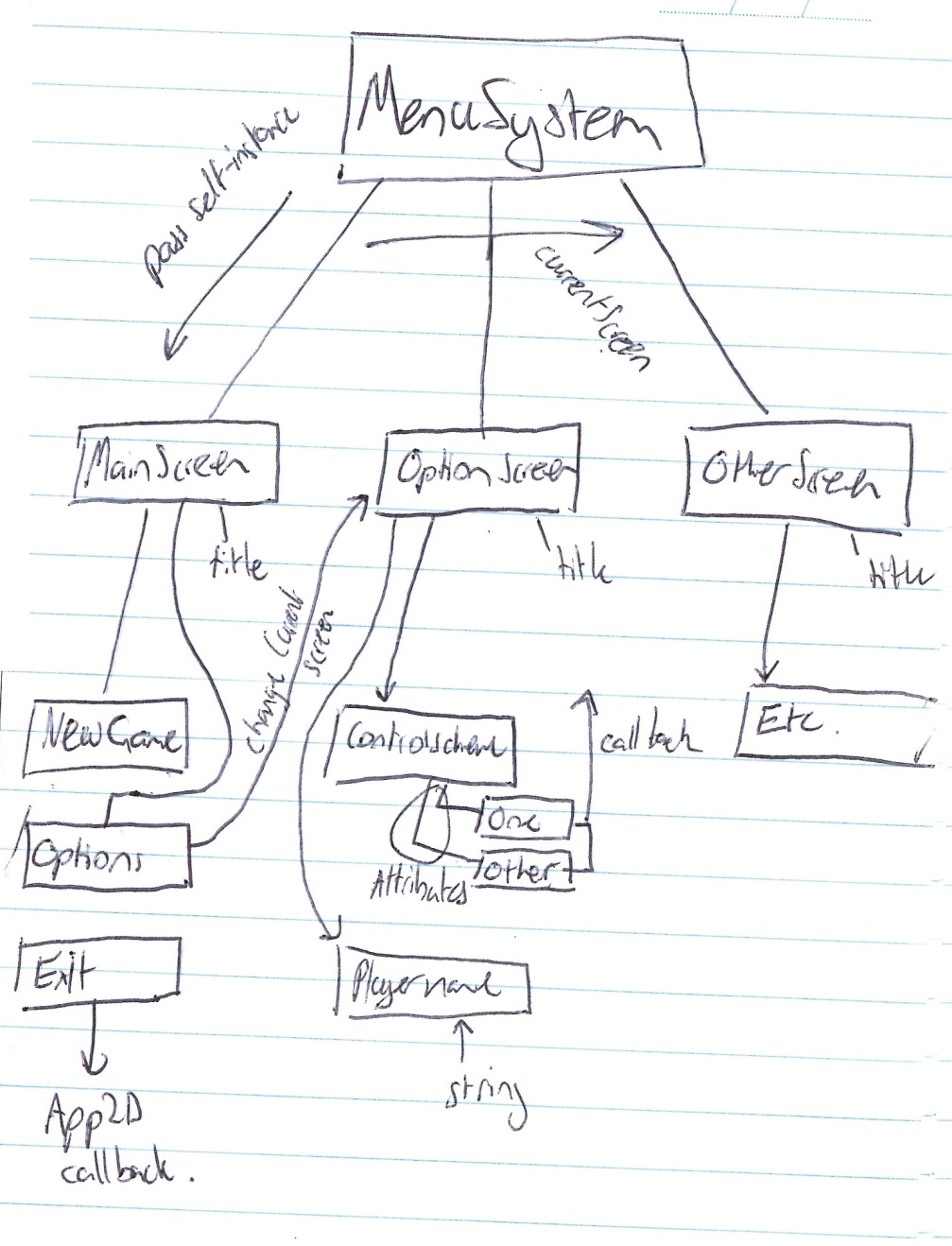
**Menu System** – Inside Seb’s Mind

*Seb Holzapfel*

Right, so; menu system eh? Better get to it, this isn’t much of a game without a menu..

