ESE 224 Project

Hangman

12/8/2014

Michael Cassano 108543535

Brandon Conklin 109351481

Daniel Peterson 109091561

Suphasith Usdonvudhikai 109219981

**Introduction**

This program simulates the game “hangman.” The purpose of the game is to correctly guess the computer-assigned 7-letter word either one letter at a time, or by guessing the entire word at one time. Each correct letter gets filled into the blank spaces, while incorrect guesses result in a new body part of the hanging man to appear on the output screen. If the user correctly guesses the word in the allotted number of guesses, then they have won the game. However if the user uses up all of their guesses without correctly obtaining the answer, then they have lost and the game is over. This program comes equipped with a single player, as described above, and a double player mode. The double player mode is essentially the same game, only the user is playing against an AI who is also trying to correctly guess the word. The program also has the capability of interacting with an admin, allowing the user to add words to the word list if they have the proper password.

**Include Files**

1) <iostream> is required for inputting and outputting in our main function, mainly cout and cin.

2) <fstream> is required for ifstream and ofstream.

3) <string> is required to access the string class.

4) <string.h> is required to manipulate C style strings.

5) <cstdlib> is required for srand() and rand().

6) <ctime> is required for to manipulate time information.

7) <iomanip> is required for parametric manipulation, namely setw().

8) <vector> is required to use the vector class.

9) ‘using namespace std’ tells the compiler to use the library filenames declared in the namespace std.

**Functions**

1) binary\_search

bool binary\_search(const vector<string>& sorted\_vec, string key)

{

size\_t mid, left = 0 ;

size\_t right = sorted\_vec.size(); // one position passed the right end

while (left < right) {

mid = left + (right - left)/2;

if (key > sorted\_vec[mid]){

left = mid+1;

}

else if (key < sorted\_vec[mid]){

right = mid;

}

else

{

return true;

}

}

return false;

}

The binary\_search function is implemented during the administrative phase of the program. If the user inputs the word “admin” for their name when they are asked by the program, then the computer recognizes that an administrator may be trying to access the word list. Once “admin” is entered, and the correct password is entered, the user has the option of adding a word to the word list. The binary\_search function checks to see if the entered word is already a part of our word list. If the new word is already on the list, then the function returns a value of ‘true.’ If the word is not on the list, then the function returns a value of ‘false.’

2) rand\_int

int rand\_int(int a, int b)

{

return rand()%(b-a+1) + a;

}

The rand\_int function is one that is used to create a random integer to be used by the computer. It is implemented to generate a random number corresponding to the index of a word in our word list. Once the number is generated, it is returned to the main function.

3) linear\_search

bool linear\_search(string word, char letter)

{

int length = word.length();

for(int i = 0; i < length; i++)

{

if(word[i] == letter)

{

return true;

}

}

return false;

}

The linear\_search function is implemented to see if the user has correctly inputted a word or letter. We decided to use linear search since the word is only 7 or more chracters long and would suffice for run time. It uses a for loop to run through each letter of the correct word and checks to see if the user-inputted letter matches any of the letters in the answer, or if the user-inputted word completely matches the correct answer. If a letter, or the entire word, is correctly guessed, the function returns a value of ‘true’, and returns a value of ‘false’ otherwise.

4) make\_man

void make\_man(int num)

{

switch (num)

{

case 7:

cout<< "\n ================|\n // |\n || |\n || |\n || \_^\_\n || / ..\\\n || [ \_ ]\n || \\\_\_\_/\n ||\n || || - \*Snap!!!\*\n || /||\\\n || //||\\\\\n || // || \\\\\n || \* || \*\n || //\\\\\n || // \\\\\n /\\ // \\\\\n//\\\\ \*\*\* \*\*\*\n/||\\\\\n\_||\_\\\\\n";

break;

case 6:

cout<< "\n ================|\n // |\n || |\n || |\n || \_^\_\n || / ..\\\n || [ \_ ]\n || \\\_\_\_/\n || ||\n || /||\\\n || //||\\\\\n || // || \\\\\n || \* || \*\n || //\n || //\n || //\n /\\ \*\*\*\n//\\\\ \n/||\\\\ \n\_||\_\\\\\n";

break;

case 5:

cout<< "\n ================|\n // |\n || |\n || |\n || \_^\_\n || / ..\\\n || [ \_ ]\n || \\\_\_\_/\n || ||\n || /||\\\n || //||\\\\\n || // || \\\\\n || \* || \*\n ||\n ||\n ||\n /\\\n//\\\\\n/||\\\\\n\_||\_\\\\\n";

break;

case 4:

cout<< "\n ================|\n // |\n || |\n || |\n || \_^\_\n || / ..\\\n || [ \_ ]\n || \\\_\_\_/\n || ||\n || /||\n || //||\n || // ||\n || \* ||\n ||\n ||\n ||\n /\\\n//\\\\\n/||\\\\\n\_||\_\\\\\n";

break;

case 3:

cout<< "\n ================|\n // |\n || |\n || |\n || \_^\_\n || / ..\\\n || [ \_ ]\n || \\\_\_\_/\n || ||\n || ||\n || ||\n || ||\n || ||\n ||\n ||\n ||\n /\\\n//\\\\\n/||\\\\\n\_||\_\\\\\n";

break;

case 2:

cout<< "\n ================|\n // |\n || |\n || |\n || \_^\_\n || / ..\\\n || [ \_ ]\n || \\\_\_\_/\n || ||\n ||\n ||\n ||\n ||\n ||\n ||\n ||\n /\\\n//\\\\\n/||\\\\\n\_||\_\\\\\n";

break;

case 1:

cout<< "\n ================|\n // |\n || |\n || |\n || \_^\_\n || / ..\\\n || [ \_ ]\n || \\\_\_\_/\n ||\n ||\n ||\n ||\n ||\n ||\n ||\n ||\n /\\\n//\\\\\n/||\\\\\n\_||\_\\\\\n";

break;

case 0:

cout<<"\n ================\n //\n ||\n ||\n ||\n ||\n ||\n ||\n ||\n ||\n ||\n ||\n ||\n ||\n ||\n ||\n /\\\n//\\\\\n/||\\\\\n\_||\_\\\\\n";

break;

}

}

The make\_man function is used to display the hanging man in our program. Each incorrect answer in the game results in a new body part of the man to become displayed. With each incorrect answer comes the man’s head, neck, body, right arm, left arm, right leg, and left leg, in that order. Once the left leg is displayed and the man is fully intact, the user has run out of guesses and lost the game, which then displays the hanging man’s neck being snapped. The function uses the switch statement to display the correct image for the user. It takes the integer variable num\_chances from the main function and uses this value to jump to the correct case of the switch function. Each incorrect answer increments the num\_chances variable and thus the higher the number within our function make\_man, the more body parts will be displayed.

