Project #3 : **Polymorphism in Action**

CpSc 4160/6160: Data-Driven 2D Video Game Development
Computer Science Division, Clemson University
Brian Malloy, PhD
February 18, 2018

Due Date:

To receive credit for this assignment, your solution must meet the requirements specified in this document and be submitted, using the `handin` facility, by 8 AM, Friday, March 2nd, 2018. The handin close date is set at three days after the due date. If you submit after the due date but before the handin close date there will be a ten point deduction. No submissions will be accepted after the handin close date and no submissions will be accepted by email.

Project Submission:

To submit your solution through handin, copy the README file from the project directory in the repo to your project directory, fill in the blanks in the README, make clean in your project directory, and compress the project directory using tar or zip.

Project Specification:

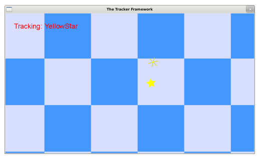


Figure 1: Tracker

The purpose of this assignment is to: help you to become familiar with: (1) C^{++} language constructs including inheritance and polymorphism; (2) design patterns including factory and singleton; (3) data-driven programming, and (4) a tracker framework, included in the directory for project #3. An execution of the tracker framework is illustrated in Figure 1 where it is "tracking" a yellow star. Once you begin this project you should consider the tracker framework to be yours, since you will be modifying and extending it for the rest of the course.

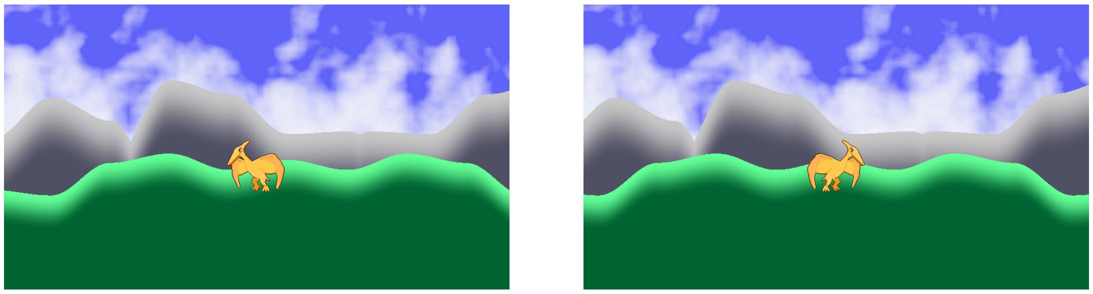
To complete this project, get the tracker framework, and complete the following tasks:

1. Convert the while loops, in the following functions, into ranged for loops:
   - ParseXML::display(),
   - Gamedata::displayData()
2. Convert the first 3 ranged for loops in the ImageFactory destructor to while loops without using *auto*.
3. Convert ImageFactory from a GoF singleton into a Meyers singleton.
4. Use IoMod and C^{++} stringstreams to write the fps of your game to the screen, and write your name in the lower left corner of the screen.
5. Add generality to IoMod by permitting the user to write text in a different color font. You can do this by overloading IoMod::writeText to accept an additional parameter, the new font color, and use this font to write the text.
6. Incorporate parallax scrolling into the animation by using at least two backgrounds. Please note that readTexture and readSurface routines in IoMod require that your image has an *alpha channel*. If your sprite sheet doesn't have one, you must add it. If you use gimp, simply: (1) Make sure the *image* → *mode* is **RGB**. Then (2) choose *layer* → *transparency* → *color to alpha*. Click *ok*.

7. Use **delete** to “explicitly disallow compiler generated functions” in Engine, and ImageFactory.

8. Convert the current *tracker framework* into a coherent animation, using as many sprites as you need, but you must use (instantiate) one of your sprites at least 7 times; I will illustrate this by instantiating the yellow star, currently in the tracker framework, 7 times. To do this, I will modify game.xml, and I will modify the constructors in class Sprite. If I wanted to instantiate many MultiSprites I would also have to modify those constructors. **if you don’t modify these constructors your sprites will appear on top of each other**. You may also have to modify your ImageFactory class, since your ImageFactory should create and destroy all the images in your animation.

   Store **all** sprites in a polymorphic vector of Drawable*. Creating a coherent animation entails replacing the images currently used in your *tracker framework* with your images. **You may not use my images in your animation**. You may use images from the internet but you must cite the source in your README. Creating your own images is time consuming but always makes for a more authentic animation and will earn more points.



(a) Pter flying left.　　(b) Pter flying right.

Figure 2: This figure illustrates a two-way multi-frame sprite.

9. With some small changes, your Engine class in the *tracker framework* will contain a polymorphic vector of type Drawable* consisting of either Sprite or MultiSprite.

   Extend your animation, and this vector, to contain a third type of spite: a two-way multi-frame sprite. If you have an idea for an additional “sprite type’,’ you may substitute for the two-way multi-frame sprite, or simply implement both as an added feature, with the instructor’s permission. A two-way multi-frame sprite is a sprite that can travel in two directions, such as my pterodactyl in Figure 2. You must have at least two different types of sprites in your polymorphic vector and one of them must not be Sprite or MultiSprite, but a sprite of your construction.

10. Your submission must contain no warnings (USE **g++**!) and no memory leaks in user code.

11. A video of your animation in mp4 format, where the filename uses your username as the prefix. Submit your mp4, with your username as prefix, with your project. You can make the video by:

    (a) using your FrameGenerator class to generate frames, and then use avconv to make an mp4; Read the HOWTO file in the frames directory.

    (b) using a video capture tool, for example, simplescreenrecorder.