```
Yasith Ellewela and Seungkwon Lim

ORDER OF FILES IN SUBMISSION:
1. gooseEscapeActors.hpp
2. gooseEscapeUtil.hpp
3. gooseEscapeConsole.hpp
4. gooseEscapeGamePlay.hpp
5. gooseEscapeGamePlay.cpp
6. gooseEscapeMain.cpp


------------gooseEscapeActors.hpp-------------------------

#ifndef GOOSE_ESCAPE_ACTORS
#define GOOSE_ESCAPE_ACTORS
#include <cmath>
#include <BearLibTerminal.h>
#include "gooseEscapeUtil.hpp"

/*
    Modify this class to contain more characteristics of the "actor".  Add
    functions that will be useful for playing the game that are specific to
    the Actor.

    Feel free to add additional Classes to your program.
*/

/*
    Going further:  Learn the other syntax for implementing a class that is
    more appropriate for working with multiple files, and improve the class
    code.
*/

class Actor
{
  private:
    int actorChar;
    int location_x, location_y;
    // energy has to be from 1-100. If energy is below 50, player will go slower
    int energy;

  public:
    Actor()
    {
        actorChar = int('A');
        location_x = MIN_SCREEN_X;
        location_y = MIN_SCREEN_Y;
        energy = MAX_ENERGY;

        put_actor();
    }

    Actor(char initPlayerChar, int x0, int y0)
    {
        change_char(initPlayerChar);
        location_x = MIN_SCREEN_X;
        location_y = MIN_SCREEN_Y;
        update_location(x0,y0);
    }

    int get_x() const
    {
        return location_x;
    }

    int get_y() const
    {
        return location_y;
    }
```

```
70
71      int get_energy() const
72      {
73
74          return energy;
75      }
76      // sets energy of actor
77      int set_energy(int new_energy)
78      {
79          energy = new_energy;
80      }
81
82      //updates energy of actor when needed
83      void update_energy(int energy_change)
84      {
85          energy += energy_change;
86      }
87
88      //used to respawn the actor at a given location.
89      void respawn_location(int x_coord, int y_coord)
90      {
91          terminal_clear_area(location_x, location_y,1 ,1);
92          location_x = x_coord;
93          location_y = y_coord;
94          put_actor();
95      }
96
97      string get_location_string() const
98      {
99          char buffer[80];
100         itoa(location_x,buffer,10);
101         string formatted_location = "(" + string(buffer) + ",";
102         itoa(location_y,buffer,10);
103         formatted_location += string(buffer) + ")";
104         return formatted_location;
105     }
106
107     void change_char(char new_actor_char)
108     {
109         actorChar = min(int('~'),max(int(new_actor_char),int(' ')));
110     }
111
112     bool can_move(int delta_x, int delta_y) const
113     {
114         int new_x = location_x + delta_x;
115         int new_y = location_y + delta_y;
116
117         return new_x >= MIN_BOARD_X && new_x <= MAX_BOARD_X
118           && new_y >= MIN_BOARD_Y && new_y <= MAX_BOARD_Y;
119     }
120
121     void update_location(int delta_x, int delta_y)
122     {
123         if (can_move(delta_x, delta_y))
124         {
125             terminal_clear_area(location_x, location_y, 1, 1);
126             location_x += delta_x;
127             location_y += delta_y;
128             put_actor();
129         }
130     }
131
132     void put_actor() const
133     {
134         terminal_put(location_x, location_y, actorChar);
135         terminal_refresh();
136     }
137
138 };
```

```
#endif
```

```
------------gooseEscapeUtil.hpp-------------------------
```

```
//STUDENTS: Be very careful if you decide to change these values

#ifndef GOOSE_UTIL
#define GOOSE_UTIL
/*
With graphics, screens are given an x,y coordinate system with the origin
in the upper left corner.  So it means the coordinate axes are:
---------------->   x direction
|
|
|
|
|
V

y direction
*/

// Screen layout
const int NUM_SCREEN_X = 80;
const int NUM_SCREEN_Y = 25;
const char SETUP_MESSAGE[] = "window: title='Escape the Goose', size=80x25";
const int MIN_SCREEN_X = 0;
const int MIN_SCREEN_Y = 0;
