Name: Dean Fletcher
Date: 04/25/2021

Project Title: Snake Game

Summary of Project:

This program is my version of a game and implements various C++ programing techniques. My intent is to show how an Object Orientated program can be structured. In this version of Snake, a collision with the walls, a rock, or itself will result in a game over. The object of this game is to grow, which is done by eating, but in result makes movement inside the confined playing area increasingly difficult. In this program there are two version to play, either as a Viper or Mamba. I consider the Viper mode to be an arcade mode that follows the classic rules of the game. The Mamba mode is considered challenge mode. As a Mamba you gain the ability to pass through walls and play in a larger area but be careful because these come at a cost. In this mode you will move much quicker and there are new obstacles in the form of rocks that will need to be avoided at all costs.

For the structure of this program, it will be broken into several classes to work with each stage of the game. The game runs on a while loop until game over status is reached. Each iteration a Render function will display the board and its components to the console. Following Render is an Input function that will deal with movement of the snake. This function uses the curses library to access keyboard input to move the snake via arrows. Next a game Logic function will deal with implementing the rules the game, such as growth and collisions.

Classes:

1. **Snake:** This class is the parent class of *Viper* and *Mamba* and holds things necessary for dealing with a snake such as the body and head positions. It also controls various operations of the snake such as growth and movement via an x and y coordinate system, which is provided by objects of the *Coord* class.
2. **Viper:** This class is a subclass of *Snake* and will the arcade mode of the game. It holds a set speed and area for this version. There is no special ability such as going through walls.
3. **Mamba:** This class is a subclass of *Snake* and will be a more challenging game of the game. Through polymorphism it will modify the collision rules of the game. It plays in a larger area then *Viper* but will have the ability to go through walls and hit new obstacles in the form of rocks.
4. **Food:** This class generates food objects that are randomly placed in the playing area. This class also works with the *Coord* class to make x, y coordinates.
5. **Board:** This class holds the boards dimensions which are set by the selected game mode as well as variables like score and the game over status. This is a general use class for of the game layout.
6. **Coord:** This class is used as a base case for the x and y coordinates used in the *Snake* and *Food* classes.
7. **FileIO:** This class is used for reading/writing the High Score file and tracking high scores. This assists in keeping the main.cpp file clean.
8. **Player:** This class is used for creating player objects. This is used by the FileIO class when working with the text file. It creates a player by name and score.

Board Class

Abstract class: No
Subclass of: N/A
Composed of: N/A

Data Members

Variable Name	Data Type	Static	Description
mapArea	int	no	Holds map dimensions
gameOver	bool	no	Holds T/F the game status
score	int	no	Holds the games score

Member Functions

Prototype	Static	Virtual	Overloading Operator	Friend of this Class	Description
int getArea()	no	no	no	no	Gets playing area
void setArea(int)	no	no	no	no	Sets playing area
int getScore()	no	no	no	no	Returns current score
bool currentStatus()	no	no	no	no	Return the current game status
void changeStatus()	no	no	no	no	Changes games current status
void increaseScore()	no	no	no	no	Increments players score
friend void outputScore (Board board)	no	no	no	yes	Outputs the games current score

Snake Class

Abstract class: Yes
Subclass of: N/A
Composed of: Coord

Data Members

Variable Name	Data Type	Static	Description
advance	int	no	Contains a value used for moving the snake inthe plane
direction	char	no	Contains the current direction of the snake
snakeBody	vector<Coord>	no	Container of {x,y} coordinates for the snakeposition
snakeHead	Coord	no	Contians the current snakes head position

