C29 SD:ST Assignment2 – Doomenstein Foundation

Prof. Eiserloh

# **Overview**

* Create a new Doomenstein project, branched from Protogame3D.
* Clone the feature set of the A2 Demo project provided in Canvas.
* Implement a basic 2.5D tile-based data-driven first-person map loading & rendering pipeline for our Doomenstein game, including basic HUD.

This assignment’s focus is twofold: (1) cloning the Demo’s behavior precisely; and (2) ensuring you are ready to move forward with the project.

# **Specification**

It is important that we all use the same conventions and coordinate systems in this course, as we will be establishing a common shared map file format that will be freely interchangeable amongst the professor and students.

* **(Up to 20 points off) A1 Requirements intact:** All coordinate system, camera, control, movement, and drawing conventions from A1 still apply.

In addition, we have the following new requirements for A2:

* **(7 points) Debug Draw**
  + F1 toggles onscreen debug drawing on/off; this includes debug UI elements (e.g. camera xyz/ypr) and 3d elements (world axes, compass).
  + All significant game events (including data file parsing, entering a map, etc.) are printed to the developer console (regardless of F1 setting).
  + The developer console is not flooded / spammed with any messages that print every frame.
  + The current frametime (ms/frame) is displayed onscreen, always. Can **also** show FPS (frames per second), but must have ms/frame.
* **(8 points) Default Map & Map/Warp Command**
  + The starting map name is specified in your Data/GameConfig file and can easily be changed.
  + A “map” or “warp” command in your developer console, which can be used to change maps.
  + If used without arguments, must print a list of valid maps as well as a correct usage example.
  + NOTE :- Mine is pretty much a duplicate of the Demo

*For your “<command> <key1>=<value1> <key2>=<value2>” format, I recommend the following usage:*

> warp // Prints correct usage examples (like these) and also lists all map names

> warp pos=5.5,7.5 // Warps to (5.5,7.5) on the current map (e.g. the center of tile 5,7 in a TileMap)

> warp map=TwistyMaze // Warps to the map “TwistyMaze” at the map-specified default position and yaw

> warp map=EmptyRoom pos=3,7 yaw=90 // Warps to the map “EmptyRoom” at the specified position & yaw override

* **(10 points) Multiple Map Support**
  + App owns Game, which owns World, which owns several instances of Map (abstract base class), implemented currently as TileMap (derived).
  + World also keeps a pointer to the “current map”, which may be nullptr if not in any map.
  + All maps are fully parsed/loaded/instantiated/populated, but only the current map is updated and rendered.
  + It is possible to switch from map to map (via the “map” or “warp” console command, or at least through a debug cheat key).
* **(10 points) Map Material Types**
  + The Data/Definitions/MapMaterialTypes.xml file is parsed, actually used, and works correctly (would still work if I changed its content).
  + The MapMaterialTypes.xml file provided with the Demo should be able to be parsed as-is, without requiring any changes.
  + Each <MaterialType> root child element creates a new MapMaterialType definition, registered (in a map or vector), findable by name.
  + The sprite sheet used (Data/Images/Terrain\_8x8.png, with layout=“8,8”) may be hard-coded, or parsed from the <MaterialsSheet>.
  + The spriteCoords= attribute is parsed and used to look up the UVs for the sprite coordinates specified.
  + Each MapMaterialType keeps a pointer to its SpriteSheet (or the SpriteSheet’s Texture) as well as the UV bounds for its sprite.

