07 Implementation Guidelines

Functional and Implementation Guidelines

Functional Guidelines

1. Abstract away rendering logic from the Engine
   1. Create a new object that deals with those operations
2. Abstract away resource loading information from the Engine
   1. Create a object that abstract away this logic
3. Abstract away the Flyweight “Core” implementation
   1. Other entities such as Engine/Game should not have direct access to it
   2. This should happen through some limited API
4. Implement an Image & Text classes (set of functionalities)
   1. The implementation should let the user of the functionality invoke simple commands such as
   2. create
   3. draw
   4. destroy
   5. move
   6. etc...

Implementation Guidelines

1. **Singleton - Lecture**
2. Renderer, ImageContainer, and TextContainer
   1. They are too low-level
   2. Users should not have direct access to it
   3. We need to abstract them away
3. Implement managers
   1. They “encapsulate” types of logic (about drawing, about hardware, about...money, etc...).
   2. Singleton in one word
4. Create a new folder ‘managers’
5. Implement MgrBase class - will be used polymorphically

| class MgrBase {  public:  virtual ~MgrBase() = default;  //forbid the copy and move constructors  //forbid the copy and move assignment operators  virtual void deinit() = 0;  virtual void process() = 0;  }; |
| --- |

1. Implement DrawMgr
   1. Structure

| class DrawMgr: public MgrBase {  Renderer \_renderer;  MonitorWindow \_window;  //Hold maximum frame rate cap  uint32\_t \_maxFrames { 0 };  };  extern DrawMgr \*gDrawMgr; |
| --- |

* 1. Public API - everything needed for the renderer

| void clearScreen();  void finishFrame();  void addDrawCmd(const DrawParams &drawParams); |
| --- |

1. Remove the frameTextures from the Engine struct. It is no longer needed.
   1. drawEngineFrame() should only have a call to \_game.draw()
2. Implement RsrcMgr
   1. Structure

| class RsrcMgr: public MgrBase,  public ImageContainer,  public TextContainer {  //forbid the copy and move constructors  //forbid the copy and move assignment operators  }; |
| --- |

* 1. How can the functionalities behind the ImageContainer and TextContainer be exposed? Do we need something else?

1. Implement getTargetTexture() in the Renderer using the RsrcMgr.

| SDL\_Texture\* getTargetTexture(const DrawParams &widgetInfo) |
| --- |

* 1. Remove any related TODO’s;

1. Expand the DrawMgr functionalities using the getTargetTexture() method
   1. Additional Public API

| void setWidgetBlendMode(const DrawParams &drawParams, BlendMode blendMode);  void setWidgetOpacity(const DrawParams &drawParams, int32\_t opacity); |
| --- |

* 1. Remove related TODO’s

1. Implement ManagerHandler class
   1. Combines the functionalities for all managers (soon to be more)
   2. Part of the Engine
   3. Public API

| int32\_t init(const MgrHandlerConfig &cfg);  void deinit();  void process(); |
| --- |

1. Implement drawing functionalities (Images & Texts)
2. Implement Widget struct (Base struct)
   1. Structure

| struct Widget {  DrawParams \_drawParams;  bool \_isCreated = false;  bool \_isVisible = true;  bool \_isAlphaModulationEnabled = false;  }; |
| --- |

* 1. Public API

| void draw();  //setters & getters  void reset();  void setOpacity(int32\_t opacity);  int32\_t getOpacity() const;  void activateAlphaModulation();  void deactivateAlphaModulation();  //move methods could be added for convenience  //show & hide methods could also be implemented for convenience |
| --- |

* 1. Use the newly created managers in the implementation

| void Widget::draw() {  if (\_isVisible) {  gDrawMgr->addDrawCmd(\_drawParams);  }  } |
| --- |

1. Implement Image class
   1. Reuse Widget
   2. Structure

| class Image: public Widget {  ~Image();  bool \_isDestroyed = false;  }; |
| --- |

* 1. Public API

| void create(int32\_t textureId, const Point &pos = Point::ZERO);  void destroy(); |
| --- |

1. Text class
   1. Reuse Widget
   2. Structure

| class Text : public Widget {  ~Text();  std::string \_textContent;  Color \_color = Colors::BLACK;  int32\_t \_fontId = -1;  bool \_isDestroyed = false;  }; |
| --- |

* 1. Public API

| void create(const char \*text, int32\_t fontId,  const Color& color, const Point &pos = Point::ZERO);  void destroy();  void setText(const char \*text);  void setColor(const Color& color);  std::string getTextContent() const; |
| --- |

1. Reformat the Engine and Game to use the Text/Image classes.
   1. Remove the remaining TODOs
   2. Remove the isTextHidden bool flag and use the hide()/show() methods (or isVisible flag)
2. Implement mouseTargetText that shows where you’ve clicked on the screen and set the text to that position
3. Bonus question: why does the text not draw on the exact same pixel that I’ve clicked?