```cpp
// Lost and Found: Survival Game
//          By: Zach Hollis

// Unless otherwise stated, all code is written by Zach Hollis and is
//      copywrited as of 2016. It may only be used with the authors permission!
//  If you wish to use any of the code below you may contact me at: zhollis21@gmail.com

/* !!! This game was designed and tested on Windows OS it may not function correctly on other OS !!! */

#include <iostream>
#include <string>
#include <vector>
#include <map>
#include <random>
#include <ctime>
#include <conio.h>
#include <sstream>
#include <cctype>
#include <climits>
#include <fstream>

using namespace std;



            // Class Independent Global Variables

const double instant = 0; // Used to for things we want to be printed "instantly" like the map
const double fast_type_speed = 0.02; // Used for repeatitive and long messages
const double norm_type_speed = 0.04; // Used for the normal messages and story
const double slow_type_speed = 0.08; // Used for the important and dramatic story lines
const int max_health = 10; // Used to limit the amout you can regain your health to
const int max_hunger = 5; // Used to limit the amount you can increase your hunger
const int max_defenses = 5; // Used to limit the amount you can build camp's defense
bool starve_warning = false; // Player is given a warning when starving, this keeps track of if they were warned
map <string, int>::iterator mit; // Global Map Iterator
string player_name = "";
const char enter = 13; // Represents the char given via the enter key
const int max_cycles = 8; // Number of turns in a day
const int bear_health = 10; // Used to build Animal bear obj health
const int bear_attack = 5; // Used to build Animal bear obj attack
const int lion_health = 8; // Used to build Animal lion obj health
const int lion_attack = 4; // Used to build Animal lion obj attack
const int gator_health = 6; // Used to build Animal gator obj health
```

```cpp
const int gator_attack = 3; // Used to build Animal gator obj attack
const int wolf_health = 4; // Used to build Animal wolf obj health
const int wolf_attack = 2; // Used to build Animal wolf obj attack
const int snake_health = 2; // Used to build Animal snake obj health
const int snake_attack = 1; // Used to build Animal snake obj attack
int days = 1; // Keeps track of the day (each day is 6-9 turns, depending on when they sleep)
bool escaped = false;
bool leaderboards = false;
bool load = false;
bool ret_bear, ret_lion, ret_gator, ret_wolf, ret_snake;
int missing_emblems;

//--------------------------------------------------------------------------------------------------

            // Class Definitions

// Class which defines the main character or person
class Person
{
public:
    friend class Database;

    Person() : hunger(max_hunger), health(max_health), poisoned(false), poison_recovery(5) {};
    int get_hunger() { return hunger; }
    int get_health() { return health; }
    bool get_poison() { return poisoned; }
    void track_poison();
    void decay_hunger(int decay_amount);
    void decay_health(int decay_amount);
    void build_hunger(int build_amount);
    void build_health(int build_amount);
    void change_poisoned(bool TorF) { poisoned = TorF; }
    void revert_poison_recovery() { poison_recovery = poison_recovery_turns; }
    void status();
    void escape();
    int get_attack() { return attack; }
    string chance();
    bool poss_encounter();
    void poss_build_health(int amount);


protected:
    int hunger;
    int health;
    const int attack = 3;
    bool poisoned;
    int poison_recovery;
    const int poison_recovery_turns = 5;
};
```

```
 94
 95 // Class which sets up a choice and the consequenses of that choice
 96 class Choice
 97 {
 98 public:
 99     char orgin_choice();
100     char menu_choice();
101     void help_choice();
102     char explore_choice();
103     void build_camp_choice();
104     void gather_choice();
105     char food_choice(string name, int total);
106     void open_inventory();
107 };
108
109 // Camp is a class which keeps track of the players base/camp
110 class Camp
111 {
112     friend class Database;
113 public:
114     Camp() : defenses(0), noose(false), spring(false), hole(false) {};
115     void build_defenses(int build_amount);
116     void decay_defenses(int decay_amount);
117     int get_defenses() { return defenses; }
118     void woken(string it);
119     bool get_noose() { return noose; }
120     bool get_spring() { return spring; }
121     bool get_hole() { return hole; }
122     void set_noose(bool val) { noose = val; }
123     void set_spring(bool val) { spring = val; }
124     void set_hole(bool val) { hole = val; }
125
126 protected:
127     int defenses;
128     bool noose;
129     bool spring;
130     bool hole;
131 };
132
133 // Tracks and manages stored items
134 class Inventory
135 {
136     friend class Database;
137 public:
138     bool food_empty() { return food_inventory.empty(); }
139     bool item_empty() { return item_inventory.empty(); }
140     void add(char type, bool secret, string name, int num);
141     void use(char type, string name, int num);
142     void remove(char type, string name, int num);
143     void clear() { food_inventory.clear(); item_inventory.clear(); }
144     void print_inv(char type);
145
```

```
146 protected:
147     map <string, int> food_inventory;
148     map <string, int> item_inventory;
149 };
150 
151 // The Time_Write class is a heavily modified version of a class originally
152 // developed by Dr. Henry Suters of Carson-Newman University. Said class was
153 // originally designed to just track time. I have added output and input
      functions
154 // which take advantage of the tracked time.
155 class Time_Write
156 {
157 public:
158     Time_Write();
159     void write(string s);
160     void set_type_speed(double speed) { type_speed = speed; }
161     string num_to_string(int num); // Modified version of a function I found on
        "cplusplus.com"
162     int search(string &key, char c, int size);
163     char get_input(); // Very tricky solution to player hitting too many buttons
        and computer still going off old input
164     bool time_key(bool boss_battle);
165     bool key_race(string an);
166     string replace_spaces(string s);
167     string restore_spaces(string s);
168     double elapsed_time();
169     void reset();
170 
171 private:
172     clock_t start_time;
173     double type_speed;
174     const string norm_battle_key = "1234567890";
175     const string boss_battle_key = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
176     const double press_time = 1.35;
177     const double run_speed = .5;
178 };
179 
180 // Class which defines food values and what to do with them
181 class Food
182 {
183     friend class Database;
184 public:
185     void random_food();
186     void poss_poison(int food, string name);
187     void found_berries();
188     void found_mushrooms();
189     void found_eggs();
190     void found_insects();
191     void leave(string food);
192     void eat(string name, int num);
193 };
194 
```

```
195 // Tracks the player and key locations, also prints off a grid for the player to ↵
       see
196 class Map
197 {
198     friend class Database;
199 public:
200     void rand_set_coords();
201     void print_map();
202     char check_self(int x, int y, bool decode);
203     char check_camp(int x, int y, bool decode);
204     char check_others(int x, int y, bool decode, bool filter);
205     bool move_camp();
206     void move_self(char command);
207     void hit_wall(string direction);
208     bool not_camp() { return self_x != camp_x || self_y != camp_y; }
209     void print_move(string direction);
210     string key_str(int i);
211     void reset();
212     void unlock();
213     string spec_loc();
214     void investigate();
215     bool use(string emblem);
216 
217     char snake_c = 'T';
218     char wolf_c = 'H';
219     char bear_c = 'B';
220     char cabin_c = 'L';
221     char water_c = 'W';
222     char temple_c = 'A';
223 
224 protected:
225     void rand_num_loop(int &x, int &y);
226     int columns = 53;
227     int rows = 19;
228     int self_x, self_y;
229     int camp_x, camp_y;
230     int snake_x, snake_y;
231     int wolf_x, wolf_y;
232     int bear_x, bear_y;
233     int cabin_x, cabin_y;
234     int waterfall_x, waterfall_y;
235     int temple_x, temple_y;
236     bool found_snake;
237     bool found_wolf;
238     bool found_bear;
239     bool found_cabin;
240     bool found_waterfall;
241     bool found_temple;
242     int unexplored_map[81]; // Keeps track of where the player has been
243 };
244 
245 // Manages turn based events such as hunger, day, night, and sleep
```

```
246 class Turn
247 {
248     friend class Database;
249 public:
250     Turn() : turn(0) {};
251     int get_turn() { return turn; }
252     void next_turn();
253     void reset_turn();
254     void set_records();
255     void check_records(int turndays, string name, bool lead_escaped, bool
          secret);
256     void print_leaderboard();
257
258 protected:
259     int turn;
260     vector<string> days_survived_names;
261     vector<int> days_survived;
262     vector<string> days_escaped_names;
263     vector<int> days_escaped;
264 };
265
266 // Class used to keep track of animals and their interaction with the player
267 class Animal
268 {
269     friend class Database;
270 public:
271     Animal(string name_stat, int attack_stat, int health_stat, bool boss_stat) :
         name(name_stat),
272         attack(attack_stat), health(health_stat), org_health(health_stat), boss
            (boss_stat)  {};
273     void battle();
274     string get_name() { return name; }
275     void decay_health(int amount);
276     void reset_health() { health = org_health; }
277     void poss_poison(string name);
278     bool boss_prep();
279     bool dead() { return health < 1; }
280
281 private:
282     string name;
283     int attack;
284     int health;
285     bool boss;
286     const int org_health;
287 };
288
289 // Class designed to load in and back up most recent game data
290 class Database
291 {
292 public:
293     bool load_in();
294     void load_in_leaderboards();
```

```
295     void back_up();
296
297 private:
298     string filename = "PDB.txt"; // Name of data storage file
299     string lfilename = "Leaderboards.txt"; // Name of leaderboards storage file
300
301     // Strings for printing and testing in load_in() and back_up().
302     const string pDays = "Day";
303     const string pStarve_Warning = "Starve_Warning";
304     const string pPlayer_Name = "Player_Name";
305     const string pLeaderboards = "Leaderboards";
306     const string pHealth = "Health";
307     const string pHunger = "Hunger";
308     const string pPoisoned = "Poisoned";
309     const string pPoison_Rec = "Poison_Rec";
310     const string pDefenses = "Defenses";
311     const string pNoose = "Noose";
312     const string pSpring = "Spring";
313     const string pHole = "Hole";
314     const string pFood_Inv = "Food_Inv";
315     const string pItem_Inv = "Item_Inv";
316     const string pSelf_X = "Self_X";
317     const string pSelf_Y = "Self_Y";
318     const string pCamp_X = "Camp_X";
319     const string pCamp_Y = "Camp_Y";
320     const string pSnake_X = "Snake_X";
321     const string pSnake_Y = "Snake_Y";
322     const string pSnake_Health = "Snake_Health";
323     const string pFound_Snake = "Found_Snake";
324     const string pWolf_X = "Wolf_X";
325     const string pWolf_Y = "Wolf_Y";
326     const string pWolf_Health = "Wolf_Health";
327     const string pFound_Wolf = "Found_Wolf";
328     const string pBear_X = "Bear_X";
329     const string pBear_Y = "Bear_Y";
330     const string pBear_Health = "Bear_Health";
331     const string pFound_Bear = "Found_Bear";
332     const string pCabin_X = "Cabin_X";
333     const string pCabin_Y = "Cabin_Y";
334     const string pLion_Health = "Lion_Health";
335     const string pFound_Cabin = "Found_Cabin";
336     const string pWaterfall_X = "Waterfall_X";
337     const string pWaterfall_Y = "Waterfall_Y";
338     const string pGator_Health = "Gator_Health";
339     const string pFound_Waterfall = "Found_Waterfall";
340     const string pTemple_X = "Temple_X";
341     const string pTemple_Y = "Temple_Y";
342     const string pFound_Temple = "Found_Temple";
343     const string pRet_Bear = "Ret_Bear";
344     const string pRet_Lion = "Ret_Lion";
345     const string pRet_Gator = "Ret_Gator";
346     const string pRet_Wolf = "Ret_Wolf";
```

```
347     const string pRet_Snake = "Ret_Snake";
348     const string pMissing_Emblems = "Missing_Emblems";
349     const string pEscaped = "Escaped";
350     const string pTurn = "Turn";
351     const string pDays_Survived = "Days_Survived";
352     const string pDays_Escaped = "Days_Escaped";
353     const string pUnexplored_Map = "Unexplored_Map";
354     const string pEnd = "End";
355 };
356
357 //------------------------------------------------------------------------------
    ------------------------------------------------
358
359             // Global Class Variable Definitions
360
361 Person person;  // Global Person Variable
362 Camp camp;      // Global Camp Variable
363 Time_Write tw;  // Global Time_Write Variable
364 Food food;      // Global Food Variable
365 Choice pick;    // Global Choice Variable
366 Inventory inv;  // Global Inventory Variable
367 Map m;          // Global Map Variable   *Note it is very different from a "map"
       type
368 Turn turn;      // Global Turn Variable
369 Database db;    // Global Database Variable
370
371 // Global Animal Variables
372 Animal bear("bear", bear_attack, bear_health, false);
373 Animal lion("mountain lion", lion_attack, lion_health, false);
374 Animal gator("alligator", gator_attack, gator_health, false);
375 Animal wolf("wolf", wolf_attack, wolf_health, false);
376 Animal snake("snake", snake_attack, snake_health, false);
377 Animal boss_bear("Great Bear", bear_attack, 20, true);
378 Animal boss_lion("Mountain Lion", lion_attack, 18, true);
379 Animal boss_gator("Giant Alligator", gator_attack, 16, true);
380 Animal boss_wolf("Alpha Wolf", wolf_attack, 14, true);
381 Animal boss_snake("Giant Snake", snake_attack, 12, true);
382
383 //------------------------------------------------------------------------------
    ------------------------------------------------
384
385             // Global Functions (excluding main() of course)
386
387 // Function which prints a random number between and including 1 and "end"
388 int random(int end)
389 {
390     int n = (((rand() % end) * (clock() % end) + person.get_hunger() * (rand() %
          end))) % end + 1;
391     return n;
392 }
393
394 // Resets data at the start of every new game
```

```
395 void reset_data()
396 {
397     person.build_health(max_health);
398     person.build_hunger(max_hunger);
399     person.change_poisoned(false);
400     person.revert_poison_recovery();
401     camp.decay_defenses(max_defenses);
402     camp.set_noose(false);
403     camp.set_spring(false);
404     camp.set_hole(false);
405     boss_snake.reset_health();
406     boss_wolf.reset_health();
407     boss_gator.reset_health();
408     boss_lion.reset_health();
409     boss_bear.reset_health();
410     escaped = ret_bear = ret_lion = ret_gator = ret_wolf = ret_snake = false;
411     missing_emblems = 5;
412     m.reset();
413     srand(time(NULL));
414     inv.clear();
415     turn.reset_turn();
416     leaderboards = false;
417     days = 1;
418     db.back_up();
419 }
420
421 // Database which holds the storyline strings
422 void print_intro()
423 {
424         tw.write("\n\n\tYou: ...Uggh... Where am I???\thello... Hello?!... Can
          anyone hear me?!...\n\n"
425             "\tYou have woken up in the middle of a dark forest with no one in
              sight.\n\t"
426             "You have absolutely no memory of who you are or where you are at.\n
              \t"
427             "You examine yourself trying to figure out something about yourself
              and why you are here...\n\n\t"
428             "You are wearing worn out jeans, a dirt covered tee-shirt, and some
              old tennis shoes\n\t"
429             "You check your pockets... No luck, just an empty map...");
430
431         tw.write("\n\n\tYou are really starting to panic now!\n\t"
432             "You have no clue, who you are, where you are, or even what is going
                on!\n\n\t"
433             "Okay... Okay... You try to calm yourself down...\n\t"
434             "You tell yourself to just sit down and figure it out.\n\n\t");
435
436         tw.write("Okay, so you are lost in the middle of a forest...\n\t");
437 }
438
439 // encodes data so you cannot cheat
440 string encode(string uncoded)
```

```
441 {
442     string coded = "";
443     char c;
444     for (int i = 0; i < uncoded.size(); i++)
445     {
446         c = uncoded[i];
447         coded.append("x" + tw.num_to_string((int)c));
448     }
449     return coded;
450 }
451
452 // decodes the input from database file
453 string decode(string coded)
454 {
455     string decoded = "";
456     string s = "";
457     char c;
458     for (int i = 0; i < coded.size(); i++)
459     {
460         c = coded[i];
461         if (c == 'x')
462         {
463             if (!s.empty())
464             {
465                 decoded += (string(1, (char)atoi(s.c_str())));
466                 s = "";
467             }
468         }
469         else
470         {
471             s.append(string(1,c));
472         }
473         if (i == coded.size()-1)
474             decoded += (string(1, (char)atoi(s.c_str())));
475     }
476     return decoded;
477 }
478
479 //------------------------------------------------------------------------------ ↵
    ----------------------------------------------
480
481             // Person Class Functions
482
483 // Increments Players health by build_amount up to max_helath
484 void Person::build_health(int build_amount)
485 {
486     for (int i = 0; i < build_amount && health < max_health; i++)
487         health++;
488 }
489
490 // Increments Players hunger by build_amount up to one past max_hunger.
