    }
    if(getFaction() == "SHADOWSTALKER"){
        if(isTame() == true){
            if(getHitpoints() > 1){
                setHitpoints(getHitpoints() -1);
            }
            if(getHitpoints() == 1){
```

```
101                 setTame(false);
102             }
103         }
104 
105     }
106     if(getCategory() == "UNDEAD" && !(getFaction() == "FLESHGORGER") && !(getFaction() ==
    "SHADOWSTALKER")){
107         setTame(true);
108         setHitpoints(getHitpoints() + 1);
109     }
110     return false;
111 }
112 
113 
114 
```

Ghoul.hpp

Download

```
/*
CSCI235 Spring 2024
Project 2 - Derived Classes
Georgina Woo
Dec 23 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      virtual void display() const override;
89
90      virtual bool eatMycoMorsel() override;
91    private:
92      int decay_;
93      Faction faction_;
94      bool transformation_;
95  };
96
97  #endif // GHOUL_HPP
```

Makefile Download

```
CXX = g++
CXXFLAGS = -std=c++17 -g -Wall -O2

PROG ?= main
OBJS = Creature.o Dragon.o Ghoul.o Mindflayer.o Cavern.o main.o

all: $(PROG)

.cpp.o:
	$(CXX) $(CXXFLAGS) -c -o $@ $<

$(PROG): $(OBJS)
	$(CXX) $(CXXFLAGS) -o $@ $(OBJS)

clean:
	rm -rf $(EXEC) *.o *.out main

rebuild: clean all

```

Mindflayer.cpp Download

```cpp
/*
CSCI235 Spring 2024
Project 2 - Derived Classes
Georgina Woo
Dec 23 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:
```

```
            return "TELEPATHIC";
        case ILLUSIONARY:
            return "ILLUSIONARY";
        default:
            return "NONE";
    }
}

void Mindflayer::display() const{
    std::cout
    << "MINDFLAYER - " << getName() << "\n"
    << "CATEGORY: " << getCategory() << "\n"
    << "HP: " << getHitpoints() << "\n"
    << "LVL: " << getLevel() << "\n"
    << "TAME: " << (isTame()? "TRUE" : "FALSE") << "\n"
    << "SUMMONING: " << (getSummoning()? "TRUE" : "FALSE") << "\n";
    for(int i =0; i < getProjectiles().size(); i++){
        auto temp = getProjectiles()[i];
        std::cout << variantToString(temp.type_)<< " : " << temp.quantity_ << "\n";
    }

    if(getAffinities().size()){
        std::cout << "AFFINITIES:\n";
        for(int i =0; i < getProjectiles().size(); i++){
            std::cout << variantToString(getAffinities()[i]) << "\n";
        }
    }
}

bool Mindflayer::eatMycoMorsel(){
    if(getCategory() == "UNDEAD"){
        setTame(true);
        setHitpoints(getHitpoints() +1);
        return false;
    }

    if(getCategory() == "MYSTICAL"){
        if(!getSummoning()){
            if(isTame()){
                if(getHitpoints() == 1){
                    setTame(false);
                    return false;
                }
                if(getHitpoints() > 1){
                    setHitpoints(getHitpoints() -1);
                    return false;
                }
            }
        if(!isTame()&& getHitpoints() == 1){
            return true;
            }
        }
```

```
151     }
152
153     if(getCategory() == "ALIEN"){
154         for(int i =0; i < getAffinities().size(); i++){
155             if(variantToString(getAffinities()[i]) == "TELEPATHIC"){
156                 setHitpoints(getHitpoints() +1);
157                 return false;
158             }
159         }
160         for(int i =0; i < getProjectiles().size(); i++){
161             if((getProjectiles()[i]).type_ == TELEPATHIC){
162                 if((getProjectiles())[i].quantity_ > 1 ){
163                     (getProjectiles()[i]).quantity_ -= 1;
164                     return false;
165                 }
166                 if((getProjectiles()[i]).quantity_ == 1){
167                     projectiles_.erase(projectiles_.begin() + i);
168                     return false;
169                 }
170             }
171         }
172     setHitpoints(getHitpoints() + 2);
173     setTame(true);
174     return false;
175     }
176 }
```

Mindflayer.hpp

Download

```
/*
CSCI235 Spring 2024
Project 2 - Derived Classes
Georgina Woo
Dec 23 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;

      virtual void display() const override;

      virtual bool eatMycoMorsel() override;

   private:
      std::vector<Projectile> projectiles_;
      bool summoning_;
      std::vector<Variant> affinities_;

};
#endif // MINDFLAYER_HPP
```

README.md Download

```
[![Review Assignment Due Date](https://classroom.github.com/assets/deadline-readme-button-24ddc0f5d75046c5622901739e7c5dd533143b0c8e959d652212380cedb1ea36.svg)](https://classroom.github.com/a/ncdduum7)
# Project4

The specification for this project can be found on Blackboard

```

creatures.csv

Download

```
TYPE,NAME,CATEGORY,HITPOINTS,LEVEL,TAME,ELEMENT/FACTION,HEADS,FLIGHT/TRANSFORM/SUMM
DRAGON,JHARYX,UNDEAD,5,11,1,FIRE,5,0,0,NONE,NONE
DRAGON,DRIFON,ALIEN,3,8,0,WATER,3,1,0,NONE,NONE
GHOUL,ZYRAJA,MYSTICAL,11,11,1,FLESHGORGER,1,0,1,NONE,NONE
MINDFLAYER,NYLTHOR,ALIEN,2,2,1,NONE,1,0,0,PSIONIC,PSIONIC-2
DRAGON,QUIVARA,UNDEAD,3,2,1,EARTH,3,0,0,NONE,NONE
GHOUL,LYTHARA,ALIEN,2,10,1,PLAGUEWEAVER,1,0,1,NONE,NONE
GHOUL,ZEPHYX,MYSTICAL,10,4,1,SHADOWSTALKER,1,1,1,NONE,NONE
MINDFLAYER,FAELAN,MYSTICAL,1,4,0,NONE,1,1,0,TELEPATHIC,ILLUSIONARY-3
DRAGON,VYNTHOR,UNDEAD,9,4,1,AIR,9,0,0,NONE,NONE
MINDFLAYER,QUIXARA,ALIEN,11,4,0,NONE,1,1,0,NONE,NONE
GHOUL,THALYN,MYSTICAL,5,8,0,FLESHGORGER,1,1,0,NONE,NONE
MINDFLAYER,XYLIX,UNDEAD,11,4,1,NONE,1,0,0,ILLUSIONARY,TELEPATHIC-1;ILLUSIONARY-2
DRAGON,ZEPHYRA,UNDEAD,10,11,1,WATER,10,0,0,NONE,NONE
DRAGON,VYLTHOR,ALIEN,8,10,1,EARTH,8,0,0,NONE,NONE
GHOUL,QUIRIN,MYSTICAL,2,10,0,SHADOWSTALKER,1,1,0,NONE,NONE
MINDFLAYER,ZYRANA,ALIEN,5,6,0,NONE,1,1,0,PSIONIC;TELEPATHIC,NONE
GHOUL,MYTHOS,UNDEAD,11,9,0,PLAGUEWEAVER,1,0,0,NONE,NONE
DRAGON,KRYLIX,ALIEN,6,1,0,AIR,6,1,0,NONE,NONE
GHOUL,VORYN,MYSTICAL,2,4,1,FLESHGORGER,1,0,1,NONE,NONE
MINDFLAYER,JHRISMAS,MYSTICAL,1,1,0,NONE,1,1,0,NONE,NONE
```