Member Functions

Prototype	Static	Virtual	Overloading Operator	Friend of this Class	Description
void grow()	no	no	no	no	Adds a new segment to the snake
void move()	no	no	no	no	According to the set direction updates the coordinates of each segment of the snake
int headX()	no	no	no	no	Returns the heads x coordinate
int headY()	no	no	no	no	Returns the heads y coordinate
void coordSwap()	no	no	no	no	Helper function for move() whichupdates the coordinates
virtual bool collision()	no	yes	no	no	Checks the snake's position relative to itself and the map edges
virtual int getArea() = 0	no	yes	no	no	Pure virtual function for the playing area of the selected play mode
virtual int getSpeed () = 0	no	yes	no	no	Pure virtual function for the snake speed of the selected play mode
bool bodyPosition(int x, int y)	no	no	no	no	Used for checking the segments provided an x, y position. This will help with drawing the snake
void setDirection(c har dir, int adv)	no	no	no	no	Helper function for move() which setsthe direction and advance upon a keyboard input

Viper Class

Abstract class: No
Subclass of: Snake
Composed of: N/A

Data Members

Variable Name	Data Type	Static	Description
speed	const int	yes	Holds the viper snakes speed
viperArea	const int	yes	Holds the maps dimensions for Viper snake version

Member Functions

Prototype	Static	Virtual	Overloading Operator	Friend of this Class	Description
Viper()	no	no	no	no	Constructor for setup of the Viper play mode
int getSpeed()	no	no	no	no	Returns the Viper snakes speed
int getArea()	no	no	no	no	Returns the Vipers map dimensions

Mamba Class

Abstract class: No
Subclass of: Snake
Composed of: N/A

Data Members

Variable Name	Data Type	Static	Description
speed	const int	yes	Holds the Mamba speed
mambaArea	const int	yes	Holds the maps dimensions for Mamba snake version

Member Functions

Prototype	Static	Virtual	Overloading Operator	Friend of this Class	Description
Mamba()	no	no	no	no	Constructor for setup of the Mamba play mode
int getSpeed ()	no	no	no	no	Returns the mamba snakes speed
int getArea ()	no	no	no	no	Returns the mamba map dimensions
bool collision()	no	no	no	no	This snake type has the ability to go through walls but invokes hitting rocks in the play area
void enterWall ()	no	no	no	no	Deals with the updating coords for going through a wall

Food Class

Abstract class: No
Subclass of: N/A
Composed of: Coord

Data Members

Variable Name	Data Type	Static	Description
food	Coord	no	Food object of type Coord that has a coordinate for the food x, y position
area	int	no	This holds the area of the board for random food locations

Member Functions

Prototype	Static	Virtual	Overloading Operator	Friend of this Class	Description
Food (int area)	no	no	no	no	Creates a food object in a random location on the board
int foodX()	no	no	no	no	Return the food x coordinate
int foodY()	no	no	no	no	Return the food y coordinate
void newFood()	no	no	no	no	Updates the random location for food

Coord Class

Abstract class: No
Subclass of: N/A
Composed of: N/A

Data Members

Variable Name	Data Type	Static	Description
coord	vector<int>	no	used for x and y positions

Member Functions

Prototype	Static	Virtual	Overloading Operator	Friend of this Class	Description
void setCoord (int x, int y)	no	no	no	no	Sets x and y coordinates
int& operator [ ] (int x)	no	no	yes	no	Overload operator for accessing each element of the object

Player Class

Abstract class: No
Subclass of: N/A
Composed of: N/A

Data Members

Variable Name	Data Type	Static	Description
name	string	no	Holds a player's name
score	int	no	Holds a player's score

Member Functions

Prototype	Static	Virtual	Overloading Operator	Friend of this Class	Description
Player(string name, int score)	no	no	no	no	Constructor to set a player's name and score
string playerName()	no	no	no	no	Getter for player name
int playerScore()	no	no	no	no	Getter for player score

FileIO Class

Abstract class: No
Subclass of: N/A
Composed of: Player

Data Members

Variable Name	Data Type	Static	Description
file	const string	no	Holds the file name

Member Functions

Prototype	Static	Virtual	Overloading Operator	Friend of this Class	Description
vector<Player> read()	no	no	no	no	Reads from the high score file and returns the players
write(vector <Player>)	no	no	no	no	Updates the high score text file