5) print\_line

void print\_line(char characters[], string word, int k)

{

char letters[word.size()];

for (int i = 0; i <word.size(); i++)

{

letters[i] = '-';

for(int j = 0; j < k; j++)

{

if(word.at(i) == characters[j])

letters[i] = characters[j];

}

}

for(int i = 0; i <word.size(); i++)

cout << letters[i];

cout << endl;

return;

}

The print\_line function is used to print the dashed line in our program which is used to display the number of letters in the word. Each dashed line represents a letter that the user or the AI has not yet guessed. Each letter that has been guessed replaces its corresponding dash in the line generated by our function. The function first generates all dashed lines corresponding to the number of letters in the word, then checks to see if any of the letters in the word have been properly guessed. Once a letter in the word is guessed, the function replaces the dash in that letter’s place with the letter itself.

5) win

bool win(char characters[], string word, int length)

{

bool found = true;

for(int i = 0; i<word.size();i++)

{

found = false;

for(int j = 0; j<length;j++)

{

if(characters[j] == word.at(i))

found = true;

}

if (!found)

return false;

}

return true;

}

The win function is implemented to check whether or not the user has won the game. After each correct answer in the game, the program runs through the win function to see if the user has guessed the last of the remaining letters. If the user has correctly guessed every letter of the word, the function returns a value of true to the main function. If the user has not correctly guessed every letter, then the function returns a value of false to the main function.

7) index\_search

int index\_search(const vector<string>& sorted\_vec, string key)

{

size\_t mid, left = 0 ;

size\_t right = sorted\_vec.size();

while (left < right) {

mid = left + (right - left)/2;

if (key > sorted\_vec[mid]){

left = mid+1;

}

else if (key < sorted\_vec[mid]){

right = mid;

}

else

{

return mid;

}

}

return 0;

}

The index\_search function is a binary search for the word within our vector to see what number index the word is associated with. Once the user wins the game and receives a congratulatory message, they are also told what index number the word was in our alphabetical word list. The function looks through our word list and finds out the index, then returns the index to our main function.

7) selection\_sort

void selection\_sort(vector<string>& vector)

{

int index;

int length = vector.size();

for(int i = 0; i < length; i++)

{

index = i;;

for(int j = i; j < length; j++)

{

if(vector[j] < vector[index])

index = j;

}

swap(vector[i], vector[index]);

}

}

The selection\_sort function is a sorting function that organizes the entries in our word-list, including any new entries. It takes all of the words in our list and sorts them alphabetically. Once a new word is added, it takes the whole list and reorganizes it alphabetically so there are no words out of place.

**Main Function**

The main function of our program starts with the initializing of a multitude of variables of type string, int, and char. Its first operation is an if statement which checks whether or not our word list can be accessed and prints an error statement if it cannot. Our word list is a text file that contains 4,563 words from an online dictionary source. Next, our main function prints out our introductory image of “HANGMAN” followed by the amount of words imported from our word list. It then asks for the user to input their name. If the user types their name, they are then asked to choose a game mode: single player or double player. If the user enters the word “admin,” then the program welcomes the administrator and asks them to enter the password, which in our case is the word “password.” If they do not enter the correct password, they receive a message telling them so and the program exits. If the correct password is entered, the user is then prompted to enter <1> to add a word to the word list. If anything else is entered, then the program exits in this administrative phase. If the user enters <1> they must then add a word to the word list. The program then performs a binary search to check if the word is unique. If the word already exists in the list, the program informs the user. If the word is new to the list, it goes through a sorting algorithm that places it alphabetically in the word list. This is done by loading the entire Dictionary.txt file into a vector. The selection sort algorithm then sorts the entire list. This has to be made possible so that the binary search can be used.

If the administrative phase is bypassed and the user enters their name, they are prompted to choose single or double player mode. Once the choice is entered, the program uses our random integer-creating function to create a number that corresponds to the index of one of the words on our word list. The program then equates the randomly generated integer to the index of a word, and sets that word as the answer to our game.

If the player chooses single player mode, then the program prints out a message telling the user that it has chosen a random word and that the user has 7 guesses to get the answer. Following this, the program prints the apparatus from which our man is hanged. Then a message is printed informing the user of the amount of guesses they have remaining, and prompts them to enter the <0> to guess a word, or <1> to guess an individual letter. If anything besides these two characters is entered, the message repeats until either 0 or 1 is chosen this concept is known has exception handling. If the player chooses to guess a word, two possible outcomes follow. They either correctly guess the entire word, in which case the screen displays a congratulatory message; or the user incorrectly guesses the word, in which case their number of guesses decrements, and a new message is displayed prompting the user once again to choose either to guess the word or a singular letter. This process repeats until the word is guessed, either as a whole word or by individual letter, or until the user runs out of guesses. Keep in mind that each correct letter guess does not deplete your number of guesses, i.e. if the user guesses a letter that is in the word, and they had 5 guesses remaining, the next turn the user will still have 5 guesses remaining. The number of guesses actually represents the number of incorrect guesses remaining that the user may use. If the player runs out of guesses, the program will print a message to the screen telling them that they have lost the game, and it will show our hanging man with a detached neck. Once the game ends, the program displays all previous images until the user presses enter, which terminates the program.

The previous paragraph described the intricacies of single player mode of our game. Double player mode has the same basic idea as single player, only this time the user is playing against an AI who is also trying to guess letters. Once the player selects double player mode, they are prompted to guess a word or letter. Just as in single player mode, correct letter guesses do not count against the user. Each player, the user and the AI, takes turns guessing letters, and the user has the option of guessing the entire word. The user may win by guessing the last letter of the word, or the word itself. The user will lose if the AI guesses the last letter of the word, or if the user runs out of guesses. The AI chooses letters based on our random integer generator. The program generates a random number which corresponds to a letter in the alphabet, which the AI then uses to make its guess.

**Source Code**

#include <iostream>

#include <fstream>

#include <string>

#include <string.h>

#include <cstdlib> //Required for srand(), rand().

#include <ctime>

#include <iomanip> //setw()

using namespace std;

bool binary\_search(const vector<string>& sorted\_vec, string key);

int rand\_int(int a, int b);

bool linear\_search(string word, char letter);

void make\_man(int num);

void print\_line(char characters[], string word, int k);

bool win(char characters[], string word, int length);

int index\_search(const vector<string>& sorted\_vec, string key);

void selection\_sort(vector<string>& sorted\_vec);

int main(void)

{

int str\_len, index, input, input1;

string admin = "admin";

string pass = "password";

string tmp\_word;

string password;

string text;

string name;

string word;

string guess;

string man;

int num\_guess;

int num\_chances = 0;

int k = 0;

char letter;

vector<string> Dictionary;

ifstream ifile ("Dictionary.txt");

char alphabet[26] = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n',

'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z'};

char characters[26];

char buffer[26];

if (!ifile .is\_open())

{

cout<<"File ""Dictionary.txt"" could not be read"<<endl;

exit(0);

}

while(ifile >> text)

{

Dictionary.push\_back(text);

