Programming Project #2
Polymorphism in Action
The Tracker Framework
CpSc 4160/6160: Data-Driven 2D Game Development
Computer Science Division, Clemson University
Brian Malloy, February 29, 2016

Due Date and Submission:

To receive credit for this assignment, your solution must meet the requirements specified in this document, and be submitted, using the department `handin` facility, by 8 AM on Friday, March 4^{th}, 2016. If you cannot complete the project by the due date, you can receive 90% of the grade if you submit the assignment within three days after the due date.

Your submission should be a directory, compressed with either tar or zip, containing your program files, assets, README, and a Makefile. Be sure to 'make clean' before compressing the directory. Your program files must be written in C^{++}, use the Simple DirectMedia Layer (SDL), and contain no memory leaks in code written by you. You can compress the directory in 1 of 2 ways:

```
tar zcvf asg2.tar.gz asg2
zip -r asg2.zip asg2
```

Project Specification:

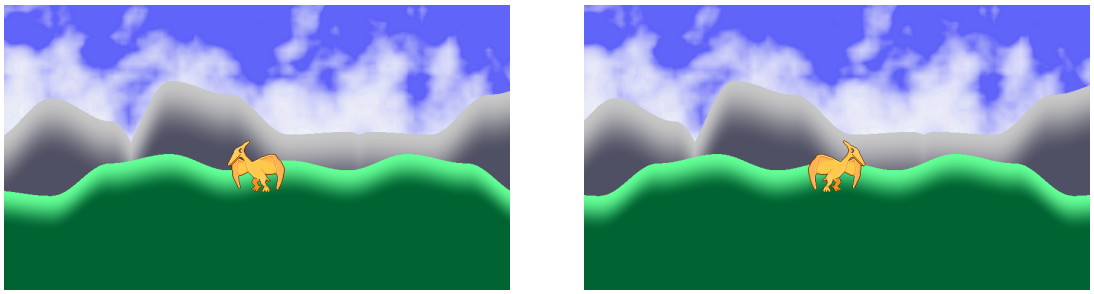
As a starting point for this assignment, you should study the data-driven *tracker framework* code found in the course repo. This framework consists of 13 classes that include singleton classes, a factory class, and a manager class that contains a polymorphic vector of drawables. To succeed in this assignment, you should complete the following tasks using the *tracker framework* as a starting point:

1. Print your name and the title of your animation in the lower left corner of the screen.
2. Some of the classes in the *tracker framework*, for example Sprite and Viewport, use global constants WORLD_WIDTH and WORLD_HEIGHT. Convert all classes in the animation to use data-driven programming, so that all constants used in the framework are read from the XML file game.xml.
3. Some of the classes in the *tracker framework* use the GoF singleton. Convert them to the Meyer's singleton.
4. A warning is generated for class Sprite because it doesn't have an overloaded assignment operator. Write an overloaded assignment operator for class Sprite.
5. Incorporate parallax scrolling into the animation by using at least two backgrounds.
6. Extend the Clock class so that the animation can be paused with the p key.
7. Class IOManager has a template function printMessageValueAt:

```
template <typename T>
void printMessageValueAt(const string& msg, T value,
       Sint16 x, Sint16 y) const;
```

This template function uses the Explicit Instantiation Model to manage the instantiation of this template function. Convert printMessageValueAt to use the Inclusion Model to manage instantiation.

8. Convert the current *tracker framework* into a meaningful animation. Examples will be provided during lecture. This will entail replacing the images currently used in the *tracker framework* to use your images; you may not use my images in your animation. You may use images from the internet but you must cite your source in your README. Making your own images always makes for a more authentic animation. Adding new images will also require that you modify game.xml, since all constants are read from this file. You may also have to modify the frameFactory class, since the frameFactory should manage all frames and SDL_Surfaces in your animation.



(a) Pter flying left.

(b) Pter flying right.

Figure 1: This figure illustrates a two-way multi-frame sprite.

9. The Manager class in the *tracker framework* contains a polymorphic vector of type Drawable* consisting of either Sprite or MultiSprite sprites. Extend your animation, and this vector, to contain two additional types of objects. These two additional types of sprites must be either (1) a two-way multi-frame sprite, (2) a smart sprite implemented using the *Observer* pattern, or (3) an exploding sprite implemented using *chunks*. If you have an idea for an additional "sprite type',' you may substitute for the above with the instructor's permission. A two-way multi-frame sprite is a sprite that can travel in two directions, such as the pterodactyl in Figure 1. You must have at least three different types of sprites in your polymorphic vector and at least two of them must be chosen from: *two-way multi-frame sprite*, *smart sprite*, *exploding sprite*, or an *instructor approved sprite*.

Generating Frames:

Class Manager contains code that, when F4 is pressed, will generate the number of frames specified in game.xml, currently specified as 300 frames. You can change this number if you need more frames to capture your animation. The inital viewport size is 854x480 pixels; if you keep close to these dimensions your animation will blend nicely with the others. Finally, modify game.xml so that the generated frame file names use your *username*, rather than mine. Your frames should be named as follows:

```
<username>.0000.bmp
...
<username>.0299.bmp
```