Coal Project Report: Developing Candy Crush in x8086

Introduction

Candy crush is a single-player free-to-play match-three puzzle video game which was originally make for Facebook online gaming. Later on, as the fan base for the game grew, versions for Android, IOS and Windows were made. Candy crush has flourished by following the fermium model, being one of the first video games to inculcate it. The aim of the game is to "crush candies" by *swapping candies* given in a *randomly generated board* to form *combinations*. A combination is defined as the same type of candies lying in the same row, or column, consecutively 3 times or more. When a combination is made, the candies which are forming the combination are *popped*; they vanish and leave behind an empty space. The candies in the columns above the empty spaces are *compressed down* such that the candy just above the empty space takes the place of the empty space. New randomly generated candies are *dropped down* into the columns from the top of the board. A specific score is added to the main score every time a combination is made and candies are popped. The amount of candies popped in a combination is directly proportional with the score attained (a combination of 3 will give less reward than a combination of 10).

How did I do it?

We kept in mind the ideology of "divide and conquer"; separate problems, challenges, aspects associated with the game were handled by separate procedures and functions. This greatly increased code reusability and problem solving capability.

We identified that there were going to be basically two sides two our project, the back end and the front end. The back end consisted of basic logic of how a grid can be represented as a *matrix* (2D array) of numbers, where corresponding to every number there will exist a shape (candy). The manipulations on that matrix could simply be reflected on the front end to make the game interactive. The functions to manipulate the matrix and other helping functions are described below.

1. Random Number Generator: this function took the clock ticks of the CPU since midnight as a seed and manipulated the seed to generate a random number in a given range. This function was used to create random candies to fill the matrix.
2. Column Compressor: this function took a column number and pushed the candies down to fill up empty spaces, essentially bringing the empty spaces to the top.
3. Column Empty Space Checker: this function took a column number and returned 1 if the given column had an empty space, else it returned 0.
4. Column Candy Dropper: this function took a column number and simply dropped a random candy at the bottom most empty space of the column
5. Swap Candy: this function took coordinates of two candies (in the matrix) and swapped their places
6. Column Combo Checker:
7. Row Combo Checker:
8. Grid Combo Checker:

9. Grid Filler: this function simply calls column compressor function for every column and then column candy dropper function for every column till every column is not empty.
10. Pop Candies:

The functions which were used to form the front end grid, reflect the background manipulations on the matrix onto the front end interactive grid, and the functions to provide a user friendly environment to play the game are described below.

Interrupts Used

Following is a list of the interrupts used:

INT 10H - AH=02H - set keyboard cursor position
INT 33H - AX=07H, CX=x1, DX=x2 - set horizontal limit to cursor
INT 33H - AX=08H, CX=x1, DX=x2 - set vertical limit to cursor
INT 21H - AH=04CH - return DOS control
INT 21H - AH=02H, DL=char - print a character to screen
INT 21H - AH=03DH, AL=02H, DX=filename - open a file
INT 21H - AH=042H, AL=02H, CX=0H, DX=0H - move file cursor
INT 21H - AH=040H, CX=strLen, DX=str - write string to file
INT 10H - AX=0EH - make a video screen
INT 10H - AH=06H, AL=0H, BH=0DH, CX=0, DH=01E, DL=064H - clear video screen
INT 21H - AH=0CH, AL=0H - flush buffer
INT 16H - AH=01H - check for keystroke input
INT 33H - AX=03H - get mouse position
INT 33H - AX=0H - hide mouse
INT 33H - AX=04H - set mouse cursor position
INT 33H - AX=01H - make mouse visible
INT 33H - AX=05H, BX=0H - get mouse click stats
INT 15H - AH=086H, DX:CX=delayTime - cause the required amount of sleep
INT 10H - AH=0CH, CX=x, DX=y - write graphics pixel
INT 1AH - AH=00H - get system clock ticks
INT 21H - AH=01H - get keyboard character input