Jeremy Barzas
10/14/2015
ADGP105

A. Requirements Documentation

1. Name: Wumpus World Redo

Problem Statement:

The player must navigate their character around a 4x4 grid and get the gold and then return to the starting point to win.

Problem Specification:

If the player's character moves to a cell that has a Wumpus, Pit, or a cell that isn't on the grid your character dies and you lose. You can kill the Wumpus with an arrow so you can pass on the cell it is on. You can use your remote to scan the area and tell you where things are located near your location.

B. Input Methods

1. Description: The user is prompted to give an input with a "cout" message then the input is store into the correct variable using "cin".

2. Type: A char variable named input is used to store the user's input. The game accepts the inputs of: "w" - move north, "s" - move south, "a" - move west, "d" - move east, "r" - use scanner, "f" - shoot arrow, "q" - save game file, "e" - load game file.

C. Output Items

1. Description: The program outputs information to the user with "cout".

2. Type: Strings of characters and integers are output types used by the program.

D. User Interface Information

1. Description: The player is displayed a message explaining how to input a command and what each input does. Displays a message telling the player, "You enter the cave of the Wumpus monsters in search of gold...". Each time the player moves it displays the direction they moved and their current x and y position.

E. Design Documentation

1. System Architecture
   Description: The game includes a Cell class, a Player class, a Gold class, a Pit Class, a Wumpus class, and a position struct.

2. Object Information

I. Name: position

Type: Struct

Description: This struct gives an x and y integer variable to be used as points on a grid for other objects to have location.

Attributes:
integer - x

integer - y.

II. Name: Player

Type: Class

Description: A class for the player's character.

Attributes:
struct - position

boolean - alive

boolean - gold

constructor function - Player()

III. Name: Cell

Type: Class

Description: A class for the grid cells.

Attributes:
struct - position
constructor function - Cell()

IV. Name: Wumpus

Type: Class

Description: A class for Wumpus creature.

Attributes:
struct - position

boolean - alive

constructor function - Wumpus(struct position)

V. Name: Pit

Type: Class

Description: A class for the Pit hazards.

Attributes:
struct - position

constructor function - Pit(struct position)

VI. Name: Gold

Type: Class

Description: A class for the Gold.

Attributes:
struct - position

boolean - onGround

constructor function - Gold()

VII. Name: Arrow

Type: Class

Description: A class for the Arrow.

Attributes:
struct - position

constructor function - Arrow()

boolean function - shootArrow(class Arrow, class Player, class Wumpus)

VIII. Name: Scanner

Type: Class

Description: A class for the Scanner.

Attributes:
integer - charge

constructor function - Scanner()

void function - useScanner(class Arrow, class Player, class Wumpus)

F. Implementation Documentation

1. Source Code

I. Name: Objects_Header.h

File Type: Header

Code:

```
#ifndef CLASSES_H
#define CLASSES_H

#include <iostream>
#include <fstream>
#include "time.h"

using namespace std;

// creates a struct that contains two integers for other classes to use as a position for location.
struct position
{
```

```
        int x;
        int y;
};

// creates a class that is a grid cell that contains a position struct for location,
// contains a function to create cells to build the grid with.
class Cell
{
public:
        position location;
        Cell();
};

// creates a class for the players character that contains a position struct for location,
// contains a function to create a player character,
// contains two bool variables to hold the information on wether the player is alive or has gold.
class Player
{
public:
        position location;
        bool alive;
        bool gold;
        Player();
};

// creates a class for the Gold that contains a position struct for location.
// contains a function to create the gold.
class Gold
{
public:
        position location;
        bool onGround = true;
        Gold();
};

// creates a class for the Wumpus that contains a position struct for location.
// contains a function to create the Wumpus.
class Wumpus
{
public:
        position location;
        bool alive;
        Wumpus(position);
```

```
};

// creates a class for the Wumpus that contains a position struct for location.
// contains a function to create the Wumpus and give it position.
class Pit
{
public:
	position location;
	Pit(position);
};

// creates a class for the Arrow that contains a position struct for location.
// contains a function to create an Arrow.
// contains a function to shoot an arrow.
class Arrow
{
public:
	position location;
	Arrow();
	bool shootArrow(Arrow, Player, Wumpus);
};

// creates a class for the Scanner that contains an integer to hold the amount of charges.
// contains a function to create a Scanner.
// contains a function to use the scanner.
class Scanner
{
public:
	int charge;
	Scanner();
	void useScanner(Scanner, Player, Pit[], Wumpus, Gold);
};

#endif
```