}

//Introduction

cout <<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" <<endl;

cout << " HH HH AAAA NN N GGGGGGG MMM MMM AAAA NN N " <<endl;

cout << " HH HH AA AA N N N GG GG MMM MMM AA AA N N N " <<endl;

cout << " HH HH AA AA N N N GG MM M M MM AA AA N N N " <<endl;

cout << " HHHHHH AAAAAA N N N GG GGG MM M M MM AAAAAA N N N " <<endl;

cout << " HH HH AA AA N N N GG G MM M M MM AA AA N N N " <<endl;

cout << " HH HH AA AA N N N GG G MM MMM MM AA AA N N N " <<endl;

cout << " HH HH AA AA N N GGGGGGG MM MM AA AA N N " <<endl;

cout <<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" <<endl;

cout << setw(33) << Dictionary.size() <<" Words Imported"<<endl;

cout << endl;

cout << "What is your name: ";

cin >> name;

if (admin.compare(name) == 0)

{

int tmp;

cout<<endl<<endl;

cout<<"Welcome administrator"<<endl;

cout<<"Enter password: ";

cin >> password;

cout <<endl;

if(pass.compare(password) == 0)

{

cout<<"What would you like to do?"<<endl;

cout<<"Enter <1> to add a word"<<endl;

cout<<"Enter else to Exit"<<endl;

while(!(cin >> tmp))

{

cout<<"Enter <1> to add a word"<<endl;

cout<<"Enter else to Exit"<<endl;

cin.clear();

cin.ignore(100, '\n');

}

if(tmp == 1)

{

cout<<"Please enter a word to add: ";

cin >>tmp\_word;

if (!binary\_search(Dictionary, tmp\_word))

{

Dictionary.push\_back(tmp\_word);

cout<<tmp\_word<<" has been added"<<endl;

selection\_sort(Dictionary);

ofstream ofile("Dictionary.txt");

for(int i = 0; i < Dictionary.size(); i++)

{

ofile << Dictionary[i] <<endl;

}

ofile.close();

}

else

{

cout<<tmp\_word<<" is already in Dictionary.txt"<<endl;

}

}

else

{

cout<<"Exited!"<<endl;

exit(0);

}

}

else

{

cout<<"You entered the wrong password"<<endl;

exit(0);

}

ifile .close();

exit(0);

}

selection\_sort(Dictionary);

ofstream ofile("Dictionary.txt");

for(int i = 0; i < Dictionary.size(); i++)

{

ofile << Dictionary[i] <<endl;

}

ofile.close();

ifile.close();

cout << endl << endl <<endl;

cout<<" Welcome " <<name << ", how would you like to play? " << endl;

cout<<" Press (1) for Single Player or Press (2) for Double Player " <<endl;

//randomize word pick from library array

srand(time(NULL));

do

{

index = rand\_int(0, Dictionary.size());

word = Dictionary[index];

str\_len = word.length();

}

while(str\_len < 7);

num\_guess = 7;

//cout <<word<<endl;

while(!(cin >> input))

{

cout<<" Press (1) for Single Player or Press (2) for Double Player " <<endl;

cin.clear();

cin.ignore(100, '\n');

}

if(input == 1)

{

cout<<" Welcome to Single Player Mode " <<endl;

cout<<"A word has been randomly picked and you have " << str\_len<< " chances to guess " <<endl;

cout<<endl;

cout<<"\n ================\n //\n ||\n ||\n ||\n ||\n ||\n ||\n ||\n ||\n ||\n ||\n ||\n ||\n ||\n ||\n /\\\n//\\\\\n/||\\\\\n\_||\_\\\\\n";

cout<<" "<< endl;

for(int i = 0; i < str\_len; i++)

{

cout<<'-';

}

cout<<endl;

while(num\_guess > 0)

{

cout<<" You have "<< num\_guess << " guess(es) left " <<endl;

cout<<" Enter(0) to Guess a word or Enter(1) to guess a letter " <<endl;

while(!(cin >> input1))

{

cout<<" You have "<< num\_guess << " guess(es) left " <<endl;

cout<<" Enter(0) to Guess a word or Enter(1) to guess a letter " <<endl;

cin.clear();

cin.ignore(100, '\n');

}

if(input1==0)

{

cout <<"Word: ";

cin >> guess;

if(word.compare(guess)==0)

{

cout<<" Congratulations you have won! " <<endl;

int x;

x = index\_search(Dictionary, word);

cout<<" The Word " << word << " was index at " << x <<endl;

exit(0);

}

else

{

cout<<" Incorrect! " <<endl;

num\_guess--;

num\_chances++;

make\_man(num\_chances);

characters[k] = letter;

k++;

print\_line(characters, word, k);

}

}

if(input1==1)

{

cout <<"Letter: ";

cin >> letter;

if(linear\_search(word, letter))

{

cout<<" Correct! "<<endl;

make\_man(num\_chances);

characters[k] = letter;

k++;

print\_line(characters, word, k);

if(win(characters, word, k))

{

int x;

cout<<" Congratulations you have won! " <<endl;

x = index\_search(Dictionary, word);

cout<<" The Word " << word << " was index at " << x <<endl;

exit(0);

}

}

else

{

cout<<" Incorrect! "<<endl;

num\_guess--;

num\_chances++;

make\_man(num\_chances);

characters[k] = letter;

k++;

print\_line(characters, word, k);

}

}

}

cout <<"You have ran out of chances. YOU LOSE!" << endl;

}

if(input == 2)

{

int AI\_guess = 7;

cout<<" Welcome to Double Player Mode " <<endl;

cout<<"A word has been randomly picked and you have " << str\_len<< " chances to guess " <<endl;

cout<<" You will be playing against the AI " <<endl;

cout<<endl;

cout<<"\n ================\n //\n ||\n ||\n ||\n ||\n ||\n ||\n ||\n ||\n ||\n ||\n ||\n ||\n ||\n ||\n /\\\n//\\\\\n/||\\\\\n\_||\_\\\\\n";

cout<<" "<< endl;

for(int i = 0; i < str\_len; i++)

{

cout<<'-';

}

cout<<endl;

while(num\_guess > 0)

{

cout<<" You have "<< num\_guess << " guess(es) left " <<endl;

cout<<" Enter(0) to Guess a word or Enter(1) to guess a letter " <<endl;

while(!(cin >> input1))

{

cout<<" You have "<< num\_guess << " guess(es) left " <<endl;

cout<<" Enter(0) to Guess a word or Enter(1) to guess a letter " <<endl;

cin.clear();

cin.ignore(100, '\n');

}

if(input1==0)

{

cout <<"Word: ";

cin >> guess;

if(word.compare(guess)==0)

{

cout<<" Congratulations you have won! " <<endl;

int x;

x = index\_search(Dictionary, word);

cout<<" The Word " << word << " was index at " << x <<endl;

exit(0);

}

else

{

cout<<" Incorrect! " <<endl;

num\_guess--;

num\_chances++;

make\_man(num\_chances);

characters[k] = letter;

k++;

print\_line(characters, word, k);

int AI = rand\_int(0, 26) + 1;

char s = alphabet[AI];

cout<<endl;

cout<<" The AI guessed " << s << " " <<endl;

if(linear\_search(word, s))

{

cout<<" Correct! "<<endl;

make\_man(num\_chances);

characters[k] = s;

k++;

print\_line(characters, word, k);

if(win(characters, word, k))

{

int x;

cout<<" Sorry the AI has won " <<endl;

x = index\_search(Dictionary, word);

cout<<" The Word " << word << " was index at " << x <<endl;

exit(0);