491 //  It goes one past because if it stops at max_hunger, next_turn() will
```

```
492 //  decrease to max_hunger - 1, so the player would never be full
493 void Person::build_hunger(int build_amount)
494 {
495     for (int i = 0; i < build_amount && hunger <= max_hunger; i++)
496         hunger++;
497     if (hunger > 0)
498         starve_warning = false;
499 }
500
501 // Decays Players health by decay amount to max of 0
502 void Person::decay_health(int decay_amount)
503 {
504     for (int i = 0; i < decay_amount && health > 0; i++)
505         health--;
506 }
507
508 // Decays Players hunger by decay_amount to max of 0
509 void Person::decay_hunger(int decay_amount)
510 {
511     for (int i = 0; i < decay_amount && hunger > 0; i++)
512         hunger--;
513 }
514
515 // Prints a status report to let the player know thier
516 // character and enviroment's condition in the game.
517 void Person::status()
518 {
519     int defense_stat = camp.get_defenses();
520     int curr_turn = turn.get_turn();
521
522     if (hunger > max_hunger) // Because person.build_hunger goes 1 past
      max_hunger
523         hunger = max_hunger; // we correct it here(after next_turn()) so the
        player can be full
524
525     if (health > 0 && !escaped)
526     {
527         tw.set_type_speed(fast_type_speed);
528         tw.write("\n\t\t\tDay: " + tw.num_to_string(days));
529         tw.write("\n\tYour Health: ");
530         tw.write(tw.num_to_string(health) + " / " + tw.num_to_string
          (max_health));
531         if (person.get_poison())
532             tw.write(" - Poisoned");
533
534         tw.write("\n\tYour Hunger: ");
535         if (hunger == max_hunger)
536             tw.write("Full");
537         else if (hunger == 4)
538             tw.write("Good");
539         else if (hunger == 3)
540             tw.write("Moderate");
```

```
541         else if (hunger == 2)
542             tw.write("Hungry");
543         else if (hunger == 1)
544             tw.write("Famished");
545         else if (hunger == 0)
546             tw.write("Starving");
547 
548         tw.write("\n\tCamp Strength: ");
549         if (defense_stat == max_defenses)
550             tw.write("Fort Knox");
551         else if (defense_stat == 4)
552             tw.write("Brick House");
553         else if (defense_stat == 3)
554             tw.write("Stick House");
555         else if (defense_stat == 2)
556             tw.write("Straw House");
557         else if (defense_stat == 1)
558             tw.write("House of Cards");
559         else if (defense_stat == 0)
560             tw.write("No Camp");
561 
562         tw.write("\n\tSun Positon: ");
563         if (curr_turn == 1)
564             tw.write("Sunrise");
565         else if (curr_turn == 2)
566             tw.write("Midway Rising");
567         else if (curr_turn == 3)
568             tw.write("Highest Point");
569         else if (curr_turn == 4)
570             tw.write("Midway Setting");
571         else if (curr_turn == 5)
572             tw.write("Sunset");
573         else if (curr_turn >= 6)
574             tw.write("Night");
575 
576         tw.write("\n\t");
577     }
578     else
579     {
580         db.back_up();
581         tw.set_type_speed(slow_type_speed);
582         if (escaped)
583         {
584             tw.write("\n\n\t\tCongratulations!!!\n\t\t"
585             "\n\t\tYou escaped! It took you " + tw.num_to_string(days - 1) + "
                  days "
586                 "and " + tw.num_to_string((turn.get_turn()- 1) * 3) + " hours!
                    \n");
587         }
588         else
589         {
590             tw.write("\n\t\tYou Died! You survived " + tw.num_to_string(days -
```

```
                1) + " day");
591             if (days > 1)
592                 tw.write("s");
593             tw.write(" and " + tw.num_to_string((turn.get_turn() - 1) * 3) + "
                  hours!\n\n\t\tGAME OVER!!!\n");
594         }
595         if (!leaderboards)
596         {
597             int turndays = (days - 1) * max_cycles + (turn.get_turn() - 1);
598             turn.check_records(turndays, player_name, escaped, false);
599         }
600         db.back_up();
601         pick.menu_choice();
602     }
603 }
604
605 // Prints out the storyline for escape and sets Escaped to True
606 void Person::escape()
607 {
608     escaped = true;
609     tw.set_type_speed(slow_type_speed);
610     tw.write("\n\tYou approach the Temple yet again, determined to gain entry!\n
          \t"
611         "After a few hours of frustration and failure, you give up and start to
              head back...\n\t"
612         "\n\tHowever, as you walk back, you pass by the wall with the now
              complete circle of animal emblems.\n\t"
613         "For the first time you notice that there is an emblem in the middle of
              the circle.\n\t"
614         "The emblem appears to be in the shape of a man!\n\t"
615         "You remove the emblem from the animal emblems circling it and take a
              few steps back...\n\t"
616         "\n\tThe ground starts shaking!!!\n\t"
617         "You can hear sounds coming from inside the Temple as if it is
              rearranging itself...\n\t"
618         "\n\tSuddenly the wall with the animal emblems splits in two and slides
              down into the ground!\n\t"
619         "\n\tThere, where the wall used to be is a passageway.\n\t"
620         "The end of the passageway is shining like the sun, too bright to see
              what is at the end...\n\t"
621         "\n\tYou walk towards the end, hand over your eyes to shield yourself
              from the brightness.\n\t"
622         "\n\tSuddendly you are falling... Falling through space and time until
              you cease to exist...\n\t");
623     tw.set_type_speed(instant);
624     for (int i = 0; i < 10; i++)
625     {
626         tw.reset();
627         while (tw.elapsed_time() < 3); // empty loop which stalls for a second
628         tw.write("\n\tBeep!\n\t");
629     }
630     tw.set_type_speed(slow_type_speed);
```

```
631     tw.write("\n\tYou slowly regain consciousness and try to open your eyes...\n\t");
632     tw.reset();
633     while (tw.elapsed_time() < 3); // Another empty loop to stall
634     tw.set_type_speed(instant);
635     tw.write("\n\tBeep!\n\t");
636     tw.set_type_speed(slow_type_speed);
637     tw.write("\n\tYou open your eyes and try to take it all in...\n\t"
638         "\n\tYou appear to be in a hospital room.\n\t"
639         "You realize the high pitched beeping you keep hearing is the machine you are hooked up to.\n\t"
640         "\n\tThere are dozens of flowers that appear to have been dead for quite some time...\n\t"
641         "\n\tAs you still taking in everything, a woman walks into the door with a plate of food.\n\t"
642         "The woman sees you are awake, she drops the food and runs over to you! \n\t"
643         "She embraces you tightly as she weeps!\n\t"
644         "\n\tAfter a while, the woman finally calms down and the two of you start to talk.\n\t"
645         "It turns out you were in a terrible accident and have been in a comma for "
646         + tw.num_to_string(days - 1) + " days and " + tw.num_to_string((turn.get_turn() - 1) * 3) + "!\n\t"
647         "The woman is actually your wife and has been coming to visit you everyday on her lunch break.\n\t");
648 }
649
650 // If hunger is full and their health isn't low because of poisoning we increment health
651 void Person::poss_build_health(int amount)
652 {
653     if ((max_health - health) > (poison_recovery_turns - poison_recovery))
654         build_health(amount);
655 }
656
657 // Keeps up with poison recovery including health recovery
658 void Person::track_poison()
659 {
660     if (!poisoned || health == 0)
661         return;
662     if (poison_recovery < 1)
663     {
664         poisoned = false;
665         build_health(1);
666         revert_poison_recovery();
667         return;
668     }
669     if (poison_recovery < 5)
670         build_health(1);
671     poison_recovery--;
672 }
```

```
673
674 // Simulates the chance of running into a wild animal
675 string Person::chance()
676 {
677     if (random(21) == 11)
678         return "bear";
679     else if (random(19) == 10)
680         return "mountain lion";
681     else if (random(17) == 9)
682         return "alligator";
683     else if (random(15) == 8)
684         return "wolf";
685     else if (random(13) == 7)
686         return "snake";
687     else
688         return "nada";
689 }
690
691 bool Person::poss_encounter()
692 {
693     string s = person.chance();
694     int inv_num = 0;
695     char command;
696     bool ran_away = false;
697
698     if (s == "nada")
699         return true;
700     tw.set_type_speed(slow_type_speed);
701     tw.write("\n\tWhile traveling you were set upon by a " + s + "!\n\t");
702     tw.set_type_speed(fast_type_speed);
703     tw.write("\n\tWhat should you do?\n\t\t"
704         "Press 'F' to Fight\n\t\t"
705         "Press 'R' to Run\n\t");
706
707     do {
708         command = tw.get_input();
709     } while (command != 'r' && command != 'f');
710
711     if (command == 'r')
712         if (tw.key_race(s))
713         {
714             tw.set_type_speed(slow_type_speed);
715             tw.write("\n\n\tYou managed to run away from the " + s +
716                 ",\n\tbut ended up right back where you started!\n\t");
717             return false;
718         }
719     if (s == "bear")
720     {
721         bear.battle();
722         inv_num = 3;
723     }
724     else if (s == "mountain lion")
```

```
725     {
726         lion.battle();
727         inv_num = 2;
728     }
729     else if (s == "alligator")
730     {
731         gator.battle();
732         inv_num = 2;
733     }
734     else if (s == "wolf")
735     {
736         wolf.battle();
737         inv_num = 2;
738     }
739     else
740     {
741         snake.battle();
742         inv_num = 1;
743     }
744
745     if (person.get_health() > 0)
746     {
747         tw.write("\n\tYou managed to kill the " + s + "!\n");
748         inv.add('f', false, "raw " + s + " meat", inv_num);
749     }
750     if (health == 0)
751         return false;
752     else
753         return true;
754 }
755
756 //------------------------------------------------------------------------------
      ------------------------------------------------
757
758             // Choice Class Functions
759
760 // Prints standard string everytime the player chooses to gather.
