Chamber Crawler 3000

Haobei Song (Aaron)
Shangshang Zheng (Daniel)
Junheng Zhang (Eason)

Introduction

Chamber Crawler 3000 is a classical rogue-like game played on the terminal emulator on Linux supported platforms. Players are free to choose from a variety of different characters and fight their way out by combatting against fierce monsters along their adventure to the last floor. Meanwhile, players have to come up with appropriate strategies such as enhance their different abilities by picking up potions or trade with merchants, in order to survive the brutal, merciless cc3k environment created by CS246 staff and professors.

Overview

In our design, we adopted the Model-View-Controller pattern to organize the whole game structure in order to potentially accommodate to various platforms. Following general purpose programming principles in object-oriented software design, any objects on the floor such as players, enemies, potions, even tiles, doors, walls etc. were implemented as inherited classes from the superclass “Object” (which is also a class inherited from Subject). Besides, the visitor pattern was not only implemented between players and enemies but among any objects you can see whether movable or not when you play the game, so that in principle any two objects can interact with each other distinctively, which furthers the generalization purpose residing in object-oriented design. Observer pattern was also used quite often in our program to achieve interesting gameplay mechanism and to implement the interactive display with players. Another functionality we implemented successfully is the strategy pattern on the game controller such as to have user decide if they want the downloadable content extension during run-time.

Design

All kinds of Interactions(Attack, Move, Pickup) Among Objects (Visitor Pattern)

In our design, all the objects are inherited from the same class “Object”, which is a subclass of “Subject”, so that any change in objects can be observed by corresponding observers (class “Observer”) such as text display and ncurses display (DLC)

Potion (Decorator Pattern) automatically be reset upon reaching next floor

Game control (Model-View-Controller pattern (and Strategy pattern to implement DLC))

Interactive Display (Observer Pattern) observes every object on the floor

Relation between Dragon and Dragon Hoard (Observer Pattern)

Other Algorithms

Initialize objects on the floor

Chamber building

Enemy chasing

Resilience to Change

Model change

Our program is based on the much general visitor patterns among all the objects on the floor and could in principle realize any interactions between any two objects regardless of their mobility or distance with others. We are able to add any new players which could have different abilities against different enemies by simply creating a new class and add methods dealing with all the different enemies inside of that newly built class. Moreover, we could also change the player's behaviour upon picking up different treasures (such as get extra money), using different potions (special enhancement) or even distinct behavior when moving on the cells (going through the wall or even fly over to the other chamber) by simply adding the methods in the corresponding classes with which you want the newly added player to have different behavior. This is the power of Visitor Pattern where we got rid of almost all the tedious if conditions (except for the displaying, which for the general design purpose have to be able to accommodate different platform or different styles on the same platform) and still were able to modify both the visitor class and visited class exclusively without violating encapsulation rules compared to modifying everything outside of the to be modified class itself such as in the game controller.

Display Change

Since we implemented Model-View-Controller pattern in which the input, output and model are separated from each other, we are able to change the display of any objects into different style or format freely to adapt to various platform (terminal line or ncurses window (DLC)). We could just change the corresponding display characters (or colors (DLC)) without any modification in the model, as we assign each object in our main (Model) program a distinct enumerator, which could be identified by the display (View) and decide what to display as needed.

Game Control Change

In order to facilitate multiple usage of our program by different users, we resort to strategy pattern and create three different game controller under the same base class. Namely, we built one game controller for normal user, one for test user and one for DLC user, but the commands are of the same as they would override corresponding virtual methods inherited from the super game controller class. Therefore, users can choose different version of the game based on their own need during run-time and don't have to remember too many commands of the same functionality within different versions.