Anders Mikkelsen

Breakout

Assignment:

Produce a Breakout clone with certain parameters.

Dependencies:

- SDL2 (PG4400_SDL2_PATH)
- SDL_Image (PG4400_SDL2_IMAGE_PATH)
- SDL_TTF (PG4400_SDL2_TTF_PATH)
- SDL_Mixer (PG4400_SDL2_MIXER_PATH)

Documents:

- This
- Doxygen documentation in $/docs/doxygen (Tools -> &View DoxyGen)
- Performance.psess in $/docs/Profiling
- Performance Report in $/docs/Profiling
- Dr. Memory results in $/docs/Profiling/results.txt
- Class Diagram in $/docs/ (accessible in Breakout Project in filter Class Diagram)

Preconditions:

As I usually follow the google code style, I used it for C++ as well with a few differences. I have later figured out its slightly outdated, but still probably ok for my needs as this point in time. I use a 100-character breadth when I code. Most of the sources if used for information have pretty much banned “using namespace” because of version issues and so does the google style guide so I have refrained from using it myself. I have instead used typdef’s in the case where a lot of std:: makes the code hard to read. In the .cpp’s I always type out the types verbosely.

I also researched my way to find out that pragma once is compiler and system specific but usually quicker so I made my own ifndefs as well as using pragma once so my game is more multi-platform.

Design:

I have previously used a pattern of having a singleton Game Manager as a primary communicator in the games I have produced and I think it works quite well while upholding Low Coupling and High Cohesion so I ended up doing that here as well. I also did this to highlight that I know how to implement and use a singleton. Only difference in C++ is really the privatization of the constructor, copy and move assignments. I suppose one could implement it in the constructor as is customary in C#/Java but I think the Meyers pattern is simple and elegant. The Game Manager ensures Low Coupling by being the sole object any

other object in the game communicates with for all interaction and has itself High Cohesion as it only does computing of info from one object to another when necessary.

I believe I have generally been able to create Low Coupling and High Cohesion throughout my application. The SDLvideo and SDLsound facades are the most prominent examples as they only do work on their respective pieces of SDL and abstract away the logic when communicating with other objects.

Next, I figured I wanted to play around with Multithreading/Lambdas and more game specific DP's.

Multithreading was much easier than with pthreads in C, and a lot more intuitive especially when coupled with lambda expressions. I did some tests with threading the updates of components individually but that required a ridiculous locking system and severely decreased performance. This was to be expected as the time it takes to start and finish a single thread is probably longer than it takes to handle any of my components, except maybe the movables.

Implementation:

Game design patterns I have implemented, heavily influenced by this site:

- Flyweight
- Singleton
- State (To some degree, but very simple)
- Game Loop
- Update Method
- Component
- Data Locality
- Spatial Partition (Slightly modified, I only check the grids of movables. When you do not have physics and "everything" is a movable, I believe this increases performance over iterating the entire grid.)

Just as when I started learning C I have spent quite some time just getting used to the syntax and way C++ is put together. Especially when to use const correctly has been an interesting adventure. (I believe I am using it correctly now, but I would like to know if I am wrong.)

What particularly started my learning experience when it comes to understanding C++'s strengths was when I implemented the Data Locality pattern and understand that the power to allocate memory contiguously and exactly as large as you want to can greatly improve performance. This was around the same time I started work on the Collision Manager with its Spatial Partitioning Grid. This was a great way to get familiar with a new DP and at the same time get a good feel for how passing arguments in functions and accessing objects work in C++. Great learning experience!

I eventually decided to try to get a good idea of when you should inline. As I understand it, you should inline short methods, used very often, in few places. If you inline methods that

are used many times in many places you can quickly bloat your executable. As far as I understand this method should then rather be made static and directly accessible from a single area in memory, if that is even possible. The short methods I have been unsure about I have added the inline keyword to and left that decision up to the compiler.

In general I have used the header to document the general functionality of my classes and their functions, and written detailed comments in .cpp where needed to explain core functionality.

I have tried to implement as many subjects of C++ as I could so some calls might seem unnecessarily different, like calling a thread on a method or a lambda. Where this is the case it is simply to highlight I understand the differences.

Profiling:

I have run the game through Dr. Memory for leak tests of which there was none. There were some errors for UNINITIALIZED READ, but internet research suggests these are simple false positives. Either way it does not affect performance at all. I am not entirely able to understand everything that goes on in the profiler in visual studio but there is a report in the doc's folder. Some old school tests via my debug settings show an average game loop speed of 0 ms, and task manager shows a CPU usage of 0 – 0.1% usage. I believe this is very good! If I turn off FPS synchronization the game runs at ~1500 fps on my machine.

Code Analysis is turned off for faster build. Clears most cases, only a few unimportant info messages.

Conclusion:

This has been fun! I like C++, or at least have come to like it. I think it really removes most of the annoying details of C, which cause frustration, and lets you get on with creating solid, high performing software.

Known Issues:

After implementing my custom timer I saw a clear "jump" or jitter/stutter on my rendering about every 1 sec. I did thorough tests and found that all my objects were moving as they should. Eventually after a lot of research I found out that the problem is in OS scheduling. No matter what level of precision I have on a timer it's always going to be "at least" that time. This paired with my time not being exactly vsynced caused this lag. So what I ended up doing was having SDL sync for vsync and reducing the time of my timed wait in the game loop by a small amount. This makes my game loop wait its excess time, and then it will wait for the async SDL thread that is waiting for vsync. This way I get very little CPU usage and at the same time ensure sync with vsync. This scales with different refresh rates, on primary display. I would appreciate feedback on whether you think this is a good solution.

Upon changing to a very low resolution (about 640 x 480) you will start experiencing bugs. The game scales a lot better on higher resolutions. It might seem weird from a gameplay

standpoint that a bigger resolution generates more bricks but I was more interested in showing I could scale the game than making it gameplayably epic.

The collision detection is not perfect. It could use a narrower scope. You will see this when the ball collides with a corner. It might take another brick with it or in the case on single bricks simply pass right through. I would probably have to implement a pixel based narrow scope or something like that, but I think I have proven that I can make a simple CollisionManager.

Extras:

The installer project builds an installer to /BreakoutSetup/Release. Finished build is NOT included in zip. Just right click the project when in release and press build to build the installer. This is because I would surpass the MB limit of its learning.

External resources:

Sprite sheet: http://opengameart.org/content/breakout-graphics

DP info: http://gameprogrammingpatterns.com/contents.html

Font: http://www.dafont.com/

Sounds: http://www.freesound.org/

Music: Free Chip tunes from way back in the day.