761 void Choice::gather_choice()
762 {
763     tw.set_type_speed(fast_type_speed);
764     tw.write("\n\tYou decide it would be best to replenish your food supply.\n
        \t"
765         "You start off into the forest keeping an eye out for anything edible,\n
          \t"
766         "while making sure you remember how to get back where you started.\n
          \t");
767     food.random_food();
768 }
769
770 // Builds camp and keeps track of what defenses are armed
771 void Choice::build_camp_choice()
772 {
```

```
773     if (camp.get_defenses() > 0)
774     {
775         tw.set_type_speed(fast_type_speed);
776         tw.write("\n\tYou decide to better fortify your camp to guard you
              against\n\t"
777             "predators and whatever else might be lurking out there...\n\n\t");
778     }
779     tw.set_type_speed(norm_type_speed);
780     if (camp.get_defenses() < max_defenses)
781     {
782         if (camp.get_defenses() == 0)
783         {
784             tw.write("\n\tYou spend a few hours scouting a good place to set up
                a new camp.\n\t"
785                 "You figure you're not sure how long you are going to be lost, "
786                 "so might as well have a place to make camp.\n\n\t"
787                 "You find a defensible place with a water source nearby, then
                  gather\n\t"
788                 "nearby leaves and straw and make a soft place for you to sleep
                  tonight.\n\t");
789         }
790         else if (camp.get_defenses() == 1)
791         {
792             tw.write("You realize you will need a fire to stay warm in the cold
                nights.\n\t"
793                 "First, spend hours gathering leaves and dry wood of all shapes
                  and sizes.\n\t"
794                 "Then you use the friction of two of the smaller sticks to
                  create a hot ember.\n\t"
795                 "You quickly transfer the ember to your pile of leaves and use
                  this to create a small fire.\n\t"
796                 "The fire should keep you warm on cold nights and provide
                  limited cooking capabilities.\n\t");
797         }
798         else if (!camp.get_noose())
799         {
800             tw.write("You find nearby thorn bushes and begin weaving the
                branches together.\n\t"
801                 "You then use those woven thorny ropes and some well chosen
                  sticks,\n\t"
802                 "to create a thorned tree noose trap.\n\t");
803             camp.set_noose(true);
804         }
805         else if (!camp.get_spring())
806         {
807             tw.write("You find some sharp rocks and proceed to sharpen a dozen
                or so wooden poles.\n\t"
808                 "You design a trap which will, when disturbed, spring forward
                  and rapidly project the sharp wooden pole.\n\t"
809                 "Hopefully killing small animals or predators nearby.\n\t"
810                 "You then place these traps all around your camp.\n\t");
811             camp.set_spring(true);
```

```
812         }
813         else if (!camp.get_hole())
814         {
815             tw.write("You find more sturdy wooden poles and proceed to sharpen
                  them.\n\t"
816                 "Then you dig a wide hole about 5 feet deep and plant the sharp
                    wooden poles in the hole.\n\t"
817                 "Finally you place small sticks across the hole and cover it
                    with leaves.\n\t"
818                 "You are left with a trap which has the potential to protect you
                     from large predators.\n\t");
819             camp.set_hole(true);
820         }
821         camp.build_defenses(1);
822     }
823     else
824         tw.write("You try for a few hours to think of a way to make your camp
              more defensible.\n\t"
825             "However, after a few hours you are forced to give up.\n\tYou
                conclude this must be the "
826             "limit of your camp's defensive capabilities.\n\t");
827 }
828
829 //Prints the help instructions for the user
830 void Choice::help_choice()
831 {
832     tw.set_type_speed(norm_type_speed);
833     tw.write("\n\nHelp:\n\t"
834         "\n\tLost and Found is a text-based survival game designed by Zach
              Hollis.\n\t"
835         "\n\tLost and Found has only been tested on Windows OS it may not
              function correctly on other OS.\n\t"
836         "\n\tLost and Found is all about making smart choices.\n\t"
837         "The game gives you the freedom to do anything you wish, but those
              choices might have swift consequences...\n\t"
838         "\n\tHaving trouble surviving?\n\t"
839         "Health can be recovered by sleeping or keeping your hunger level
              moderate to full.\n\t"
840         "Try to limit your exploring until to have some food stored up.\n\t"
841         "When exploring, try making a sturdy camp in the center of a region to
              use as a base of operations.\n\t"
842         "\n\tDoes your game freeze up when you select \"Continue\"?\n\t"
843         "Most likely your game data file has been moved or corrupted.\n\t"
844         "Unfortunately you will need to start a new game.\n\t"
845         "\n\tEncountering bugs or glitches?\n\t"
846         "Contact the developer at: zhollis21@gmail.com\n\t"
847         "Be sure to include a detailed discription of the problem and if
              possible a screenshot.\n\t"
848         "\n\tThanks for playing!"
849         "\n\nPress anything to return to the Main Menu...\n");
850     tw.get_input();
851 }
```

```
852
853 // Exploration interface
854 char Choice::explore_choice()
855 {
856     m.print_map();
857     char command;
858     tw.set_type_speed(fast_type_speed);
859     tw.write("\n\tExplore Menu:\n\t\t"
860         "Press 'N' to explore North  (^)\n\t\t"
861         "Press 'E' to explore East   (->)\n\t\t"
862         "Press 'S' to explore South  (v)\n\t\t"
863         "Press 'W' to explore West   (<-)\n\t\t"
864         "Or Press 'B' to go Back to your choices\n\t");
865     do {
866         command = tw.get_input();
867     } while (command != 'n' && command != 'e' && command != 's' && command !=
      'w' && command != 'b');
868     return command;
869 }
870
871 // Main menu options and processing
872 char Choice::menu_choice()
873 {
874     char command, ans;
875     ans = ' ';
876     do {
877         do {
878             tw.set_type_speed(fast_type_speed);
879             tw.write("\n\nMain Menu:\n\t"
880                 "Press 'S' to Start a New Game\n\t"
881                 "Press 'C' to Continue a Previous Game\n\t"
882                 "Press 'L' to View the Leaderboards\n\t"
883                 "Press 'H' for Help and FAQ\n\t"
884                 "Press 'Q' to Quit\n");
885             do
886             {
887                 command = tw.get_input();
888             } while (command != 's' && command != 'c' && command != 'l' &&
                command != 'h' && command != 'q');
889
890             if (command == 'q')
891                 throw 21;
892             else if (command == 'h')
893                 help_choice();
894             else if (command == 'l')
895                 turn.print_leaderboard();
896         } while (command != 'c' && command != 's');
897
898         if (command == 'c')
899         {
900             load = db.load_in();
901             if (!load)
```

```
902             {
903                 cerr << "\nUnable to load in saved file!" << endl;
904             }
905             else
906                 tw.write("\n\n\tWelcome back, " + player_name + "...\n");
907         }
908         else if (command == 's')
909         {
910             tw.set_type_speed(norm_type_speed);
911             tw.write("\n\tAre you sure you want to start a new game? ('Y' or
                'N')\n\t"
912                 "This will erase your previous survival game data!\n\t");
913             do {
914                 ans = tw.get_input();
915             } while (ans != 'n' && ans != 'y');
916 
917             if (ans == 'y')
918             {
919                 tw.write("\n\tPlease enter your full name and then press
                    <Enter>: ");
920                 getline(cin, player_name);
921                 reset_data();
922                 db.back_up();
923                 tw.write("\n\tDo you want to hear the intro story? ('Y' or 'N')
                    \n\t");
924                 do {
925                     command = tw.get_input();
926                 } while (command != 'n' && command != 'y');
927                 if (command == 'y')
928                 {
929                     tw.set_type_speed(slow_type_speed);
930                     print_intro();
931                 }
932                 person.status();
933                 return command;
934             }
935         }
936     } while (ans != 'y' && !load);
937     person.status();
938     return command;
939 }
940 
941 // Gives in game choices and preforms based on input
942 char Choice::orgin_choice()
943 {
944     char command;
945     char ecom = 'b';
946     bool change = false;
947     string special = m.spec_loc();
948     bool is_spec = (special != "");
949     do {
950         tw.set_type_speed(fast_type_speed);
```

```
951         tw.write("\n\tWhat do you want to do now?\n\t\t");
952         if (is_spec)
953             tw.write("Press <Enter> to Investigate the " + special + "\n\t\t");
954         tw.write("Press 'E' to Explore\n\t\t"
955             "Press 'G' to Gather Food\n\t\t"
956             "Press 'B' to Build/Fortify Camp\n\t\t"
957             "Press 'I' to go to your Inventory\n\t\t"
958             "Press 'M' to go to the Main Menu\n\t");
959         do
960         {
961             command = tw.get_input();
962         } while (command != 'e' && command != 'g' && command != 'b' &&
963             command != 'm' && command != 'i' && (!is_spec || command != enter));
964 
965         if (command == 'm')
966             menu_choice();
967         else if (command == 'i')
968             open_inventory();
969         else if (command == 'e')
970             ecom = explore_choice();
971         else if (command == 'b')
972             change = m.move_camp();
973     } while (command != 'g' && !change && ecom == 'b' && command != enter);
974     if (command == 'b')
975         build_camp_choice();
976     else if (command == 'g')
977         gather_choice();
978     else if (command == 'e')
979         m.move_self(ecom);
980     else if (command == enter)
981         m.investigate();
982     return command;
983 }
984 
985 // When gathering gives choice as what to do with found food
986 char Choice::food_choice(string name, int total)
987 {
988     tw.set_type_speed(fast_type_speed);
989     tw.write("\n\tWhat do you want to do?\n\t\t"
990         "Press 'E' to eat it\n\t\t"
991         "Press 'S' to store it for later\n\t\t"
992         "Press 'L' to leave it alone\n\t");
993     char command;
994     do {
995         command = tw.get_input();
996     } while (command != 'e' && command != 's' && command != 'l');
997 
998     int num = 1;
999 
1000    if (command == 'e')
1001    {
1002        if (total > 1)
```

```
1003            {
1004                tw.set_type_speed(norm_type_speed);
1005                tw.write("\n\tHow many would you like to eat right now?\n\t");
1006                do {
1007                    num = tw.get_input();
1008                    num -= 48;
1009                } while (num > 9 || num < 0);
1010                if (num > total)
1011                    num = total;
1012            }
1013            food.eat(name, num);
1014            if (total - num > 0)
1015            {
1016                tw.write("\n\tThen you stored the rest to your invertory.\n\t");
1017                inv.add('f', false, name, total - num);
1018            }
1019        }
1020        else if (command == 's')
1021            inv.add('f', false, name, total);
1022        else
1023            food.leave(name);
1024        return command;
1025    }
1026
1027    // Inventory Interface
1028    void Choice::open_inventory()
1029    {
1030        tw.set_type_speed(fast_type_speed);
1031        if (inv.food_empty() && inv.item_empty())
1032        {
1033            tw.write("\n\tYour inventory is empty!\n\t");
1034            person.status();
1035            return;
1036        }
1037        else
1038        {
1039            char command;
1040            tw.write("\n\tInventory Menu:\n\t\t");
1041            if (!inv.food_empty())
1042                tw.write("Press 'F' to open your Food Inventory\n\t\t");
1043            if (!inv.item_empty())
1044                tw.write("Press 'I' to open your Item Inventory\n\t\t");
1045            tw.write("Press 'R' to Return from the Inventory\n\t");
1046            do {
1047                command = tw.get_input();
1048            } while ((command != 'f' || inv.food_empty()) && (command != 'i' ||
                  inv.item_empty()) && command != 'r');
1049            if (command == 'f')
1050                inv.print_inv('f');
1051            else if (command == 'i')
1052                inv.print_inv('i');
1053        }
```

```
1054 }
1055
1056 //------------------------------------------------------------------------------
       ----------------------------------------------
1057
1058             // Camp Class Functions
1059
1060 // Decay defenses stat by build amount up to max_defenses
1061 void Camp::build_defenses(int build_amount)
1062 {
1063     for (int i = 0; i < build_amount && defenses < max_defenses; i++)
1064         defenses++;
1065 }
1066
1067 // Decay defenses stat by decay_amount to at most zero
1068 void Camp::decay_defenses(int decay_amount)
1069 {
1070     for (int i = 0; i < decay_amount && defenses > 0; i++)
1071         defenses--;
1072 }
1073
1074 // Simulates an attacking animal waking the player
1075 void Camp::woken(string it)
1076 {
1077     tw.set_type_speed(slow_type_speed);
1078     tw.write("\n\tYou are woken by a strange noise.\n\t"
1079         "You look around and see a ");
1080     if (it == bear.get_name())
1081     {
1082         tw.write("bear charging at you!\n\t");
1083         if (noose)
1084             tw.write("The bear rips through your thorned tree noose trap.\n\t");
1085         if (spring)
1086         {
1087             tw.write("The charging bear is impaled by a few of your pole spring
                traps.\n\t"
1088                 "It barely phases the bear, it is still charging at you!\n\t");
1089             bear.decay_health(3);
1090         }
1091         tw.write("\n\tYou quickly get to your feet and prepare to do battle!\n
              \t");
1092         if (hole)
1093             tw.write("\n\tHowever, when the bear was almost upon you,\n\t"
1094                 "he fell into the hole and onto the many sharp poles.\n\t");
1095         else
1096             bear.battle();
1097         if (person.get_health() > 0)
1098         {
1099             bear.reset_health();
1100             hole = false;
1101             spring = false;
1102             noose = false;
```

```
1103                defenses = 2;
1104                tw.write("\n\tYou managed to kill the bear!\n\t"
1105                    "You proceed to cut off 3 big slabs of raw meat and then bury
                          the rest of the bear.\n\t");
1106                inv.add('f', false, "raw bear meat", 3);
1107            }
1108        }
1109        else if (it == lion.get_name())
1110        {
1111            tw.write("mountain lion running at you!\n\t");
