Monster Evolution Game

Game Overview:

You have woken up in a strange dungeon. It is full of weird monsters that you are able to catch and look after! You will wander the dungeon, finding monsters and evolving them into more powerful and special versions of themselves. Can you evolve a monster that you can fight against the evil boss that guards the exit of the dungeon?

Game Implementation code:

The game code is divided into four main classes:

1. The class **Monster**.
2. The class **Player**.
3. The class **Spot**.
4. The class **Game**.

We will discuss every class in detail:

- Class Player:
  Description: this class created to define a player with it's information of name and skills, healthy degrees and number of monster of each type he have and lists to store his monsters.

  Constructors:
  1. Constructor with no argument.
  2. Constructor with one argument of his name.

  Date field:
  1. String store his name.
  2. Integer store his skills degree.
  3. Integer store his healthy degree.
  4. Integer store number of his Bugbear monsters.
  5 Integer store number of his Platypie monsters.
  6. Integer store number of his Emoo monsters.
  7. Integer store number of his Evolved monsters.
  8. List of object from monster class to store his Bugbear monsters.
  9. List of object from monster class to store his Platypie monsters.

10. List of object from monster class to store his Emoo monsters.
11. Object of monster class to store his Evolved monster.

Methods:
1. Getter of his name.
2. Getter and setter of his skills degree.
3. Getter and setter of his healthy degree.
4. Getter and setter of his monster number of Bugbear.
5. Getter and setter of his monster number of Platypie.
6. Getter and setter of his monster number of Emoo
7. Getter and setter of his monster number of Evolved.
8. Adding monster to the right list the monster belong.
9. Getter of the evolved monster.

- Class Monster
  Description: this class created to define a monster with it's information of name and skills degree and healthy degree and special power modifier.

  Constructors:
  1. Constructor with no argument.
  2. Constructor with one argument of his name.
  3. Constructor with three argument name, skills, healthy and power.

  Date field:
  1. String to store monster name.
  2. integer to store his skills degree.
  3. integer to store his healthy degree.
  4. integer to store his special power modifier.

  Methods:
  1. Getter and setter of his skills degree.
  2. Getter and setter of his healthy.
  3. Getter and setter of his special power modifier.
- Class Spot:
  Description: this class created to define a single cell until of the game board

each cell has an identification id that tell us meaningful information about the object that inhabit in this cell (player or monster) so it make easy when the player move from cell to another and know if there is a monster there or not. The id takes 1 if the player inhabit in this cell or 2 if monster or 3 if its free space cell.

Constructors:
1. Constructor with no argument.
2. Constructor with one argument object of class player.
3. Constructor with one argument object of class monster.

Date field:
1. Integer to describe the id.
2. Object of player class.
3. Object of monster class.

Methods:
1. Getter and setter of the integer id.
2. Getter and setter of the player object.
3. Getter and setter of the monster object.

- Class Game:
  Description: this class created:
  1. Initializing the game when an object created by calling method (init) that randomly inhabit the player in the leftmost edge of the board by calling (placePlayer) method and inhabit the monster in the board randomly and take into account that the Skeletor monster must be in the rightmost edge of the board.
  2. Manage the whole game by calling the main method (startGame) that take decision depending on the input provided by the player, when the player enter specific move (up, down, right, left) this method call the method (move) that move the player forward his direction and call method (playWithMonster) that discuss the issue of have monster in this cell if there is a friend monster call method (catchMonster) to catch monster by playing with his and if the monster is Octopod the call method (fightOctopod) to

fight this monster and if the monster is Skeletor then call method (fightSkeletor) to fight his if the player has evolved monster only, finally if the cell don't have monster so the player continue playing.

Date field:

1. Matrix describe the board of the dungeon.
2. Matrix describe the visited cell of the player.
3. Lists define the direction (up, left, down, right).
4. Integer to define the length of the dungeon.
5. Integer to define the width of the dungeon.
6. Point to store the start point of the player at the first of the game.
7. Flage the discuss the issue of the view of the dungeon.

Methods:

1. Method for initializing the whole game.
2. Method for initializing of the direction.
3. Method for inhibiting the player in the leftmost of the board.
4. Method for inhibiting the monsters in the board randomly.
5. Method take a point and check whether if it's in the board
6. Method return random potion in the x-axis.
7. Method return random potion in the y-axis.
8. Method that print the board of the game.
9. Method that detects that move of the player.
10. Method that handle the player with a friend monster to catch him.
11. Method that handle the fight between the player and unfriend monster.
12. Method that handle the fight between the player and Skeletor monster.
13. Method the manage the whole game.

Game Implementation issues:

1. Language used in implementation is C++.
2. Structures: different kind of structures used to store data
   - vector: we use it as a list to store objects of monsters in class Player and used in class game as matrix (vector<vector<Spot> > , vector<vector<bool> >) to represent the board of the game and also to mark the visited position

- pair: we use it in class Game (pair<int, int>)to store the current position and the starting position of the player at the first of the game to make the player return to this point if the player find the Skeletor monster and he doesn't have evolved monster.
- Class spot: this structure used to store information of single unit position of the board game for example in this position player or monster or free space and if there is a monster or player so store its object.
- Class Player: this structure used to store player information his name, skills degree, healthy degree, the number of each monster he catch, lists of his catches monster and also object of his evolved monster.
Class monster: this structure used to store monster information his type (name), skills degree, healthy degree and his special power modifier.

3. Movement of the player:
The player press (u, d, s, a) to move, which means (up, right, down, left) respectively so let represent them as integers (0, 1, 2, 3) respectively and also define di={-1, 0, 1, 0} and dj={0, 1, 0, -1}, to move the player to his wanted direction the simple technique is to determine which integer represent his next move let 'nxt' so we have to increase the current position in x-axis to the value in the di[nxt] and in y-axis to the value in the the dj[nxt].
For example is the current position in cell (5, 7) and the player press 'd' so the corresponding integer to this move is 1 (i.e. nxt = 1) so the next position in x-axis will be {current position in x-axis + di[1] = 5 + 0 = 5} and next position in y-axis will be {current position in y-axis + dj[1] = 7 + 1 = 8} so the next position is (5, 8).
4. Evolving this implementation in future:
This code is designed in such a way to be able to enhanced many time again as it's written in efficient way because it's divided into understood classes its objective clear and can be changed to reflect the customer needs.
5. Changing this design:
The class Game may be divided into three classes:
- class to represent the move of the player.
- class to fight an unfriend monster.
- class to catch a friend monster.