II. Name: Class_Function_Definitions.cpp

File Type: Header

Code:

```
#include"Objects_Header.h"
```

```
// constructor function for the Cell class.
Cell::Cell()
{
}

/* constructor function for the Player class.
defines the Players position on creation as being at (0, 0) to give it a fixed start location.
defines the Player on creation as alive.
defines the Player on creation as not having the gold.
*/
Player::Player()
{
	location = { 0,0 };
	alive = true;
	gold = false;
}

// constructor function for the Gold class.
// defines the Golds position on creation as being at (3, 1) to give a fixed start location.
Gold::Gold()
{
	location = { 3,1 };
	onGround = true;
}

// constructor function for the Wumpus class.
// defines the Wumpus position on creation as being at (2, 1) to give a fixed start location.
Wumpus::Wumpus(position p)
{
	location = p;
	alive = true;
}

// constructor function for the Pit class.
// function takes in the argument of a position struct to be bale to make multiple pits at different
locations.
Pit::Pit(position p)
{
	location = p;
}

// constructor function for the Arrow class.
```

```
Arrow::Arrow()
{
	location = { 0, 0 };
}

// constructor function for the Scannerclass.
Scanner::Scanner()
{
	charge = 3;
}

void Scanner::useScanner(Scanner s, Player c, Pit p[], Wumpus w, Gold g)
{
	if (s.charge >= 1)
	{
		cout << "Your X, Y position is: " << c.location.x << ", " << c.location.y << endl;

		// loops through the array of Pit class instances.
		// displays the direction of the Pit from the Player with a warning.
		for (int i = 0; i < sizeof(p); i++)
		{
			if ((c.location.x + 1 == p[i].location.x) && (c.location.y == p[i].location.y))
			{

				cout << "The scanner shows a big elevation drop to the East." <<
endl;
				cout << "" << endl;
			}

			else if ((c.location.x - 1 == p[i].location.x) && (c.location.y ==
p[i].location.y))
			{
				cout << "The scanner shows a big elevation drop to your West."
<< endl;
				cout << "" << endl;
			}

			else if ((c.location.x == p[i].location.x) && (c.location.y + 1 ==
p[i].location.y))
			{
				cout << "The scanner shows a big elevation drop to the North." <<
endl;
				cout << "" << endl;
```

```
                        }

                        else if ((c.location.x == p[i].location.x) && (c.location.y - 1 ==
p[i].location.y))
                        {
                                cout << "The scanner shows a big elevation drop to the South." <<
endl;
                                cout << "" << endl;
                        }

                        else if ((c.location.x + 1 == p[i].location.x) && (c.location.y + 1 ==
p[i].location.y))
                        {
                                cout << "The scanner shows a big elevation drop to the
Northeast." << endl;
                                cout << "" << endl;
                        }

                        else if ((c.location.x - 1 == p[i].location.x) && (c.location.y + 1 ==
p[i].location.y))
                        {
                                cout << "The scanner shows a big elevation drop to the
Northwest." << endl;
                                cout << "" << endl;
                        }

                        else if ((c.location.x + 1 == p[i].location.x) && (c.location.y + 1 ==
p[i].location.y))
                        {
                                cout << "The scanner shows a big elevation drop to the
Southeast." << endl;
                                cout << "" << endl;
                        }

                        else if ((c.location.x - 1 == p[i].location.x) && (c.location.y + 1 ==
p[i].location.y))
                        {
                                cout << "The scanner shows a big elevation drop to the
Southwest." << endl;
                                cout << "" << endl;
                        }
                }
```

```
// displays the direction of the Wumpus from the Player with a warning.
if ((c.location.x + 1 == w.location.x) && (c.location.y == w.location.y))
{
	cout << "The scanner detects a large life form to the East..." << endl;
	cout << "" << endl;
}

else if ((c.location.x - 1 == w.location.x) && (c.location.y == w.location.y))
