# TEXT-BASED RPG GAME

**Introduction**

The game is prototype of a simple RPG played on the console using only text. This game incorporates some of the basic RPG systems such as loot, exploration, and small-scale fights. The goal is to survive as long as possible. At present, this is a very basic game. It has only one enemy type who seems to be a little difficult to deal with until proper loot is found, and only one type of every item. There is no levelling system as well. These will be implemented later.

**Code Details**

The game has eight classes (excluding the child classes), each representing a separate item in the game. Here is the list of classes:

| **Class Name** | **Function** |
|---|---|
| Item | Abstract Class. Represents an equippable item. Derived into six child classes. |
| Inventory | A linked list that stores items. Can hold up to five items. |
| Potion | Heals player by 25 points. |
| PotionPouch | Linked list to store potions. Can hold up to five items |
| Person | Base class for Player and Enemy |
| Player | The player. Has 100HP and deals 20 damage per hit |
| Enemy | The enemy. Has 100HP and deals 15 damage per hit. |
| Game | Manages the key elements of the game. |

**Item Class**

The item class is responsible for various items that the player can equip throughout the game. The player can use these items to their advantage and increase their offence, defence, or both. There are a total of six classes that are derived from the item class:

| **Class Name** | **Function** |
|---|---|
| Sword | Boosts damage by 8% |
| Shield | Boosts defence by 8% |
| Armor | Boosts defence by 5% |
| Helmet | Boosts defence by 3% |
| Boots | Boosts damage by 3% and defence by 2% |
| Braces | Boosts damage by 4% and defence by 2% |

Item class and its children have four methods. The GetNext and SetNext are used to return and set pointers for the next item in the inventory. The GetDamageRes and GetAttackBoost return the amount of damage boost and defence each item provides to the player.

**Inventory Class**

The inventory class is made using a linked list to hold item objects. A player can hold up to five items which offers cumulative attack boost or defence boost. Player is not restricted to hold only one item of one item type. He can stack up to five items of the same type, filling up his inventory. Example, I can hold up to five swords so I could gain a 40% increase in attack but have 0 defence, stack five shields to have 40% defence boost but have no attack boost, or I could manage my inventory in such a way where I can have a decent damage boost and defence boost by equipping different items or item types.

The inventory class has a total of eleven different methods. Below is the list of methods, return types and their function:

| Method Name | Return Type | Function |
|---|---|---|
| Inventory | constructor | Instantiates head pointer, tail pointer and the maximum number of items that the inventory can hold. |
| ~Inventory | destructor | Clears the entire inventory. |
| GetMaxSize | int | Returns the maximum number of items that the inventory can hold |
| GetCurrentSize | int | Returns the number of items that the inventory is holding now |
| GetTotalBoost | float | Returns the cumulative damage boost offered by all the items in the inventory. |
| GetTotalRes | float | Returns the cumulative defence boost offered by all the items in the inventory. |
| isFull | bool | Returns true if the number of items in the inventory are equal to the maximum number of items it can hold. |
| GetHead | Item* | Returns the first item in the inventory |
| GetTail | Item* | Returns the last item in the inventory |
| AddItem | void | If the inventory is empty, item is added as the first node of the inventory. Else, it is placed next to the last node if the number of items present are lower than maximum amount. |

| | | |
|---|---|---|
| Clear | void | Deletes all the instances of items that are created on the heap, clearing the inventory |
| RemoveItem | void | Removes the specified item from the inventory |
| SwapItem | void | If the inventory is full, it deletes the specified item and adds a new item to the inventory. |

**Potion Class**

The potion class is similar in design to the base item class. But, instead of providing any damage boost or defence boost, it heals the player by 25 points. The remaining methods are like that of the item. Instead of GetAttackBoost or GetDamageRes, we have GetHeal which returns the amount by which the player is healed.

**Potion Pouch Class**

The potion pouch is used to hold potion objects. Just like inventory, you can only hold up to five potions. The potion pouch class is very similar to inventory in terms of design. The only difference is that there is no method that returns cumulative heal amount as one potion can be used at a time and are independent, and there is no method to swap items since there is only one type of potion now. The other methods are like that of inventory.

**Person Class**

This is the base class for both Player and Enemy. It has five methods. `GetHealth` returns the person's health and `isDead` returns if the person is dead or alive. The other three, `DealDamage`, `ReceiveDamage`, and `Dead` are virtual functions, that have different behaviours depending on the player or the enemy. They are overridden in their classes.

**Enemy Class**

The enemy class is simple. It creates an enemy object with 100HP and 15 Damage if nothing is specified during its instantiation. It is a derived class of person. It inherits the behaviour of the functions `GetHealth` and `isDead`. The remaining virtual methods are simple. `DealDamage` returns the amount of damage it can do, `ReceiveDamage` takes the damage from a person object and reduces the health by the amount of damage done by its opponent. `Dead` prints out a statement stating the enemy has been killed.