1112            if (noose)
1113            {
1114                tw.write("\n\tThe mountain lion runs right through your thorned tree
                       noose trap.\n\t"
1115                    "The trap catches it's hind end, but is not strong enough to
                          hoist up the mountain lion.\n\t"
1116                    "It breaks away from the thorny rope,\n\t"
1117                    "but you can see a small amount of blood trickling from the
                          animal.\n\t");
1118                lion.decay_health(2);
1119                defenses--;
1120                noose = false;
1121            }
1122            tw.write("You brace yourself for a heavy fight with the mountain lion!\n
                  \t");
1123            if (spring)
1124            {
1125                tw.write("\n\tHowever as the mountain lion nears you, it is caught
                      by your spring pole traps.\n\t"
1126                    "The sharp wooden poles pierces it's chest as it falls to the
                          ground and breaths it's last breath!\n\t");
1127                spring = false;
1128                defenses--;
1129            }
1130            else
1131                lion.battle();
1132            if (person.get_health() > 0)
1133            {
1134                lion.reset_health();
1135                tw.write("\n\tYou killed the mountain lion!\n\t"
1136                    "You cut out 2 large chunks of raw mountain lion meat and bury
                          the animal away from your camp.\n\t");
1137                inv.add('f', false, "raw mountain lion meat", 2);
1138            }
1139        }
1140        else if (it == gator.get_name())
1141        {
1142            tw.write("alligator darting toward you!\n\t");
1143            if (noose)
1144            {
1145                tw.write("\n\tThe alligator runs right through your thorned tree
                      noose trap.\n\t"
```

```
1146                    "The trap catches it's tail, but is not strong enough to hoist
                          up the alligator.\n\t"
1147                    "It breaks away from the thorny rope,\n\t"
1148                    "but you can see a small amount of blood trickling from the
                          animal's tail.\n\t");
1149                lion.decay_health(2);
1150                defenses--;
1151                noose = false;
1152            }
1153            tw.write("You brace yourself for a heavy fight with the alligator!\n
                  \t");
1154            if (spring)
1155            {
1156                tw.write("\n\tHowever as the alligator nears you, it is caught by
                      your spring pole traps.\n\t"
1157                    "The sharp wooden poles pierces it's body and it falls to the
                          ground and breaths it's last breath!\n\t");
1158                spring = false;
1159                defenses--;
1160            }
1161            else
1162                gator.battle();
1163            if (person.get_health() > 0)
1164            {
1165                gator.reset_health();
1166                hole = false;
1167                spring = false;
1168                noose = false;
1169                defenses = 2;
1170                tw.write("\n\tYou managed to kill the alligator!\n\t"
1171                    "You proceed to cut off 3 big slabs of raw meat and then bury
                          the rest of the alligator.\n\t");
1172                inv.add('f', false, "raw alligator meat", 3);
1173            }
1174        }
1175        else if (it == wolf.get_name())
1176        {
1177            tw.write("wolf running at you!\n\t");
1178            if (noose)
1179            {
1180                tw.write("\n\tThe wolf runs through your thorned tree noose trap.\n
                      \t"
1181                    "The trap catches it's hind leg, but is not strong enough to
                          hoist up the wolf.\n\t"
1182                    "It breaks away from the thorny rope,\n\t"
1183                    "but you can see a small amount of blood coming from it's leg.\n
                          \t");
1184                wolf.decay_health(2);
1185                defenses--;
1186                noose = false;
1187            }
1188            tw.write("You brace yourself for a heavy fight with the wolf!\n\t");
```

```
1189            if (spring)
1190            {
1191                tw.write("\n\tHowever as the wolf nears you, it is caught by your
                      spring pole traps.\n\t"
1192                    "The sharp wooden poles pierces it's chest as it falls to the
                          ground and breaths it's last breath!\n\t");
1193                spring = false;
1194                defenses--;
1195            }
1196            else
1197                wolf.battle();
1198            if (person.get_health() > 0)
1199            {
1200                wolf.reset_health();
1201                tw.write("\n\tYou killed the wolf!\n\t"
1202                    "You cut out 2 large chunks of raw wolf meat and bury the wolf
                          away from your camp.\n\t");
1203                inv.add('f', false, "raw wolf meat", 2);
1204            }
1205        }
1206        else if (it == snake.get_name())
1207        {
1208            tw.write("snake slithering quickly toward you!\n\t");
1209            tw.write("You prepare to face the snake!\n\t");
1210            if (noose)
1211            {
1212                tw.write("\n\tHowever the snake triggers your thorned tree noose
                      trap.\n\t"
1213                    "It is slung up in the air by it's neck and writhes as the
                          thorns pierce it's skin!\n\t");
1214                defenses--;
1215                noose = false;
1216            }
1217            else
1218                snake.battle();
1219            if (person.get_health() > 0)
1220            {
1221                snake.reset_health();
1222                tw.write("\n\tYou killed the snake!\n\t"
1223                    "You cut off a large chunk of raw snake meat and bury the snake
                          far from camp.\n\t");
1224                inv.add('f', false, "raw snake meat", 1);
1225            }
1226        }
1227        else
1228            tw.write(">" + it + "<");
1229    }
1230
1231    //------------------------------------------------------------------------------
          ----------------------------------------------
1232
1233                // Inventory Class Functions
```

```
1234
1235 // Interface for managing inventory
1236 void Inventory::print_inv(char type)
1237 {
1238     string id = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
1239     char command;
1240     map <string, int> *temp;
1241     if (type == 'f')
1242         temp = &food_inventory;
1243     else
1244         temp = &item_inventory;
1245
1246     do {
1247         int i = 0;
1248         tw.set_type_speed(fast_type_speed);
1249         tw.write("\n\t\t\tID\tAmount\tName\n");
1250
1251         for (mit = temp->begin(); mit != temp->end(); mit++)
1252         {
1253             tw.write("\n\t\t\t" + id.substr(i, 1) + "\t" + tw.num_to_string(mit-
                  >second) + "\t" + mit->first);
1254             i++;
1255         }
1256         tw.write("\n\n\tEnter the ID of the item you want to select\n\tOr press
              ENTER to go back to the Main Menu\n\t");
1257         do {
1258             command = tw.get_input();
1259         } while (tw.search(id, command, temp->size()) == INT_MAX && command !=
              enter);
1260         if (command == enter)
1261         {
1262             person.status();
1263             return;
1264         }
1265         int select = tw.search(id, command, temp->size());
1266         mit = temp->begin();
1267         for (int i = 0; i < select; i++)
1268             mit++;
1269         string name = mit->first;
1270         int total = mit->second;
1271
1272         if (type == 'f')
1273         {
1274             tw.write("\n\tWhat do you want to do with the " + name + "?\n\t\t"
1275                 "Press 'U' to Use\n\t\t"
1276                 "Press 'D' to Dicard\n\t\t");
1277             if (name.substr(0, 3) == "raw")
1278                 tw.write("Press 'C' to Cook\n\t\t");
1279             tw.write("Press 'N' to do Nothing\n\t");
1280             do {
1281                 command = tw.get_input();
1282                 if (command == 'c' && name.substr(0, 3) == "raw")
```

```
1283                        break;
1284                } while (command != enter && command != 'u' && command != 'd' &&
                      command != 'n');
1285                if (command != 'n')
1286                {
1287                    int num;
1288                    if (total > 1 && command != 'c')
1289                    {
1290                        tw.write("\n\tHow many (up to nine at a time)?\n\t");
1291                        do {
1292                            num = tw.get_input();
1293                            num -= 48;
1294                        } while (num > 9 || num < 0);
1295                        if (num > total)
1296                            num = total;
1297                    }
1298                    else
1299                        num = total;
1300                    string hand;
1301                    if (num > 1)
1302                        hand = " handfuls of ";
1303                    else
1304                        hand = " handful of ";
1305                    if (command == 'u')
1306                    {
1307                        bool warn = true;
1308                        if (name.substr(0, 3) == "raw")
1309                        {
1310                            tw.write("\n\tAre you sure you want to consume raw meat?
                              ('Y' or 'N')\n");
1311                            char c;
1312                            do {
1313                                c = tw.get_input();
1314                            } while (c != 'y' && c != 'n');
1315                            warn = c == 'y';
1316                        }
1317                        if (warn)
1318                        {
1319                            use('f', name, num);
1320                            person.status();
1321                        }
1322                    }
1323                    else if (command == 'd')
1324                    {
1325                        remove('f', name, num);
1326                        tw.write("\n\tYou discarded " + tw.num_to_string(num) + hand
                              + name + ".\n\t");
1327                    }
1328                    else if (command == 'c' && name.substr(0, 3) == "raw")
1329                    {
1330                        if (!m.not_camp() && camp.get_defenses() > 1)
1331                        {
```

```
1332                        remove('f', name, num);
1333                        add('f', true, "cooked" + name.substr(3, name.size() -
                        3), num);
1334                        tw.write("\n\tYou cooked " + tw.num_to_string(num) +
                        hand + name + ".\n\t");
1335                        }
1336                        else
1337                        tw.write("\n\tYou must be at camp and have a fire to
                        cook!\n\t");
1338                    }
1339                }
1340            }
1341            else
1342            {
1343                if (m.use(name))
1344                    remove('k', name, 1);
1345                else
1346                    tw.write("\n\tYou take out the " + name + " and try to think of
                      a way to use it here...\n\t"
1347                      "However, after a while of unsuccessful brainstorming, you
                      put it back in your bag.\n\t");
1348            }
1349        } while (command != enter && person.get_health() > 0 && temp->size() != 0);
1350        db.back_up();
1351    }
1352
1353    // "Uses" the item by calling a function made to handle it's typeitems
1354    void Inventory::use(char type, string name, int num)
1355    {
1356        if (type == 'f')
1357        {
1358            if (num > food_inventory[name])
1359                num = food_inventory[name];
1360            food.eat(name, num);
1361            remove(type, name, num);
1362        }
1363    }
1364
1365    // Used to add things to the inventory, "secret" is true if we don't want to
      print the add message
1366    void Inventory::add(char type, bool secret, string name, int num)
1367    {
1368        tw.set_type_speed(norm_type_speed);
1369        if (type == 'f')
1370        {
1371            string hand;
1372            if (num > 1)
1373                hand = " handfuls of ";
1374            else
1375                hand = " handful of ";
1376            if (!secret)
1377                tw.write("\n\tYou added the " + tw.num_to_string(num) + hand + name
```

```
                + " to your inventory.\n\t");
1378        for (int i = 0; i < num; i++)
1379            food_inventory[name]++;
1380    }
1381    else if (type == 'k')
1382    {
1383        if (!secret)
1384        {
1385            tw.write("\n\tYou found an stone emblem carved in the shape of a " + name + "!\n\t"
1386                "\n\tYou added the " + name + " Emblem to your inventory.\n\t");
1387            name += " Emblem";
1388        }
1389        item_inventory[name]++;
1390    }
1391    else if (type == 'i')
1392    {
1393        if (!secret)
1394            tw.write("\n\tYou found a " + name + "!\n\t"
1395                "\n\tYou added the " + name + " to your inventory.\n\t");
1396        item_inventory[name]++;
1397    }
1398 }
1399
1400 // Removes elements from an inventory. Type is the inv type so 'f' for food inv
1401 void Inventory::remove(char type, string name, int num)
1402 {
1403     map <string, int> *temp;
1404     if (type == 'f')
1405         temp = &food_inventory;
1406     else
1407         temp = &item_inventory;
1408     for (int i = 0; i < num && (*temp)[name] > 0; i++)
1409         (*temp)[name]--;
1410     if ((*temp)[name] == 0)
1411         (*temp).erase(name);
1412 }
1413
1414 //------------------------------------------------------------------------------------------------
1415
1416             // Time_Write Class Functions
1417
1418 Time_Write::Time_Write()
1419 {
1420     start_time = clock();
1421 }
1422
1423 double Time_Write::elapsed_time()
1424 {
1425     clock_t end_time = clock();
1426     return ((double)(end_time - start_time)) / ((double)CLOCKS_PER_SEC);
```

```
1427 }
1428
1429 void Time_Write::reset()
1430 {
1431     start_time = clock();
1432 }
1433
1434 char Time_Write::get_input()
1435 {
1436     reset();
1437     char command;
1438     while (elapsed_time() < .1)
1439         command = _getch();
1440     return command;
1441 }
1442
1443 void Time_Write::write(string s)
1444 {
1445     for (int i = 0; i < s.size(); i++)
1446     {
1447         cout << s[i] << flush;
1448         Time_Write t;
1449         t.reset();
1450         while (t.elapsed_time() < type_speed);
1451     }
1452 }
1453
1454 // This function is a modified function from "cplusplus.com".
1455 string Time_Write::num_to_string(int num)
1456 {
1457     ostringstream ss;
1458     ss << num;
1459     return ss.str();
1460 }
1461
1462 int Time_Write::search(string &key, char c, int size)
1463 {
1464     for (int i = 0; i < size; i++)
1465         if (tolower(key[i]) == c)
1466             return i;
1467     return INT_MAX;
1468 }
1469
1470 // Function which returns true if the player hit the correct button in time.