{
	cout << "The scanner detects a large life form to the West..." << endl;
	cout << "" << endl;
}

else if ((c.location.x == w.location.x) && (c.location.y + 1 == w.location.y))
{
	cout << "The scanner detects a large life form to the North..." << endl;
	cout << "" << endl;
}

else if ((c.location.x == w.location.x) && (c.location.y - 1 == w.location.y))
{
	cout << "The scanner detects a large life form to the South" << endl;
	cout << "" << endl;
}

else if ((c.location.x + 1 == w.location.x) && (c.location.y + 1 == w.location.y))
{
	cout << "The scanner detects a large life form to the Northeast" << endl;
	cout << "" << endl;
}

else if ((c.location.x - 1 == w.location.x) && (c.location.y + 1 == w.location.y))
{
	cout << "The scanner detects a large life form to the Northwest" << endl;
	cout << "" << endl;
}

else if ((c.location.x + 1 == w.location.x) && (c.location.y + 1 == w.location.y))
{
	cout << "The scanner detects a large life form to the Southeast" << endl;
	cout << "" << endl;
}
```

```
		else if ((c.location.x - 1 == w.location.x) && (c.location.y + 1 == w.location.y))
		{
			cout << "The scanner detects a large life form to the Southwest" << endl;
			cout << "" << endl;
		}

		// displays the direction of the Wumpus from the Gold with a message.
		if ((c.location.x + 1 == g.location.x) && (c.location.y == g.location.y))
		{
			cout << "The scanner detects a large deposit of metals to the East..." <<
endl;
			cout << "" << endl;
		}

		else if ((c.location.x - 1 == g.location.x) && (c.location.y == g.location.y))
		{
			cout << "The scanner detects a large deposit of metals to the West..." <<
endl;
			cout << "" << endl;
		}

		else if ((c.location.x == g.location.x) && (c.location.y + 1 == g.location.y))
		{
			cout << "The scanner detects a large deposit of metals to the North..." <<
endl;
			cout << "" << endl;
		}

		else if ((c.location.x == g.location.x) && (c.location.y - 1 == g.location.y))
		{
			cout << "The scanner detects a large deposit of metals to the South" <<
endl;
			cout << "" << endl;
		}

		else if ((c.location.x + 1 == g.location.x) && (c.location.y + 1 == g.location.y))
		{
			cout << "The scanner detects a large deposit of metals to the Northeast"
<< endl;
			cout << "" << endl;
		}

		else if ((c.location.x - 1 == g.location.x) && (c.location.y + 1 == g.location.y))
```

```
		{
			cout << "The scanner detects a large deposit of metals to the Northwest"
<< endl;
			cout << "" << endl;
		}

		else if ((c.location.x + 1 == g.location.x) && (c.location.y + 1 == g.location.y))
		{
			cout << "The scanner detects a large deposit of metals to the Southeast"
<< endl;
			cout << "" << endl;
		}

		else if ((c.location.x - 1 == g.location.x) && (c.location.y + 1 == g.location.y))
		{
			cout << "The scanner detects a large deposit of metals to the Southwest"
<< endl;
			cout << "" << endl;
		}
	}

	else if (s.charge <= 0)
	{
		cout << "The scanner's battery is drained." << endl;
	}
}

bool Arrow::shootArrow(Arrow a, Player c, Wumpus w)
{
	char input;
	a.location.x = c.location.x, a.location.y = c.location.y;

	cout << "You draw your bow..." << endl;
	cin >> input;

	switch (input)
	{
	case 'w':
		cout << "You shot an arrow to the North..." << endl;
		cout << "" << endl;
		a.location.x, a.location.y += 1;
		break;
	case 's':
```

```
                cout << "You shot an arrow to the South..." << endl;
                cout << "" << endl;
                a.location.x, a.location.y -= 1;
                break;

        case 'a':
                cout << "You shot an arrow to the South..." << endl;
                cout << "" << endl;
                a.location.x, a.location.y -= 1;
                break;

        case 'd':
                cout << "You shot an arrow to the South..." << endl;
                cout << "" << endl;
                a.location.x, a.location.y -= 1;
                break;
        }

        if ((a.location.x == w.location.x) && (a.location.y == w.location.y) && (w.alive == true))
        {
                return true;
        }

        else
        {
                return false;
        }
}
```