const int MAX_SCREEN_X = NUM_SCREEN_X - 1;
const int MAX_SCREEN_Y = NUM_SCREEN_Y - 1;

// Play area layout
const int NUM_BOARD_X = 80;  // needs to be <= NUM_SCREEN_X
const int NUM_BOARD_Y = 21;  // needs to be < NUM_SCREEN_Y
const int MIN_BOARD_X = 0;
const int MIN_BOARD_Y = 0;
const int MAX_BOARD_X = NUM_BOARD_X - 1;
const int MAX_BOARD_Y = NUM_BOARD_Y - 1;

// Console message area layout
const int NUM_CONSOLE_X = 79;  // needs to be <= NUM_SCREEN_X
const int NUM_CONSOLE_Y = NUM_SCREEN_Y - NUM_BOARD_Y;
const int MIN_CONSOLE_X = 1;
const int MIN_CONSOLE_Y = MAX_BOARD_Y + 1;
const int MAX_CONSOLE_X = MAX_SCREEN_X;
const int MAX_CONSOLE_Y = MAX_SCREEN_Y;

//Safe Zone
const int SAFE_ZONE_LENGTH = 2;

//Wall
const int WALL_LENGTH = 10;

//Energy
const int MAX_ENERGY = 100;
const int CHANGE_ENERGY = -10;

//Buffer
const int BUFFER_X = 2;
const int BUFFER_Y = 5;
```

```
#endif
```

```
------------gooseEscapeConsole.hpp-------------------------
/*
    STUDENTS: Don't change the code in this file (unless you are very
                comfortable with using the BearLibTerminal)
*/

#ifndef GOOSE_CONSOLE
#define GOOSE_CONSOLE
#include <iostream>
using namespace std;
#include <BearLibTerminal.h>
#include "gooseEscapeUtil.hpp"

/*
    Going further:  Learn the other syntax for implementing a class that is
    more appropriate for working with multiple files, and improve the
    class code.
*/
class Console
{
  private:
    string messages[NUM_CONSOLE_Y];
    int messageRow;

  public:
    Console()
    {
        /* each string element in messages is initialized to ""
        by the string constructor*/
       messageRow = 0;
    }

    void writeLine(string new_message_to_print)
    {
        //clear the whole console
        terminal_clear_area(MIN_CONSOLE_X, MIN_CONSOLE_Y,
                            NUM_CONSOLE_X, NUM_CONSOLE_Y);

        // update content of console rows
        if(messageRow < NUM_CONSOLE_Y)
        {
            messages[messageRow] = new_message_to_print;
            messageRow++;
        }
        else
        {
            for(int index = 0; index < NUM_CONSOLE_Y-1; index++)
                messages[index] = messages[index+1];
            messages[NUM_CONSOLE_Y-1] = new_message_to_print;
        }

        // output all message rows to the console
        for(int line = 0; line < NUM_CONSOLE_Y; line++)
            terminal_print(MIN_CONSOLE_X, MIN_CONSOLE_Y + line,
                            messages[line].c_str());

        terminal_refresh();
    }
```

```
277
278   /*
279       Having more than one console is a bad idea.  So you really shouldn't be
280       calling these functions.
281   */
282       Console(Console const & src)
283       {
284           /* memory allocation of array is fixed, so copy constructor
285           and assignment operator are the same */
286           *this = src;
287       }
288
289       Console& operator=(Console const & src)
290       {
291           cerr << "Warning!  You have more than one Console object" << endl;
292           for (int index = 0; index < NUM_CONSOLE_Y; index++)
293               messages[index] = src.messages[index];
294           messageRow = src.messageRow;
295           return *this;
296       }
297   };
298   #endif
299
300
301
302
303
304
305
306
307
308
309   ------------gooseEscapeGamePlay.hpp-------------------------
310
311   #ifndef GOOSE_ESCAPE_GAMEPLAY
312   #define GOOSE_ESCAPE_GAMEPLAY
313   #include "gooseEscapeUtil.hpp"
314   #include "gooseEscapeActors.hpp"
315   #include "gooseEscapeConsole.hpp"
316
317   /*This file is all about the game world.  You will modify this to add
318       constants and function prototypes.  Modify gooseGamePlay.cpp to
319       actually add functionality.
320   */
321
322   /*
323       Declare constants to indicate various game world features in the board
324       array.  Modify them to fit what you would like to do in the game.  You can
325       change the type if you choose to store your game board as something other
326       than intengers.