Thinking of a pretty ordinary menu, title up the top, various items that highlight and can be clicked on to change options, perform actions and visit other menus.

I reckon the best way of doing things is probably just to slap a fat hierarchy together and read individual item-datasets back up the chain for parents to handle. Actually, yeah that should work, I can allocate smart pointers at each level and program in different menus with only a couple of lines once the boilerplate code’s done. Looks like it’s pen & paper time for an object sketch:



So. My main class will be MenuSystem; and will keep pointers to various MenuScreens (alongside a currentScreen pointer), each MenuScreen containing various MenuItems, and each MenuItem containing a list of possible states if it is to be treated as an option (probably determined by checking whether an item actually *has* any states).

\*Brainlessly translates the paper architecture into code

Right; so now I can define and display menus using my virtual classes. Here’s a specific example of my defining of the ‘control scheme’ item from the options screen:

class ControlScheme : public MenuItem {

public:

ControlScheme( App2D &a, MenuSystem &m ) : MenuItem(a, m) {

name = "Control Scheme";

attributes.push\_back("Global Movement");

attributes.push\_back("Mouse Relative");

InitializeToCurrent(attributes, currentAttribute, "ControlScheme", app);

}

virtual void Clicked() {

CycleOptions(attributes, currentAttribute, "ControlScheme", app);

}

};

Pretty easy; set the name, push on some attributes, initialize and cycle options when clicked. Note that InitializeToCurrent and CycleOptions are functions that I wrote to [initialize an option to it’s current game-state value] and to [cycle possible options and make the result visible to the rest of the engine’s logic so other areas of the game can look at option values] – they’re only a couple of lines each but syntax-heavy so I won’t include them. The ControlScheme class above is instantiated in the OptionsMenu:

class OptionsMenu : public MenuScreen {

public:

OptionsMenu( App2D &a, MenuSystem &m ) : MenuScreen(a, m) {

title = "Options";

items.push\_back( boost::shared\_ptr<MenuItem>( new ControlScheme(app, ms) ) );

items.push\_back( boost::shared\_ptr<MenuItem>( new PlayerName(app, ms) ) );

items.push\_back( boost::shared\_ptr<MenuItem>( new Directory(app, ms) ) );

items.push\_back( boost::shared\_ptr<MenuItem>( new GoBack(app, ms) ) );

}

};

Looks complicated, but really it’s just setting the menu title and adding each of the specific menu items (You can see ControlScheme is in the first line). And then finally we have the constructor for MenuSystem, which loads menu screens:

menus.push\_back( boost::shared\_ptr<MenuScreen>( new MainMenu(app, \*this) ) );

menus.push\_back( boost::shared\_ptr<MenuScreen>( new OptionsMenu(app, \*this) ) );

DisplayMenu(Menu::MainMenu);

You can see the OptionsMenu being instantiated in the code above; even though the menu displayed at first is MainMenu instead of OptionsMenu. To jump to OptionsMenu, here’s the Clicked() function of the Options item class in MainMenu:

virtual void Clicked() {

ms.DisplayMenu(Menu::OptionsMenu);

}

Easy! And the boilerplate I wrote behind this isn’t actually that complicated either; well, at least not too much :P.

For example, here’s the largest boilerplate function used in this system (Don’t faint, it’s really not that bad)

void MenuSystem::Draw() {

currentMenu = newCurrentMenu;

// Draw title

mTextOSS.str(""); //Clear OSS

mTextOSS << currentMenu->title;

mText.SetText( mTextOSS.str() );

mText.SetSize(40);

mText.SetPosition(0, 0);

mText.SetPosition(app.GetSize().x/2 - mText.GetCharacterPos(mTextOSS.str().size()-1).x/2, app.GetSize().y/2-currentMenu->items.size()\*60 );

mText.SetColor( sf::Color(255, 255, 255, 255) );

app.Draw(mText);

