TUCKER DANE WALKER
CS 162 COMPUTER SCIENCE II
FINAL PROJECT

REFLECTION DOCUMENT

DESIGN DESCRIPTION

The design for this project truly was a culmination of a number of the assignments that we had throughout the course. I implemented the matrices we used from the doodlebug/ant assignment; a search function to locate specific items in my player's backpack, polymorphism for the rooms and objects, and each room was essentially a node that pointed to other rooms like itself.

Below is a list of my initial design. I included rough descriptions of what each class was meant to achieve and the functions that support them

Class Name	**Space – *an abstract class for rooms that contain objects***
Variables (protected)	*Int rowSize – number of rows* *Int colSize - number of columns* *Space *ptr[4]- pointers to other rooms, may be set to NULL* *Object *** space – dynamically allocated 2d matrix of pointers to objects* *Player *player – a player object* *Door (left, right, top, bottom) – doors*
Functions (public)	*get and set functions for the above variables* *printSpace() – prints the room* *movePlayer()- controls moving the player around within the room* *enterDoorWay () – handles the player moving into another space* *setSpace() – pure virtual function setting up space;* *setObjectPos(Object) – sets up an object in a space, depending on the space;*

Class Name	**Room – *a 9x9 space that contains a stone***
Variables (private)	*All variables from Space* *Stone *s – Room spaces contain stones*
Functions (public)	*Same as Space, initializes to its own size written above and contains a stone*

Class Name	**GreatRoom – *a 19x19 space that contains switches***
Variables (private)	*Switch s[x] – a number of switches*
Functions (public)	*setSwitches – sets up the switches in a greatroom* *getSwitch – gets the switches in a greatroom*

Class Name	**vHallWay and hHallway – *a 5x9 or 9x5 space that contains aarows***
Variables (private)	*Aarow a[x] – a number of aarows*
Functions (public)	*Same as Space, initializes to its own size written above and contains aarows*

Class Name	**Object – an abstract class for objects contained in a room**

Variables (protected)	*rowPos – the row position of the object* *colPos – the column position of the object* *weight – the weight of the object* *symbol – the symbol associated with the object to print to screen* *type – the type of object* *name – the name of the object*
Functions (public)	*Get/set functions for all listed above*

Class Name	**Player – an object that you directly control, the game revolves around it**
Variables (private)	*currentRoom – tracks the player's current room location* *Object *backpack[x] – contains x number of objects from the rooms* *maxCarry – the max weight the player can carry* *int numItems – a tracker for how many items the player currently has*
Functions (public)	*Get/set functions for those listed* *addObject( ) – handles picking up objects and putting them into the backpack* *dropObject() – handles dropping items out of the backpack, if possible*

Class Name	**Door – an immovable object that opens and closes gates to other rooms**
Variables (private)	*bool open – tells if the door is open or not*
Functions (public)	*setOpen(bool 0) – sets the door to open or closed* *bool getOpen() – returns the doors status*

Class Name	**Stone – a moveable object that corresponds to a switch**
Variables (private)	*String color – the color of the stone*
Functions (public)	*Get/set functions*

Class Name	**Switch – an immovable object that corresponds to a stone**
Variables (private)	*String color – the color of the switch (corresponds to a stone)* *Tripped – true if the switch has been tripped, false if not*
Functions (public)	*Get/set functions for those listed*

Class Name	**Aarow – a moveable item that is highly desired by Andrew, the game master**
Variables (private)	*Same as object*
Functions (public)	*Same as object, initialized to aarow*

Class Name	***Dungeon – holds a series of rooms, the player, and objects***

CHANGES TO ORIGINAL DESIGN

This assignment took a lot of mapping out on paper. I ended up adding a number of functions to help access the data that I wanted to, or to verify the status of an object from outside of its scope. Additionally, I didn't think a lot about the menu until I started coding, but this ended up being one of the cooler, and more frustrating aspects of the assignment. I wanted a way to clear the terminal every time that player moved, and I also wanted to make it so that you could move the player around without pressing enter after every keystroke. I got this to work really well, and it functions amazing – on a windows machine. But when I ported it over to Linux I found out that the <conio.h> library which

contains _getch() is unsupported. I fooled around trying to find a workaround or a linux equivalent and came upon the curses library. Despite this, I was unable to make curses compile on FLIP. Therefore, I fixed my program so that it would compile and work on FLIP, but you do have to press the "enter" key every time you make the player move. That said, you can overload the buffer with multiple keystrokes so that when you press enter, they are all executed at once, to save time. I recommend (after seeing that it works just fine in flip) testing it on a windows machine. It feels more smooth with the ability to refresh without pressing enter. I will include instructions on how to convert the code below if you wish to try it out.

TEST PLAN

There was less testing in this project than should have been. I mostly tested the functionality of my code as I wrote each piece. That said, I did come upon a number of issues that I ended up solving:

1. Handling dropping an item out of the player's backpack onto an empty space, a space containing an object, or a special case where the player can drop a stone onto a switch.
    a. I wrote code which set the pointers in the player's backpack to what was on the ground, then switched what was on the ground to NULL if the player was able to accept the object
    b. I then wrote code which did just the opposite, but I encountered errors when there was already an object on the ground, or I attempted to drop a NULL object onto the ground (nothing!)
    c. I fixed these errors by using a debugger and step-by-step finding out the locations that my code didn't handle a specific situation, then fixed it.
2. Player walking through walls.
    a. I created a border around my spaces so that the user can visually see the border, but also so that the player won't go outside of it.
    b. Initially, when I would move from room to room, instead of being located inside the room, the player would be on the same line, level with the walls. In these cases, the player was able to move toward a wall and walk through it, and behind it. This caused my program to crash if I walked outside of the 2D matrix
    c. I solved this problem, once again by stepping through my code and identifying where I needed to modify my player's start position within a room.

CONVERSION TO WINDOWS

My program works well on FLIP, but if you want to move through it at a faster rate than pressing enter each time, here's a way that you can convert the code for a windows machine to accept:

1. Uncomment <conio.h> in Dungeon.h and Space.h
2. Comment out the following lines in main:
    a. Line 36 // cin >> direction;
    b. Line 37 // std::system("clear")
3. Uncomment the following line so it is back in main:
    a. Line 35 // direction = _getch();

And that's it!