1471 bool Time_Write::time_key(bool boss_battle)
1472 {
1473     string battle_key;
1474     if (boss_battle && battle_key != boss_battle_key)
1475         battle_key = boss_battle_key;
1476     else if (!boss_battle && battle_key != norm_battle_key)
1477         battle_key = norm_battle_key;
1478
```

```
1479      int n = random(battle_key.size()) - 1;
1480      set_type_speed(instant);
1481
1482      write("\n\tQuickly Press '" + battle_key.substr(n, 1) + "'!   ");
1483      reset();
1484      char key = tw.get_input();
1485      double pTime = elapsed_time();
1486      if (isalpha(key))
1487          key = toupper(key);
1488      if (pTime < press_time && battle_key[n] == key)
1489          return true;
1490      else
1491          return false;
1492  }
1493
1494  bool Time_Write::key_race(string an)
1495  {
1496      double t;
1497      string escaped = " -> -> -> -> -> -> -> -> -> -> Escape!";
1498      string attempt = " -> -> -> You!";
1499      string spaces = "   ";
1500
1501      set_type_speed(slow_type_speed);
1502      write("\n\tTap the <spacebar> as fast as you can to try to escape!\n\t");
1503
1504      set_type_speed(instant);
1505      write("\n\t" + an + escaped +
1506          "\n\t" + an + attempt + spaces);
1507      get_input(); // This is to ensure they don't start hitting the button before ↵
            the timer starts
1508      do {
1509          reset();
1510          _getch();
1511          _getch();
1512          _getch();
1513          t = elapsed_time();
1514          for (int i = 0; i < (attempt.size() + spaces.size()); i++)
1515              cout << "\b";
1516          if (t < run_speed)
1517              attempt = " ->" + attempt;
1518          else
1519              attempt = attempt.substr(3, attempt.size() - 3);
1520
1521          write(attempt + spaces);
1522          if (attempt.size() < 6)
1523              return false;
1524
1525      } while (attempt.size() != (escaped.size() - 3));
1526      return true;
1527  }
1528
1529  string Time_Write::replace_spaces(string s)
```

```
1530 {
1531     for (int i = 0; i < s.size(); i++)
1532         if (s[i] == ' ')
1533             s[i] = '_';
1534     return s;
1535 }
1536
1537 string Time_Write::restore_spaces(string s)
1538 {
1539     for (int i = 0; i < s.size(); i++)
1540         if (s[i] == '_')
1541             s[i] = ' ';
1542     return s;
1543 }
1544
1545 //------------------------------------------------------------------------------
      ----------------------------------------------
1546
1547             // Food Class Functions
1548
1549 void Food::poss_poison(int food, string name)
1550 {
1551     int x = random(9);
1552     tw.set_type_speed(norm_type_speed);
1553     if (x == 5)
1554     {
1555         if (!person.get_poison()) // Makes sure they have not already been
            poisoned
1556         {
1557             tw.write("\n\tYou ate the handful of " + name + " and soon after you
                become very weak and pale.\n\t"
1558                 "You vomit up everything and hope the poisonous effects will
                  fade with time...\n\t");
1559             person.decay_health(5);
1560             person.change_poisoned(true);
1561         }
1562         else
1563         {
1564             tw.set_type_speed(fast_type_speed);
1565             tw.write("\n\tYou ate the handful of " + name + " and it contained
                poisonous elements!\n\t"
1566                 "You were already poisoned and the combined effects were fatal!
                  \n\n\t");
1567             person.decay_health(max_health);
1568         }
1569     }
1570     else
1571     {
1572         tw.write("\n\tYou ate the handful of " + name + " and were fine!\n\t");
1573         person.build_hunger(food);
1574     }
1575 }
```

```
1576
1577  void Food::leave(string food)
1578  {
1579      tw.set_type_speed(norm_type_speed);
1580      tw.write("\n\tYou decide to leave the " + food + " where it is and head back
             to camp.\n\t");
1581  }
1582
1583  void Food::found_berries()
1584  {
1585      tw.write("\n\tAfter hours of searching, you find a blackberry bush with 3
           handful of berries...!\n\t");
1586      pick.food_choice("blackberries", 3);
1587  }
1588
1589  void Food::found_mushrooms()
1590  {
1591      tw.write("\n\tJust when you were about to give up and head back to camp,\n
           \t"
1592          "you find a log with 3 handfuls of mushrooms on it.\n\t"
1593          "However, you don't know much about mushrooms, you could pick a
             poisonous one...\n\t");
1594      pick.food_choice("mushrooms", 3);
1595  }
1596
1597  void Food::found_eggs()
1598  {
1599      int num = random(3)+2;
1600      tw.write("\n\tAs you are scouting for food, you find a bird nest with " +
1601          tw.num_to_string(num) + " eggs in it.\n\t");
1602      pick.food_choice("bird eggs", num);
1603  }
1604
1605  void Food::found_insects()
1606  {
1607      tw.write("\n\tAfter searching for hours without a hint of anything edible
           you start to get desperate...\n\t"
1608          "You turn over rocks and logs, grabbing the largest insects before they
             squirm away.\n\t"
1609          "After about an hour of this, you end up with a handful of slimy,
             crushed up bugs.\n\t");
1610      pick.food_choice("bug guts", 1);
1611  }
1612
1613  void Food::eat(string name, int num)
1614  {
1615      int value = 1;
1616
1617      if (name == "mushrooms" || name.substr(0,3) == "raw")
1618          for (int i = 0; i < num; i++)
1619              poss_poison(value, name);
1620      else
```

```
1621      {
1622          string hand;
1623          if (num > 1)
1624              hand = " handfuls of ";
1625          else
1626              hand = " handful of ";
1627          tw.write("\n\tYou ate " + tw.num_to_string(num) + hand + name + "\n\t");
1628          for (int i = 0; i < num; i++)
1629              person.build_hunger(value);
1630      }
1631  }
1632
1633  void Food::random_food()
1634  {
1635      int x = random(5);
1636      tw.set_type_speed(norm_type_speed);
1637
1638      if (x == 1)
1639          found_mushrooms();
1640      else if (x == 2)
1641          found_berries();
1642      else if (x == 3)
1643          found_eggs();
1644      else if (x == 4)
1645          found_insects();
1646      else
1647      {
1648          tw.write("\n\tAfter a few hours of searching, you were unable to find
            anything.\n\t"
1649              "You return empty handed...\n\t");
1650      }
1651  }
1652
1653  //------------------------------------------------------------------------------
      ------------------------------------------------
1654
1655              // Map Class Definitions
1656
1657  void Map::print_map()
1658  {
1659      char c;
1660      string s;
1661      tw.set_type_speed(instant);
1662      tw.write("\nYour Map:\n\t");
1663
1664      for (int i = 0; i < rows; i++)
1665      {
1666          for (int j = 0; j <= columns; j++)
1667          {
1668              if (i % 2 == 0)
1669                  c = '-';
1670
```

```
1671                else
1672                    if (j % 6 == 0)
1673                        c = '|';
1674                    else if (j % 6 == 2)
1675                        c = check_self(j, i, true);
1676                    else if (j % 6 == 3)
1677                        c = check_others(j, i, true, true);
1678                    else if (j % 6 == 4)
1679                        c = check_camp(j, i, true);
1680                    else
1681                        c = ' ';
1682
1683                tw.write(string(1, c));
1684            }
1685            if (i % 2 != 0)
1686            {
1687                c = '|';
1688                s = key_str(i);
1689            }
1690            else
1691                s = "";
1692
1693            tw.write(string(1, c) + s + "\n\t");
1694        }
1695    }
1696
1697    // Checks if we are at a special location and the location hasn't already been explored
1698    string Map::spec_loc()
1699    {
1700        char c = check_others(self_x, self_y, false, false);
1701        if (c == snake_c && !boss_snake.dead())
1702            return "Tree";
1703        else if (c == wolf_c && !boss_wolf.dead())
1704            return "Hole";
1705        else if (c == bear_c && !boss_bear.dead())
1706            return "Cave";
1707        else if (c == cabin_c && !boss_lion.dead())
1708            return "Destroyed Log Cabin";
1709        else if (c == water_c && !boss_gator.dead())
1710            return "Waterfall";
1711        else if (c == temple_c)
1712            return "Temple";
1713        else
1714            return "";
1715    }
1716
1717    // Returns map key strings based on what the player has found
1718    string Map::key_str(int i)
1719    {
1720        static int last; // Used to keep track of what we have printed last
1721
```

```
1722      if (i == 1)
1723      {
1724          last = 1;
1725          return "\tMap Key:";
1726      }
1727      if (last == 1) // last goes up by odd values because of map layout
1728      {
1729          last = 3;
1730          return "\t\t'X' | Your Location";
1731      }
1732      if (last == 3)
1733      {
1734          last = 5;
1735          if (camp.get_defenses() > 0)
1736              return "\t\t'C' | Camp Location";
1737      }
1738      if (last == 5)
1739      {
1740          last = 7;
1741          if (found_snake)
1742              return "\t\t'" + string(1, snake_c) + "' | Biggest Tree in Forest";
1743      }
1744      if (last == 7)
1745      {
1746          last = 9;
1747          if (found_wolf)
1748              return "\t\t'" + string(1, wolf_c) + "' | Hole at Bottom of a
                Cliff";
1749      }
1750      if (last == 9)
1751      {
1752          last = 11;
1753          if (found_bear)
1754              return "\t\t'" + string(1, bear_c) + "' | Bloody Cave";
1755      }
1756      if (last == 11)
1757      {
1758          last = 13;
1759          if (found_cabin)
1760              return "\t\t'" + string(1, cabin_c) + "' | Destroyed Log Cabin";
1761      }
1762      if (last == 13)
1763      {
1764          last = 15;
1765          if (found_waterfall)
1766              return "\t\t'" + string(1, water_c) + "' | Waterfall";
1767      }
1768      if (last == 15)
1769      {
1770          last = 17;
1771          if (found_temple)
1772              return "\t\t'" + string(1, temple_c) + "' | Aztec Temple";
```

```
1773      }
1774      return "";
1775  }
1776
1777  // loop used to set rand non repeating coords for map variables
1778  void Map::rand_num_loop(int &x, int &y)
1779  {
1780      int nx, ny;
1781      do {
1782          nx = random(9); ny = random(9);
1783      } while ((check_self(nx, ny, false) != ' ' && check_self(nx, ny, false) !=
          '?') ||
1784          (check_others(nx, ny, false, false) != ' ' && check_others(nx, ny,
            false, false) != '?'));
1785      x = nx; y = ny;
1786  }
1787
1788  // Sets all map variables to rand non repeating coords at start of game
1789  void Map::rand_set_coords()
1790  {
1791      self_x = random(9); self_y = random(9);
1792      camp_x = 0; camp_y = 0;
1793
1794      rand_num_loop(snake_x, snake_y);
1795      rand_num_loop(wolf_x, wolf_y);
1796      rand_num_loop(bear_x, bear_y);
1797      rand_num_loop(cabin_x, cabin_y);
1798      rand_num_loop(waterfall_x, waterfall_y);
1799      rand_num_loop(temple_x, temple_y);
1800  }
1801
1802  void Map::reset()
1803  {
1804      m.rand_set_coords();
1805      found_snake = false;
1806      found_wolf = false;
1807      found_bear = false;
1808      found_cabin = false;
1809      found_waterfall = false;
1810      found_temple = false;
1811      for (int i = 0; i < 81; i++)
1812      {
1813          unexplored_map[i] = 0;
1814      }
1815  }
1816
1817  // Checks if x and y is the same as the players coords. Decode is used for map
      printing
1818  char Map::check_self(int x, int y, bool decode)
1819  {
1820      if (decode)
1821      {
```

```
1822            x = ((x + 1) / 6) + 1;
1823            y = ((y - 1) / 2) + 1;
1824        }
1825        if (x == self_x && y == self_y)
1826        {
1827            if (!unexplored_map[9 * (y - 1) + (x - 1)])
1828                unexplored_map[9 * (y - 1) + (x - 1)] = 1;
1829            return 'X';
1830        }
1831        else if (!unexplored_map[9 * (y - 1) + (x - 1)]) // Checks if the location
              is unexplored
1832            return '?';
1833        else
1834            return ' ';
1835    }
1836
1837    // Checks if x and y is the same as the camp coords. Decode is used for map
          printing
1838    char Map::check_camp(int x, int y, bool decode)
1839    {
1840        if (decode)
1841        {
1842            x = ((x + 1) / 6) + 1;
1843            y = ((y - 1) / 2) + 1;
1844        }
1845        if (x == camp_x && y == camp_y)
1846            return 'C';
1847        else if (!unexplored_map[9 * (y - 1) + (x - 1)]) // Checks if the location
              is unexplored
1848            return '?';
1849        else
1850            return ' ';
1851    }
1852
1853    // Checks if x and y is the same as the boss and item locations. Decode and
          filter is used for map printing
1854    char Map::check_others(int x, int y, bool decode, bool filter)
1855    {
1856        if (decode)
1857        {
1858            x = ((x + 1) / 6) + 1;
1859            y = ((y - 1) / 2) + 1;
1860        }
1861        if (filter && !unexplored_map[9 * (y - 1) + (x - 1)]) // Checks if the
              location is unexplored
1862            return '?';
