SD1 A7: Adventure Foundation

*Version 2.0 – updated 2019-11-11*

# Overview

Assignment SD1-A7: Adventure Foundation begins our next (and last) SD1 project: Adventure, an early prototype for a simple 2D top-down RPG with a graphical style similar to Stardew Valley or The Legend of Zelda: A Link To The Past, with data-driven gameplay systems and a mix of authored and procedural content pipelines.

For SD1-A7 Adventure Foundation, our goal is twofold:

1. Establish basic gameplay data pipelines, and
2. Add and upgrade Engine features as required to support the game.

I recommend you implement the engine requirements in your **TestGame** project first (you *did* create a TestGame project from Protogame2D, yes? If not, why wait?), get features tested and working, and THEN clone (“Branch Files” in Perforce Helix P4V) Protogame2D into a new **Adventure** project.

# Specification and Requirements

## Engine requirements (65 points)

1. **(13 points) DevConsole, tier 1** - print, log, toggle, render

A built-in development & debugging engine console (and, in later assignments, command prompt).

See **Example Code** at the end of this document for expected class header signatures.

* 1. (3) Allows [game and engine] code to print colored strings to a log
  2. (2) Allows [game and engine] code to set/toggle the console log’s visibility open or closed
  3. (3) Shows the last N colored log print lines on screen, with the most recently printed at the bottom
  4. (2) Draws only log print lines that fit onscreen (doesn’t draw unseen lines off the top of the screen)
  5. (1) Is, and acts like, a new engine subsystem (like Renderer, Audio, and Input) in these ways:
     1. Has Startup(), BeginFrame(), EndFrame(), Shutdown() methods
     2. Is owned (created, managed, and destroyed) by the App in game code
  6. (2) Is different than other engine subsystems in these ways:
     1. Has a global pointer “g\_theConsole” in EngineCommon, NOT GameCommon
     2. Engine systems – not just game code – can all see, know about, and use g\_theConsole

For example, the Renderer may g\_theConsole->**PrintString**( “Loaded texture…” );

1. **(6 points) class Image**

An Image loading (and, later, writing) , **unrelated** to Textures or Rendering.

See **Example Code** at the end of this document for expected class header signatures.

* 1. (2) Constructs from an imageFilePath (assumed relative to “Run/”, e.g. “Data/Images/Castle.png”)
  2. (2) Allows users to Get or Set any specific Texel’s color in the image
  3. (2) Allows users to ask about the source Image file’s dimensions and original file path

1. **(10 points) Aligned text-in-box drawing**

The ability to draw text strings arbitrarily aligned within an AABB2.

See **Example Code** at the end of this document for expected class header signatures.

* 1. (2) BitmapFont provides at least one new method for creating font text verts aligned to a box:

void **AddVertsForTextInBox2D**( *std*::*vector*<Vertex\_PCU>& vertexArray, **const AABB2& box**, float cellHeight,

const *std*::*string*& text, const Rgba8& tint=Rgba8::WHITE, float cellAspect=1.f,

**const Vec2& alignment=ALIGN\_CENTERED** );

* 1. (2) BitmapFont also provides a new method for determining the 2D size of text IF it were drawn:

Vec2 **GetDimensionsForText2D**(float cellHeight, const *std*::*string*& text, float cellAspect=1.f );

* 1. (2) Allows drawing of a specified text string within a specified AABB2

**Note**: actual drawing still happens later, when calling renderContext->DrawVertexArray( verts… );

* 1. (3) Supports **2D alignment** as Vector2 (0,0 means bottom-left, .5,.5 centered, 1,1 top-right).
  2. (1) Definitions for global named Vec2 alignment constants, provided by EngineCommon:
     1. const Vec2 ALIGN\_CENTERED = Vec2( 0.5f, 0.5f );
     2. const Vec2 ALIGN\_BOTTOM\_LEFT = Vec2( 0.0f, 0.0f );
     3. …as well as ALIGN\_CENTER\_RIGHT, etc. (9 named alignments in total)

**Note**: these are “ALIGN\_” instead of “TEXT\_ALIGN\_” as we will use them for other alignment purposes

1. **(4 points) SplitStringOnDelimiter() string utility function**
   1. Create the following function in Engine/Core/StringUtils.cpp, declared (advertised) as:

Engine/Core/StringUtils.hpp

typedef *std*::*vector*< *std*::*string* > Strings;

kFr

Strings **SplitStringOnDelimiter**( const *std*::*string*& originalString, char delimiterToSplitOn );

…with the following example usage:

Game/Main.cpp or wherever

Strings s1 = **SplitStringOnDelimiter**( "Amy,Bret,Carl", ',' ); // split into 3 substrings: "Amy", "Bret", "Carl"

Strings s2 = **SplitStringOnDelimiter**( " -7.5, 3 ", ',' ); // split into 2: " -7.5" and " 3 " (including whitespace!)

