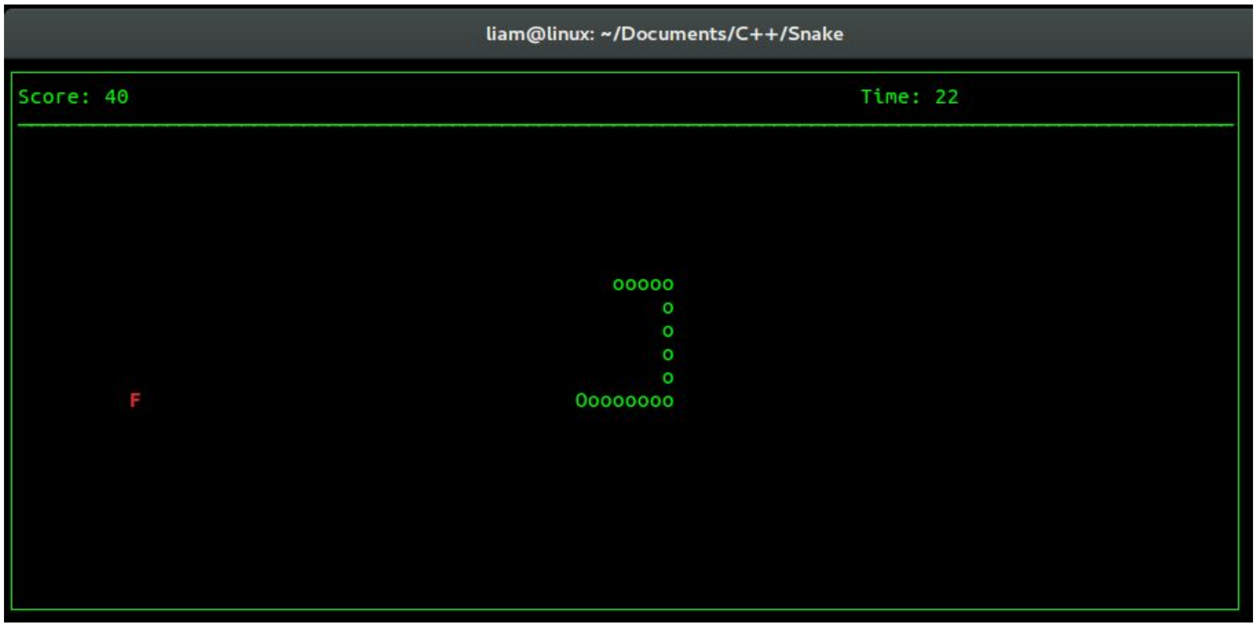
Source code: https://github.com/LiamTyler/Console_Snake .
Branches: ‘Master’ = solution, ‘Starter’ = starter code for students

Remaking the arcade classic ‘Snake’ in your terminal



```
Score: 40                                                           Time: 22
----------------------------------------------------------------------------------------------------



                                                ooooo
                                                    o
                                                    o
                                                    o
                                                    o
         F                                   Oooooooo
```

For this lab, you will be adding code to the game that is already provided for you. For those of you that dont know, “Snake” is an old classic game, where you move your snake around the screen trying to collect fruit. Each time you get a fruit, your snake grows longer, you get points, and depending on the game extra effects can happen.

Already provided: We already provide the code for following things:

1. Creating a game and playing it (main.cpp and game.h/cpp)
    a. Don't change main.cpp
2. Creating the window (windowmanager.h/cpp)
    a. You don't need to change this unless you want, just take a look at what functions it provides though
3. Moving a snake around (snake.h/cpp)
    a. Use either the arrow keys, or the w,a,s,d keys to move the snake.
4. Displaying the score and time (windowmanager.h
5. Quitting (press q) and pausing (press p, then p again to unpause)

Basically, you’re given everything but the fruits. Our fruits are going to be awesome and have all sorts of effects when you pick them up. First though, we should just make a basic fruit!

Quick note: If you segfault happen, the graphics will look messed up, and you have to open up a new terminal

A.

Make a Fruit: Make a Fruit class. Now, think about a couple things here. What do all fruits have in this game? Well, they are a 1 character symbol ('F' in the example above) that sits at a specific position on the screen. They also have a color, and a point value. Sound's like 5 parameters to me. Write a class to have these parameters. It also needs to the following methods:

1. Constructor: default and non default
2. Accessor (Getter) functions for your member variables
3. Init. This function will randomly select an empty spot to place the fruit on the screen and draw it
4. Draw. Draw the fruit's symbol on the screen at its position, with its color.
    a. For ease, we are only using 8 defined colors. See "window_manager.h" for a note about how to use colors
5. Destroy. This is what should be called when a fruit is eaten. Currently, all a fruit should do when it is eaten, return its point value.
6. Update. Our fruit currently doesn't do anything other than sit there, but future fruits will. Make the update for this class redraw the fruit.