1863        else if (x == snake_x && y == snake_y && (!filter || found_snake))
1864            return snake_c; // T for Tree
1865        else if (x == wolf_x && y == wolf_y && (!filter || found_wolf))
1866            return wolf_c; // Hole at the Bottom of Cliff
1867        else if (x == bear_x && y == bear_y && (!filter || found_bear))
1868            return bear_c; // B for Bloody Cavern
```

```
1869      else if (x == cabin_x && y == cabin_y && (!filter || found_cabin))
1870          return cabin_c; // L for Log cabin
1871      else if (x == waterfall_x && y == waterfall_y && (!filter ||
            found_waterfall))
1872          return water_c; // W for Waterfall
1873      else if (x == temple_x && y == temple_y && (!filter || found_temple))
1874          return temple_c; // A for Aztec Temple which
1875      else
1876          return ' ';
1877  }
1878
1879  // Simply prints an explore message
1880  void Map::print_move(string direction)
1881  {
1882      tw.set_type_speed(norm_type_speed);
1883      tw.write("\n\tYou cautiously move " + direction + ", watchful of unfriendly
            foes.\n\t");
1884  }
1885
1886  // Handles storyline for investigation and itiitiates boss attacks
1887  void Map::investigate()
1888  {
1889      char location = check_others(self_x, self_y, false, false);
1890      bool victory;
1891      tw.set_type_speed(slow_type_speed);
1892
1893      if (location == snake_c)
1894      {
1895          tw.write("\n\tYou approach the giant, bone riddle tree.\n\t");
1896          tw.write("As you do, a giant snake rears up out of the leaves!\n\t");
1897          victory = boss_snake.boss_prep();
1898          if (victory)
1899          {
1900              tw.write("\n\tAfter a fierce battle, you managed to slay the giant
                    snake!\n\t");
1901              inv.add('f', false, "raw snake meat", 2);
1902              inv.add('k', false, "Snake", 1);
1903          }
1904          else if (person.get_health() > 0)
1905              tw.write("\n\tYou managed to get away from the giant snake!\n\t");
1906      }
1907      else if (location == wolf_c)
1908      {
1909          tw.write("\n\tAs you approach the hole at the base of the cliff,\n\t"
1910              "a large pack of about 2 dozen wolves file out and surround you!\n
                    \t"
1911              "\n\tThey hold their ground as a giant scarred wolf walks through
                    their ranks.\n\t"
1912              "The alpha wolf crouches and prepares to attack...\n\t");
1913          victory = boss_wolf.boss_prep();
1914          if (victory)
1915          {
```

```
1916                tw.write("\n\tAfter a knockdown drag out fight, you emerge
                      victorious!\n\t"
1917                    "The other wolves scatter and run into the forest after seeing
                          their alpha defeated.\n\t");
1918                inv.add('f', false, "raw wolf meat", 3);
1919                inv.add('k', false, "Wolf", 1);
1920            }
1921            else if (person.get_health() > 0)
1922                tw.write("\n\tYou somehow managed to run away from an entire pack of
                       wolves!\n\t");
1923        }
1924        else if (location == bear_c)
1925        {
1926            tw.write("\n\tYou approach the bloody and scarred cave.\n\t"
1927                "As you are nearing the entrence, you hear a wild roar!\n\t"
1928                "Then a bear the size of a large car charges out of the cave!\n\t");
1929            victory = boss_bear.boss_prep();
1930            if (victory)
1931            {
1932                tw.write("\n\tAfter a fierce battle, you manage to bring down this
                      humongous beast!\n\t");
1933                inv.add('f', false, "raw bear meat", 4);
1934                inv.add('k', false, "Bear", 1);
1935            }
1936            else if (person.get_health() > 0)
1937                tw.write("\n\tAfter a few close calls, you finally got away from the
                       bear!\n\t");
1938        }
1939        else if (location == cabin_c)
1940        {
1941            tw.write("\n\tAs you search the destroyed log cabin, you hear something
                  rapidly approaching!\n\t"
1942                "You spin around and see a massive mountain lion sprinting toward
                      you!\n\t");
1943            victory = boss_lion.boss_prep();
1944            if (victory)
1945            {
1946                tw.write("\n\tAfter an intense fight you stand proud, the mountain
                      lion dead at your feet!\n\t");
1947                inv.add('f', false, "raw mountain lion meat", 3);
1948                inv.add('k', false, "Mountain Lion", 1);
1949            }
1950            else if (person.get_health() > 0)
1951                tw.write("After running non-stop and with a few well timed dodges,
                      you were able to get away!\n\t");
1952        }
1953        else if (location == water_c)
1954        {
1955            tw.write("\n\tAs you are explore around the waterfall, you hear
                  splashing.\n\t"
1956                "You spin around to see a giant, battle scarred alligator darting
                      toward you!\n\t");
```

```
1957            victory = boss_gator.boss_prep();
1958            if (victory)
1959            {
1960                tw.write("\n\tAfter a brutal battle of beasts, you emerge victorious!\n\t");
1961                inv.add('f', false, "raw alligator meat", 3);
1962                inv.add('k', false, "Alligator", 1);
1963            }
1964        }
1965        else if (location == temple_c)
1966        {
1967            char command;
1968            tw.set_type_speed(slow_type_speed);
1969            tw.write("\n\tYou approach the temple and admire the impressive structure...\n\t");
1970            do {
1971                tw.set_type_speed(norm_type_speed);
1972                tw.write("\n\tWhat do you want to do?\n\t\t"
1973                    "Press 'E' to Enter the Temple\n\t\t"
1974                    "Press 'I' to Open your Inventory\n\t\t"
1975                    "Press 'Q' to Quit exploring the Temple\n\t");
1976
1977                do {
1978                    command = tw.get_input();
1979                } while (command != 'e' && command != 'i' && command != 'q');
1980
1981                if (command == 'q')
1982                    return;
1983
1984                else if (command == 'i')
1985                    pick.open_inventory();
1986
1987                else if (command == 'e')
1988                {
1989                    if (ret_bear && ret_lion && ret_gator && ret_wolf && ret_snake)
1990                        person.escape();
1991                    else
1992                        tw.write("\n\tYou search the temple again trying to find an secret door.\n\t"
1993                            "However, the only thing you can find is the circle of animal embems with "
1994                            + tw.num_to_string(missing_emblems) + " missing...\n\t");
1995                }
1996            } while (!escaped && command != 'q');
1997        }
1998    }
1999
2000    // Checks to see if the player has "hit the wall" if not we move the player
2001    void Map::move_self(char command)
2002    {
2003        if (command == 'n')
```

```
2004            if (self_y > 1)
2005            {
2006                print_move("North");
2007                if (person.poss_encounter())
2008                {
2009                    self_y--;
2010                    unlock();
2011                    print_map();
2012                }
2013            }
2014            else
2015                hit_wall("North");
2016        else if (command == 'e')
2017            if (self_x < 9)
2018            {
2019                print_move("East");
2020                if (person.poss_encounter())
2021                {
2022                    self_x++;
2023                    unlock();
2024                    print_map();
2025                }
2026            }
2027            else
2028                hit_wall("East");
2029        else if (command == 's')
2030            if (self_y < 9)
2031            {
2032                print_move("South");
2033                if (person.poss_encounter())
2034                {
2035                    self_y++;
2036                    unlock();
2037                    print_map();
2038                }
2039            }
2040            else
2041                hit_wall("South");
2042        else
2043            if (self_x > 1)
2044            {
2045                print_move("West");
2046                if (person.poss_encounter())
2047                {
2048                    self_x--;
2049                    unlock();
2050                    print_map();
2051                }
2052            }
2053            else
2054                hit_wall("West");
2055    }
```

```
2056
2057 // Function which simulates the player putting the emblems in place
2058 bool Map::use(string emblem)
2059 {
2060     if (check_others(self_x, self_y, false, false) == temple_c)
2061     {
2062         tw.set_type_speed(slow_type_speed);
2063         tw.write("\n\tYou walk up the wall with the Stone Emblems on the Temple.
           \n\t"
2064             "You gently place the " + emblem + " back in it's spot.\n\t"
2065             "\n\tYou hear a faint clicking as it slides into place...\n\t");
2066         if (emblem == "Snake Emblem")
2067             ret_snake = true;
2068
2069         else if (emblem == "Wolf Emblem")
2070             ret_wolf = true;
2071
2072         else if (emblem == "Alligator Emblem")
2073             ret_gator = true;
2074
2075         else if (emblem == "Mountain Lion Emblem")
2076             ret_lion = true;
2077
2078         else if (emblem == "Bear Emblem")
2079             ret_bear = true;
2080
2081         missing_emblems--;
2082
2083         string num_missing;
2084         if (missing_emblems < 1)
2085             num_missing = "are no more emblems";
2086         else if (missing_emblems == 1)
2087             num_missing = "is only one more emblem";
2088         else
2089             num_missing = "are still " + tw.num_to_string(missing_emblems) + "
             more emblems";
2090         tw.write("\n\tThere " + num_missing + " missing...\n\t");
2091
2092         return true;
2093     }
2094     return false;
2095 }
2096
2097 // Prints out designated string for new location and sets their bool values to
     true
2098 void Map::unlock()
2099 {
2100     tw.set_type_speed(slow_type_speed);
2101     char location = check_others(self_x, self_y, false, false);
2102
2103     if (location == ' ' || location == '?' || person.get_health() == 0)
2104         return; // If we are not at a special location we return
```

```
2105      else if (location == snake_c && !found_snake)
2106      {
2107          found_snake = true;
2108          tw.write("\n\tAs you are exploring, you come across what must be the
                tallest tree in the forest!\n\t"
2109              "However, scattered all around this tree is small to medium sized
                  animal bones\n\t"
2110              "and what appears to be a snake skin about as long as a school
                  bus...\n\t");
2111      }
2112      else if (location == wolf_c && !found_wolf)
2113      {
2114          found_wolf = true;
2115          tw.write("\n\tWhile exploring you come across a large cliff.\n\t"
2116              "At the base of this cliff is a sizable hole with large marks
                  leading to it.\n\t"
2117              "The marks indicate many heavy things were forcibly dragged into the
                   hole...\n\t");
2118      }
2119      else if (location == bear_c && !found_bear)
2120      {
2121          found_bear = true;
2122          tw.write("\n\tAfter a while of exploring, you see a cave jutting out of
                a mountainside.\n\t"
2123              "From afar, you could see the blood stains and claw marks adorned
                  the cave walls...\n\t");
2124      }
2125      else if (location == cabin_c && !found_cabin)
2126      {
2127          found_cabin = true;
2128          tw.write("\n\tWhile exploring you find the remains of what appears to be
                 a log cabin!\n\t"
2129              "Most of it has been destroyed by fire, but you can see the claw
                  marks of some gigantic animal...\n\t");
2130      }
2131      else if (location == water_c && !found_waterfall)
2132      {
2133          found_waterfall = true;
2134          tw.write("\n\tWhile exploring you find a cascading waterfall with a
                small lake at the bottom.\n\t"
2135              "However, there is something very creepy about this dark, murky
                  water...\n\t");
2136      }
2137      else if (location == temple_c && !found_temple)
2138      {
2139          found_temple = true;
2140          tw.write("\n\tWhile exploring you come across what appears to be an
                Aztec Temple!\n\t"
2141              "You search the temple for an entrence or some secret passageway,
                  but no luck!\n\t"
2142              "You notice on one side of the temple there is a circle with animal
                  emblems carved into the stone.\n\t"
```

```
2143                "There appear to be 5 stone emblems missing...\n\t");
2144        }
2145    }
2146
2147
2148    // Simply prints a string describing why the player can't go this way
2149    void Map::hit_wall(string direction)
2150    {
2151        tw.set_type_speed(norm_type_speed);
2152        tw.write("\n\tYou head " + direction + ", but soon come across a great wall.