Strings s3 = **SplitStringOnDelimiter**( "3~7", '~' ); // split into 2: "3" and "7"

Strings s4 = **SplitStringOnDelimiter**( "255, 128, 40", ',' ); // split into 3: "255", " 128", and " 40" (including spaces!)

Strings s5 = **SplitStringOnDelimiter**( "apple", '/' ); // split into 1: "apple"

Strings s6 = **SplitStringOnDelimiter**( "8/2/1973", '/' ); // split into 3: "8", "2", and "1973"

Strings s7 = **SplitStringOnDelimiter**( ",,", ',' ); // split into 3: "", "", and ""

Strings s8 = **SplitStringOnDelimiter**( ",,Hello,,", ',' ); // split into 5: "", "", "Hello", "", and ""

Strings s9 = **SplitStringOnDelimiter**( "", ',' ); // split into 1: ""

1. **(8 points) SetFromText() methods**
   1. (3) For each of your fundamental Engine classes (e.g. Vec2, IntVec2, Rgba8…) you should write a new class method (member function) called “SetFromText”:

Engine/Math/Vec2.cpp, etc.

void Vec2::**SetFromText**( const char\* text ) // Parses “6**,**4” or “ -.3 **,** 0.05 ” to (x,y)

* 1. (1) Use **atof**() or equivalent to convert text to floats, and **atoi**() to convert text to ints.
  2. (1) For Vec2, there must be exactly one comma (‘,’) in the “text” string; call atof() on the first half (before the comma) for x, and atof() again on the second half (after the comma) for y.
  3. (1) IntVec2 also requires a single comma, which likewise separates the x-string from the y-string.
  4. (2) For **Rgba8::SetFromText**(), alpha is user-optional. Therefore it requires **either 2 or 3 commas** in the text string. If there are 2 commas, parse it as “redByteText,greenByteText,blueByteText” (e.g. “255,120,0”) with alpha assumed to be 255 (1.0). If there are 3 commas, parse it as “redByteText,greenByteText,blueByteText,alphaByteText” (e.g. “255,120,0,255”).

**Note**: Make sure your parsing approach correctly ignores whitespace before and/or after each number.

**Note**: Consider writing your **SplitStringOnDelimiter**() utility function first, then use for easy parsing!

1. **(10 points) XML Support and Utilities**
   1. (2) Add a third-party C/C++ XML parser library to your engine.
      1. I recommend TinyXml2: <http://www.grinninglizard.com/tinyxml2/>

(You will need to get tinyxml2.cpp & tinyxml2.h from the top github page, which is linked from the TinyXML2 homepage)

* + 1. On disk, these files should go in SD/Engine/Code/ThirdParty/TinyXML2/
    2. Add these (“existing”) files to your Engine project; make sure they compile.
  1. (6) Create Engine/Core/**XmlUtils.cpp**,**hpp** and implement these (standalone) utility functions:

Engine/Core/XmlUtils.hpp

int **ParseXmlAttribute**( const XmlElement& element, const char\* attributeName, int defaultValue );

char **ParseXmlAttribute**( const XmlElement& element, const char\* attributeName, char defaultValue );

bool **ParseXmlAttribute**( const XmlElement& element, const char\* attributeName, bool defaultValue );

float **ParseXmlAttribute**( const XmlElement& element, const char\* attributeName, float defaultValue );

Rgba8 **ParseXmlAttribute**( const XmlElement& element, const char\* attributeName, const Rgba8& defaultValue );

Vec2 **ParseXmlAttribute**( const XmlElement& element, const char\* attributeName, const Vec2& defaultValue );