* **(10 points) Map Region Types**
  + The Data/Definitions/MapRegionTypes.xml file is parsed, actually used, and works correctly (would still work if I changed its content).
  + The MapRegionTypes.xml file provided with the Demo should be able to be parsed as-is, without requiring any changes.
  + Each <RegionType> root child element in the file creates a new RegionType definition, put in a registry (map or vector), findable by name.
  + Each RegionType parses (and stores) whether it is a solid block or not.
  + Each RegionType points to one or two MapMaterial types; one for solid regions (“side” wall type), two for non-solid (“ceiling” and “floor”).
  + New region types can be added to the file – and referenced in any Map’s <Legend> - and will work correctly.
* **(10 points) Map Parsing**
  + Your Engine provides a file utility function for getting a list of all files matching a pattern (e.g. “\*.xml”) in a folder (e.g. “Data/Maps”).
  + For this you may use \_findfirst and \_findnext, or the new std::filesystem introduced in C++17 (e.g. std::filesystem::directory\_iterator).
  + Each \*.xml file in the “Data/Maps” folder is assumed to be a map; for each, a Map (TileMap in this case) is constructed and populated.
  + The map .xml files provided with the Demo should be able to be parsed as-is, without requiring any changes.
  + A factory function should read the type= attribute from the map .xml’s root note, if it is “TileMap”, new off a TileMap.
  + The TileMap constructor should take (at least) an XmlElement const& mapDef (to the root element of the current map file being parsed).
  + The TileMap constructor should parse whatever it needs from the map XML root element, including the dimensions= attribute (IntVec2).
  + The <Legend> element is parsed, is actually used, and works correctly (i.e. would still work if I tried it with content modifications).
  + The Legend need not be stored permanently; it may be created and used temporarily during the Map construction process, then discarded.
  + The <MapRows> element is parsed, is actually used, and works correctly (i.e. would still work if I tried it with content modifications).
  + Each glyph in the MapRow’s tiles= attribute is looked up in this map’s legend, to find which RegionType that tile/block of the map uses.
  + The <Entities> element is parsed, is actually used, and works correctly (for now, just to establish the default camera position & yaw).
* **(15 points) Map Rendering**
  + Map rendering looks the same as in the Demo. Specifically:
  + The map is drawn in a single draw call; all map faces (walls, ceilings, floors) are pushed into a single vertex array and then drawn.
  + Solid tiles have side faces (only), which are counter-clockwise wound with front (CCW) sides facing “outward”
  + Non-solid tiles have floor and ceiling faces (only), which are CCW-wound with front sides facing “inward” (ceilings down, floors up).
  + Textures should use “Nearest” or “Point” sampling for their magnification texture sampling setting, so they look crisp & pixelated.
  + Depth-testing and depth-writing (z-buffer) are enabled for map rendering, such that near faces always “cover up” far faces behind them.
* **(8 points) Dev Console Error Print function/method**
  + Add a convenience function – either on your engine’s DevConsole or in game code – to easily print out an error message to the console.
  + The message should automatically be colored red (i.e. doesn’t need a color argument)
  + The dev console should be forced into the open state anytime an Error print occurs
  + Thus, if errors occur during data file loading / validation, the game will start with the dev console open, and red message(s) showing.
* **(8 points) Data Validation**
  + For Map .xml files, MapRegionTypes.xml, and MapMaterialTypes.xml, issue a dev console error if:
    - Any required file (MapRegionTypes.xml, MapMaterialTypes.xml) is missing
    - Any file’s XML root node element does not have the correct/expected name
    - Any required XML elements or attributes were missing
    - Any unrecognized/misspelled XML elements or attributes are present
    - A MapRow’s tile=“” string has a different number of characters than the root element’s dimensions= attribute stated for map width
    - A map has a different number of MapRow elements than the root element’s dimensions= attribute stated for map height
    - Any character in a MapRow’s tile=“” string is not found in that map’s Legend
    - Any regionType=“” name referenced in a map’s Legend is not a valid RegionType (as parsed prior from MapRegionTypes.xml)
    - Any material=“” name referenced in MapRegionTypes.xml is not a valid MaterialType (as parsed prior from MapMaterialTypes.xml)
    - Any MaterialType has spriteCoords=“” that reference a sprite outside the sprite sheet’s layout (from 0,0 through 7,7 presently).
* **(6 points) HUD (Heads-Up display) elements**
  + Display the Hud\_Base.png image provided in the demo across the bottom of the screen (per the Demo).
  + Display sprite 0,0 (top-left) from the ViewModelsSpriteSheet\_8x8.png sprite sheet on the screen, centered, above the Hud\_Base.png.
* **(8 points) Hidden Surface Removal (HSR)**
  + Solid tiles (only) will, for each of their cardinal (NSEW) wall side faces, only add those faces to the vertex array IF the neighboring tile in that direction is NOT solid. i.e. if the tile to my east is solid, I will NOT push my east face vertexes.
  + Any tile coordinates “outside the map” (e.g. -1,3) should be treated as being solid, so outermost wall faces are always removed.
  + Non-solid tiles are not affected by HSR, and always have both of their (floor, ceiling) faces added to the map’s vertex array.

# **Submission**

*See “Submitting Assignments” in the course syllabus (available in Canvas) for details on assignment submission.*