}

}

else

{

cout<<" Incorrect! " <<endl;

make\_man(num\_chances);

characters[k] = s;

k++;

print\_line(characters, word, k);

}

}

}

if(input1==1)

{

cout <<"Letter: ";

cin >> letter;

if(linear\_search(word, letter))

{

cout<<" Correct! "<<endl;

make\_man(num\_chances);

characters[k] = letter;

k++;

print\_line(characters, word, k);

if(win(characters, word, k))

{

int x;

cout<<" Congratulations you have won! " <<endl;

x = index\_search(Dictionary, word);

cout<<" The Word " << word << " was index at " << x <<endl;

exit(0);

}

int AI = rand\_int(0, 26) + 1;

char s = alphabet[AI];

cout<<endl;

cout<<" The AI guessed " << s << " " <<endl;

if(linear\_search(word, s))

{

cout<<" Correct! "<<endl;

make\_man(num\_chances);

characters[k] = s;

k++;

print\_line(characters, word, k);

if(win(characters, word, k))

{

int x;

cout<<" Sorry the AI has won " <<endl;

x = index\_search(Dictionary, word);

cout<<" The Word " << word << " was index at " << x <<endl;

exit(0);

}

}

else

{

cout<<" Incorrect! " <<endl;

make\_man(num\_chances);

characters[k] = s;

k++;

print\_line(characters, word, k);

}

}

else

{

cout<<" Incorrect! "<<endl;

num\_chances++;

num\_guess--;

make\_man(num\_chances);

characters[k] = letter;

k++;

print\_line(characters, word, k);

int AI = rand\_int(0, 26) + 1;

char s = alphabet[AI];

cout<<endl;

cout<<" The AI guessed " << s << " " <<endl;

if(linear\_search(word, s))

{

cout<<" Correct! "<<endl;

make\_man(num\_chances);

characters[k] = s;

k++;

print\_line(characters, word, k);

if(win(characters, word, k))

{

int x;

cout<<" Sorry the AI has won " <<endl;

x = index\_search(Dictionary, word);

cout<<" The Word " << word << " was index at " << x <<endl;

exit(0);

}

}

else

{

cout<<" Incorrect! " <<endl;

make\_man(num\_chances);

characters[k] = s;

k++;

print\_line(characters, word, k);

}

}

}

}

cout <<"You have ran out of chances. YOU LOSE!" << endl;

}

return 0;

}

bool binary\_search(const vector<string>& sorted\_vec, string key)

{

size\_t mid, left = 0 ;

size\_t right = sorted\_vec.size(); // one position passed the right end

while (left < right) {

mid = left + (right - left)/2;

if (key > sorted\_vec[mid]){

left = mid+1;

}

else if (key < sorted\_vec[mid]){

right = mid;

}

else

{

return true;

}

}

return false;

}

int rand\_int(int a, int b)

{

return rand()%(b-a+1) + a;

}

bool linear\_search(string word, char letter)

{

int length = word.length();

for(int i = 0; i < length; i++)

{

if(word[i] == letter)

{

return true;

}

}

return false;

}

void make\_man(int num)

{

switch (num)

{

case 7:

cout<< "\n ================|\n // |\n || |\n || |\n || \_^\_\n || / ..\\\n || [ \_ ]\n || \\\_\_\_/\n ||\n || || - \*Snap!!!\*\n || /||\\\n || //||\\\\\n || // || \\\\\n || \* || \*\n || //\\\\\n || // \\\\\n /\\ // \\\\\n//\\\\ \*\*\* \*\*\*\n/||\\\\\n\_||\_\\\\\n";

break;

case 6:

cout<< "\n ================|\n // |\n || |\n || |\n || \_^\_\n || / ..\\\n || [ \_ ]\n || \\\_\_\_/\n || ||\n || /||\\\n || //||\\\\\n || // || \\\\\n || \* || \*\n || //\n || //\n || //\n /\\ \*\*\*\n//\\\\ \n/||\\\\ \n\_||\_\\\\\n";

break;

case 5:

cout<< "\n ================|\n // |\n || |\n || |\n || \_^\_\n || / ..\\\n || [ \_ ]\n || \\\_\_\_/\n || ||\n || /||\\\n || //||\\\\\n || // || \\\\\n || \* || \*\n ||\n ||\n ||\n /\\\n//\\\\\n/||\\\\\n\_||\_\\\\\n";

break;

case 4:

cout<< "\n ================|\n // |\n || |\n || |\n || \_^\_\n || / ..\\\n || [ \_ ]\n || \\\_\_\_/\n || ||\n || /||\n || //||\n || // ||\n || \* ||\n ||\n ||\n ||\n /\\\n//\\\\\n/||\\\\\n\_||\_\\\\\n";

break;

case 3:

cout<< "\n ================|\n // |\n || |\n || |\n || \_^\_\n || / ..\\\n || [ \_ ]\n || \\\_\_\_/\n || ||\n || ||\n || ||\n || ||\n || ||\n ||\n ||\n ||\n /\\\n//\\\\\n/||\\\\\n\_||\_\\\\\n";

break;

case 2:

cout<< "\n ================|\n // |\n || |\n || |\n || \_^\_\n || / ..\\\n || [ \_ ]\n || \\\_\_\_/\n || ||\n ||\n ||\n ||\n ||\n ||\n ||\n ||\n /\\\n//\\\\\n/||\\\\\n\_||\_\\\\\n";

break;

case 1:

cout<< "\n ================|\n // |\n || |\n || |\n || \_^\_\n || / ..\\\n || [ \_ ]\n || \\\_\_\_/\n ||\n ||\n ||\n ||\n ||\n ||\n ||\n ||\n /\\\n//\\\\\n/||\\\\\n\_||\_\\\\\n";

break;

case 0:

cout<<"\n ================\n //\n ||\n ||\n ||\n ||\n ||\n ||\n ||\n ||\n ||\n ||\n ||\n ||\n ||\n ||\n /\\\n//\\\\\n/||\\\\\n\_||\_\\\\\n";

break;

}

}

bool win(char characters[], string word, int length)

{

bool found = true;

for(int i = 0; i<word.size();i++)

{

found = false;

for(int j = 0; j<length;j++)

{

if(characters[j] == word.at(i))

found = true;

}

if (!found)

return false;

}

return true;

}

void print\_line(char characters[], string word, int k)

{

char letters[word.size()];

for (int i = 0; i <word.size(); i++)

{

letters[i] = '-';

for(int j = 0; j < k; j++)

{

if(word.at(i) == characters[j])

letters[i] = characters[j];

}

}

for(int i = 0; i <word.size(); i++)

cout << letters[i];

cout << endl;

return;

}

int index\_search(const vector<string>& sorted\_vec, string key)

{

size\_t mid, left = 0 ;

size\_t right = sorted\_vec.size();

while (left < right) {

mid = left + (right - left)/2;

if (key > sorted\_vec[mid]){

left = mid+1;

}

else if (key < sorted\_vec[mid]){

right = mid;

}

else

{

return mid;

}

}

return 0;

