Programming Project #3
CpSc 8700: Software Development
Computer Science Department
Clemson University
Polymorphism, Factories, and
The Tracker Framework
Brian Malloy, PhD
September 28, 2015

In order to receive credit for this assignment, your solution must, as a minimum, meet the requirements specified in this document, and be submitted with the web `handin` command, by 8 AM, Thursday, October 22nd, 2015. If you cannot complete the project by the due date, you can receive 90% of the grade if you submit the assignment within a week after the due date.

In the third project, you began with a basic animation framework composed of 9 classes that permitted you to efficiently bounce a bunch of sprites around on a screen or canvas. The efficiency derived from reusing the Frame, which wraps an *SDL_Surface* for all of the sprites that use the same *SDL_Surface*; the Sprite class wraps the frame, and by setting different (x, y) coordinates in the Sprite class, you drew each frame at a different (x, y) location on the screen. Minimizing memory and resource usage by sharing data in this manner is an example of the Flyweight design pattern, and reusing the Frame class makes it a Flyweight.

Additional Design Features:

In this fourth project, you will use a tracker framework that extends the basic animation framework by adding 4 new classes: FrameFactory, World, Viewport, and MultiSprite. The FrameFactory class, when completed, formalizes the reuse of the Frame by encapsulating it in a factory that uses an STL *map* to store frames, and another *map* that stores surfaces. The idea here is that all frames are obtained from the FrameFactory, which when asked for a frame first searches the map to see if the frame has already been constructed; if the frame is in the map a pointer to the frame is returned; if the frame is not in the map the FrameFactory will build the frame, insert it in the map for subsequent requests, and return a pointer to the frame to fulfill the request. Note that there is an SDL_Surface for every frame.

Probably the best approach to migrating from your third project to the fourth project would entail incorporating your assets from Project #3 into the new tracker framework. By doing this, you will be moving frame and surface management out of the Manager class and into a central location, FrameFactory, written expressly for this purpose. The 4 new classes are described in the next section and the code can be found in the course repository.

Additional Animation Features, Tracking a Sprite:

Two of the additional classes, World andViewport, form the tracker part of the framework, where the World class is a container for a background sprite, and the Viewport class is used to provide a view into the world, and tracks a specific sprite in your animation. The tracker framework includes functionality to permit the player to track any or all of the sprites in your animation.

The intention is that the background sprite will be much larger than the size of the viewport and the World class will draw the background based on the size and location of the Viewport, which tracks one of

the sprites in your animation. In the current implementation of the World and Viewport classes, the Viewport will track one of the sprites and will follow it wherever it goes; pressing the *t* key will make the viewport track a different sprite. The sprite that is being tracked can move vertically from y = 0 to y = width of the background sprite, but if your background sprite is seamless on the edges, you can scroll endlessly in the horizontal, x, direction. For example, I have set the width of the world in the tracker framework to be $1,000$, so the Viewport will track a sprite from $x = 0$ to $x = 1,000$. If you prefer, you can make the viewport wrap so that the sprite moves from $x = 1,000$ to $x = 0$. The world and Viewport classes can be modified or extended to scroll endlessly in the y direction, or both the x and y direction. To see this, change the world/width parameter in the file game.xml. On the web page for our course, you will find a video, made by Kara Gunderson, that describes how to use gimp to make a background that is seamless and will enable you to scroll endlessly in the x direction and, by generalizing, in the y direction.

The final two classes in the tracker framework are MultiSprite and FrameFactory. Incidentally, in this framework we also use ExtractSurface, which was included in the 9 classes in the Basic Framework. The MultiSprite class contains a vector of frames that is used to draw a multi-frame sprite. The "frames" for a multi-frame sprite are generally taken from a *sprite sheet*, which is a canvas upon which a graphic artist draws many sprites. However, to facilitate some important actions that we want to incorporate into our game, such as per-pixel collision detection, and rotating multi-frame sprites, we need to extract each frame out of the sprite sheet into an *SDL_Surface*, and then put each surface into a Frame class, and then insert each frame into the vector stored in the FrameFactory and supplied to MultiSprite. The ExtractSurface class is currently being used by the Manager, but you should move this responsibility to the FrameFactory class, which should extract frames of a multi-frame sprite from a sprite sheet.

You are not required to build your own sprites for this project but you can receive additional points if you do. Also, please note that you **may not** use the sprites that I use for demonstration purposes. If you use sprites or sprite sheets that you did not build yourself, then you must acknowledge the author or site from which you obtained the sprites in the ASCII README file included with your submission.

Requirements Summary:

1. Provide an option in class Clock to cap the framerate at 60 fps. Test thoroughly.
2. The IOManager class uses the *explicit instantiation model* for making a class from a template class. Change IOManager so that it uses the *Inclusion Model*.
3. Use the Tracker Framework to provide a viewport into a larger world of your animation. Provide a facility to permit the user to track at least two different objects in your animation.
4. Incorporate at least three kinds of Sprites into your animation, including a two-way multi-frame sprite. You should use a vector of Drawable* in your manager to polymorphically store the two kinds of sprites. If you do not build your sprites, you **must** cite the source of your sprites.
5. Animate lots of sprites, including one or more two-way multi-frame sprites.
6. Refactor your design to move the responsibility for managing your animation assets into a *factory*, which is partially written for you. To complete the *factory*, you must complete the code for the destructor in the FrameFactory, and you must modify classes in the framework to use the FrameFactory; this will entail changes to Manager, Sprite, MultiSprite, and World.

7. Currently, the multi-frame sprite doesn't move. Fix this by adding code to MultiSprite::update.
8. Virtually all of the constants and parameters to your animation should be read from an XML file so that they may be adjusted without re-compiling your program.
9. Include an ASCII **README** report.
10. Submit, using web handin a directory containing your source code, your animation assets (which may **not** include any of my sprites), a README report, and a subdirectory containing all animation assets.
11. **video** (10 points) (1) Do not generate frames, (2) filename has format <**username**>.{*nnnn*}.*bmp* (Clemson userid), (3) set frameMax to 200,
    (4) Your name printed clearly on the lower left corner of the screen.
12. A Makefile that uses the Meyer's flag, -Weffc++, and -Wall.
13. All warnings eliminated from user code.
14. No memory leaks in user code.

Of course your assignment must be written in $C^{++}$and you must use the Simple Directmedia Layer (SDL) gaming API. You should submit the entire directory that contains your code, Makefile, animation assets, and a README file, using the web handin facility.

Future Projects:

1. Project #4
   - A HUD
   - Parallax scrolling
   - Player
   - Painter's Algorithm
2. Project #5
   - Object Pooling
   - Projectiles and shooting
   - Collision detection
   - Explosions
   - Music and Sound
3. Project #6 (final project)
   - A menu of options: help, modifications, ...
   - AI
   - Restart
   - Score/outcome