David Inglis dainglis
Jason Williamson j43willi

ChamberCrawler3000

The order in which we have decided to build the modules for our application is as follows:

Phase 1 Estimated completion date: Saturday, Nov 28/Sunday, Nov 29

- Reading command-line arguments and then user input, and printing a floor
- Creating a character, enemies, potions, gold (general models), and printing player data
- Creating framework for different race/item types (however their implementations won't actually be different, yet)
- Random generation of items/enemies on floors

Phase 2 Estimated completion date: Tuesday, Dec 1/ Wednesday, Dec 2

- Player/enemy movement and interactions (possibly without consideration for racial traits)
- Player - item interactions

Phase 3 Estimated completion date: Thursday, Dec 3/ Friday, Dec 4

- Implementing specific item types (potions, dragon gold, merchant gold)
- Implementing specific enemy types (and specific racial traits for those interactions)

Our plan of attack is similar to the one suggested under the "If all else fails" header in the assignment outline, except that we have moved random generation of items, etc, closer to the beginning. We are attempting to have the entire board generated, with player, stairs, all the other randomized aspects, before we deal with the interaction between those entities, so that we can test that the entire board layout works first.
David Inglis will be responsible for command-line arguments, the display, and passing user input to the Player class, and Jason Williamson will be responsible for creating a general character, enemy, item model, and randomly generating their locations.

We will then implement the interactions between those entities, and as stated above, possibly without consideration for racial traits because we are going to actually make the different races *different* near the end of the project.
David Inglis will be responsible for player – item interactions, and Jason Williamson will be responsible for enemy movement and player – enemy interactions.

Lastly we'll fill in the framework for the different races/item types with actual differentiation (aka racial abilities/hindrances) to complete the game.
David Inglis will be responsible for creating the specific player/enemy races, and Jason Williamson will be responsible for the specific item types.

For DLC, assuming we have time to implement any, hopefully we can allow the player to interact with merchants, in order to buy potions

David Inglis dainglis
Jason Williamson j43willi

Questions

How could your design your system so that each race could be easily generated? Additionally, how difficult does such a solution make adding additional races?

In our system each race will be spawned with use of a Factory Pattern, this abstracts any knowledge of Character's race away from the client. To invoke creation of each the races We create an abstract Level Class that is a superclass to any concrete level implementation.

This makes adding new a race simple. If races are added to either one of the Enemy or Player character hierarchy we only have to make a small change in our concrete Level class.

How does your system handle generating different enemies? Is it different from how you generate the player character? Why or why not?

Generation of enemies and the player character are handled different. As it is necessary to spawn many enemies in a level (20, not including Dragons) we must also consider there random placement in each chamber (with equal probability), and each tile in the chamber must have an equal probability for an enemy to be spawned there as well. Not to mention that player character's race is chosen by the user before hand, in contrast to enemies race that is randomized based on probability between the races.

For these reasons our system must handle enemy generation separate from player. Creating 20 twenty randomized enemy races in bulk will be handled by our Factory Pattern. However, our player character does get created by the Factory Pattern as well, but for the aforementioned reasons these two character's generation specifics must differ.

How could you implement the various abilities for the enemy characters? Do you use the same techniques as for the player character races? Explain.

Using the Template Method Pattern we can override the behaviour of our super class Enemy, but all of the behaviour. In this way we can re-define certain steps or various specific abilities without changing the Enemy class structure. Enemy provides some abstract operation that subclasses can override to provide concrete behaviour. We do use the same techniques for the Player character races as well. Both Player and Enemy represent superclasses that will present abstract classes for subclasses (different races) to provide race specific behaviour for.

The Decorator and Strategy patterns are possible candidates to model the effects of potions so that we do not need to explicitly track which potions the player character has consumed on any particular floor? In your opinion, which pattern would work better? Explain in detail, by weighing the advantages/dis- advantages of the two patterns.

David Inglis dainglis
Jason Williamson j43willi

We claim that Decorator Pattern would work best to model the effects of potions, as the pattern provides an elegant way to wrap a core component, in this case the player character, with one or more decorators that modify the core component slightly. This makes adding additional decorators very easy, simply by adding a new concrete decorator we have extended the pattern. Using the concrete decorators to decorate a component only requires knowledge of the concrete decorator class name, everything else is abstracted.

The Strategy Pattern enables us to define a family of algorithms and make each one interchangeable. Although this pattern could be configured to work for this problem, the idea of effects of a potion on a player is better mimicked by Decorator Pattern.

How could you generate items so that the generation of Treasure and Potions reuses as much code as possible? That is, how would you structure your system so that the generation of a potion and then generation of treasure does not duplicate code?

Game items that include potions and gold in our game will be spawned through a Factory Pattern, this is a level specific pattern. As mentioned before this makes adding new potions and treasure an easy task. That being said potions and gold differ from characters, as they are representations of values. For this reason the potion and gold classes will be instantiated with a type value, all potion and gold type specifics will be defined based on this type value. In this way we are not duplicating any code for each item, also, adding an extension (an extra gold, or potion type) would require only a small change in potion or golds respective class.