}

void selection\_sort(vector<string>& vector)

{

int index;

int length = vector.size();

for(int i = 0; i < length; i++)

{

index = i;;

for(int j = i; j < length; j++)

{

if(vector[j] < vector[index])

index = j;

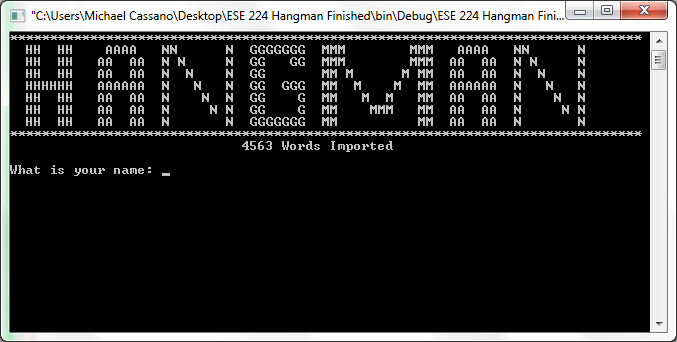
}

swap(vector[i], vector[index]);

}

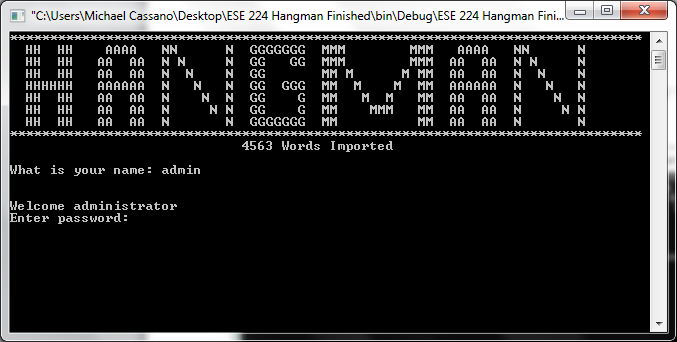
}

**Output Examples**

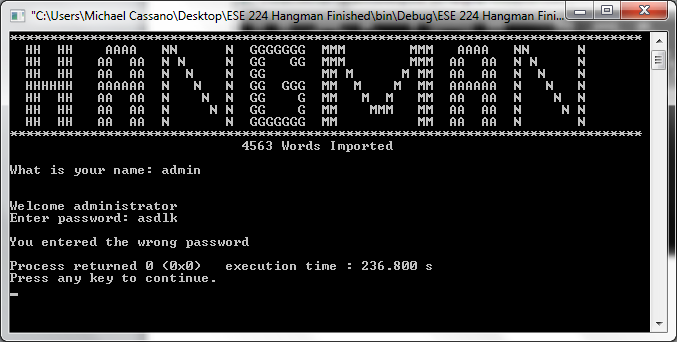
Section 1: When the program is first run, this screen is displayed, prompting the user to enter their name.

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

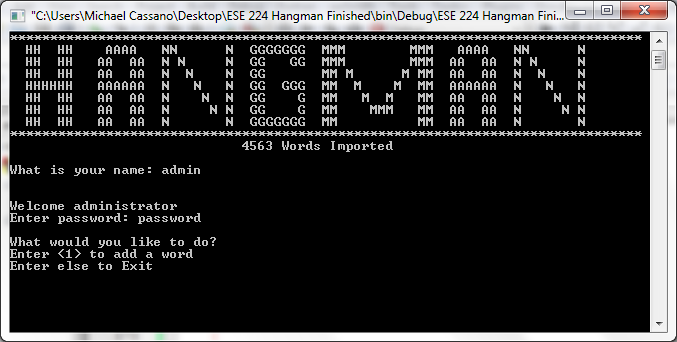
Section 2: Case 1: User enters ‘admin’ as their name.

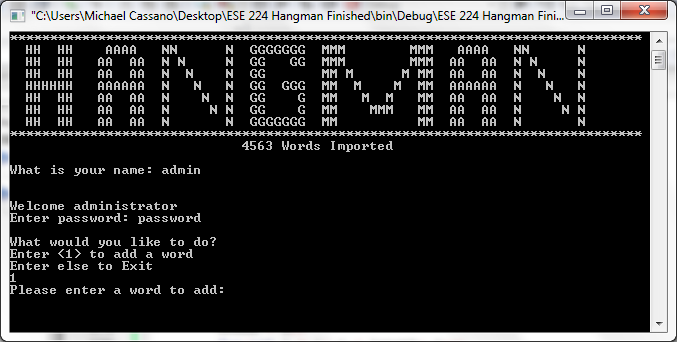


Section 2: Case 1.a: User enters incorrect password.

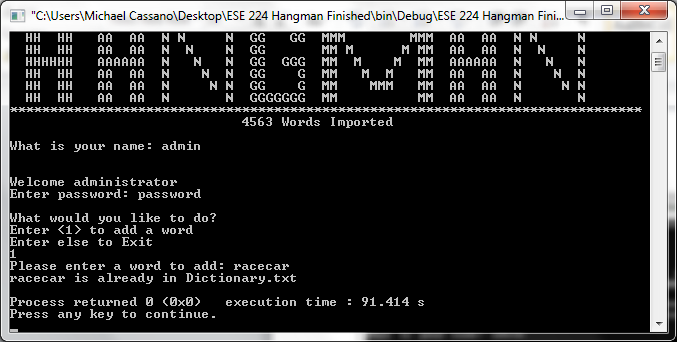


Section 2: Case 1.b: User enters the correct password, which is ‘password’, then presses 1.

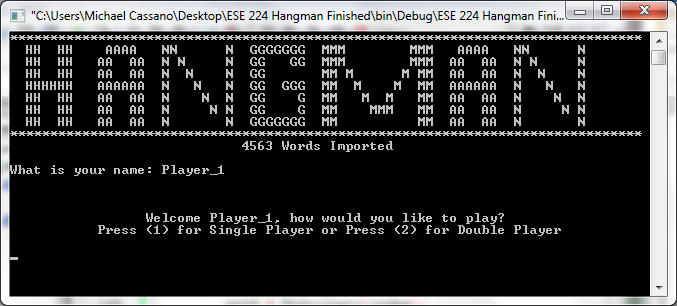




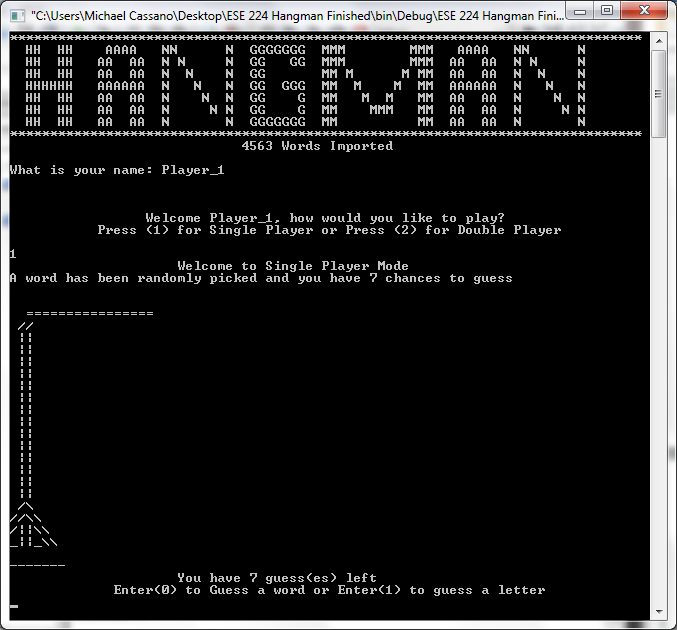
At this point, the user may enter a new word to be added to the word list. If a new word is added, our sorting function places the new word into the list alphabetically. If the word is in the list already, the following message is displayed.