// Iterate through each item and draw them

for(int i = 0; i != currentMenu->items.size(); ++i ) {

mTextOSS.str(""); //Clear OSS

mTextOSS << currentMenu->items[i]->name;

// Do attribute processing

if(!currentMenu->items[i]->attributes.empty()) {

mTextOSS << " [" << currentMenu->items[i]->attributes[currentMenu->items[i]->currentAttribute] << "]";

}

mText.SetText( mTextOSS.str() );

mText.SetSize(25);

mText.SetPosition(0, 0);

mText.SetPosition(app.GetSize().x/2 - mText.GetCharacterPos(mTextOSS.str().size()-1).x/2, i\*50 + 50 + app.GetSize().y/2 - currentMenu->items.size()\*50 );

//Check whether item is hovered over, pressed, or clicked

if( mText.GetPosition().x < app.GetInput().GetMouseX() &&

mText.GetPosition().x+mText.GetCharacterPos(mTextOSS.str().size()-1).x > app.GetInput().GetMouseX() &&

mText.GetPosition().y < app.GetInput().GetMouseY() &&

mText.GetPosition().y+mText.GetSize() > app.GetInput().GetMouseY() ) {

currentMenu->items[i]->hovering = true;

if(app.GetInput().IsMouseButtonDown(sf::Mouse::Left))

currentMenu->items[i]->downed = true;

if(!app.GetInput().IsMouseButtonDown(sf::Mouse::Left) && currentMenu->items[i]->downed) {

currentMenu->items[i]->Clicked();

currentMenu->items[i]->downed = false;

}

} else {

currentMenu->items[i]->hovering = false;

currentMenu->items[i]->downed = false;

}

// If an item is hovered over, change it's alpha

if( currentMenu->items[i]->hovering )

mText.SetColor( sf::Color(255, 255, 255, 255) );

else

mText.SetColor( sf::Color(255, 255, 255, 190) );

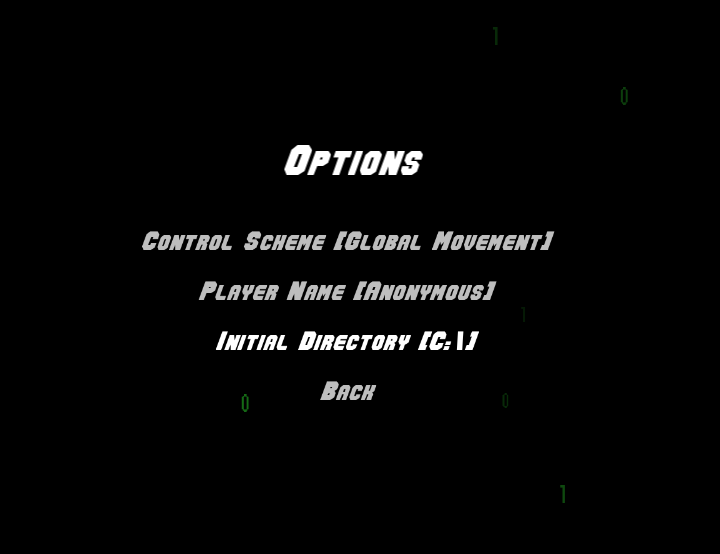
app.Draw(mText);

}

}

Follow the comments and it shouldn’t be too hard to understand what’s going on; C++ just likes to tease normal people with its syntax, heh. Basically top down, make sure the current menu is correct, draw the title, draw each item, check hovering and call callbacks. The whole ‘downed’ business is to make sure an item is only ‘Clicked’ if the mouse is pushed down *and* released on an item.

And; for the finale: fruits of labour! – (At the time my mouse was hovering over the 2nd last option but fraps didn’t display the cursor for some reason)



Spent a couple of minutes linking it to the ‘escape’ button as well for pausing:



It was a little more difficult than I thought to do this, I needed to figure out a way to freeze everything in the game except UI items. So I stuck a Boolean in the global event ‘step’ function checking for UI and a paused state, withdrawing from calls if the game is paused – worked a treat, amazingly!