IntVec2 **ParseXmlAttribute**( const XmlElement& element, const char\* attributeName, const IntVec2& defaultValue );

*std*::*string* **ParseXmlAttribute**( const XmlElement& element, const char\* attributeName, const *std*::*string*& defaultValue );

*Strings* **ParseXmlAttribute**( const XmlElement& element, const char\* attributeName, const *Strings*& defaultValues );

Note: **These functions are very easy to write** for your custom Engine class types:

* + 1. Hint: write **ParseXmlAttribute( … std::string )** FIRST; all the other types can call it!
    2. Create a stack-local temporary object, copy-constructed from “defaultValue”
    3. Get the attribute value as a text string
    4. Call **SetFromText**( attributeValueAsText ) on the temporary object, then return the object
  1. (2) Create also a **custom special-case** function for getting an attribute as a std::string, even if you provide the “default” value as traditional / hardcoded const char\* text (will be important later):

Engine/Core/XmlUtils.hpp

*std*::*string* **ParseXmlAttribute**( const XmlElement& element, const char\* attributeName, const char\* defaultValue );

1. **(10 points) Game Config Blackboard**
   1. (1) Create a **class** **NamedStrings** in Engine/Core/NamedStrings.cpp,hpp
   2. If you don’t already have it, create Engine/Core/**EngineCommon.cpp,hpp**
   3. (1) In EngineCommon.cpp,hpp create a (global-by-value) game config blackboard object, e.g.:

Engine/Core/EngineCommon.hpp

extern NamedStrings **g\_gameConfigBlackboard**; // declared in EngineCommon.hpp, defined in EngineCommon.cpp

* 1. (1) A NamedStrings should store a private dictionary (std::**map**) of string key/value pairs:

Engine/Core/NamedStrings.hpp

*private:*

*std*::***map***< *std*::*string*, *std*::*string* > **m\_keyValuePairs**;

* 1. (4) Your NamedStrings class should offer the following methods (member functions):

Engine/Core/NamedStrings.hpp

void **PopulateFromXmlElementAttributes**( const XmlElement& element );

void **SetValue**( const *std*::*string*& keyName, const *std*::*string*& newValue );

bool **GetValue**( const *std*::*string*& keyName, bool defaultValue ) const;

int **GetValue**( const *std*::*string*& keyName, int defaultValue ) const;

float **GetValue**( const *std*::*string*& keyName, float defaultValue ) const;

*std*::*string* **GetValue**( const *std*::*string*& keyName, *std*::*string* defaultValue ) const;

*std*::*string* **GetValue**( const *std*::*string*& keyName, const char\* defaultValue ) const;

Rgba8 **GetValue**( const *std*::*string*& keyName, const Rgba8& defaultValue ) const;

Vec2 **GetValue**( const *std*::*string*& keyName, const Vec2& defaultValue ) const;

IntVec2 **GetValue**( const *std*::*string*& keyName, const IntVec2& defaultValue ) const;

Again, these methods should be very easy to write, as they can use **SetFromText**() for Rgba8, Vec2, etc.

Your globally-visible **g\_gameConfigBlackboard** should therefore be able to be populated with key/value pairs of string data from an external XML file (see below).

* 1. (3) In your Startup() function in Main\_Windows.cpp – **before** you create your OpenGL window, App, or engine subsystems – use TinyXML2 to parse the (newly-created) file **Data/GameConfig.xml** (if it exists), and populate the game config blackboard from that file’s root XML element’s attributes. Each game project should have its own GameConfig.xml file; make sure you add this file to your game’s Visual Studio project (and Perforce), so it gets included in any FindInFiles (Entire Solution) search results. The following is an example GameConfig.xml file which defines 3 key/value entries:

Run/Data/GameConfig.xml

<GameConfig

startLevel="WizardTower3"

windowAspect="1.777"

isFullscreen="false"

/>

**The game config blackboard holds universal global settings used by engine AND game code alike.**

1. **(4 points) Font (and texture) management**
   1. (1) Your RenderContext class should offer the following methods for creating bitmap fonts:

Engine/Renderer/RenderContext.cpp

BitmapFont\* RenderContext::**CreateOrGetBitmapFont**( const char\* bitmapFontFilePathNoExtension );