327   */
328   // Going further:  Learn how to use an enum for these values
329   const int EMPTY = 0;
330   const int SHALL_NOT_PASS = 1;
331   const int WINNER = 2;
332   const int FREEZE = 3;
333   const int ENERGY = 4;
334   /*
335       A few examples of characters both for actors and for the game board
336       itself are shown.
337   */
338   //display characters
339   const int PLAYER_CHAR = int('@');
340   const int MONSTER_CHAR = int('G');
341   const int WALL_CHAR = int('o');
342   const int WIN_CHAR = int('%');
343   const int FREEZE_CHAR = int('*');
344   const int ENERGY_CHAR = int('+');
345
```

```
346  /*
347      Game play function prototypes are give below.
348  */
349
350  // print the game board function protype
351
352  /*
353      Do something when the goose captures the player
354
355      If you want to attack or something else, this is the function you
356      need to change.  For example, maybe the two touch each other and
357      then fight.  You could add a health to the Actor class that is
358      updated.  Run, use weapons, it's up to you!
359  */
360  bool isTouch(Actor const & player, int x_coord, int y_coord);
361
362
363  bool captured(Actor const & player, Actor const & monster);
364
365  /*
366      Move the player to a new location based on the user input.  You may want
367      to modify this if there are extra controls you want to add.
368
369      All key presses start with "TK_" then the character.  So "TK_A" is the a
370      key being pressed.
371
372      A look-up table might be useful.
373      You could decide to learn about switch statements and use them here.
374  */
375
376
377  void movePlayer(int key, Actor & player,
378                  int game_board[NUM_BOARD_Y][NUM_BOARD_X]);
379
380  void gooseChase(Actor & player, Actor & monster,
381                  int game_board[NUM_BOARD_Y][NUM_BOARD_X]);
382
383  void printWorld(int game_board[NUM_BOARD_Y][NUM_BOARD_X]);
384
385  void printLives(int num_of_lives);
386
387  void printEnergy(Actor const & player);
388
389  void makeWall(int game_board[NUM_BOARD_Y][NUM_BOARD_X], bool orientation);
390
391  void makeSafeZone(int game_board[NUM_BOARD_Y][NUM_BOARD_X]);
392
393  int randNumGen(int max);
394
395  bool hasWon(Actor const & player, int winner_x_location, int winner_y_location);
396
397
398  /*
399      What other functions do you need to make the game work?  What can you
400      add to the basic functionality to make it more fun to play?
401  */
402
403
404  #endif
405
406
407
408
409
410
411
412
413
414
```

```
415
416 ------------gooseEscapeGamePlay.cpp-------------------------
417
418 #include <iostream>
419 #include <cmath>
420 #include <cstdlib>
421 #include <string>
422 #include <sstream>
423 using namespace std;
424 #include <BearLibTerminal.h>
425 #include "gooseEscapeUtil.hpp"
426 #include "gooseEscapeActors.hpp"
427 #include "gooseEscapeConsole.hpp"
428 #include "gooseEscapeGamePlay.hpp"
429
430 extern Console out;
431 /*
432 With graphics, screens are given an x,y coordinate system with the origin
433 in the upper left corner.  So it means the coordinate axes are:
434 ---------------->   x direction
435 |
436 |
437 |
438 |
439 |
440 V
441
442 y direction
443 */
444
445 /*
446     Print the game world
447
448     The functions should draw characters to present features of the game
449     board, e.g. win location, obstacles, power ups
450 */
451 //this function is to create a wall at a random location
452 // and make it either vertical or horizontal with a constant length of 10 tiles
453 void makeWall(int game_board[NUM_BOARD_Y][NUM_BOARD_X], bool orientation)
454 {
455     int random_x = rand() % (NUM_BOARD_X- WALL_LENGTH);
456     int random_y = rand() % (NUM_BOARD_Y- WALL_LENGTH);
457     cout << "X = " << random_x << endl;
458     cout << "Y = " << random_y << endl;
459     if (orientation) // horizontal wall
460     {
461         int bound = random_x+WALL_LENGTH;
462         for (int index = random_x; index <= bound; index ++)
463         {
464             game_board[random_y][index] = SHALL_NOT_PASS;
465         }
466     }
467
468     else // vertical wall
469     {
470         int bound = random_y+WALL_LENGTH;
471         for (int index = random_y; index <= bound; index ++)
472         {
473             game_board[index][random_x] = SHALL_NOT_PASS;
474         }
475     }