Demonstration of OOP Concepts

1. Encapsulation:
    a. Look to Board.h lines 9 - 11: The Board class uses encapulation by having the private data members score, gameOver, and mapArea. These data members are hidden from other areas of the program but are accessible via getters and setters.
    b. Look to Coord.h line 9: This class has private container coord.
    c. Look to FileIO.h line 12: This class has the file held as private string constant.
    d. Look to Food.h lines 12 & 13: This class had the private data members food of type Coord and the area as an integer.
    e. Look to Mamba.h and Viper.h lines 9 & 10: Mamba and Viper classes have their map area and speed data members held as private integers.
    f. Look to Player.h lines 10 & 11: This class has private data members name and score.
    g. Look Snake.h lines 12 & 13: This class has the private members advance and direction. It also has protected data members snakeBody and snakeHead.
    h. All private data members have public functions to work with the private data members as needed.

2. Inheritance:
    a. Look to Mamba.h and Viper.h line 6: Mamba and Viper both inherit the Snake class. Treating the Snake class as a base case provides the ability to add different game modes via the subclasses.

3. Polymorphism:
    a. The collision function in the Snake class is a virtual function that Mamba uses to enforce additional rule such as rock collisions. Look to Snake.h line 20 and Mamba.h line 15.
    b. The getSpeed and getArea functions in Snake are pure virtual functions which Mamba and Viper use to set up their different game modes. Look to Snake.h lines 21 & 22 as well as Mamba.h and Viper.h lines 13 & 14.
    c. In main() Snake is initialized to reference a Mamba or Viper (Look to Main.cpp lines 46 & 47). This gives the ability to treat a Snake as a Mamba or Viper. When a player chooses the gameplay mode the desired snake is passed by reference to the necessary functions. By doing this a Mamba or Viper can be passed as a Snake data type.