**Player Class**

The player class is a derived class from person with its own behaviour. It has some unique functionalities that the person and enemy classes lack. While the player class inherits five functions from the person class, it has seven methods that are exclusive to the player class. Here is a list of all the methods and their function:

| Method Name | Return Type | Function |
|---|---|---|
| Player | constructor | Instantiates a player object with 100HP, 20 Damage, an inventory and potion pouch of size 5 |
| ~Player | destructor | Deletes player's inventory and potion pouch |
| FindItemByNumber | Item* | Return the item pointer placed as the nth node of the inventory |
| AskMoveInput | char | Asks and returns player actions |
| UpdateStats | void | When an item is picked up, the stat buffs are applied to the player |
| PickUp | void | Picks up and places the item in player's inventory. Calls inventory's Swap or Add Item depending on the inventory size. |
| PickPotion | void | Picks up and places potion into potion pouch. If there is no place, a message is triggered |
| Move | void | Takes the value returned by AskMoveInput and performs necessary actions. WSAD would trigger a narrative placeholder, H to heal, R to manage inventory, and I to lookup the inventory. Q would allow player to force quit the game at any given instance. |
| Heal | void | Increases player health by 25 points if below 75 and above 0. If above 75, it would increase the health to 100. |
| Drop | void | Drops an item from the inventory |
| SetForceQuit | void | Sets forcequit flag to true when q is pressed |
| IsForceQuit | bool | Returns true if forcequit is true |
| DealDamage | void | Returns the amount of damage the player can deal. This also adds boost to the damage if there are any items that provide a damage buff. |
| ReceiveDamage | void | Takes damage from the opponent and reduces health accordingly. Lower damage is taken if the player has any defence boost items equipped. |
| Dead | void | Triggers "Player Killed" text |

**Game Class**

The game class is responsible for all the events that happen in the game. It is responsible for spawning enemies, items, potions, displaying information etc. The `Instructions()` method is responsible to display player instructions when the game begins. After that `Begin()` and `Battle()` handle main events and player-enemy battles respectively. `Begin()` is responsible to spawn enemies, potions, item, either two, or all three of them, or none of them, which is randomized.

`SpawnItem()` spawns any of the five items that the player can pick up throughout the game and calls player's `PickUp()` method. `SpawnPotion` simply spawns a potion and calls `PickPotion()` method of the player. `SpawnAndFight()` spawns an enemy. Later, the player is given a choice to fight the enemy or not. Even though the player chooses not to fight the enemy, it is highly possible (2 out 3 times) that the player is forced to fight. If the player decides to fight or is forced to fight, the `Battle()` method is called.

`Battle()` method handles player input to attack, heal and force quit the game. The function runs until one of the players is dead or the player force quits the game. It is turn-based battle system where both fighters take turns to perform certain moves. If the player decides to heal and not attack, this is also considered as move and the next move will be performed by the enemy. Health of both the fighters are updated after their respective moves and is displayed on the screen.

**Additional Methods**

The code also has friend functions and a function that manage narrative placeholder statements. The friend functions belong to Item, Inventory, Potion and Potion Pouch classes. They are responsible to display the stats of potions and items, and contents of inventory and potion pouch. AssignNarrative function opens the **narrative.txt** provided with the code and place all its contents in a vector.

**Inputs**

After running the code, the game will be playable in the terminal itself. The inputs depend on different situations. During the Narrative or Move section, you can press "W", "S", "A", or "D" to trigger a narrative placeholder and either item, potions, or any enemy is spawned. Additionally, you can also use R to remove an item from the inventory, I to display inventory or H to heal. But none of these actions will trigger the narrative placeholder or spawn items, enemy, or potions.

During battle, the player can press "E" to attack the enemy or "H" to heal. Since Battles are turn-based, you can either heal or attack. Enemy's move is followed by yours irrespective of your move.

Player can also quit the game by press "q" at most parts of the game. Remember to press enter and wait for the console to print the text completely to avoid unintentional actions.

**Outputs**

The output is game played on the terminal. The game is divided into 4 parts. Title and Instructions, Move, Narrative Placeholder, and Battle and/or item pick-up. Most of these parts feature animated text to provide an immersive effect. Every action performed by the player gives visual feedback to let the player know that their action was successfully performed. The game displays item stats, so it becomes easy for the player to decide what to keep and what to remove. The narrative placeholder is a set of descriptions of the places that the player might be seeing and are displayed in a random order. So, it is possible to see one description twice or thrice in a row. These descriptions were only placed to give the player an immersive feel and to provide an imaginative experience through various kinds of place descriptions.

**Know Issues**

During my playtest session, I haven't encountered any unexpected behaviours. Only small issues such as the enemy's death message being displayed twice. But there is still a minor chance that unexpected behaviours might be encountered after going farther into the game.