Sprite Machine

By: David Silva

17B-48597

**Summary**: My project reads sprites from xml files and then displays the usage of them with the keybinds Z, X,C and arrow keys.

**Statistics:**

Line count: ~1200

Comment: Most of my work is in battlearea.cpp and battlearea.h

**Controls:**

Move: up, left,down,right arrow keys

Other moves: Z, X, C

**What’s used and where:**

#include <QLabel> - used to create splashscreen

#include <QWidget> - Used throughout project

#include <QFile> - Used to read the images into qdomdocument

#include <QPixmap> - used to store images

#include <QtXml/QDomElement> - used to parse xml files

#include <QDebug> - used for debugging

#include <QtXml/QDomNodeList> - for parsing xml files

#include <QtXml/QDomDocument> - for parsing xml files

#include <QtXml/QDomNode> - for parsing xml files

#include <QTimer> - used to go to the next frame

#include <QObject> - used throughout

#include <QPainter> - backbone of my project to instantiate paintEvents

#include <QElapsedTimer> - to time the fps

#include <QImage> - used to flip images

#include <QtGui> - gui

**How my project works:**

**Reading the xml files:**

I start by reading the xml files, which I had to learn xml for and learn how to use Dom Documents to parse xml, that tell where each individual sprite is located on the given source sprite sheet. With those numbers, x position, y position, width, and height, put into a QRect I translate the whole sprite sheet into a QPixmap to ready the program to crop out the individual sprites. Then, I continually crop out each of size nRow.length(), which is the number of sprites needed to complete the given sprite animation. I also read in offset points so that each sprite is aligned with the previous sprite that was just displayed so my animations do not look choppy. From then I put each array holding a given sprite animation into an array belonging to the class that is ready to store the information. I have one function that reads all xml files ( load(Qstring File) located in battlearea.cpp lines 264 - 433). Since I have one function that reads all xml files all xml files had to be uniform in format.

**Movement and Animation:**

Movement was tricky. I had to check whether a button was being held down, which qt provides no way to do, change the sprite each time my timer timed out, and then move the animation at a reasonable pace. I used a QPaintEvent( located in battlearea.cpp lines 85-134) to paint the sprite to my widget. I had to also have a control enumerated variable to tell my paintEvent which sprite array to paint from because I have multiple sprite animations. With all that preparation I was ready to start the animations.

**paintEvent explained lines(85-134) of battlearea.cpp:**

I create a temporary QPixmap and QPoint because I know I need to hold the individual sprite of whatever animation is requested to start. After I copy the correct sprite I paint it to the widget. However what if the user required my sprite to go left? Then I had to flip the image and also had to create another variable to tell my program which way the sprite was already facing. For holding which way they were facing I used the enumerated value player1 = right by default and then another variable to hold current direction which could be either right or left, but is right by default. Now, in order to mirror the image QPixmap has no way of doing (that I found). I had to go through a large amount of image handling documentation find out that to mirror the image I had to convert it to a QImage first (refer to lines 121-124 of battlearea.cpp).

**keyPressEvent & keyReleaseEvent explained lines( 136-262 ):**

Here I handle the timer->start(), x positioning ( no vertical movement in program), and direction of the player. I had to find out about the event being auto repeat because otherwise my timer would keep restarting as long as the key was pressed, which I did not want. I just wanted it to start on the first key press to tell the timer to start and to move the sprite as long as one of the movement keys was pressed. In order to do that I had to have separate parts within the vent; event->isAutoRepeat() and !event->isAutoRepeat in ! auto repeat, which is the first press of the key I would start the timer. In auto repeat I would keep incrementing the x position if the keys pressed were movement sprite animations. In the keyReleaseEvent I would just tell the timer to stop, reset the sprite index to 0 and to repaint the event with the index at 0 so the sprite is in a resting state.

**How I animate:**

I connect the timer’s timeout signal to my nextFrame ( lines 435 - 469) slot. When my timer time’s out I increment my index and repaint the object if the user is still holding down the button and the index is about to go out of bounds (based on which animation is playing the index for each full sprite animation is different) then I reset the index to a specified point.

**Splash Screen (lines 14-27):**

I use a qpixmap to store the splash screen image, use a qlabel with the flags qtframelesswindowhint and qtwindowstaysontophint and set the attribute of the qlabel to qt::WA\_translucentbackground so that way it appears se through except for the image itself. It was time consuming to figure out because even though qt does provide a way to make a widget appear opaque except for the image itself, that way is bugged (according to forum posts I found). The other challenging thing is that centering the splash screen on the main screen was also time consuming to figure out. The QLabel itself does not center onto the main computer screen automatically. So I had to create a QDesktopWidget and feed the dimensions of only the mainscreen of a given computer into a QRect. The reason I am mentioning the main screen or primary monitor so much is because if a given person has more than 1 screen (as I do) then centering the splash screen based on qDesktopWidget::availablegeometry will end up placing the image where the two screens meet because that is the middle of all of the available geometry for qt to work with. So I had to figure out how to only get the dimensions of the screen set to primary on windows and then center the splash screen based on those dimensions.

**Settings:**

I never got them to work properly. It writes user input to a file and reads them from the same file but doesn’t actually change the settings.