Special Characters: Dungeon Linux

Broderick Molcak, Zach Nelson, Darsh Thanki, Nathan Yarbrough, Jared Sasco

University Of Lethbridge

CPSC 2720

30 November 2017

Special Characters: Dungeon Linux

In our project, our aim was to create and implement a text based role-playing game, that will allow a player to traverse through a virtual dungeon collecting items and equipment to enhance their character, whilst fighting and defeating bosses and monsters throughout the map. This game will be using a text based read and response system, This text base input will help with the movement of the character throughout the map. The environment will be made in a ASCII symbol format and be pre-generated. The character, and enemies will also be using the ASCII character format.

Implementation

List of Features

- Items/Armour/Potions
- Enemies (Monsters/Bosses)
- Combat
- Chests
- Movement

ASCII Characters Used

- Walls:	| -
- Chest:	C
- Player:	O
- Monsters:	X

Implementation Changes

There were many changes that had to occur throughout the duration of our programs creation that differed from our initial plan. These changes had to be made due to time constraints and conflicts within our program files. Our initial design was incomplete, and many new classes had to be added to allow for the proper stability and reliability of the program. Additions include changes to such classes as the Entity class which now has two subclasses; Player and Enemy. Each of these classes inherit an x and y location and an updateHP function from the entity class.

Separately, the player class has a vector of items, while the Enemy class remains the same from the original implementation design.

The Item and Chest classes work symbiotically; when a player opens a chest it calls on the Item class and ites subclasses to populate the chest. While the Item class had a complete rework making it an abstract base class for the Weapon, Armour and Potion classes. Each subclass of Item has a string and three int variables which corresponds to a name and an attack, defense, and health value respectfully. These classes inherit the open function from the parent Item class allowing for the item to be added to the player's inventory, which facilities an equip function from the player class.The Chest class stayed the same from our implementation phase.

Few addition and changes where applied to the Battle class. A new function, fight, was added which is where the actual battle information is stored and passed to both the enemy and entity classes, and the battleAction function was made into a bool rather than accepting a boolean parameter.

The Room, World, and Factory classes work together to create the environment, entities, and the different rooms. The Room class controls all of the information of the room, while also checking for player collision with both the walls and enemies. The World class now functions as a builder class to build and populate the rooms with the declared entities within the room, we did however change the implementation of the rooms from an int to an array.

And the factory was added to be used for the building of entities.

Assessments and Measures

Coding and Naming Conventions

Coding Style

```
funtionName(){
    code
    if(param){
        code
    }
}
```

Naming Conventions

```
fileName
funcationName()
varName;
```

Documentation Style

Header Documentation

```
/**********************************
/// \name.h/cc
/// \author
/// \date
///
///
/// description
**********************************/
```

Program Documentation

```
/// description of function to follow
void function();

/// Description of var to follow
var;
```

Doxygen style documentation was also used where necessary

Error Handling. Throughout the progression of the project we used GDB, and manual break points using cout statements. GDB was an appropriate error handling strategy to be used because when exceptions are un-caught by the compiler they then run through GDB.. If a command handled has an error or exception, GDB will terminate it and print out an error message containing the exception name, the associated value, and a backtrace to the point where the exception occurred. Another form of error-handling that we used were manual cout statements, this was used rarely and only if GDB did not help us pinpoint the issue.

Testing Strategies. We used the Iterative development testing methodology. We decided to use this methodology as with each submission throughout the process of the project we obtained feedback which could then be incorporated into the next steps. This feedback gave us knowledge of any inefficient ideas, designs, and program coding, which could be changed in the next steps to further enhance and improve the software. To test for any errors within the code we used GDB, and getting other members of the group to read over the code to look for grammatical and format errors that we might have missed. GDB helped identify defects by looking through the code after compiling and finding any errors that the compiler might have missed, while getting other members of the group to read over the code helped us improve the efficiency of the coding and program.

Debugging and Optimization. Issues that we found were that the player could enter more than one input, the Chest class was unable to call from Item to create build the items within a chest, and possible memory leaks within the program itself. The multiple character input was fixed by putting a cin.ignore at the end of each loop to ignore everything but the first character input. The chest issue was a fix by creating a function build call for each item that would populate the chest, this fixed the issue with the item building but we have noticed that it sometimes it causeed memory leaks.

Results

Lessons Learned

Throughout the process of the project there were many things that we as a group could have been improved upon and implemented differently. Having a more rigorous planning stage, creating a detailed list of everythings that needs to be completed, using a conservative methodology for the amount of items and classes. Also creating a visual mind-map to have a visual representation of the classes and functions and how they interact with each other. Communication both within the group to ensure that we were keeping on track, and meeting at least once a week to make certain that everyone was working on what needed to be done and that no double work was being completed. Consulting with Howard more to make sure that we were on the right path and not doing anything overcomplicated and unneeded was being done. Last but not least using doxygen more frequently should have played a integral role throughout the coding process. Not using doxygen at times made it difficult to read over anothers code to figure out what each function was doing and required to properly run. When we as a group began using doxygen more frequently the efficiency of our workload increased dramatically.

Defects

There are few known defects within the final project and they are as follows, you can walk around enemies and access the chest without needing to battle, potions can increase you maximum health, and there are possible memory leaks within the program itself.