III. Name: Main_Game.cpp

Type: cpp

Code:

```
#include "Objects_Header.h"

int RNG(int mod)
{
        int temp = rand() % mod;

        return temp;
}
```

```
// creates a function that creates a grid and returns void.
// takes in the arguments of an interger and an array of instances of the Cell class.
 void createGrid(int size, Cell g[])
{
	// defines the "x" location of the "i" position in the array of Cell instances as the value of "i".
	// defines the "y" location of the "j" position in the array of Cell instances as the value of "j".
	for (int i = 0; i < size; i++)
	{
		for (int j = 0; j < size; j++)
		{
			g[j].location.y = j;
			g[i].location.x = i;
		}
	}
}

void main()
{
	srand(time(NULL));

	// dynamic memory allocation
	// creates an array of instances of the Cell class with a dynamic size.
	// this is setup in a way so that you could ask the user for an array size and it would allocate the required amout of memory.
	Cell* cell;
	cell = new Cell[16];

	// calls the function that creates the grid to be used as the game map defined as a 4 x 4 grid stored inside the dynamic cell array.
	createGrid(4, cell);

	// creates a varible to be used as a text file.
	fstream gameSave;

	// makes a bool variable used to define the default win condition as false until changed when certain criteria is met.
	bool winCondition = false;

	// creates an instance of the Player class named player defined as the constructor function.
```

```
	Player player = Player();

	// creates an instance of the Scanner class name scanner defined as the constructor function.
	Scanner scanner = Scanner();

	// creates an instance of the Gold class named gold defined as the constructor function.
	Gold gold = Gold();

	// creates an instance of the Wumpus class named wumpus defined as the constructor function.
	Wumpus wumpus = Wumpus({ RNG(4),RNG(4) });

	cout << "Wumpus location: " << wumpus.location.x << " = x, " << wumpus.location.y << " = y" << endl;
	cout << "\n";

	// creates an array of instances of the Pit class.
	Pit pits[4] = { Pit({1,1}), Pit({ 1,3 }), Pit({ 3,0 }), Pit({ 3,3 }) };

	// creates an instance of the Arrow class name arrow defined as the constructor function.
	Arrow arrow = Arrow();

	// displays instructions for the user to the console.
	cout << "Type your input then press enter to do actions" << endl;
	cout << "Use 'w','a','s','d' as the arrow keys to move." << endl;
	cout << "Use 'f' to draw your bow then 'w','a','s','d' to shoot in a direction ." << endl;
	cout << "Use 'r' to check your scanner." << endl;
	cout << "Use 'q' to save, and 'e' to load a save" << endl;
	cout << "\n";

	cout << "You enter the cave of the Wumpus monsters in search of gold..." << endl;
	cout << "\n";

	cout << "Which direction do you want to go?" << endl;
	cout << "\n";

	// starts the game loop.
	do
	{
		// makes a char variable used to store user input to be used in the movement switch case statements.
		char input;
```

```
		// takes in the users input to be check against the switch/case statments.
		cin >> input;
		cout << "\n";

		// tells the user that the wrong input was entered and lists the correct inputs if the wrong input was entered.
		if (!(input == 'w' || input == 'a' || input == 's' || input == 'd' || input == 'q' || input == 'e' || input == 'f' || input == 'r'))
		{
			cout << "That is not a valid input..." << endl;
			cout << "Use 'w','a','s','d' as the arrow keys to move." << endl;
			cout << "Use 'f' to draw your bow then 'w','a','s','d' to shoot in a direction ." << endl;
			cout << "Use 'r' to check your scanner, you only have 2." << endl;
			cout << "Use 'q' to save, and 'e' to load a save." << endl;
			cout << "\n";
		}

		// creates a switch/case statment thats accepts the character variable "input".
		switch (input)
		{
			// adds 1 to the players "y" cooridinate if 'w' was the input.
		case 'w':
			player.location.x, player.location.y += 1;
			cout << "You moved to the North..." << endl;
			cout << "Your X, Y position is: " << player.location.x << ", " << player.location.y << endl;
			cout << "\n";
				break;

			// subtracts 1 from the players "y" cooridinate if 's' was the input.
		case 's':
			player.location.x, player.location.y -= 1;
			cout << "You moved to the South..." << endl;
			cout << "Your X, Y position is: " << player.location.x << ", " << player.location.y << endl;
			cout << "\n";
				break;

			// subtracts 1 to the players "x" cooridinate if 'a' was the input.
		case 'a':
			player.location.x -= 1, player.location.y;
```

```
                cout << "You moved to the West..." << endl;
                cout << "Your X, Y position is: " << player.location.x << ", " <<
player.location.y << endl;
                cout << "\n";
                        break;

                // adds 1 to the players "x" cooridinate if 'd' was the input.
        case 'd':
                player.location.x += 1, player.location.y;
                cout << "You moved to the East..." << endl;
                cout << "Your X, Y position is: " << player.location.x << ", " <<
player.location.y << endl;
                cout << "\n";
                        break;

                // calls the useScanner function to scan the area if 'r' was the input.
        case 'r':
                scanner.useScanner(scanner, player, pits, wumpus, gold);
                scanner.charge -= 1;
                break;

                // calls the shootArrow function to shoot an arrow in the direction of the