Note: 3 - 6 above will have to take in a reference (use a pointer) to the WindowManager object so that they can draw on it, find an open space, etc.

Once you have your fruit.cpp and fruit.h ready, open up both the header and source fills for game and snake. Look at the comments titled "TODO(PartA)" for places that you will have to change to include your fruit.

B.

Moving fruit

We want to introduce some new fruit, that for one reason or another moves around. We'll call it "MovingFruit". Creative, I know. It will not only have a position, but also a velocity in its X and Y directions, as well as the other characteristics of a fruit. This means that it should inherit from Fruit. Here are the rules it follows:

1. It needs a different symbol, different color, and a higher point value than plain fruit.
2. It can only move at most 1 unit in 1 direction per frame.
3. It needs an update function, that will be called once per frame to update its position, and then draw the new position.
    a. Position is updated in many games as follows: new_x = old_x + velocity * time.
    b. For ours, this is really just x = x + vx;

4. At initialization, and every 5 frames following, it should randomly change its velocity. (Hint: Update gets called once per frame, and you can keep track of how often it is called)
5. If it is going to hit the snake or a wall (or anything other than whitespace), it should just not move (until its velocities change, and then maybe it will be able to move again)

Keep in mind that **you will have to change game.cpp again** to use the new fruit, not the default one
Also, tip: This can be annoying to debug. Before you generate the random velocities, you might want to just test it by giving it constant velocities. It shouldn't ever move into the border. Also do similar tests to see if it dies when it hits the snake. Then add the random velocities.

C.
Refactor

Did you notice any similarity between classes Snake, Fruit, and MovingFruit? They all have a position, and two of them have velocities. When you see similarity between two classes like this, it's often a sign to separate those similarities out into a parent class. Here's the reason: if we ever wanted to change the way positions are represented in the future with how our code is so far, we would have to go to every single class and make the change. If the movement was abstracted out into its own class though, we would only have to change it in one place. For this problem, do the following:

1. Move the x and y positions, as well as the x and y velocities out from Snake, Fruit, and MovingFruit into a class called "Moveable"
2. Add constructors, then add getters / setters for each of the 4 member variables
3. Make Snake and Fruit inherit from Moveable
4. Make sure your code works as it did at the end of partB

Keep in mind that you will also have to get rid of some getter/setter functions in snake and fruit, and move them to moveable.

D. SnakeFruit
Lets say we now want to make an enemy snake, that tries to eat the fruit before you can. Everytime it gets a fruit, it grows, and speeds the game up. It will keep going until it manages to eat itself, or you eat the head. Sounds kind of hard right? It's actually not. We already have a Snake class for growing and moving around like a snake, and detecting if you eat yourself. We also have a fruit class, which makes objects 'eatable' in a sense, and has effects when eaten. The problem I described above needs exactly these two things at the same time. As a result, you need to write a SnakeFruit class, that inherits from both Snake and Fruit.

Now, don't worry, you really don't have to write much code for this. The hardest part, the auto targeting the fruit, is already provided for you in "target.txt". Just copy it over into your snakefruit.cpp file. In terms of new member variables, we really only need two: a pointer to the fruit we are targeting, and a boolean to tell if the snake is dead if it eats itself. As for the functions, all of them are similar to other code, and only a handful of lines long:

1. Constructor
2. Init(WindowManager* win) -- Randomly place the snake when it spawns
3. Void GotoTarget(WindowManager* win) -- already provided for you in “target.txt”
4. Void SetTarget(Fruit* f) -- just set the target member variable to be the passed in fruit
5. Bool Dead() -- return whether the snake is dead or not
6. Void Update(WindowManager* win) -- Overload the update function to be like Snake’s update, but calling “GotoTarget” instead of “ProcessInput” since we aren’t controlling this snake
7. Void EatFruit(Fruit* fruit, WindowManager* win) -- Grow the snake, add the fruit’s points to its score, and also speed up the FPS by 1 using the WindowManager
8. Int Destory(WindowManager* win) -- when it dies, erase the snake off the screen, lower the FPS back to what it was (you can tell how many times it increased by looking at its size)

Once you have done that, see game.h and game.cpp for the “TODO(partD)”s you see, so we can actually add the snake fruit into the game.