# RPG Phase 2 – Combat and Monsters

In role-playing games (RPGs), a player controls a character for all in-game interactions. In Phase 1, we built the base character (which we called a Hero). However, in most RPGs, characters are expected to have adventures wherein they fight monsters for experience, loot, and story development. In this phase, we’ll add some simple combat mechanics and some monsters to fight.

## Requirements

We will be adding to and modifying the code from RPG Phase 1. With this said, this phase assumes you have completed the requirements from the previous phase. Below, you will find new and “adjusted” requirements for the classes you made in Phase 1. You will need to make some new classes as well. So, without further ado….

First, we need a few new classes for potions!

HealthPotion

* Has a constructor that randomly chooses a number between 1 and 10 inclusive. It multiplies that number by 10 and stores it in a field called points.
* Has a use() method that returns the value contained in the points field. It also resets the points field to zero. You’re going to have to think about the order of operations here to make this work correctly.

MagicPotion

* Has a constructor that randomly chooses a number between 1 and 10 inclusive. It multiplies that number by 7 and stores it in a field called points.
* Has a use() method that returns the value contained in the points field. It also resets the points field to zero. You’re going to have to think about the order of operations here to make this works correctly.

Next, we’ll need to update our Hero class with some new data and functionality.

Hero - modified

The Hero follows all the same rules as it did before regarding attributes and bonuses (see Phase 1 for more info), but now is able to hold potions in inventory, take damage, and take potions to restore HP and MP. The Hero will be able to attack using a normal attack and a special attack.

**Potions**

* **Health Potions: The Hero may hold up to 5 Health Potions. This must be done using an array. You will also need a way for outside classes to get the number of Health Potions the Hero currently has. Make sure your solution does NOT simply return the array of potions to the outside world.**
* **Magic Potions: The Hero may hold up to 5 Magic Potions. This must be done using an array. You will also need a way for outside classes to get the number of Magic Potions the Hero currently has. Make sure your solution does NOT simply return the array of potions to the outside world.**

**Methods**

* **toString() – The toString override method should be used to output all information about the Hero in a useful, well-formatted way. It should also include potion inventory data, displaying the number of health and mana potions the Hero currently has. The values should never be lower than 0 nor higher than the expressed bounds above.**
* **receiveDamage(int damage) – This method will take in an int and subtract it from the current HP. If the Hero’s current HP is equal to or less than zero, the Hero is dead.**
* **useHealthPotion() – If there is at least one Health Potion in the Hero’s inventory, a potion will be used and its value in HP will be added to the Hero’s current HP pool (up to, but never exceeding the Hero’s base HP value). The Health Potion should be removed from the array after it is used. You may decide how Health Potions are used (meaning in what order they are pulled from the array). This method returns the potion’s HP value.**
* **receiveHealthPotion(HealthPotion potion) – This method takes in an instance of HealthPotion and adds it to the next available slot in the Health Potion array. If there is no room left in the Health Potion array, the new Health Potion is immediately used, adding its value to the Hero’s current HP. This method returns true if the potion was stored in the array, or false if there was no room left and the potion was automatically used.**
* **useMagicPotion() - If there is at least one Magic Potion in the Hero’s inventory, a potion will be used and its value in MP will be added to the Hero’s current MP pool (up to, but never exceeding the Hero’s base MP value). The Magic Potion should be removed from the array after it is used. You may decide how Magic Potions are used (meaning in what order they are pulled from the array). This method returns the potion’s MP value.**
* **receiveMagicPotion(MagicPotion potion) – This method takes in an instance of MagicPotion and adds it to the next available slot in the Magic Potion array. If there is no room left in the Magic Potion array, the new Magic Potion is immediately used, adding its value to the Hero’s current MP. This method returns true if the potion was stored in the array, or false if there was no room left and the potion was automatically used.**
* **isAlive() – This method returns a boolean representing whether or not the character is still alive. “Alive” is defined as having a current HP value greater than zero.**
* **attackNormal() – This returns the value of the Hero’s attack combined with the Hero’s Damage Bonus. In the future, the base attack damage will be determined by the weapon being used. For now, the base attack value is always 10.**
* **attackSpecial() – This returns the value of the Hero’s special attack combined with the Hero’s Spell Strength Bonus. This attack costs 10 MP each time it is used. The 10 MP are deducted from the current MP pool (not the base MP pool). In the future, the base attack damage will be determined by the spell being used. For now, the base attack value is always 15. However, if there is not enough MP to use this attack, the total attack value returned is zero and no additional MP is lost.**

