THE SNAKE GAME USING C++

Features of OOP implemented in the project:

- Basic inheritance
- Polymorphism
- Demonstration of protected attributes accessed by derived classes
- Parametrized constructors
- Use of this pointer
- Setter/Getters
- Pure virtual function
- Virtual function
- Header files made and used together for each class
- Base constructor used by derived class constructor
- Use of objects to call class methods
- Using base pointer to call methods of derived classes

OVERVIEW

The primary issue we faced with our project was creativity while trying to understand how to link together different games using features of OOP. The obstacle we kept coming up against was the difficulty in finding mutuality in different games, since this would be the main use of OOP (inheritance, polymorphism, etc).
We solved this problem by deciding to work on one game, which would have different alterations(versions). This way we could implement features of OOP for the now definite mutual aspects of all versions.

BASIC STRUCTURE

Following is the basic structure of the code:
A base class having a few protected attributes, and a public parametrized constructor as well as some methods.
3 classes derived from this base class, one for each version of the game, all using public inheritance to access base class's properties and methods.
Base class having a virtual function, redefined in one derived class.
Base class having a few pure virtual functions, redefined in all derived classes.
Main function uses a simple switch statement to handle the different types of versions the user may want to play. In case of each version, the object of appropriate class is used to call relevant methods and make use of required attributes. There are 2 basic conditions for ending the game, either by winning (scoring 3 points), or by pressing escape key and ending the game.

ARRAY INITIALIZER

This is our pure virtual function. It is responsible for assigning values to the (21*11) 2-dimensional array after the array has been declared. The reason for it being pure virtual is because the maze will have a different design in each version, hence re-definition of the function is required in each class.

MOVEMENT

This is our virtual function. It is responsible for taking user input and controlling the movement of the snake based on that input. It is declared virtual since it must be re-defined in select versions only and remains the same in other versions. Version 3 has a different definition of this function due to an added aspect in the game which modifies the functionality of this function slightly. The other 2 versions use the original function which has been defined in base class.

MAIN

Uses a switch statement to handle user input related to which version they wish to play. Each case in the switch statement makes use of the relevant class's object and methods. In case of each version, the object of the appropriate class is used to call relevant methods and make use of required attributes. There are 2 basic conditions for ending the game, either by winning (scoring 3 points), or by pressing escape key and ending the game.

OTHER METHODS

Generator:

Generates 2 random integers for variables x and y for a specific range for each. Following is the code used in generator:

```
x = rand() % 9 + 1;   // Generates a random value between 1 and 9
y = rand() % 19 + 1;  // Generates a random value between 1 and 19
```

This randomization is done based on the acceptable x and y range that are used as dimensions of the maze. Since the dimensions are 11 by 21 for x and y (rows and columns) respectively, the generated value needs to be between 0 and 10. However, as the design of each maze includes a boundary, the outermost elements of each column are not empty. Hence the range for the value to be generated becomes 1 to 9 and 1 to 19 respectively for x and y.

Food Generator:

This function calls the generator function in a loop. The purpose of this function is to control generation of the food. Generator function returns x and y values and then this function checks whether the element at (x, y) position in the array is empty (having " ") or not. In case it is non-empty, a while loop calls the generator function repeatedly until the generated location is empty. Then the value 0 is assigned to this location.

Food Checker:

This method checks whether a food item already exists in our maze. It includes a simple algorithm which searches for a specific character ('.') at each element in the array. This is done by 2 nested For loops, which run for rows and columns respectively. It returns the value true in case food exists at any element in the maze.