* 1. (1) Calling a Create**OrGet**…() function loads the requested object only the first time it is called for that object name; subsequent calls with the same name simply return the already-loaded object.
  2. (1) The CreateTextureFromFile() and CreateBitmapFont() functions are now private, used internally by the RenderContext only, and only called once per unique Texture or Font being loaded.
  3. (1) Your RenderContext therefore needs to keep a registry of loaded textures & fonts; for example:

Engine/Renderer/RenderContext.hpp

*std*::*vector*< Texture\* > **m\_loadedTextures**;

*std*::*vector*< BitmapFont\* > **m\_loadedFonts**;

*// or, if you prefer, you can use named dictionaries using std::map, mapping names to pointers:*

*std*::*map*< *std*::*string*, Texture\* > **m\_loadedTextures**;

*std*::*map*< *std*::*string*, BitmapFont\* > **m\_loadedFonts**;

## Game-specific requirements (35 points)

1. **(10 points) Tile & TileDefinition**
   1. (1) A flyweight **Tile** class which, for now, holds only two data members:

Game/Tile.hpp

IntVec2 **m\_tileCoords**;

TileDefinition\* **m\_tileDef** = nullptr;

* 1. (1) Tile:: Methods for changing the tile’s type, getting its world bounds (computed on-demand), and anything else you find yourself wanting to be able to ask a Tile.
  2. A **TileDefinition** class which represents a single *type* of tile/terrain (e.g. “grass”), and which:
     1. (2) are all created at startup, exactly one per <TileDefinition> child element under the root <TileDefinition**s**> element in **Data/Gameplay/TileDefs.xml**
     2. (1) only one constructor, which takes only a <TileDefinition> const XmlElement&
     3. (2) sets its member variables by parsing the value of each of the following XML attributes:

Run/Data/Gameplay/TileDefs.xml

<**TileDefinition**

name="Grass"

spriteCoords="0,1"

spriteTint="180,180,180"

allowsSight="true"

allowsWalking="true"

allowsFlying="true"

allowsSwimming="false"

/>

* + 1. (2) parses spriteCoords as an IntVec2, but internally stores the actual AABB2 of UV texture coordinates for the sprite (determined by asking the game’s global g\_tileSpriteSheet)
    2. (1) is registered by name in a class-static map/dictionary of TileDefinitions, e.g.

Game/TileDefinition.hpp

static *std*::***map***< *std*::*string*, TileDefinition\* > **s\_definitions**;

1. **(15 points) Map & MapDefinition**
   1. A **Map** class, similar to that of the Incursion project, which:
      1. (1) knows its name, dimensions, and MapDefinition\*
      2. (1) owns its Tiles in an std::vector<Tile>, one-dimensionally, by-value
      3. (1) offers any map-convenience methods as needed by game code
      4. (3) has only one constructor, which takes the map instance’s name (e.g. “Maru Village”) and which MapDefinition to use (by MapDefinition type name, or by MapDefinition pointer), and:
         1. initializes its tiles, giving each tile its m\_tileCoords, and setting its tiles’ types based on settings given by its MapDefinition
         2. (temporary for A7 only) sprinkles a few tiles of the MapDefinition’s edgeTile type randomly throughout the map (for physics testing)
   2. A **MapDefinition** class, which represents a single *type* of map (e.g. “Caves”), and which:
      1. (2) are all instantiated at startup, exactly one per <MapDefinition> element in Data/Gameplay/MapDefs.xml (each child element under the <MapDefinition**s**> root element)
      2. (1) has only one constructor, which takes only a <MapDefinition> const XmlElement&
      3. (2) sets its member variables by parsing the following XML data (attributes and children):

Run/Data/Gameplay/MapDefs.xml

<**MapDefinition** name="Island" width="20" height="30" fillTile="Grass" edgeTile="Water">

</MapDefinition>

* + 1. (1) sets all Tiles initially to the TileDefinition type named by the fillTile attribute
    2. (1) sets all outer-edge border tiles initially to the TileDefinition type named by the edgeTile attribute, if present
    3. (1) parses width and height as ints
    4. (1) is registered in a class-static map/dictionary of MapDefinitions, e.g.