476 }
477
478 // this function is called to print the game world at the start of the game
479 void printWorld(int game_board[NUM_BOARD_Y][NUM_BOARD_X])
480 {
481     // for each row in the array
482     for (int index1 = 0; index1 < NUM_BOARD_Y; index1 ++)
483     {
```

```
484             //for each column in the array
485             for (int index2 = 0; index2 < NUM_BOARD_X; index2 ++)
486             {
487                 /* print the value of the game board, at the specified
488                  row and column index, to the console */
489                 //terminal_put(index2, index1, game_board[index1][index2]);
490                 if (game_board[index1][index2] == WINNER)
491                 {
492                     terminal_put(index2, index1, WIN_CHAR);
493                 }
494 
495                 else if (game_board[index1][index2] == SHALL_NOT_PASS)
496                 {
497                     terminal_put(index2, index1, WALL_CHAR);
498                 }
499 
500                 else if (game_board[index1][index2] == FREEZE)
501                 {
502                     terminal_put(index2, index1, FREEZE_CHAR);
503                 }
504 
505                 else if (game_board[index1][index2] == ENERGY)
506                 {
507                     terminal_put(index2, index1, ENERGY_CHAR);
508                 }
509             }
510         }
511 
512         terminal_refresh();
513     }
514 
515     //generates a random number to be used for random spawn locations
516     int randNumGen(int max)
517     {
518         return rand() % (max-1) + 1;
519     }
520 
521     //this function prints the current lives of the player on the screen
522     void printLives(int num_of_lives)
523     {
524         ostringstream lives;
525         lives << num_of_lives;
526         out.writeLine("Number of lives: " + lives.str());
527     }
528 
529     //this function prints the current energy of the player on the screen
530     void printEnergy(Actor const & player)
531     {
532         ostringstream energy;
533 
534         if(player.get_energy() <= 0)
535         {
536             energy << 0;
537         }
538 
539         else
540         {
541             energy << player.get_energy();
542 
543         }
544 
545         out.writeLine("Energy Left: " + energy.str());
546 
547     }
548 
549     //this function determines whether the player is in contact with an
550     //item on the gameboard
551     bool isTouch(Actor const & player, int x_coord, int y_coord)
552     {
```

```
553       if (abs(x_coord - player.get_x()) <= 1
554           && abs(y_coord - player.get_y()) <= 1)
555       {
556           return true;
557       }
558
559       return false;
560   }
561
562   /*
563       Do something when the goose captures the player
564
565       If you want to attack or something else, this is the function you
566       need to change.  For example, maybe the two touch each other and
567       then fight.  You could add a health to the Actor class that is
568       updated.  Run, use weapons, it's up to you!
569   */
570
571   /* this function was edited to suit our own custom win condition. We determined
572      that the goose can capture the player by being adjacent to the player as
573      well. Thus, the conditions were updated to accomodate this */
574   bool captured(Actor const & player, Actor const & monster)
575   {
576       return (abs(player.get_x() - monster.get_x()) <= 1
577            && player.get_y() == monster.get_y()) ||
578            (abs(player.get_y() - monster.get_y()) <= 1
579            && player.get_x() == monster.get_x());
580   }
581
582   //returns true if the player reaches the safe zone
583   bool hasWon(Actor const & player, int winner_x_location, int winner_y_location)
584   {
585       if(abs(player.get_x() - winner_x_location) <= 1
586          && player.get_y() == winner_y_location ||
587         (abs(player.get_y() - winner_y_location) <= 1
588          && player.get_x() == winner_x_location))
589       {
590           return true;
591       }
592       return false;
593   }
594
595   /*
596       Move the player to a new location based on the user input.  You may want
597       to modify this if there are extra controls you want to add.
598
599       All key presses start with "TK_" then the character.  So "TK_A" is the a
600       key being pressed.