Section 2: Case 2: User enters their name.

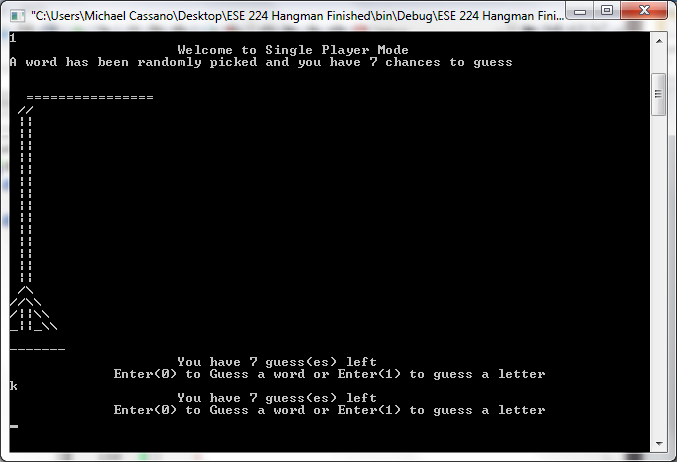


Section 3: Single Player Mode: If the user enters (1), the program initiates single player mode.

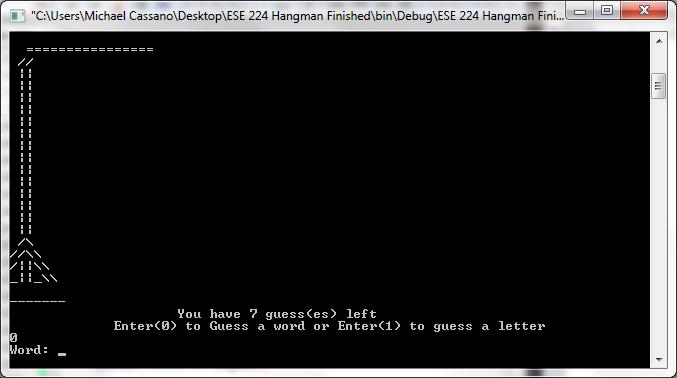


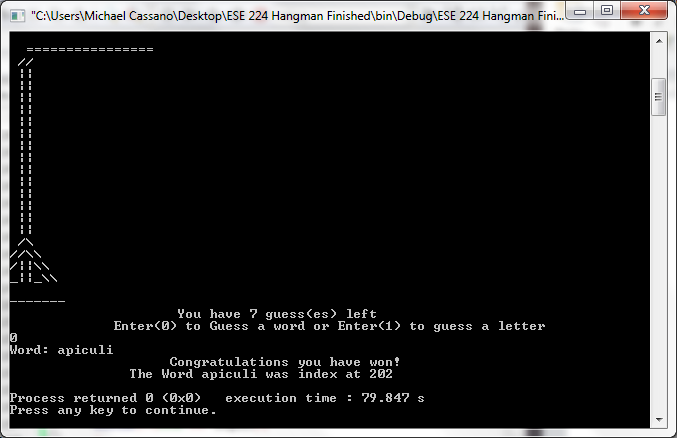
From this point on, the user has the option of entering a single letter, or an entire word.

Section 3: Case 1: Invalid entry: If the user enters anything besides 0 or 1 at this stage, the message displaying the number of guesses, as well as the prompt, repeats.

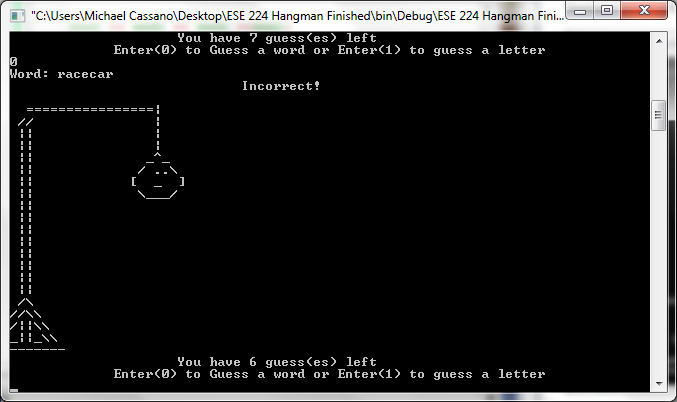


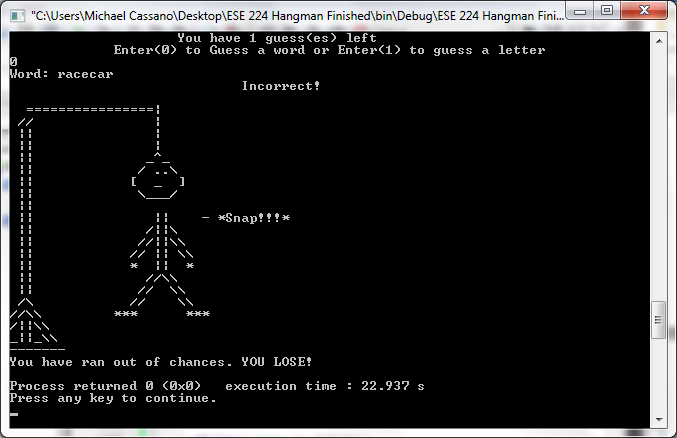
Section 3: Case 2: The user enters 0 to guess a word.

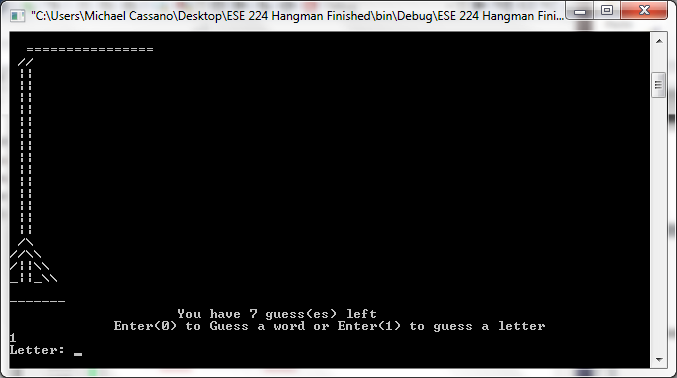


Section 3: Case 2.a: User correctly guesses the word. 