user input.
        case 'f':
                if (arrow.shootArrow(arrow, player, wumpus) == true)
                {
                        wumpus.alive = false;
                        cout << "You have killed the Wumpus monster!" << endl;
                        cout << "\n";
                }

                else
                {
                        cout << "You didnt hit anything..." << endl;
                        cout << "\n";
                }
                        break;

                // saves the players data to a file if 'q' was the input.
        case 'q':

                gameSave.open("gameSave.txt", ios_base::out);
```

```
                        gameSave << player.location.x << "\n" <<  player.location.y << "\n" <<
player.gold << "\n" << wumpus.alive;

                        gameSave.close();

                        cout << "Game has been saved." << endl;
                        cout << "\n";
                                break;

                        // loads the players data from a file if 'e' was the input.
                case 'e':

                        gameSave.open("gameSave.txt", ios_base::in);

                        gameSave >> player.location.x >> player.location.y >> player.gold >>
wumpus.alive;

                        gameSave.close();

                        cout << "Game has been loaded." << endl;
                        cout << "\n";
                        cout << "Your X, Y position is: " << player.location.x << ", " <<
player.location.y << endl;
                        cout << "\n";
                                break;

                default:
                                break;
                }

                if ((player.location.x == pits[0].location.x) && (player.location.y ==
pits[0].location.y))
                {
                        player.alive = false;
                        cout << "You fall in a giant pit and are trapped forever..." << endl;
                        cout << "\n";
                }

                if ((player.location.x == pits[1].location.x) && (player.location.y ==
pits[1].location.y))
                {
                        player.alive = false;
                        cout << "You fall in a giant pit and are trapped forever..." << endl;
```

```
                cout << "\n";
        }

        if ((player.location.x == pits[2].location.x) && (player.location.y ==
pits[2].location.y))
        {
                player.alive = false;
                cout << "You fall in a giant pit and are trapped forever..." << endl;
                cout << "\n";
        }

        if ((player.location.x == pits[3].location.x) && (player.location.y ==
pits[3].location.y))
        {
                player.alive = false;
                cout << "You fall in a giant pit and are trapped forever..." << endl;
                cout << "\n";
        }

        // displays a message to the console that you have died if the Player's "x" or "y"
coordinate is not on the game map grid it sets the Alive bool of the Player to false.
        if ((player.location.x == -1) ||
        (player.location.x == 4) ||
        (player.location.y == -1) ||
        (player.location.y == 4))
        {
                player.alive = false;
                cout << "You fall of a ledge to your death..." << endl;
                cout << "\n";
        }

        // displays a message to the console that you have died if the players "x" and "y"
coordinates are equal to that of the Wumpus it sets the alive bool of the Player to false.
        if ((player.location.x == wumpus.location.x) && (player.location.y ==
wumpus.location.y) && (wumpus.alive == true))
        {
                player.alive = false;
                cout << "A Wumpus monster bites your head off.." << endl;
                cout << "\n";
        }

        // if the Player's "x" and "y" coordinates are equal to that of the Gold it sets the
Gold bool of the Player to true.
```

```
		// dislplays a message to the colse to tell the play that they have picked up the gold and must leave the cave.
		if ((player.location.x == gold.location.x) && (player.location.y == gold.location.y) && (gold.onGround == true))
		{
			player.gold = true;
			gold.onGround = false;
			cout << "You have found the gold, now you must escape the Wumpus cave" << endl;
			cout << "\n";
		}

		// if the players alive bool is false it sets the winCondition bool to false.
		if (player.alive == false)
		{
			winCondition = false;
		}

		// if the players "x" and "y" coordinates are equal to the start position (0,0) and the gold bool is true it sets the winCondition to true.
		if ((player.gold == true) && (player.location.x == 0) && (player.location.y == 0))
		{
			winCondition = true;
		}

		// while the winCondition bool is false and the players alive bool is true continue to execute this loop.
	} while ((winCondition == false) && (player.alive == true));

	//if the winCondtion bool is true and the players alive bool is true display to the console that they have won the game.
	if ((winCondition == true) && (player.alive == true))
	{
		cout << "\n";
		cout << "\n";
		cout << "Congradulations you escaped the cave with the gold!" << endl;
		cout << "\n";

		// clears the save game file if the winCondition bool is true.
		gameSave.open("gameSave.txt", ios_base::out);

		gameSave.close();
	}
```

```
	// deallocates the dynamic memory of the Cell array.
	delete[] cell;

	system("pause");
}
```