601
602       A look-up table might be useful.
603       You could decide to learn about switch statements and use them here.
604   */
605
606   void movePlayer(int key, Actor & player,
607                   int game_board[NUM_BOARD_Y][NUM_BOARD_X])
608   {
609       int yMove = 0, xMove = 0;
610       if (key == TK_UP)
611       {
612               yMove = -1;
613       }
614
615       else if (key == TK_DOWN)
616       {
617           yMove = 1;
618       }
619
620       else if (key == TK_LEFT)
621       {
```

```
622           xMove = -1;
623       }
624
625       else if (key == TK_RIGHT)
626       {
627           xMove = 1;
628       }
629
630
631
632
633
634       /* the following code will execute so long as the position the player
635          tries to move to passes the 'can_move' method and is not encountering
636          a wall */
637       if (player.can_move(xMove, yMove)
638         && game_board[player.get_y()+yMove][player.get_x()+xMove]
639           != SHALL_NOT_PASS)
640         {
641           player.update_location(xMove, yMove);
642         }
643
644   }
645   // this algorithm allows the goose to chase the player
646   void gooseChase(Actor & player, Actor & monster,
647
648                   int game_board[NUM_BOARD_Y][NUM_BOARD_X])
649
650   {
651       /* This algorithm takes the absolute value of the change in x and y
652          distance between the goose and the player. If the magnitude of the
653          change in y is greater than the magnitude of the change in x, then we
654          will modify the y value of the goose to get it closer to the player.
655          The same applies for the opposite. */
656       int negative_change = -1;
657       int positive_change = 1;
658
659       if (player.get_energy() <= 50)
660       {
661           positive_change = positive_change*2;
662           negative_change = negative_change*2;
663       }
664       if(abs(player.get_y() - monster.get_y()) >
665           abs(player.get_x() - monster.get_x()))
666       {
667           //if the change in y values was negative
668           if (player.get_y() - monster.get_y() < 0)
669           {
670               monster.update_location(0,negative_change);
671           }
672
673           else
674           {
675               monster.update_location(0,positive_change);
676           }
677       }
678
679       else
680       {
681           if (player.get_x() - monster.get_x() < 0)
682           {
683               monster.update_location(negative_change,0);
684           }
685
686           else
687           {
688               monster.update_location(positive_change,0);
689           }
690       }
```

```
}



/*
    What other functions do you need to make the game work?  What can you
    add to the basic functionality to make it more fun to play?
*/









------------gooseEscapeMain.cpp------------------------
#include <BearLibTerminal.h>
#include <cmath>
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;
#include "gooseEscapeUtil.hpp"
#include "gooseEscapeActors.hpp"
#include "gooseEscapeConsole.hpp"
#include "gooseEscapeGamePlay.hpp"

//set up the console.   Don't modify this line!
Console out;

int main()
{
    //Set up the window.  Don't edit these two lines
    terminal_open();
    terminal_set(SETUP_MESSAGE);

/*
    The code below provides a skeleton of the game play.  You will need to
    write code for setting up the game board, and playing the game itself.
    You can modify the code given as needed.

    Call the functions that you have written in the game play file, and that
    you have added to the Actor class.
*/

    //make the player
    srand(time(NULL));

    /* The following variables are random starting x-y coordinates for
     the player, the monster(goose), and the location of the safezone(winner) */

    int player_location_x = randNumGen(NUM_SCREEN_X-BUFFER_X);
    int player_location_y = randNumGen(NUM_SCREEN_Y-BUFFER_Y);
    int monster_location_x = randNumGen(NUM_SCREEN_X-BUFFER_X);
    int monster_location_y = randNumGen(NUM_SCREEN_Y-BUFFER_Y);
    int winner_location_x = randNumGen(NUM_SCREEN_X-BUFFER_X);
    int winner_location_y = randNumGen(NUM_SCREEN_Y-BUFFER_Y);
    int freeze_location_x = randNumGen(NUM_SCREEN_X-BUFFER_X);
    int freeze_location_y = randNumGen(NUM_SCREEN_Y-BUFFER_Y);
    int energy_boost_location_x = randNumGen(NUM_SCREEN_X-BUFFER_X);
    int energy_boost_location_y = randNumGen(NUM_SCREEN_Y-BUFFER_Y);



     //initalize the game world array
```

```
760       int game_world[NUM_BOARD_Y][NUM_BOARD_X] = {0};
761
762       //place the visible win (safe zone) location in the game world
763       game_world[winner_location_y][winner_location_x] = WINNER;
764
765       const int NUM_WALLS = 4;
766       for (int num = 1; num <= NUM_WALLS; num ++)
767       {
768           int orientation = num % 2;
769           makeWall(game_world, orientation);
770       }
771
772       printWorld(game_world);
773
774       // you probably don't want to start in the same place each time
775       Actor player(PLAYER_CHAR, player_location_x, player_location_y);
776
777
778       //make the monster
779       Actor monster(MONSTER_CHAR, monster_location_x,monster_location_y);
780
781
782
783
784
785
786
787   /*
788       Initiallize locations in the game board to have game features.  What if you
789       have man things to add to the game board?  Should you use a loop?  Does it
790       make sense to store this information in a file?  Should this code be a
791       function as well?