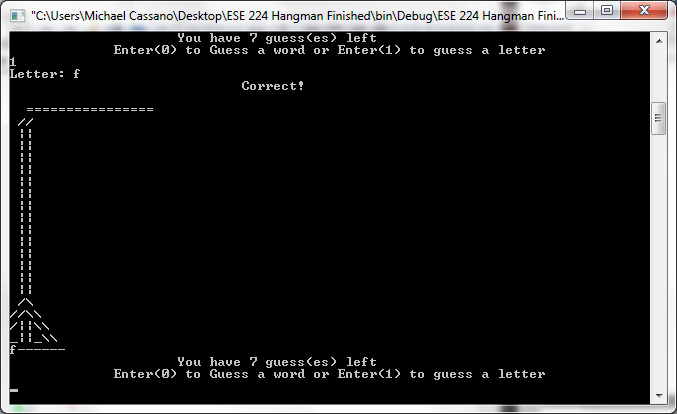
Section 3: Case 2.b: User incorrectly guesses word.

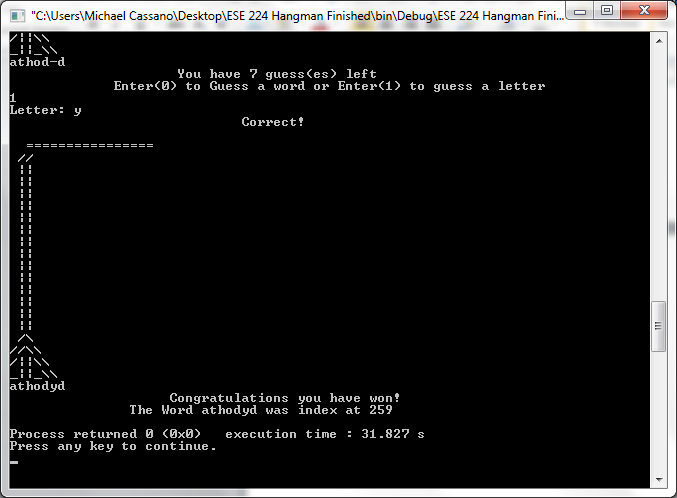


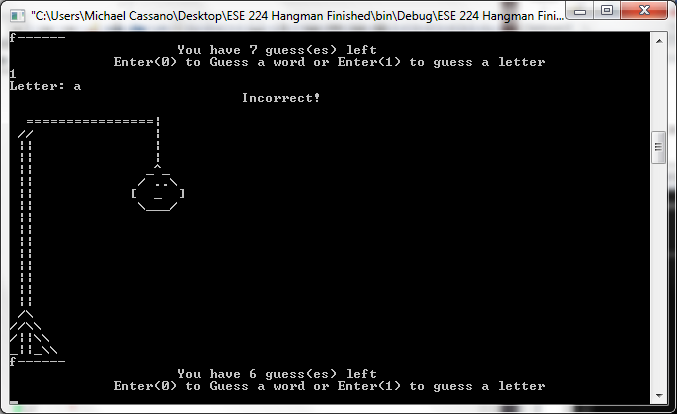
Section 3: Case 2.b.1: User incorrectly guesses the word on their last attempt. 

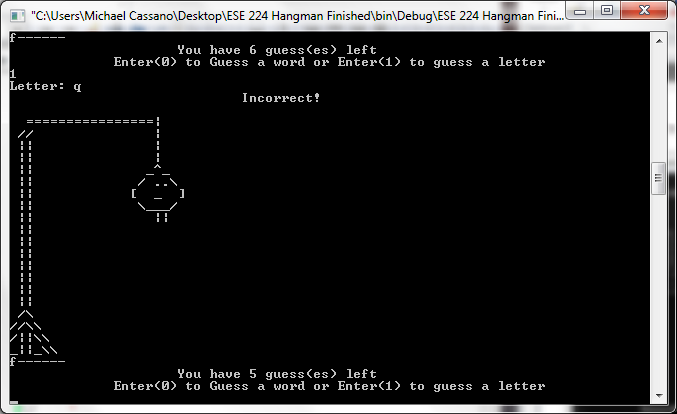
Section 3: Case 3: User enters 1 to guess a letter. 

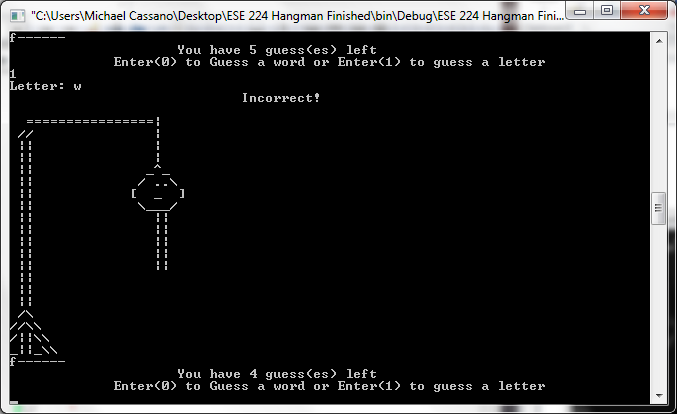
Section 3: Case 3.a: User correctly guesses a letter.

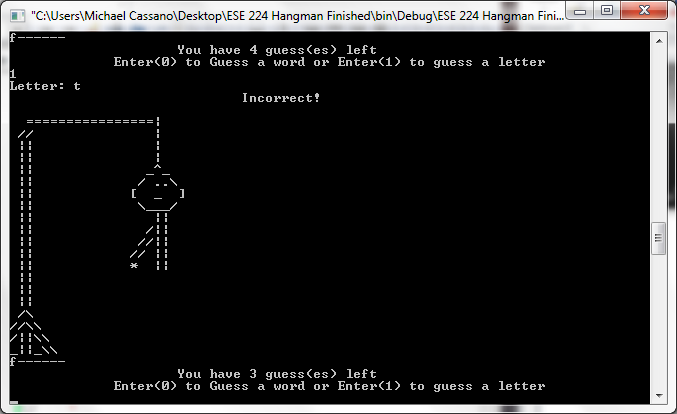


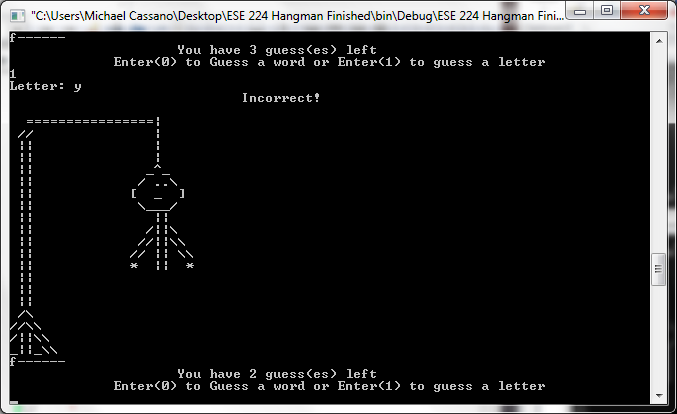
Section 3: Case 3.a.1: User correctly guesses last letter. 

