Project #3 : **Polymorphism in Action**
CpSc 4160/6160: Data-Driven 2D Video Game Development
Computer Science Division, Clemson University
Brian Malloy, PhD
February 27, 2019

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, Monday, March 4^{st}, 2019. 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, use the README file in in the repo, fill in the blanks in the README, make clean in your project directory, and compress the project directory using tar or zip.

Project Specification:

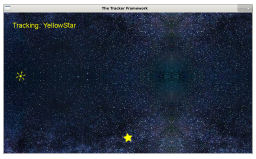


Figure 1: Tracker

The purpose of this project 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.

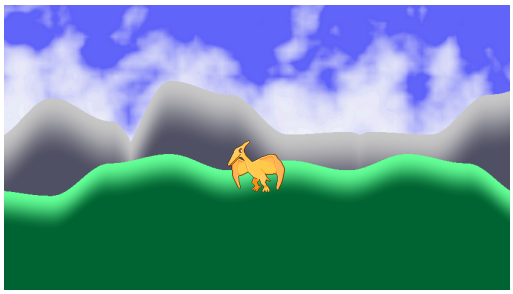
To complete this project, get and use the tracker framework to complete the following tasks:

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 all GoF singletons to Meyers singletons.
4. Use IoMod, the getFps function in Clock, 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 so that everything is readable.
5. Add generality to IoMod by permitting the user to write text in a different color font. One way to do this is 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. The readTexture and readSurface routines in IoMod work with either an *alpha channel* or a transparency bit (see transparency in game.xml). If your sprite sheet doesn't have an alpha channel, you can add it using gimp: (1) Make sure the *image* → *mode* is **RGB**; (2) choose *layer* → *transparency* → *color to alpha*.

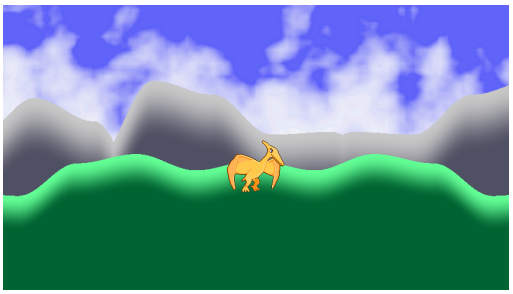
7. All classes in your framework should conform to Item #11 in *Modern* . . .
8. All classes in your framework should conform to the Rule of 3/Zero.
9. With some small changes, your Engine class in the *tracker framework* will contain a polymorphic vector of Drawable* which you must populate with at least two different kinds of sprites to demonstrate polymorphism: (1) You can use either or both of Sprite or MultiSprite; (2) but you must also design your own sprite type, a *two-way multi-frame sprite*, which you must design and implement. Of course you can have three or more different kinds of sprites that populate your polymorphic vector if you wish. You must have a total of at least 7 sprites.

   A two-way multi-frame sprite is a sprite that can travel in two directions with a different set of frames for each direction; the pterodactyl in Figure 2 is an example. You must have at least two different types of sprites in your polymorphic vector and one of them must be a two-way multi-frame sprite.
10. Your game or animation have a coherent theme. Creating a coherent animation entails replacing the images currently used in the *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 may be time consuming but always makes for a more authentic animation and will earn more points; claim ownership of the sprites in your README.

    To instantiate a sprite multiple times, you will have to modify game.xml, and the constructors in the respective classes. **if you don't modify the relevant 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.



(a) Pter flying left.



(b) Pter flying right.

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

11. Your submission must be *data driven*, with no warnings (USE **g++**), and no user memory leaks.
12. A video of your animation in mp4 format, not more than 20 seconds in length, 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.