Monsters

The GM will now have three different monster types it can pit against the Hero. These monsters are going to be YOUR design! Here are the rules:

* You should have a monster that is easy to kill, one that is of a medium challenge, and one that is difficult to defeat. One suggestion is to make classes for Orc, Troll, and Dragon respectively.
* Each monster should allow for a range of values in its stats appropriate to that monster type. This includes the three major attributes, health, and mana. The same bonus rules apply to monsters as they do to heroes. However, a monster may have different possible stat ranges than the hero. An Orc, for example may have attribute ranges that are far smaller than the hero’s while the Dragon may have ranges that are far greater by comparison.
* Each monster should have a normal attack and a special attack. Like the Hero, the special attack is stronger, but also requires the use of mana. The normal attack does not use any mana and is always available.

The GM should be able to create any one of these monsters at any time, randomly generating the stats as appropriate for that monster type

Combat

Once the user has created the Hero he/she likes best and named it, the GM will ask which monster the player wishes to fight. Present the user with your three monster options. Once the user has selected the monster type, the GM will randomly generate that monster (based on its class definition) and pit the player’s Hero against the monster. When it’s the player’s turn, you will print out the character’s current state (use the toString if you like), the monster’s current state, and then present the following menu:

* Perform Normal Attack
  + Performs the character’s normal attack, displaying the total damage done
* Perform Special Attack
  + Performs the character’s special attack, displaying the total damage done
* Use Health Potion
  + Uses a Health Potion from the character’s inventory. If there are no potions available for use, this option should NOT appear in the menu.
* Use Magic Potion
  + Uses a Magic Potion from the character’s inventory. If there are no potions available for use, this option should NOT appear in the menu.

After the player has chosen an action, the results are printed to the screen. The monster then takes its turn, randomly attacking with normal or special attacks (assuming the monster has enough mana to make a special attack). Once the encounter is over, the player should be asked if he/she wishes to fight again. If yes, then the player is presented with the same three monster choices. If no, the application closes.

When a monster dies, it drops between 0 and 4 potions (randomly determined, of course). Any potions that are dropped are randomly assorted (health and mana). Each potion should be added to the player’s appropriate inventory collection. Remember that, if a player has a full potion inventory of that type, the potion is immediately used. The player should be informed of the potions dropped and whether they were added to the inventory or used.

**COMBAT RULES:**

* The player and the monster roll what is called initiative. This is done by rolling 1d20. The highest roll wins. In the case of a tie, both roll again.
* In a single round, each opponent will take turns being the attacker and the defender. The opponent with the highest initiative attacks first with the other as the defender. Once that turn is resolved, they switch roles with the opponent with the highest initiative defending, the other attacking.
* After 5 rounds, initiative is rolled again.
* During an attack (normal or special) the attacker rolls to strike, the defender rolls to dodge. This is done by each opponent rolling 1d20 and adding to that roll their bonus to strike (if attacker) or dodge (if defender). Highest roll wins with ties going to the defender. If the attack is successful, the defender takes damage. If the defender has the higher roll, no damage is dealt. Regardless of this outcome, using a special attack always results in the expenditure of the mana.
* A natural 20 means that the die roll was a 20 before any bonuses were added. A natural 20 is referred to as a critical success (or crit for short). A crit during an attack is an automatic success, results in double damage being dealt and can only be defended against by a crit. In the case when both opponents roll a crit, the opponent with the highest total value after bonuses wins. If the attacker rolls a crit and the defender does not, it’s a success, even if the defender has a higher total roll with bonuses.
* Similarly, a natural 1 is a critical failure (also called a botch). An attacker who botches automatically fails (even if the defender also botches). If the defender botches when the attacker does not, the attack is successful and does double damage. If the attacker rolls a crit and the defender botches, the attack is successful and does QUADRUPLE damage!
* Combat continues until an opponent is dead.

DiceBag

No changes. The GM is still the only class that owns an instance of DiceBag.

Die

No changes.

## Rubric

**AUTOMATIC ZERO:** Your application uses the main method for logical code (it must ONLY start the program) or a character or monster cannot be created by the GM