QuadBoom

Or: How I learned to *never* rely on even professional-grade software to make basic sense

And then how I dealt with it anyhow

Overview

GOAL

- Implement the basics of Puyo Puyo on the Xbox 360 game console.
- Not developed for a client or assumed target audience.

WHY?

- Because it was a fun and interesting challenge working in a new environment.
- Also, the game hadn't been done on the 360 when I started...
  - But has now. Oops.

Game Summary

The game keeps itself pretty simple:

- Pairs of pieces of various colors drop into a 6*12 field. How they fall is manipulated by the player.
- Anytime four pieces of the same color are touching, they are removed from the board – pieces above the removed pieces falling to fill the gap.
  - This can result in chains of pieces being removed.
- Gameplay can be single-player, though typically it is versus.
  - Pieces one player removes get placed as difficult-to-remove 'trash' tiles into the other's field.

Data Representations

- The game, for play purposes, has just one unique data object: The actual tiles, known as Drops, given the class "DropObject"
  - Among other contents of the DropObject:
    - The location of the Drop on the field
    - The Drop's color (and what graphic represents that color)
    - What the drop is doing
- The game also heavily uses Enumerations – in fact, both the color and motion of a Drop are stored as Enumerations.

Other Structures

- The game uses two basic types of graphical structure to draw the screen:
  - The actual graphics are stored as two-dimensional textures, which are easy to manipulate.
  - Rectangles are used to specify where each texture is drawn – this allows for automatic, optimized scaling of items. (Vectors and an integer can also be used for scaling, but does *not* allow for different scales in two directions.)

Active Search

- The game needs to be able to run a basic, looping search at the end of every move to verify if there are groups of four...
  - And then run it again after those groups are cleared, to determine if a chain occurs.
  - There's many ways to do this, and I've prototyped, but not tested, a few.
    - Currently favoring a union-graph structure – put each tile in a group of one, merge adjacent groups of the same color, and if the size exceeds 4 after the search is done? QuadBoom!

And then the problem...

- Microsoft XNA Studio is the only (official) way to code for the Xbox 360 (at least, for Microsoft Indie Games). In this case, I used it with Microsoft Visual Studio 2008.
- The problem is... XNA isn't exceptional in its consistency, *especially* with draw position handling.
  - I have been stuck for *four weeks* trying to get it to accept that the same screen coordinate, drawn twice, doesn't draw in two different locations.

System Architecture

- The game's initial logic opens a general class, ScreenManager.
  - The ScreenManager in turn operates a GameScreen metatype, which is the parent class for specific screens such as:
    - The background
    - The actual gameplay
    - The pause menu
    - Etc. Etc.
  - Notably,this design can run screens on top of one another simultaneously – so the pause menu can run atop the gameplay screen, atop the background.

Prototyping

- Prototyping has been the game model from the start... Far finer than I'd initially planned.
  - Get the ScreenManager sample working, make sure I understood it.
  - Get a nicer background in place. (This one, in fact!)
  - Get the game to render the Background
  - Get the frames rendering
  - Get the icons moving
  - You get the idea...

Conclusion

- Microsoft programming resources are a pain.
  - What else is new?
- I am capable of learning a new programming language (if one simliar to one I already know) and programming environment (not similar to existing ones) in a short timeframe.
- I am not so good at estimating the time it'll take to finish assignments, unfortunately.
  - Single-player should be done by the final code demo date! (See previous statement)