FAQ for Tracker Framework
CpSc 4160/6160: 2D Game Development
Computer Science Division, Clemson University
Brian Malloy, February 8, 2019

This document is intended to provide information about the classes in the tracker framework and the use of these classes. You will likely use this framework to build your data-driven video game.

1. *What does the* Engine *class do?* The Engine class contains the *event loop* for the game as well as containers (vector/list) for the sprite objects that appear in the game.

2. *What does* RenderContext *do?* RenderContext is a singleton class that initializes the SDL framework, initializes a window, and initalizes a renderer. RenderContext also contains an instance of the ImageFactory, explained below.

3. *What does it mean to say that the* tracker framework *is data-driven?* In a data-driven approach, the programmer can make changes to the program by changing the data that describes the program. We do this by reading program constants from an XML file. Some programmers loathe XML; thus, JSON is an alternative to XML.

4. *How is data-driven programming incorporated into the* tracker framework*?* Data-driven programming is incorporated through two classes: ParseXML and Gamedata.

   ParseXML is a wrapper around the expat XML parser. An XML element consists of a start tag, end tag, and the character data listed between the start and end tags. Thus, ParseXML contains three wrapper functions that override reading of the XML *start* tag, *end* tag, and the character data contained in the corresponding element.

   Gamedata is a singleton class that contains a map that holds all of the constants in the XML file, which was read by ParseXML. Gamedata also contains functions that permit easy access to integer, float, boolean, and string constants, i.e., getXmlInt, getXmlFloat, etc.

5. *Why do we have an* Image *class?* The Image class is a wrapper around an SDL_Surface and an SDL_Texture, including functions to get or draw the surface or texture. The conversion constructor of the Image class accepts a surface, made by the ImageFactory, and uses it to create a texture; in SDL 2.0, you can create a texture from a surface but you cannot create a surface from a texture.

   In the tracker framework the Image class is an example of the *flyweight* pattern, which helps to minimize memory usage by sharing as much data as possible, in this case, an SDL surface and texture, which are large.

6. *I don't like the name of the* Image *class. Why did you call it* Image*? I think it should be called something else, like* TextureWrapper*, or something.* Please feel free to rename or change any of the classes in the tracker framework.

7. *What does the* ImageFactory *do?* The ImageFactory is a creational class, implemented as a singleton, that makes Images, described in the previous item. The ImageFactory contains STL map containers

of Images and surfaces (and unfortunately, oops, textures). If a client uses a name to request an Image, the ImageFactory first looks in the map container to see if it's been previously generated; if the Image has been previously generated, the ImageFactory returns a pointer to the Image; if the Image is not in the map container, then the ImageFactory makes the Image, stores it in the map, and returns a pointer to the Image to the user.

8. *We have an* ImageFactory *and an* FrameGenerator. *Why do we need both? What's the difference?* The ImageFactory generates Image objects, which are wrappers around SDL textures. The FrameGenerator class writes bmps to a directory called frames to permit you, the TA, or the instructor, to build a video.

9. *What does the* World *class do?* The World class is a wrapper around an Image that is used as a background in the game. The World includes a draw function that only draws the part of the world that can be seen in the viewport (kinda). World also contains an update function that permits the World to track the viewport. Textures used by the world must be the height of the world, expressed in the XML file, but can be variable, almost infinite, width. The tracker framework is implemented as a horizontal scroller but can be changed to a vertical scroller.

10. *Why isn't the* World *class called* Background*? Wouldn't that make more sense?* Please rename or change any classes in the tracker framework.

11. *What does the* Viewport *class do?* The Viewport class tries to always position itself so that the player is in the middle of the viewport. In a sense, the Viewport tracks the player. Of course when the player (or whatever image is being tracked) gets near a boundary the viewport cannot keep the image in the center of the viewport.

12. *What does* SpriteSheet *do?* Sprites frequently consist of multiple frames to simulate motion, such as a bird flapping its wings, or a figure walking. Class SpriteSheet contains functions that extract a surface from within a sprite sheet. The functions include overloaded get and parentheses functions that access the i^{th} or i, j^{th} surface within a sprite sheet. The surfaces within a sprite sheet are always numbered starting at zero and begin in the upper left corner of the sprite sheet.

13. *Why do we need to extract surfaces from within a sprite sheet?* Extracting surfaces and storing each image in a separate class makes it easier to perform some operations on the image, such as rotation and collision detection.

14. *The* tracker framework *contains an inheritance hierarchy with* Drawable *as a base class, and* Sprite *and* MultiSprite *as derived classes. What is the purpose of this hierarchy?*

    Class Drawable is an abstract base class with pure virtual functions draw and update. These two functions must be implemented by all classes derived from Drawable, thus permitting clients of Drawable to program to an interface, rather than an implementation, since all derived classes must implement the two interfaces draw and update. Drawable also contains data attributes that apply to all sprites in the game, such as a name, position, and velocity. The Sprite class is intended for single frame

sprites, and the MultiSprite class is intended for multiple frame sprites. This hierarchy can facilitate implementation of a polymorphic container.

15. *What is a polymorphic container?* A *polymorphic container* is typed as a pointer to the base class and can therefore contain any object instantiated from any class in the hierarchy.