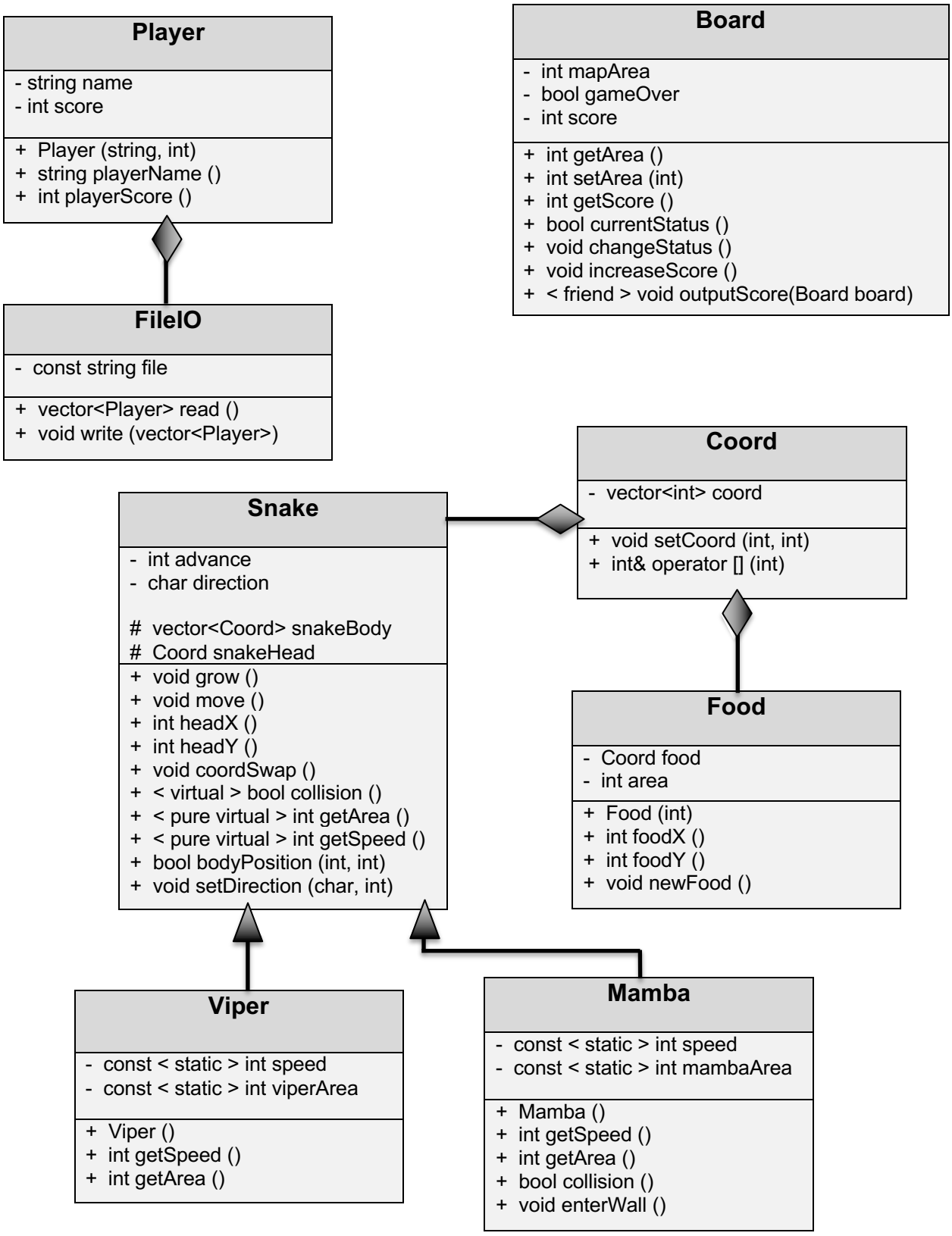
4. Static Members/Functions:
    a. Look to Mamba.h lines 9 & 10: This class has the mambaArea and speed held as constant static data members
    b. Look to Viper.h lines 9 & 10: This class has it viperArea and speed private data members held as constant static integers.

5. Friend functions:
    a. Look to Board.h line 19: The friend function is used for outputting the games current score to the console fromthe Board class.

6. Overloaded operators:
    a. Look to Coord.h line 11: The '[ ]' operator is overloaded to access each element of a Coord object.These objects will be used in the Snake and Food classes.

7. Text file(s):
    a. Look to High_Scores.txt and FileIO.h for working with that file. The text file is used for holding the 10 highest scores from past player attempts.

UML Diagram


Player
- string name
- int score
+ Player (string, int)
+ string playerName ()
+ int playerScore ()
Board
- int mapArea
- bool gameOver
- int score
+ int getArea ()
+ int setArea (int)
+ int getScore ()
+ bool currentStatus ()
+ void changeStatus ()
+ void increaseScore ()
+ < friend > void outputScore(Board board)
FileIO
- const string file
+ vector<Player> read ()
+ void write (vector<Player>)
Coord
- vector<int> coord
+ void setCoord (int, int)
+ int& operator [] (int)
Snake
- int advance
- char direction
vector<Coord> snakeBody
Coord snakeHead
+ void grow ()
+ void move ()
+ int headX ()
+ int headY ()
+ void coordSwap ()
+ < virtual > bool collision ()
+ < pure virtual > int getArea ()
+ < pure virtual > int getSpeed ()
+ bool bodyPosition (int, int)
+ void setDirection (char, int)
Food
- Coord food
- int area
+ Food (int)
+ int foodX ()
+ int foodY ()
+ void newFood ()
Viper
- const < static > int speed
- const < static > int viperArea
+ Viper ()
+ int getSpeed ()
+ int getArea ()
Mamba
- const < static > int speed
- const < static > int mambaArea
+ Mamba ()
+ int getSpeed ()
+ int getArea ()
+ bool collision ()
+ void enterWall ()