**\\*\*\* ===== C O M M A N D O ===== \*\*\* /**

**A PROGRAMMING PROJECT**

**Project No. 2**

**Submitted by:**

**Ezra P. Guinmapang**

**Professor:**

**Dr. Mark Lehr**

**Class:**

**CSC-5 (C++ Programming Logic)**

**Section:**

**42450**

**Date:**

**June 13, 2014**

**SECTION I - GENERAL DESCRIPTION AND OBJECTIVES**

The main objective of this project is to write a program that simulates and plays the game Commando. This game is based loosely on “Robots,” which is a Unix-based console game, along with elements of a Roguelike dungeon crawl game.

Written in C++, the game has in mind several key goals. First, this project is meant to demonstrate an understanding of functions, arrays, pointers, strings and vectors, and file I/O. A partial list will be provided, along with the appropriate line numbers and filenames, when necessary.

Also, this is meant to be a continuing project: this is but the first version. As such, this program is not complete, in that it is only the very basic framework of the game itself. When run, it will allow the user to move his or her character along the grid, respecting borders & “picking up” items along the way. The display should update the user’s character position correctly, and correctly display the appropriate tiles when picking up items. Also, the program generates “autonomous” enemies – these move around independently. A more detailed explanation of what the program’s current state is – what it can and cannot do – will be discussed in a later section.

**SECTION II – BASIC MECHANICS OF THE GAME**

The game itself is meant to be a combination of the game “Robots” and a Roguelike dungeon crawl game. Unlike dungeon crawl and RPG games, which employ hit points to determine whether a Player is alive, the winning conditions are taken from Robots: The game ends when the Robot enemies catch the Player – that is, when they reach a position directly adjacent to the Player’s position.

Gameplay is pretty straightforward: this is a turn-based game, meaning that the Player acts first, then the Enemies act, then the data is processed by the display. The game occurs in three phases:

1. Player Action (movement, item use, shoot, etc.)
2. Enemy Action (similar to Player action)
3. Display

The game automatically ends if a Player’s position and an Enemy’s position become

adjacent to each other.

**SECTION III – CAVEATS AND LIMITATIONS**

The most poignant caveat to this program is that it is an incomplete program in the sense that not all the features and actions have been programmed into the game. However, as this is meant to be a continuing project, future versions of this game should be expected to contain all the planned features, actions, and game concepts. From the standpoint of being able to demonstrate a working knowledge of programming concepts, though, I feel that this program, although merely a framework, is complete, in that it illustrates most, if not all of the concepts learned thus far, and it runs the way it should for its current iteration – as a framework, and not as a completely playable game.

**SECTION IV – PROGRAM DESIGN**

This section provides a partial list of programming concepts shown in this program, along with the filenames and lines that demonstrate them.

CONCEPTS:

1. File I/O - in main.cpp, lines 67 & 68:

The filenames are mapgen.dat and obstacles.dat. The first file generates the map. The second file generates the obstacles. The values read in from these files are passed into the constructor for the TileObj class, which then creates the various tile types used in the game.

1. Functions and Arrays:
2. 2D arrays and Dynamic Allocation – in main.cpp, lines 72-76 & lines 106-114:

The array type is **tile**, a user-defined type defined in the files **TileObj.h** and **TileObj.cpp**. This was implemented thus:

**tile\*\* grid;**

The 2D array is allocated dynamically and populated in the subsequent lines (78-83), using the keyword **new**.

1. Passing arrays into functions, Dynamic Allocation and the **delete** keyword – in main.cpp, lines 106-114:

The function **Destroy()** takes an array as an argument. The full function declaration is:

**Destroy(tile \*\*grid, int&ROW, int&COL)**

Since **grid** is an array, it is passed by pointer/reference here, thereby eliminating the need for a dummy variable or dummy array. The array is then deleted using the **delete** and **delete[ ]** keywords (lines 109, 111, and 113).

1. Pass-by-reference and Returning an array from a function – in main.cpp, lines 65-87:

The function is named **Populate()**, and returns the 2D array as a pointer(line 86). The function declaration is thus:

**tile \*\*Populate( int&ROW, int&COL)**

1. Pass-by-value and Returning primitive data types – in file **TileObj.cpp**, lines 115-119:

The member function **Match(int x\_coord, int y\_coord)** in the **TileObj** class takes 2 arguments of type **int**. The function call is actually found in the **Player** class, in the file **Actors.cpp**, within the member function **FindTile()**, line 62:

**if(grid[idx][ctr]->Match(x\_coord, y\_coord))**

Note: the dummy variables in the function declaration are NOT the same as the arguments being passed in, despite having the same name.

1. Pointers and Pointer addition – in file **Actors.cpp**, line 280:

This method is used to peek at the next tile to determine if it is a valid position for a Player or a Robot instance to occupy.

**tile nxt\_obj**

**nxt\_obj = \*(\*(grid+(posx+ch\_x))+(posy+ch\_y))**