              \n\t"
2153            "This great wall appears to be at least five times the size of the
                  tallest tree!\n\t"
2154            "You stand there, sizing up the wall trying to think of a way around
                  it...\n\t"
2155            "Alas, you are forced to head back with a heavy heart.\n\t"
2156            "As you walk back, you wonder what is that wall for and who built it?\n
                  \t");
2157    }
2158
2159    // Checks if the player wants to move their camp, if so resets camp coords
2160    bool Map::move_camp()
2161    {
2162        if (camp.get_defenses() == 0 || not_camp())
2163        {
2164            if (camp.get_defenses() > 0)
2165            {
2166                char command;
2167                tw.write("\n\tAre you sure you want to build a new camp here,\n\t"
2168                    "it will erase your previous camp! ('Y' or 'N')\n\t");
2169                do {
2170                    command = tw.get_input();
2171                } while (command != 'y' && command != 'n');
2172                tw.write("\n\t");
2173                if (command == 'n')
2174                    return false;
2175            }
2176            camp_x = self_x;
2177            camp_y = self_y;
2178            camp.decay_defenses(max_defenses);
2179            camp.set_noose(false);
2180            camp.set_spring(false);
2181            camp.set_hole(false);
2182        }
2183        return true;
2184    }
2185
2186    //------------------------------------------------------------------------------
          ------------------------------------------------
2187
2188                // Turn Class Definitions
2189
```

```
2190  // resets turn back to 1 and increments days and players health
2191  void Turn::reset_turn()
2192  {
2193      turn = 1;
2194      person.poss_build_health(2);
2195      days++;
2196  }
2197
2198  // Sets records at begining of game
2199  void Turn::set_records()
2200  {
2201      // Makes sure leaderboards are empty
2202      days_survived.clear(); days_survived_names.clear();
2203      days_escaped.clear(); days_escaped_names.clear();
2204
2205      // Standard records are uploaded first.  // Must be updated by programmer
2206      // This is so when program is sent to others for play they will have times
          to beat
2207
2208      // Survived Records
2209      days_survived_names.push_back("Bross Boss"); days_survived.push_back(209);
2210      days_survived_names.push_back("Ian Blevins"); days_survived.push_back(196);
2211      days_survived_names.push_back("Zach Hollis"); days_survived.push_back(166);
2212      days_survived_names.push_back("Madison Allen"); days_survived.push_back(71);
2213      days_survived_names.push_back("Andrew Benson"); days_survived.push_back(55);
2214      days_survived_names.push_back("Cammy"); days_survived.push_back(31);
2215      days_survived_names.push_back("Brandon Livingston"); days_survived.push_back
          (23);
2216      days_survived_names.push_back("Meredith Kay_Freshour");
          days_survived.push_back(15);
2217      days_survived_names.push_back("Lee Presson"); days_survived.push_back(15);
2218      days_survived_names.push_back("Z"); days_survived.push_back(5);
2219
2220      // Escaped Records
2221      days_escaped_names.push_back("Zach Hollis"); days_escaped.push_back(166);
2222      days_escaped_names.push_back("O"); days_escaped.push_back(1199);
2223      days_escaped_names.push_back("J"); days_escaped.push_back(1211);
2224      days_escaped_names.push_back("P"); days_escaped.push_back(1310);
2225      days_escaped_names.push_back("A"); days_escaped.push_back(1409);
2226      days_escaped_names.push_back("K"); days_escaped.push_back(1508);
2227      days_escaped_names.push_back("L"); days_escaped.push_back(1607);
2228      days_escaped_names.push_back("U"); days_escaped.push_back(1706);
2229      days_escaped_names.push_back("Z"); days_escaped.push_back(1805);
2230      days_escaped_names.push_back("K"); days_escaped.push_back(1904);
2231
2232      db.load_in_leaderboards();
2233  }
2234
2235  void Turn::check_records(int turndays, string name, bool lead_escaped, bool
        secret)
2236  {
2237      const string perm_name = name;
```

```
2238      const int perm_turndays = turndays;
2239
2240      int temp_num;
2241      string temp_name;
2242      bool congrats = false;
2243
2244      if (lead_escaped)
2245      {
2246          for (int i = 0; i < days_escaped.size() && i < 10; i++)
2247          {
2248              if (turndays == days_escaped[i] && name == days_escaped_names[i] &&
                    secret)
2249                  return;
2250              if (turndays < days_escaped[i])
2251              {
2252                  if (!congrats)
2253                      congrats = true;
2254                  temp_num = days_escaped[i];
2255                  temp_name = days_escaped_names[i];
2256                  days_escaped[i] = turndays;
2257                  days_escaped_names[i] = name;
2258                  name = temp_name;
2259                  turndays = temp_num;
2260              }
2261          }
2262          if (congrats && !secret)
2263              tw.write("\n\tNice Job! You earned a spot in the \"Quickest Escapes
                    \" Leaderboard!\n\t");
2264          name = perm_name;
2265          turndays = perm_turndays;
2266      }
2267      if (!secret || !lead_escaped)
2268      {
2269          for (int i = 0; i < days_survived.size() && i < 10; i++)
2270          {
2271              if (turndays == days_survived[i] && name == days_survived_names[i]
                    && secret)
2272                  return;
2273              if (turndays > days_survived[i])
2274              {
2275                  if (!congrats)
2276                      congrats = true;
2277                  temp_num = days_survived[i];
2278                  temp_name = days_survived_names[i];
2279                  days_survived[i] = turndays;
2280                  days_survived_names[i] = name;
2281                  name = temp_name;
2282                  turndays = temp_num;
2283              }
2284          }
2285          if (congrats && !secret)
2286              tw.write("\n\tNice Job! You earned a spot in the \"Longest Survival
```

```
                \" Leaderboard!\n\t");
2287    }
2288    if (!secret)
2289        leaderboards = true;
2290 }
2291
2292 void Turn::print_leaderboard()
2293 {
2294    tw.set_type_speed(norm_type_speed);
2295    tw.write("\n\nLeaderboards:\n\t"
2296        "\n\tLongest Survivals:\n\t\t"); // Loop that prints the top survival
          names with days and hours
2297
2298    for (int i = 0; i < 10 && i < days_survived.size(); i++)
2299        tw.write(tw.num_to_string(i + 1) + ":\t" + days_survived_names[i] + "
          Survived " +
2300            tw.num_to_string(days_survived[i] / max_cycles) + " Days and " +
2301            tw.num_to_string((days_survived[i] % max_cycles ) * 3) + " Hours!\n
              \t\t");
2302
2303    tw.write("\n\tQuickest Escapes:\n\t\t"); // Same as above but on quickest
        escapes
2304    for (int i = 0; i < 10 && i < days_escaped.size(); i++)
2305        tw.write(tw.num_to_string(i + 1) + ":\t" + days_escaped_names[i] + "
          Escaped in " +
2306            tw.num_to_string(days_escaped[i] / max_cycles) + " Days and " +
2307            tw.num_to_string((days_escaped[i] % max_cycles) * 3) + " Hours!\n\t
              \t");
2308
2309    tw.write("\nPress anything to return to the Main Menu...\n");
2310    tw.get_input();
2311 }
2312
2313 // handles all things associated with time change: such as sleep, freezing,
      hunger, and poison
2314 void Turn::next_turn()
2315 {
2316    if (person.get_health() < 1 || escaped)
2317    {
2318        if (turn == max_cycles)
2319        {
2320            turn = 1;
2321            days++;
2322        }
2323        person.status();
2324        return;
2325    }
2326    else if (turn < max_cycles)
2327    {
2328        tw.set_type_speed(fast_type_speed);
2329        turn++;
2330        if (turn > 5 && !m.not_camp() && camp.get_defenses() > 1)
```

```
2331            {
2332                tw.write("\n\tIt is dark and you are starting to get sleepy,\n\t"
2333                    "you curl up next to the fire and fall fast asleep.\n\t"
2334                    "All the while, the forest is stiring around you as beasts roam
                          through the trees...\n\t");
2335                string it = person.chance();
2336                if (it != "nada")
2337                {
2338                    camp.woken(it);
2339                    if (person.get_health() > 0)
2340                        reset_turn();
2341                }
2342                else
2343                {
2344                    tw.write("\n\tYou awake a little before sunrise and get ready
                          for a hard day.\n\t");
2345                    reset_turn();
2346                }
2347            }
2348            else if (turn == 6)
2349            {
2350                tw.write("\n\tNight is quickly approaching and with it,\n\t"
2351                    "the beasts and freezing temperatures!\n\t");
2352            }
2353            else if (turn == 8)
2354            {
2355                tw.write("\n\tThe temperature has dropped severely!\n\t"
2356                    "Your extremities are starting to go numb!\n\t");
2357            }
2358        }
2359        else
2360            if (camp.get_defenses() < 2 || m.not_camp())
2361            {
2362                tw.set_type_speed(norm_type_speed);
2363                tw.write("\n\n\tYou stand there, as the cold nights air slowly over
                      powers you.\n\t"
2364                    "It takes you a few minutes to even realize you aren't standing
                          anymore.\n\t"
2365                    "You finally figure out you are lying facedown in the dirt.\n\t"
2366                    "You can hear things moving around you making strange sounds...
                          \n\n\t"
2367                    "You try to get up but your frozen body won't move.\n\t"
2368                    "The sounds are getting closer and closer,\n\t"
2369                    "you listen closely trying to figure out what it is...\n\t"
2370                    "The last thing you hear is a loud ripping noise and then...\n
                          \t");
2371                person.decay_health(max_health);
2372            }
2373            else
2374                reset_turn();
2375
2376        person.track_poison();
```

```
2377
2378        if (person.get_hunger() >= max_hunger-2)
2379            person.poss_build_health(1);
2380
2381        person.decay_hunger(1);
2382
2383        if (person.get_hunger() == 0 && person.get_health() > 0)
2384        {
2385            tw.set_type_speed(norm_type_speed);
2386            if (starve_warning)
2387            {
2388                tw.write("\n\tYou collapse from starvation!\n\n\t"
2389                    "You lay there, unable to move, as your body slowly digests
                          itself\n\t"
2390                    "in its desperation for any source of energy...\n\t");
2391                person.decay_health(max_health);
2392            }
2393            else
2394            {
2395                starve_warning = true;
2396                tw.write("\n\tYou are starving!\n\t"
2397                    "Your body isn't going to be able to holdout much longer!\n\t"
2398                    "You need to eat something now!\n\t");
2399            }
2400        }
2401        else
2402            starve_warning = false;
2403
2404        person.status();
2405    }
2406
2407    //-------------------------------------------------------------------------------
        ------------------------------------------------
2408
2409                // Animal Class Definitions
2410
2411    // Decays animals health to max of 0
2412    void Animal::decay_health(int build_amount)
2413    {
2414        for (int i = 0; i < build_amount && health > 0; i++)
2415            health--;
2416    }
2417
2418    // Simulates poisonous animal attacks on the player
2419    void Animal::poss_poison(string name)
2420    {
2421        if (random(5) == 3)
2422        {
2423            if (!person.get_poison()) // Makes sure they have not already been
              poisoned
2424            {
2425                tw.write("\n\tYou are bitten by the " + name + " and soon after you
```

```
                become very weak and pale.\n\t"
2426              "Hopefully the poisonous effects will fade with time...\n\t");
2427            person.decay_health(5);
2428            person.change_poisoned(true);
2429        }
2430        else // If they are already poisoned we kill the player
2431        {
2432            tw.set_type_speed(slow_type_speed);
2433            tw.write("\n\tYou were bitten by the " + name + " while you were
                already poisoned!\n\t"
2434              "The combined effects were fatal!\n\n\t");
2435            person.decay_health(max_health);
2436        }
2437    }
2438    else
2439        tw.write("\n\tYou were bitten by the " + name + " and were fine!\n\t");
2440 }
2441
2442 // Simulates a battle with an animal. Timed key strokes determine hit and miss
2443 void Animal::battle()
2444 {
2445    int pa = person.get_attack();
2446    int num;
2447    bool on_time;
2448    do
2449    {
2450        num = random(5);
2451        on_time = tw.time_key(boss);
2452        tw.set_type_speed(norm_type_speed);
2453        if (num == 3)
2454            if (on_time)
2455            {
2456                tw.write("\tYou landed an attack against the " + name + "!\n
                    \t");
2457                decay_health(person.get_attack());
2458            }
2459            else
2460                tw.write("\tYour attack missed!\n\t");
2461
2462        else
2463            if (on_time)
2464                tw.write("\tYou dodged the " + name + "'s attack!\n\t");
2465
2466            else
2467            {
2468                tw.write("\tYou tried to dodge the " + name + "'s attack, but
                    failed!\n");
2469                person.decay_health(attack);
2470                if (name == snake.name || name == boss_snake.name)
2471                    poss_poison(name);
2472            }
2473    } while (person.get_health() > 0 && health > 0);
```

```
2474      if (!boss)
2475          health = org_health;
2476  }
2477
2478  bool Animal::boss_prep()
2479  {
2480      bool ran_away = false;
2481      char command;
2482
2483      tw.set_type_speed(slow_type_speed);
2484      tw.write("\n\tWhat should you do?\n\t\t"
2485          "Press 'F' to Fight\n\t\t"
2486          "Press 'R' to Run\n\t");
2487
2488      do {
2489          command = tw.get_input();
2490      } while (command != 'r' && command != 'f');
2491
2492      if (command == 'r')
2493          if (tw.key_race(name))
2494          {
2495              tw.write("\n\tYou managed to run away from the " + name + "!\n\t");
2496              return false;
2497          }
2498
2499      battle();
2500
2501      if (person.get_health() > 0 && health < 1)
2502          return true;
2503      else
2504          return false;
2505  }
2506
2507  //------------------------------------------------------------------------------
      ------------------------------------------------
2508
2509              // Database Class Definitions
2510
2511  // Backs up current stats to a file so a game can be resumed
2512  void Database::back_up()
2513  {
2514      // Sets up the file
2515      ofstream ofile;
2516      ofile.open(filename, ofstream::trunc);
2517
2518      // Global Stats
2519      ofile << encode(pPlayer_Name) << " " << encode(tw.replace_spaces
          (player_name)) << endl;
2520      ofile << encode(pDays) << " " << encode(tw.num_to_string(days)) << endl;
2521      ofile << encode(pEscaped) << " " << encode(tw.num_to_string(escaped)) <<
          endl;
2522      ofile << encode(pMissing_Emblems) << " " << encode(tw.num_to_string
```

```
          (missing_emblems)) << endl;
2523      ofile << encode(pLeaderboards) << " " << encode(tw.num_to_string
          (leaderboards)) << endl;
2524      ofile << encode(pStarve_Warning) << " " << encode(tw.num_to_string
          (starve_warning)) << endl;
2525      ofile << encode(pRet_Bear) << " " << encode(tw.num_to_string(ret_bear)) <<
          endl;
