The retro game ‘Snake’ asks the user to move a snake within a set boundary, eating fruit to increase the snake’s size. The snake moves constantly, changing direction when the user inputs any of the arrow keys. The snake is not allowed to touch itself or the game border which is made more difficult when the game speed increases each time a fruit is eaten.

The snake game created uses 3 classes. These classes include the Snake class, Fruit class and Game class. The Snake class includes all the logic that allows the snake to move, be drawn and the collision logic. The fruit class has functions to change the position of the fruit in relation to the position of the snake. The Game class contains the user input.

All class variables are private, with getter and setter functions, providing increased protection of the local variables. All the functions are public.

The movement and drawing of the snake is all done through the use of 2 vectors storing the snakes previous *x* and *y* positions, and 2 integers storing the snake heads current *x* and *y* positions. The use of vectors allow for data to be added without having to update the size of the container, unlike arrays which would have to be initialized to the size of the grid. The use of the vector.push\_back() function allows for elements to be added to the vector while also increasing its size. The collisions with the wall and the snake body is done using a for loop, running over the snakes head and body, ensuring at no point the co-ordinates of the snakes head and body are the same. These collision functions return a Boolean value. This Boolean value is used to determine whether the game loop should continue to run. If at any point the game loop should stop running, game over text is printed to the screen, telling the player to press ‘esc’ to close the game window.

The fruit class utilizes the rand() function to create new positions, both on initialization and when the fruit has been eaten. The game runs through a loop to check whether any of the current snake positions are equal to the fruit’s new position, if so the fruit’s position is changed.

The game ‘Snake’ has functionality to ensure the user cannot input a direction directly opposite to the snake’s current movement. This has been implemented in the Game class’s player input. The GetDirection() function takes 1 parameter, the previous direction. An if statement is then used at each directional input to override the input if it is not valid.

The game speed is changed in the main game loop. If statements are used, checking the score (how many fruits have been eaten) and decreasing *timeDelta* incrementally.

Version Control:

<https://github.com/liamslunAIE/AIE2023/tree/main/RaylibStarterCPP>

A screenshot of a computer

Description automatically generated with medium confidence