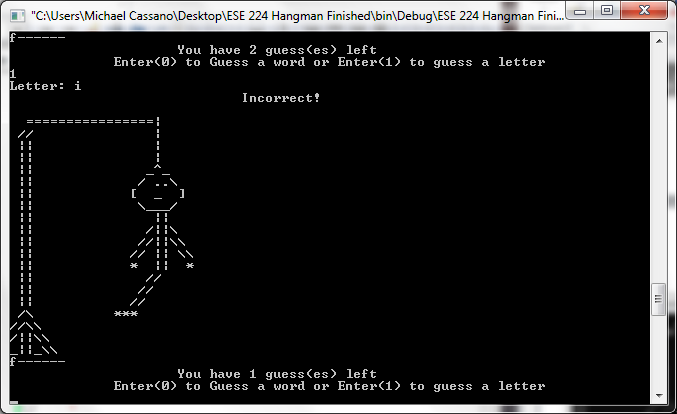
Section 3: Case 3.b: User incorrectly guesses a letter (first six incorrect guesses are shown). 

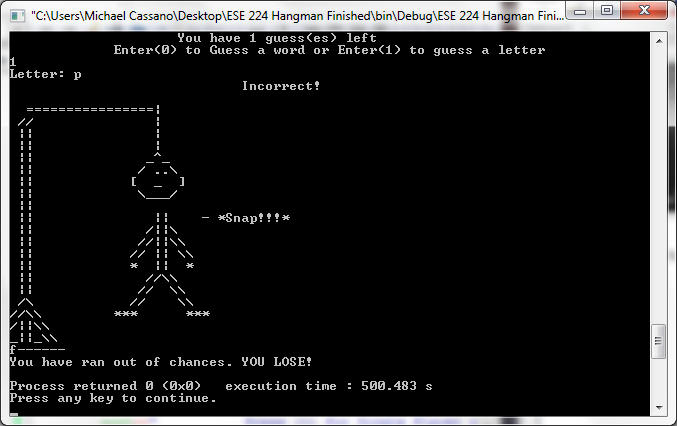


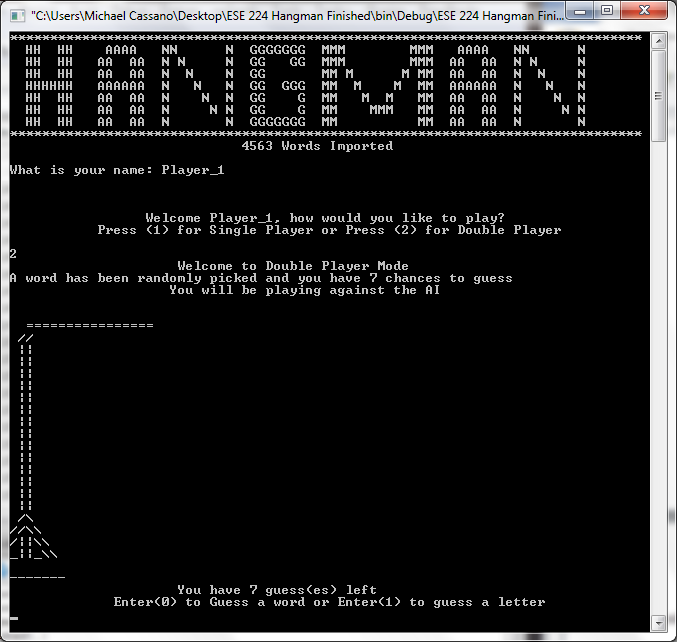






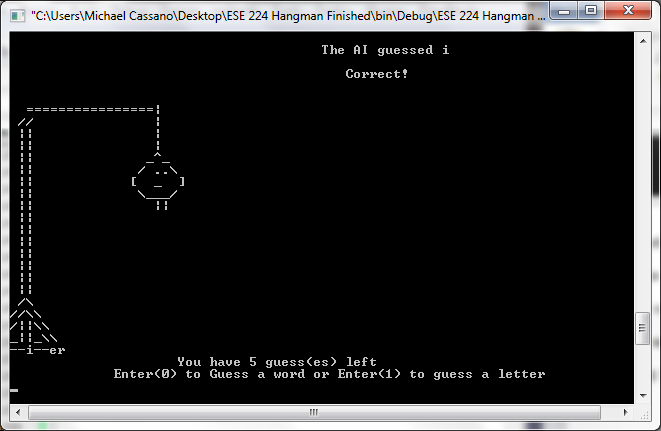


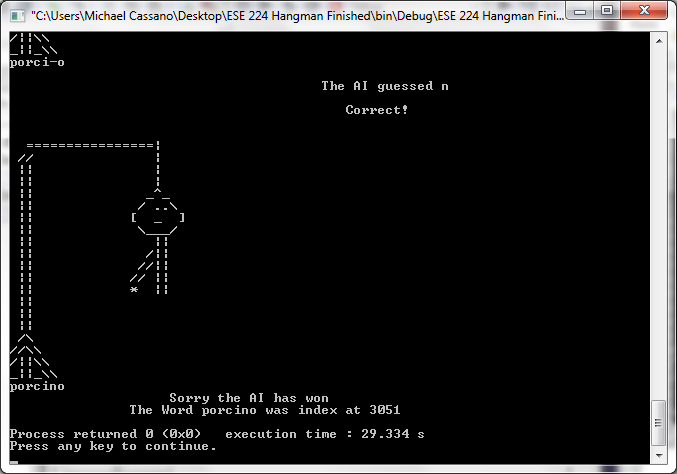
Section 3: Case 3.b.1: User incorrectly guesses a letter on their last attempt. 

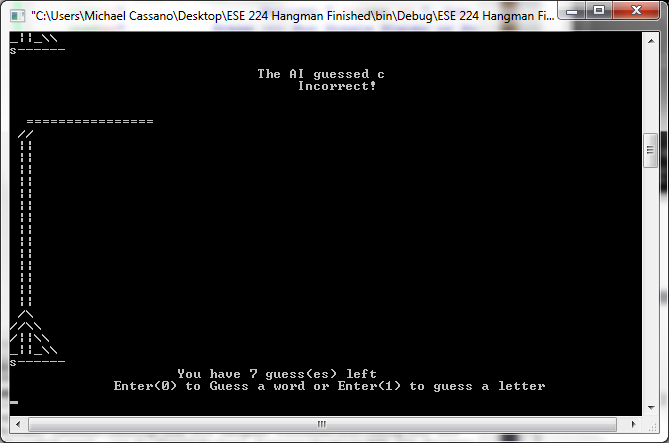
Section 4: Double Player Mode: If the user enters 2 when they are asked to choose a game mode, they have selected to play double player mode. 

During double player mode, the user and the AI go back and forth trying to guess a correct answer. The AI can only guess letters, while the user may still enter either a letter or a word as a guess. The results of correctly or incorrectly guessing a letter or word, whether it is for the win or not, is the same as it is in single player mode. The outputs will look the same, except the AI will be guessing in between the guesses of the user.

Section 4: Case 1: User input is arbitrary, AI correctly guesses a letter.



Section 4: Case 1.a: User input is arbitrary, AI correctly guesses final letter. 

Section 4: Case 2: User input is arbitrary, AI incorrectly guesses a letter. 

Section 4: Case 3: User runs out of guesses. 