**Ultimate Hero Fun Adventure Quest**

**A Game**

By The Hackstreet Boys:

Brian Lollis  
Jacob Weeden  
Sierra Huckins

**Game Description**

HARK! Mighty Hero! Whether you are questing for sick loot, gold, epic battles or…. ADVENTURE, Ultimate Hero Fun Adventure Quest (UHFAQ) is the game for you! Pick you own band of adventurers to pit against the evils monsters to show ‘em who’s the best adventurer ‘round town, YEAH!

Who do you want to play? We have eight classes to pick from: the noble Cleric, the proud Warrior, the stealthy Elf, the precise Archer, the spooky Necromancer, the righteous Paladin, the spellbound Wizard, and the tree hugging Druid! You get four of them for your party, so the combinations are endless (almost).

Each hero has their own special attack, stats, and can level up! Once you hit level three with one character, you know what happens? YOU GET ANOTHER SPECIAL ATTACK! This new special attack will seriously mess up some monsters. Try and pick a party that will work well together, because you are going to have to rely on each other to make it through this maze!

After you pick your band of merry gentlemen, you have to traverse the 2D MAZE OF DOOM! You get to pick your own map, but be warned… These mazes will seriously get you turned all sorts of backwards, upside down, get your rights and lefts confused, its crazy stuff! Your objective is to reach the exit marked by an ‘X’ on map. Once you do, you win the game. Easy enough right? NO WAY!

Every time you try and move to a spot, you have a chance to do an EPIC BATTLE with some horrific monsters (think Pokemon encounters). Once an encounter has occurred, you are going to fight a random number on enemies based on the max number of surviving heroes.

You may end up fighting the giant cyclops, the fire breathing dragon, the pesky gremlin, the super strong ogre, the brain hungry zombie, the blood thirsty orc, the scary skeleton, and the freakish manbearpig. Be careful because these guys will kill you until you are dead! They each have their own special attack, as well as a basic attack to try and eliminate your party, how rude!

Any hero that kills a monster gain experience. The monsters also have a chance to drop sick loot. I mean, who doesn’t like loot?! The hero that kills the monster will be able to pick up the loot and use it during the next encounter. There will be many encounters… If you plan to make it through the maze, your resolve will have to be as hard as you steel!

If any hero dies, they are dead, FOREVER! You can’t use ‘em to help you get to the exit. The monsters will just keep coming. They are ruthless killers, hell-bent on seeing your party’s demise… But, if you happen to make it to the exit, even with only one lonely party member alive, you will win the game! Winning the game is such a milestone, you may actually want someone there with you to document your reaction so you can relive the moment, over and over.

So, hero, if you feel capable enough to pick a competent party, navigate a topsy-turvy maze, and battle endless hordes of enemies while your party members die all around you, then I wish you luck! You will need more than a little luck to make it to the end… ULTIMATE HERO FUN ADVENTURE QUEST GO!

**Patterns Used**

Template Pattern

The first big refactoring we did after the initial writing of our game was to use the template pattern to improve our battle method. Originally, the entire battle sequence and was written as one huge method in the Battle class. We realized this was leading to a lot of duplicate code between the good guy’s turn and the bad guy’s turn, so we broke the code down into the steps that were accomplished for each turn and then rewrote those as a template called BattleTurn. Anything that is executed by both the good guy and the bad guy during their turn is written in this abstract BattleTurn class and then the different individual actions are overwritten or hooked as necessary into the concrete GoodGuyTurn and BadGuyTurn classes. All of the steps are combined in the final executeTurn() method in BattleTurn, which is then called as needed from Battle. Using the template method both saved us a lot of duplicate code and made it much easier to follow the sequence of our battle turns.

Simple Factory

One pattern that we used multiple times was that of simple factory. Of course simple factory is not a true pattern, but for our purposes it worked very well. When first writing our code we coded the generators for the items and the bad guys, as well as the choosing ability for the initial good guys party, directly into the classes that needed them. Of course this became problematic later on when we started trying to expand from our one or two tester classes for the good guys and bad guys. It was difficult to locate the code when we needed to add to it because it was lost in the mess of its calling class. It was also easy to miss updating the text that prints to screen because it wasn’t always directly near the object it was creating. By breaking each generator into its own factory, we were able to group all the interacting code for the three factories that we made. This made it much simpler to add additional characters or items because we only had to change one class when we did so. We considered using the more developed abstract factory method to link our factories together while maintaining variability, but decided against this as it added complexity that was not necessary and didn’t really contribute anything to the usability or updatability of our code. The simple factory was very helpful in clarifying and simplifying our code though, which is why we used it for the GoodGuyFactory, BadGuyFactory, and the ItemFactory.

Strategy Pattern

Strategy was one of the first ideas we had for implementing a pattern in our code. It also turned out to be a pattern that we used quite a bit. The two main places that we used the strategy pattern were for in the implementation of each character’s attacks and in the use of items. The attacks needed to use strategy pattern because the attacks available to a character change during the course of the game. Each character is initialized with a default attack (which is the same for all characters) and a special attack that is distinct to that character. When a character reaches level 3 though they earn a new attack, so there had to be a way to add this new attack in. This was solved using an AttackInventory so that if we want to in the future we can continue to add new attacks. Similarly, with the items we needed to be able to gain and use items throughout the course of the game. This was also solved using an ItemInventory. Characters gain items as they are dropped and the items are added to their inventory. When they use the item, it is easily removed from the inventory. The use of Strategy Pattern made all of these runtime changes very simple.