2020/03/04 - Candy Crush

Tuesday, March 3, 2020 10:45 AM

Synopsis

- Ok, back to "~bvanderz" directories (one more time...)

Getting Started

- Copy needed files over:
  - cp ~bvanderz/cs140/Lab6/candyCrush.hpp .
  - cp ~bvanderz/cs140/Lab6/candyCrushTest.cpp .

- You will have to compile by manually typing the g++ command: g++ -o candyCrushTest candyCrushTest.cpp candyCrush.cpp

- I have made a makefile you may find at:
  - ~ssmit285/public/code/cs140_lab6_makefile (rename to "makefile").

- Above makefile also gives you a command to auto-TAR...
  - UNIX> make package

Submission Command

tar -cvf lab6.tar candyCrush.cpp candyCrush.hpp

OR

make package

Suggested Order

- getRawLength, getScore (One-liners)
- candyCrush(const string &) (Constructor)
- fillCandyList (optional helper)
- printCandy (print function)
- play (obliterate candy)

Understanding "input.txt"

256 10 Seed Raw Length
grape cherry orange lemon toffee flavours
20 10 20% chance of sequence 1. Destroy 1 candy = 10 pts.
10 20 10% chance of sequence 2. Destroy 2 candy = 20 pts.
25 30 25% chance of sequence 3. Destroy 3 candy = 30 pts.
... And so on ...

Implementation Details

- Prepare to error check... A lot...
- getRawLength / getScore
  - Literally one line. Just return rawLength & score respectively...
- candyCrush(const string &)
  - Read in file w/ format above.
  - I recommend doing this w/ C++ ifstreams. I'll explain why if asked...

- When populating probabilities, don't fill with number from file. Instead, keep a total_prob variable and increment it by the read in probabilities. Then push that instead. (Ex. {20, 10, 25, 20, 15, 10} ⇒ {20, 30, 55, 75, 90, 100}

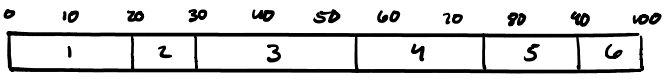
- Play smart (heh)... make helper function fillCandyList() to fill in the "candy" list. We will use this in candyCrush::play() as well.

- fillCandyList
  - Loop until candy.size() == rowLength. We will keep generating candy until this is met.

  - Call rand() IN THIS ORDER...
    1) Random candy (mod by flavors.size())
    2) Random probability (mod by 100)

  - Determine, with second random number, how many candies to generate. Let's call that number r...

  (Ex.
  probabilities = {20, 30, 55, 75, 90, 100}



  if r == 10: make 1 candy
  r == 25: make 2 candies
  r == 40: make 3 candies
  r = 95: make 6 candies
  r = 90: make 6 candies ← why? think about it.

- NEAT TRICK: resize "candy" to its size + num to insert. set second argument to the candy name. Then, if candy.size exceeds rawLength, resize so it's the same size.

- printCandy
  - Print 8 candies per line. Use printf and format as:
    - Left justified
    - Width of 10 characters
    - No manual " " space char after each name. (This is just to match the gradescript...)

- play
  - Seek-and-destroy (seriously).
  - Go to "choice", delete it and all connecting candies of the same kind.
  - Call fillCandyList to update candy with fresh new candies to crush.

(Ex. lemon grape grape grape grape orange

choice

SHIFTS LEFT

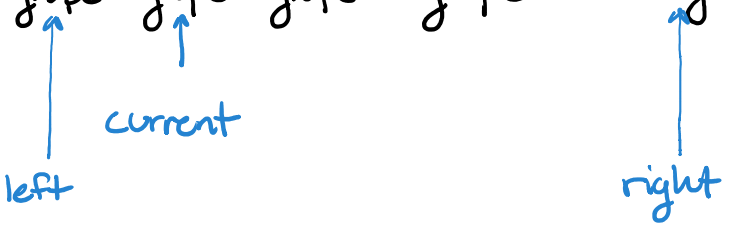
lemon ~~grape~~ ~~grape~~ ~~grape~~ ~~grape~~ orange

lemon orange toffee toffee grape grape

MADE VIA "fillCandyList"

- Make 3 iterators: current, left, right.
- Have current go to choice. Set left and right equal to current.
- Make left go to the left until candy type doesn't match, or if we are at beginning of list.
  - Ditto for right, but in opposite direction.

(Ex. lemon grape grape grape grape orange



notice "right" being one past the last "grape". this is intentional... but why?

- Use ranged deletion. Look up "list.erase c++" on Google & click cplusplus.com link for this:

1) iterator erase (iterator position);
2) iterator erase (iterator first, iterator last);

Obviously, use the second one...