# PnC Engine – A Basic **P**oint'**n**'**C**lick Engine

by Michael Gebhart – BA3 WT 19/20

## 1. Description and Manual:

This semester task, the **PnC Engine** powered by SFML, creates a 2D scene with data read from an external .xml-file ("*game.xml*"). My goal was to provide ways of creating scenes in easy and intuitive means, based on a simplified AssetId System. An .xml-file example:

```
<?xml version="1.0" encoding="utf-8"?>
<game title="Point and Click Adventure Game" width="1920" height="1080" fps="48" entryScene="1">
  <assets path="/assets">
    <player id="player" src="player_default.png" speed="5" origin="50, 50" position="100, 650"/>
    <sprite id="npc" src="npc1.png" scale="0.5, 1"/>
    <sprite id="background" src="background.jpg" scale="0.7, 0.6" origin="500, 400"/>
    <sprite id="box" src="crate.png" position="800, 950" scale="0.6, 0.6"/>
    <sprite id="sun" src="sun.png" position="40,40" scale="0.3, 0.3"/>
    <audio id="soundtrack" src="music.wav"/>
    <audio id="clickSound" src="sound01.wav" pitch="0.9"/>
    <text id="txt" src="arcadeclassic.ttf" size="40"/>
  </assets>
  <scenes>
    <scene sceneId="1" rightLevelLimit="900" upperLevelLimit="650" music="soundtrack">
      <txt position="450, 100">GAME ON!</txt>
      <sun onClickText="txt" textPosition="40, 300">I am a Sun!</sun>
      <box/>
      <player/>
      <box position="1050, 780" rotation="90"/>
      <npc position="730, 650" scale="-0.6, 0.6" onClickAudio="clickSound" pitch="0.7"
         onClickText="txt">SMALL GUY</npc>
      <npc position="400, 590" onClickAudio="clickSound" pitch="0.4" onClickText="txt"
         textPosition="400, 500">BIG GUY</npc>
      <background onClickAudio="clickSound" volume="20"/>
    </scene>
  </scenes>
</game>
```

The .xml-file is split into two sections: the Asset Library (blue) and the Scene Hierarchy (green).

In the **Asset Library**, all the assets are initially located, so that they can be loaded into `<scenes>` later. Each asset needs 1.) an unique ***assetId*** 2.) a ***src*** path for loading external data such as sprites and audio, and 3.) individual default values the assets can later be instantiated with in the scene, without the need of further descriptions there.

Each asset (/**Entity**) is to be differentiated in the following types:

- **SPRITE**: a basic visual entity. It contains the basic properties for each entity: `<position>`, `<scale>`, `<origin>` and `<rotation>`. All properties but `<rotation>` are detailed with a Vector2f value.
  - **PLAYER**: a sub-entity of Sprite. It can only be instantiated once and is the controllable instance. A unique property is `<speed>` displaying how many pixels per frame the player moves.
- **AUDIO**: sound and music, with the properties `<volume>` and `<pitch>`.
- **TEXT**: always located "in front" of each sprite. The string content is the value of the node itself and "src" should lead to the .ttf-font. Its remaining properties are `<color>` and `<size>`.

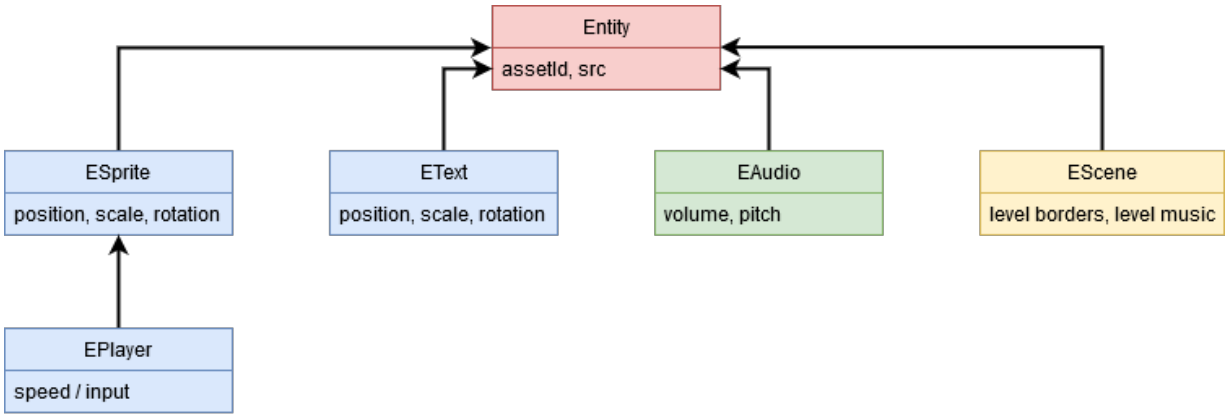
In the **Scene Hierarchy,** each asset that should be loaded to the scene can be easily introduced with stating the ***assetId*** as the node's name (e.g. `<character01/>`). The remaining information is taken from the default data declared in the Asset Library earlier. All the additional information that can be added to each asset node in the scene will overwrite the default data.
With the exception of `<player>`, every asset can be instantiated in multiple numbers and each instance can have different values. It's worth noting that the `<scene>` maintains its hierarchy, meaning that a sprite that is in the upper line of the .xml-files is always drawn "on top" of the ones in the lines below.

The scene itself can be modified with additional nodes and properties:

- **SCENE**: the `<music>` node represents background music that begins to play right after the scene is loaded, furthermore, it also contains the properties `<left->`, `<upper->`, `<right->` and `<lowerLevelLimit>` that are px-coordinates and set borders the player can't pass.
- **`<onClickAudio>`** and **`<-Text>`** are properties that play a certain sound or display a certain text when the attached sprite is clicked on. The displayed text is displayed at the sprite's position, but can be adjusted with `<textPosition>`. It will also disappear after a second click on any position.

2. Technical Realization

The code itself is based around the **Editor** which serves the purpose of a general manager:
it reads the .xml-file and creates asset constructs based on the default data in the Asset Library. Dependent on the information of the active `<scene>`, the Editor then instantiates and loads the assets to draw the scene and provides the needed data to the single entities, which they will then manage. The following inheritance diagram shows the most essential parameters to each Entity type:




To take a closer look at the drawing of the scene: all the asset nodes in <scene> are iterated through from "bottom up" to keep the layering hierarchy. For every node, the node's name is compared to the elements' assetIds in the default vector<(Entity)> constructs built out of the Asset Library. Each entity contains a bool `instantiated` that states whether it was already instantiated in the scene or not, if so, a second entity of the same type will be drawn with the default information overwritten from the scene's node.

After executing the program, the scene is then drawn (here with the code from above):


3. Personal Reflection

I learned a lot in the run of this project's development, not only with coding in C++, but also with self-organization in solo projects, especially in terms of time management and efficient ways of working. For me, it was difficult to draw a line between which functions are needed for the final project and which ones I shouldn't even consider. I also underestimated the time needed to implement certain functions, resulting in functions like Scene Management and Animations being cut due to time constraints. Furthermore, it was really interesting to think of usability for the designers of the .xml-files, even though I easily got lost with little details when it came down to convenience.

I am convinced that my code can be improved in a lot of ways i.e. naming conventions and general efficiency, nonetheless, I am really satisfied with what I achieved with this semester task. C++ is a language I want to keep learning and I really intend to stick to. This project was really helpful to improve my skills in that very language and I am excited to find out which other projects I am going to tackle with it.

Thank You!