792   */
793
794
795
796       // Call the function to print the game board
797
798       // Printing the instructions
799       out.writeLine("Escape the Goose! " + monster.get_location_string());
800       out.writeLine("Use the arrow keys to move");
801       out.writeLine("If the goose catches you, you lose!");
802       out.writeLine("Be careful! Sometimes the goose can jump through walls!");
803
804   /*
805       This is the main game loop.  It continues to let the player give input
806       as long as they do not press escape or close, they are not captured by
807       the goose, and they didn't reach the win tile
808   */
809   /*
810       All key presses start with "TK_" then the character.  So "TK_A" is the "a"
811       key being pressed.
812
813   */
814
815
816       // can be any valid value that is not ESCAPE or CLOSE
817       int keyEntered = TK_A;
818       int player_lives = 2; //player has 2 lives
819       printLives(player_lives);
820       player.set_energy(MAX_ENERGY); // players life set to 100
821       printEnergy(player);
822
823       /* this loop will execute so long as the keys entered are not
824          the escape key, or the close button, and will also continue considering
825          the player has not been captured, nor have they won */
826       //this while loop allows for multiple lives
827       while (player_lives > 0)
828       {
```

```
829         player.respawn_location(player_location_x, player_location_y);
830         monster.respawn_location(monster_location_x, monster_location_y);
831         game_world[freeze_location_y][freeze_location_x] = FREEZE;
832         game_world[energy_boost_location_y][energy_boost_location_x] = ENERGY;
833         player.set_energy(MAX_ENERGY);
834         printWorld(game_world);
835
836
837         while(keyEntered != TK_ESCAPE && keyEntered != TK_CLOSE
838               && !captured(player,monster)&&
839               !(hasWon(player,winner_location_x,winner_location_y)))
840         {
841             // get player key press
842             keyEntered = terminal_read();
843             printWorld(game_world);
844             if (keyEntered != TK_ESCAPE && keyEntered != TK_CLOSE)
845             {   //the goose is frozen for 3 turns if the player interacts
846                 //with the freeze power-up
847                 if (isTouch(player,freeze_location_x, freeze_location_y))
848                 {
849                     out.writeLine("Goose cannot move! They are frozen!");
850                     game_world[freeze_location_y][freeze_location_x] = 0;
851                     movePlayer(keyEntered,player,game_world);
852                 }
853
854                 else
855                 {
856                     // call the goose's chase function
857                     movePlayer(keyEntered,player,game_world);
858                     gooseChase(player, monster, game_world);
859                 }
860
861                 //this will restore the energy of the player back to 100
862                 if (isTouch(player, energy_boost_location_x,
863                             energy_boost_location_y))
864                 {
865                     player.set_energy(MAX_ENERGY);
866                     out.writeLine("Emergy Restored!");
867                     game_world[energy_boost_location_y]
868                               [energy_boost_location_x] = 0;
869
870                 }
871
872                 //display remaining energy after each turn
873                 player.update_energy(CHANGE_ENERGY);
874                 printEnergy(player);
875             }
876
877         }
878
879
880
881         //reduce the lives of the player if captured
882         if (captured(player,monster))
883         {
884             player_lives -= 1;
885             printLives(player_lives);
886
887         }
888
889         else
890         {
891             player_lives = 0;
892         }
893
894
895     }
896
897
```

```
898         if (keyEntered != TK_CLOSE)
899         {
900             //once we're out of the loop, the game is over
901             out.writeLine("Game has ended");
902 
903             //if the player reached the safe zone
904             if (hasWon(player,winner_location_x,winner_location_y))
905             {
906                 out.writeLine("You reached the safe zone!");
907             }
908             //only other case is that they were caught by the goose
909             else
910             {
911                 out.writeLine("You were caught by the goose!");
912             }
913 
914             // Wait until user closes the window
915             while (terminal_read() != TK_CLOSE);
916         }
917 
918         //game is done, close it
919         terminal_close();
```