2526      ofile << encode(pRet_Lion) << " " << encode(tw.num_to_string(ret_lion)) <<
          endl;
2527      ofile << encode(pRet_Gator) << " " << encode(tw.num_to_string(ret_gator)) <<
           endl;
2528      ofile << encode(pRet_Wolf) << " " << encode(tw.num_to_string(ret_wolf)) <<
          endl;
2529      ofile << encode(pRet_Snake) << " " << encode(tw.num_to_string(ret_snake)) <<
           endl;
2530      ofile << encode(pEnd) << endl << endl;
2531
2532      // Person Class Stats
2533      ofile << encode(pHealth) << " " << encode(tw.num_to_string(person.health))
          << endl;
2534      ofile << encode(pHunger) << " " << encode(tw.num_to_string(person.hunger))
          << endl;
2535      ofile << encode(pPoisoned) << " " << encode(tw.num_to_string
          (person.poisoned)) << endl;
2536      ofile << encode(pPoison_Rec) << " " << encode(tw.num_to_string
          (person.poison_recovery)) << endl;
2537      ofile << encode(pEnd) << endl << endl;
2538
2539      // Camp Class Stats
2540      ofile << encode(pDefenses) << " " << encode(tw.num_to_string(camp.defenses))
           << endl;
2541      ofile << encode(pNoose) << " " << encode(tw.num_to_string(camp.noose)) <<
          endl;
2542      ofile << encode(pSpring) << " " << encode(tw.num_to_string(camp.spring)) <<
          endl;
2543      ofile << encode(pHole) << " " << encode(tw.num_to_string(camp.hole)) <<
          endl;
2544      ofile << encode(pEnd) << endl << endl;
2545
2546      // Food Inventory Stats
2547      ofile << encode(pFood_Inv) << " " << encode(tw.num_to_string
          (inv.food_inventory.size())) << endl;
2548      for (mit = inv.food_inventory.begin(); mit != inv.food_inventory.end(); mit+
          +)
2549        ofile << encode(tw.replace_spaces(mit->first)) << " " << encode
            (tw.num_to_string(mit->second)) << endl;
2550
2551      // Item Inventory Stats
2552      ofile << encode(pItem_Inv) << " " << encode(tw.num_to_string
          (inv.item_inventory.size())) << endl;
2553      for (mit = inv.item_inventory.begin(); mit != inv.item_inventory.end(); mit+
          +)
```

```
2554            ofile << encode(tw.replace_spaces(mit->first)) << " " << encode
                  (tw.num_to_string(mit->second)) << endl;
2555
2556        // Map Class Stats
2557        ofile << encode(pSelf_X) << " " << encode(tw.num_to_string(m.self_x)) <<
              endl;
2558        ofile << encode(pSelf_Y) << " " << encode(tw.num_to_string(m.self_y)) <<
              endl;
2559        ofile << encode(pCamp_X) << " " << encode(tw.num_to_string(m.camp_x)) <<
              endl;
2560        ofile << encode(pCamp_Y) << " " << encode(tw.num_to_string(m.camp_y)) <<
              endl;
2561        ofile << encode(pSnake_X) << " " << encode(tw.num_to_string(m.snake_x)) <<
              endl;
2562        ofile << encode(pSnake_Y) << " " << encode(tw.num_to_string(m.snake_y)) <<
              endl;
2563        ofile << encode(pSnake_Health) << " " << encode(tw.num_to_string
              (boss_snake.health)) << endl;
2564        ofile << encode(pFound_Snake) << " " << encode(tw.num_to_string
              (m.found_snake)) << endl;
2565        ofile << encode(pWolf_X) << " " << encode(tw.num_to_string(m.wolf_x)) <<
              endl;
2566        ofile << encode(pWolf_Y) << " " << encode(tw.num_to_string(m.wolf_y)) <<
              endl;
2567        ofile << encode(pWolf_Health) << " " << encode(tw.num_to_string
              (boss_wolf.health)) << endl;
2568        ofile << encode(pFound_Wolf) << " " << encode(tw.num_to_string
              (m.found_wolf)) << endl;
2569        ofile << encode(pBear_X) << " " << encode(tw.num_to_string(m.bear_x)) <<
              endl;
2570        ofile << encode(pBear_Y) << " " << encode(tw.num_to_string(m.bear_y)) <<
              endl;
2571        ofile << encode(pBear_Health) << " " << encode(tw.num_to_string
              (boss_bear.health)) << endl;
2572        ofile << encode(pFound_Bear) << " " << encode(tw.num_to_string
              (m.found_bear)) << endl;
2573        ofile << encode(pCabin_X) << " " << encode(tw.num_to_string(m.cabin_x)) <<
              endl;
2574        ofile << encode(pCabin_Y) << " " << encode(tw.num_to_string(m.cabin_y)) <<
              endl;
2575        ofile << encode(pLion_Health) << " " << encode(tw.num_to_string
              (boss_lion.health)) << endl;
2576        ofile << encode(pFound_Cabin) << " " << encode(tw.num_to_string
              (m.found_cabin)) << endl;
2577        ofile << encode(pWaterfall_X) << " " << encode(tw.num_to_string
              (m.waterfall_x)) << endl;
2578        ofile << encode(pWaterfall_Y) << " " << encode(tw.num_to_string
              (m.waterfall_y)) << endl;
2579        ofile << encode(pGator_Health) << " " << encode(tw.num_to_string
              (boss_gator.health)) << endl;
2580        ofile << encode(pFound_Waterfall) << " " << encode(tw.num_to_string
              (m.found_waterfall)) << endl;
```

```
2581      ofile << encode(pTemple_X) << " " << encode(tw.num_to_string(m.temple_x)) << ↵
             endl;
2582      ofile << encode(pTemple_Y) << " " << encode(tw.num_to_string(m.temple_y)) << ↵
             endl;
2583      ofile << encode(pFound_Temple) << " " << encode(tw.num_to_string            ↵
            (m.found_temple)) << endl;
2584      ofile << encode(pUnexplored_Map) << endl;
2585      for (int i = 0; i < 81; i++)
2586          ofile << encode(tw.num_to_string(m.unexplored_map[i])) << " ";
2587      ofile << endl;
2588      ofile << encode(pEnd) << endl << endl;
2589
2590      // Turn Class Stats
2591      ofile << encode(pTurn) << " " << encode(tw.num_to_string(turn.turn)) <<      ↵
            endl;
2592      ofile << encode(pEnd) << endl;
2593
2594      // Opens leaderboard file
2595      ofstream lfile;
2596      lfile.open(lfilename, ofstream::trunc);
2597
2598      // Leaderboards
2599      for (int i = 0; i < 10 && i < turn.days_escaped.size(); i++)
2600          lfile << encode(tw.replace_spaces(turn.days_escaped_names[i])) << " " << ↵
                encode(tw.num_to_string(turn.days_escaped[i])) << endl;
2601      for (int i = 0; i < 10 && i < turn.days_survived.size(); i++)
2602          lfile << encode(tw.replace_spaces(turn.days_survived_names[i])) << " "   ↵
              << encode(tw.num_to_string(turn.days_survived[i])) << endl;
2603
2604      lfile.close();
2605      ofile.close();
2606  }
2607
2608  // Loads in previous saved data from PDB.txt. Returns false if it fails
2609  bool Database::load_in()
2610  {
2611      string input;
2612
2613      ifstream ifile(filename);
2614      if (!ifile)
2615      {
2616          tw.write("\n\tError! Could not load saved game!\n\t");
2617          return false;
2618      }
2619      // Loads in saved Global variable data
2620      do {
2621          ifile >> input; input = decode(input);
2622          if (input == pPlayer_Name)
2623          {
2624              ifile >> input; input = decode(input);
2625              player_name = tw.restore_spaces(input);
2626          }
```

```
2627
2628          else if (input == pDays)
2629              {ifile >> input; days = atoi(decode(input).c_str());}
2630
2631          else if (input == pEscaped)
2632              {ifile >> input; escaped = atoi(decode(input).c_str());}
2633
2634          else if (input == pMissing_Emblems)
2635              {ifile >> input; missing_emblems = atoi(decode(input).c_str());}
2636
2637          else if (input == pLeaderboards)
2638              {ifile >> input; leaderboards = atoi(decode(input).c_str());}
2639
2640          else if (input == pStarve_Warning)
2641              {ifile >> input; starve_warning = atoi(decode(input).c_str());}
2642
2643          else if (input == pRet_Bear)
2644              {ifile >> input; ret_bear = atoi(decode(input).c_str());}
2645
2646          else if (input == pRet_Lion)
2647              {ifile >> input; ret_lion = atoi(decode(input).c_str());}
2648
2649          else if (input == pRet_Gator)
2650              {ifile >> input; ret_gator = atoi(decode(input).c_str());}
2651
2652          else if (input == pRet_Wolf)
2653              {ifile >> input; ret_wolf = atoi(decode(input).c_str());}
2654
2655          else if (input == pRet_Snake)
2656              {ifile >> input; ret_snake = atoi(decode(input).c_str());}
2657
2658      } while (input != pEnd);
2659      // Loads in saved Person class data
2660      do {
2661          ifile >> input; input = decode(input);
2662
2663          if (input == pHealth)
2664              {ifile >> input; person.health = atoi(decode(input).c_str());}
2665
2666          else if (input == pHunger)
2667              {ifile >> input; person.hunger = atoi(decode(input).c_str());}
2668
2669          else if (input == pPoisoned)
2670              {ifile >> input; person.poisoned = atoi(decode(input).c_str());}
2671
2672          else if (input == pPoison_Rec)
2673              {ifile >> input; person.poison_recovery = atoi(decode(input).c_str
                  ());}
2674
2675      } while (input != pEnd);
2676
2677      // Loads in saved Camp class data
```

```
2678      do {
2679          ifile >> input; input = decode(input);
2680
2681          if (input == pDefenses)
2682              {ifile >> input; camp.defenses = atoi(decode(input).c_str());}
2683
2684          else if (input == pNoose)
2685              {ifile >> input; camp.noose = atoi(decode(input).c_str());}
2686
2687          else if (input == pSpring)
2688              {ifile >> input; camp.spring = atoi(decode(input).c_str());}
2689
2690          else if (input == pHole)
2691              {ifile >> input; camp.hole = atoi(decode(input).c_str());}
2692
2693      } while (input != pEnd);
2694
2695      // Loads in Food Inventory stats
2696      int total, num;
2697      string s;
2698      ifile >> input >> s;
2699      input = decode(input);
2700      total = atoi(decode(s).c_str());
2701      for (int i = 0; i < total; i++)
2702      {
2703          ifile >> input >> s;
2704          input = decode(input);
2705          num = atoi(decode(s).c_str());
2706          inv.add('f', true, tw.restore_spaces(input), num);
2707      }
2708
2709      // Loads in Item Inventory stats
2710      ifile >> input >> s;
2711      input = decode(input);
2712      total = atoi(decode(s).c_str());
2713      for (int i = 0; i < total; i++)
2714      {
2715          ifile >> input >> s;
2716          input = decode(input);
2717          num = atoi(decode(s).c_str());
2718          inv.add('k', true, tw.restore_spaces(input), num);
2719      }
2720      // Loads in Map class stats and boss health stats
2721      do {
2722          ifile >> input; input = decode(input);
2723          if (input == pSelf_X)
2724              {ifile >> input; m.self_x = atoi(decode(input).c_str());}
2725          else if (input == pSelf_Y)
2726              {ifile >> input; m.self_y = atoi(decode(input).c_str());}
2727
2728          else if (input == pCamp_X)
2729              {ifile >> input; m.camp_x = atoi(decode(input).c_str());}
```

```
2730            else if (input == pCamp_Y)
2731                {ifile >> input; m.camp_y = atoi(decode(input).c_str());}
2732
2733            else if (input == pSnake_X)
2734                {ifile >> input; m.snake_x = atoi(decode(input).c_str());}
2735            else if (input == pSnake_Y)
2736                {ifile >> input; m.snake_y = atoi(decode(input).c_str());}
2737            else if (input == pSnake_Health)
2738                {ifile >> input; boss_snake.health = atoi(decode(input).c_str());}
2739            else if (input == pFound_Snake)
2740                {ifile >> input; m.found_snake = atoi(decode(input).c_str());}
2741
2742            else if (input == pWolf_X)
2743                {ifile >> input; m.wolf_x = atoi(decode(input).c_str());}
2744            else if (input == pWolf_Y)
2745                {ifile >> input; m.wolf_y = atoi(decode(input).c_str());}
2746            else if (input == pWolf_Health)
2747                {ifile >> input; boss_wolf.health = atoi(decode(input).c_str());}
2748            else if (input == pFound_Wolf)
2749                {ifile >> input; m.found_wolf = atoi(decode(input).c_str());}
2750
2751            else if (input == pBear_X)
2752                {ifile >> input; m.bear_x = atoi(decode(input).c_str());}
2753            else if (input == pBear_Y)
2754                {ifile >> input; m.bear_y = atoi(decode(input).c_str());}
2755            else if (input == pBear_Health)
2756                {ifile >> input; boss_bear.health = atoi(decode(input).c_str());}
2757            else if (input == pFound_Bear)
2758                {ifile >> input; m.found_bear = atoi(decode(input).c_str());}
2759
2760            else if (input == pCabin_X)
2761                {ifile >> input; m.cabin_x = atoi(decode(input).c_str());}
2762            else if (input == pCabin_Y)
2763                {ifile >> input; m.cabin_y = atoi(decode(input).c_str());}
2764            else if (input == pLion_Health)
2765                {ifile >> input; boss_lion.health = atoi(decode(input).c_str());}
2766            else if (input == pFound_Cabin)
2767                {ifile >> input; m.found_cabin = atoi(decode(input).c_str());}
2768
2769            else if (input == pWaterfall_X)
2770                {ifile >> input; m.waterfall_x = atoi(decode(input).c_str());}
2771            else if (input == pWaterfall_Y)
2772                {ifile >> input; m.waterfall_y = atoi(decode(input).c_str());}
2773            else if (input == pGator_Health)
2774                {ifile >> input; boss_gator.health = atoi(decode(input).c_str());}
2775            else if (input == pFound_Waterfall)
2776                {ifile >> input; m.found_waterfall = atoi(decode(input).c_str());}
2777
2778            else if (input == pTemple_X)
2779                {ifile >> input; m.temple_x = atoi(decode(input).c_str());}
2780            else if (input == pTemple_Y)
2781                {ifile >> input; m.temple_y = atoi(decode(input).c_str());}
```

```
2782            else if (input == pFound_Temple)
2783                {ifile >> input; m.found_temple = atoi(decode(input).c_str());}
2784
2785            else if (input == pUnexplored_Map)
2786                for (int i = 0; i < 81; i++)
2787                    {ifile >> input; m.unexplored_map[i] = atoi(decode(input).c_str
                        ());}
2788
2789        } while (input != pEnd);
2790
2791        // Loads in Turn class stats
2792        do {
2793            ifile >> input; input = decode(input);
2794
2795            if (input == pTurn)
2796            {ifile >> input; turn.turn = atoi(decode(input).c_str());}
2797        } while (input != pEnd);
2798
2799        ifile.close();
2800        return true;
2801    }
2802
2803    void Database::load_in_leaderboards()
2804    {
2805        ifstream lfile(lfilename);
2806        if (!lfile)
2807        {
2808            tw.write("\n\tError! Could not load leaderboards!\n\t");
2809            return;
2810        }
2811        string input, s;
2812
2813        for (int i = 0; i < 10; i++)
2814        {
2815            lfile >> input >> s;
2816            turn.check_records(atoi(decode(s).c_str()), tw.restore_spaces(decode
                (input)), true, true);
2817        }
2818
2819        for (int i = 0; i < 10; i++)
2820        {
2821            lfile >> input >> s;
2822            turn.check_records(atoi(decode(s).c_str()), tw.restore_spaces(decode
                (input)), false, true);
2823        }
2824    }
2825
2826    //-------------------------------------------------------------------------
        ------------------------------------------------
2827
2828                // Main Function
2829
```

```
2830  int main()
2831  {
2832      try
2833      {
2834          turn.set_records();
2835
2836          tw.set_type_speed(slow_type_speed);
2837          tw.write("\n\n\t\tLost and Found...\n\t\t\tThe Ultimate Survival Game... ↵
                \n"
2838              "\n\t\t\tWritten and Developed by: Zach Hollis\n\n");
2839
2840          pick.menu_choice();
2841
2842          while (true)
2843          {
2844              pick.orgin_choice();
2845              turn.next_turn();
2846              db.back_up();
2847          }
2848      }
2849      catch(int e)
2850      {
2851          tw.set_type_speed(slow_type_speed);
2852          tw.write("\n\t\tThank you for playing!...\n\n");
2853          return 0;
2854      }
2855  }
2856
```