Game/MapDefinition.hpp

static *std*::***vector***< MapDefinition\* > **s\_definitions**;

*// or, if you prefer, you can use named dictionaries using std::map, mapping names to pointers:*

static *std*::***map***< *std*::*string*, MapDefinition\* > **s\_definitions**;

1. **(10 points) Player Actor, Entity, and Tile Physics**
   1. (1) A simple Entity class, owned by the map
   2. (1) Each Entity has a position and a physics radius, as well as an AABB2 for its local draw bounds
   3. (1) Entity-verses-tile corrective physics; an Entity is pushed out of any tile it cannot enter:
   4. (1) Each Entity has bools for whether that Entity: m\_canWalk, m\_canFly, m\_canSwim.
   5. (2) An Entity can enter a tile IF ANY of the following is true:
      1. The Entity can walk, and the tile allows walking; OR
      2. The Entity can fly, and the tile allows flying; OR
      3. The Entity can swim, and the tile allows swimming.
   6. (1) An Actor class, which derives/inherits from Entity.
   7. (3) A single instance of Actor (the player), which can be moved around with an Xbox controller.

# Example Code

from Engine/Core/Image.hpp:

class **Image**

{

public:

**Image**( const char\* imageFilePath );

const *std*::*string*& **GetImageFilePath**() const;

IntVec2 **GetDimensions**() const;

Rgba8 **GetTexelColor**( int texelX, int texelY ) const;

Rgba8 **GetTexelColor**( const IntVec2& texelCoords ) const;

private:

*std*::*string* m\_imageFilePath;

IntVec2 m\_dimensions = IntVec2(0,0);

*std*::*vector*< Rgba8 > m\_rgbaTexels;

};

from Engine/Core/DevConsole.hpp (or Engine/Console/DevConsole.hpp, or…):

class **DevConsole**

{

public:

**DevConsole**();

void **Startup**();

void **BeginFrame**();

void **EndFrame**();

void **Shutdown**();

void **PrintString**( const Rgba8& textColor, const *std*::*string*& devConsolePrintString );

void **Render**( RenderContext& renderer, const Camera& camera, float lineHeight ) const;

void **SetIsOpen**( bool isOpen );

bool **IsOpen**() const;

from Engine/Renderer/BitmapFont.hpp:

class **BitmapFont**

{

...

void AddVertsForText2D( *std*::*vector*<Vertex\_PCU>& vertexArray, const Vec2& startPos, float cellHeight,

const *std*::*string*& text, const Rgba8& tint=Rgba8::WHITE, float cellAspect=1.f);

**// ADDING THIS NEW METHOD:**

void **AddVertsForTextInBox2D**( *std*::*vector*<Vertex\_PCU>& vertexArray, **const AABB2& box**, float cellHeight,

const *std*::*string*& text, const Rgba8& tint=Rgba8::WHITE, float cellAspect=1.f,

**const Vec2& alignment=ALIGN\_CENTERED** );

# Submission

Submit your assignment by following the instructions above and checking in all the required files to Perforce (including a Release-built Adventure\_x64.exe), with the check-in comment “SD1-A7: COMPLETE” for the changelist you want me to grade. My Perforce changelist # 174954

Also, in Canvas, you should submit a .zip file as follows:

* Submit a single .zip file to Canvas under the assignment.
* Your .zip should be named: **C29\_SD1\_A7\_LastnameFirstname.zip**

*For example, Jane Smith would submit a file named* ***C29\_SD1\_A7\_SmithJane.zip***

* Your assignment submission .zip file should contain the following:
  1. A **video recording** of you playing your game (and showing your code, as needed); be sure to visually demonstrate and **verbally narrate** each feature you want credit for
     + The video should be: 1920x1080 **.mp4** at 60 FPS, under 5 minutes and < 100 MB
     + Recommend you use OBS Studio (64bit) to record; make sure you check (watch) the video!
  2. A **copy of this Word document**, with the following modifications:
     + Your submitted Perforce changelist # entered at the top of this section, at “My Perforce changelist # **???**” – this is the Perforce changelist # I should Get, run, test, and grade
     + Each line in the “Requirements” section with (X points) **must** be highlighted:
       - Fully completed requirements are highlighted cyan
         * for features you believe you’ve met/reproduced nearly exactly
       - Partially completed requirements are highlighted yellow
         * for features done but lacking or differing significantly vs. demo
       - Missing requirements are highlighted red
         * for features not implemented (not working or not attempted)