CSC
Convertible Scene Creator
Technical Document

Oleg Loshkin

December 8, 2019

1 Introduction

- **Project's Context**
  As part of the game programming cursus at SAE Institute Geneva, for the **technical module GPR5100.2**, students of the second year must **assist students from the third year in completing their bachelor's project.**

  This year, the third year's **PokFamily team** develops a **video game for the Switch** and PC using a tailored **in-house engine**.
  Second year students must **assist them** by creating various **tools they will need** in order to create their game.

  This document describes the functioning of the **Convertible Scene Creator** tool, **CSC** for short.

- **Project's Goals**
  - Create a useful tool that the PokFamily team will use to create their video game.
  - Learn to work in a non-academic environment in a team that depends on the student's performance.

- **Specific Problem**
  The PokFamily team uses the **Unity engine as an external editor**. The PokFamily team needs a tool to **export Unity scenes and prefabs** that may then be used inside the PokEngine.

2 Requirements

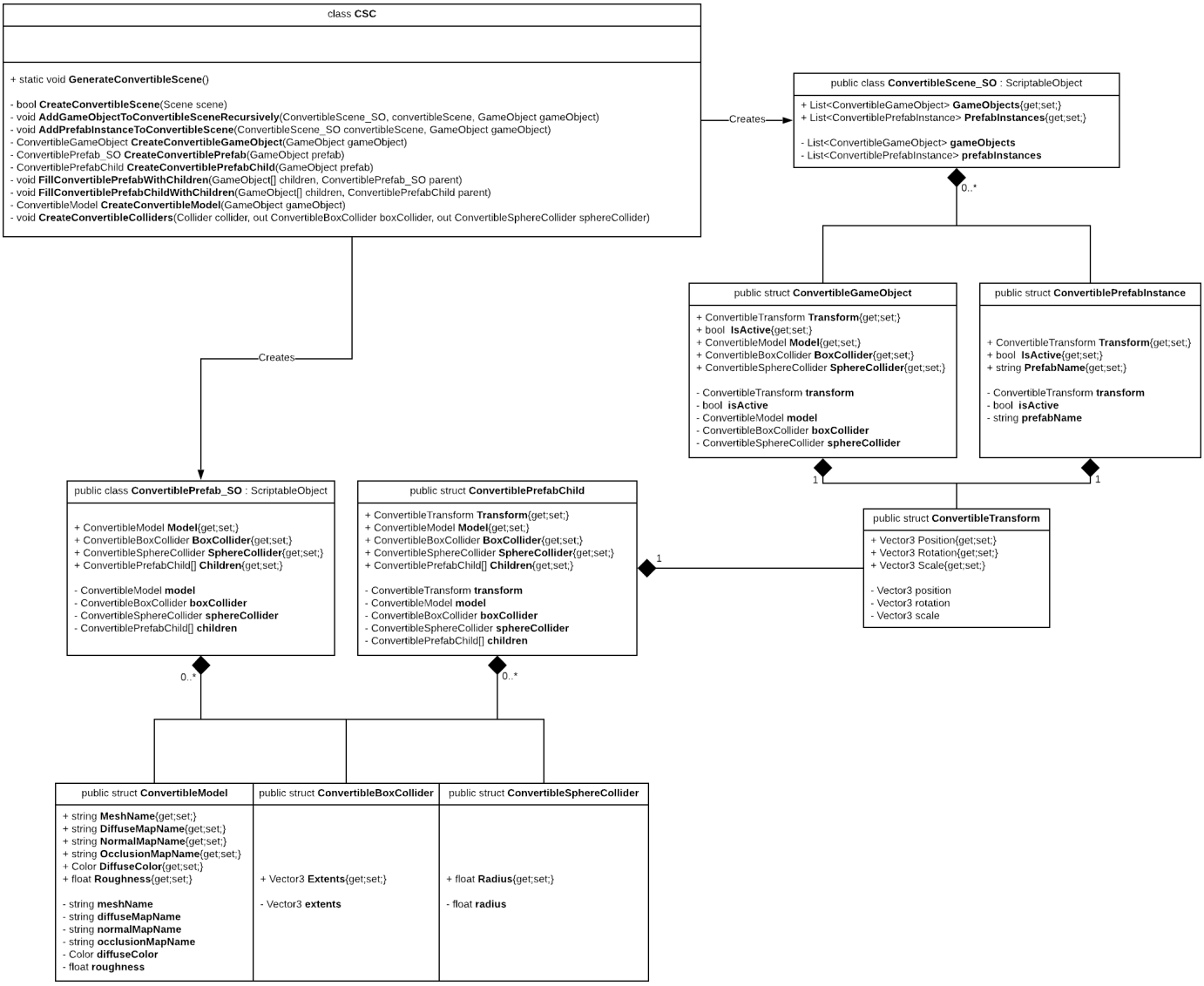
This project's requirements have two origins:

- **Academic requirements**
  - The task given by the team has been understood and done in time.
  - The tool is maintained by the student after the tool's completion.
  - The tool must be user-friendly.
  - The student understands how to manage data.
  - The student understands how a game engine interfaces with a game engine editor.
  - The student has organized himself and his work in a way to facilitate the work of others.
  - The tool's performance is reasonable.
  - The implementation is appropriately sophisticated.
  - The student understands the implications of non-academic teamwork.
- **Pragmatic requirements**
  - Convert Unity scene and prefab files to files readable by the PokEngine's parser via UPDC.
  - The user must be able to interact with the tool via Unity.
  - The code must satisfy the quality and style expected by the team. C++ coding style is defined in the Coding Style Document. C# coding style is defined in UnityWorkOrganization document.
  - The student must communicate with the team appropriately and be dependable.

3 Technologies Used

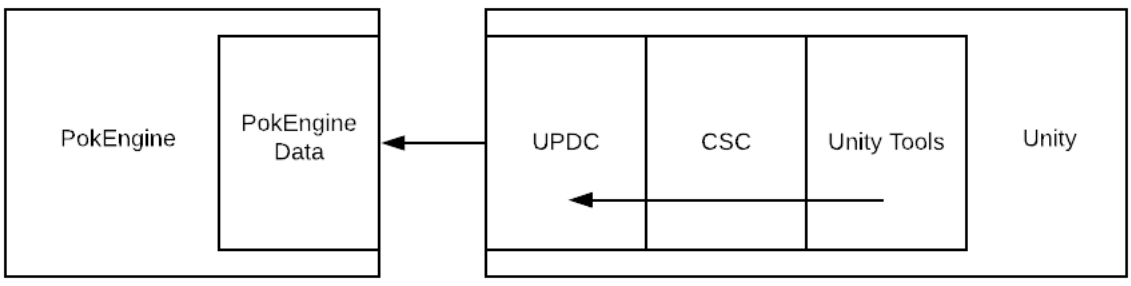
- **PokEngine**
  The **PokEngine** is the game engine developed by the PokFamily team. The engine is **written with C++ standard 2014** and **partly C++ standard 2014** for code running on the Nintendo Switch.
  The engine has a parser that is capable of reading JSON files. This parser is used to import data exported from Unity with UPDC.
- **Unity 2019.1.10f**
  Unity 2019.1.10f is used as an **external editor**.
- **Visual Studio 2017**
  Visual Studio 2017 is used for development of the PokEngine.
- **Git**
  github.com is used for versioning for the **PokEngine source code**. **gitlab.com** is used for versioning for the **Unity prototype source code**. Git bash is used for most interactions with the git framework. Merge conflicts are solved manually via text editor and git bash.

4 UML Diagram



CSC's sole purpose is to create convertible .asset files by using ScriptableObjects as is described in UPDC's documentation. The **user's only point of interaction** with the tool is a simple **drop down menu button** that launches the conversion by taking the currently active scene as input.

5 Interaction with Overall Project



CSC integrates with UPDC to export scenes and prefabs. As such, it interacts with Unity's database from which it creates .asset files for UPDC to export.

6 Exporting Prefabs

Upon the game designer's request, **this tool needed to be able to export Unity's prefabs**. This has surprisingly proven to be a challenge.

Storing whole prefabs in the `ConvertibleScene_SO` file would not have made sense since the whole point of a prefab is to define it once, and reference it later wherever it showed up.
From this came **the separation between the `ConvertiblePrefab_SO` class**, a ScriptableObject inheriting class **compatible with UPDC**, and **the struct `ConvertiblePrefabInstance`** which is **a serializable data structure** that is stored in a `ConvertibleScene_SO` and that **references an existing `ConvertiblePrefab_SO`.**

More distinctions were needed to be made **between GameObjects composing a prefab** however. **A `ConvertiblePrefabInstance` has a transform field whereas a `ConvertiblePrefab_SO` does not.** This makes sense since an instance of a prefab will have a defined position, rotation and scale relative to a scene, but will not have those by itself.
This implies however that **a `ConvertiblePrefab_SO` cannot have other `ConvertiblePrefab_SO`'s as children**: these do need to have a transform to define their positions, rotations and scales relative to their parent. From this need, was created **the `ConvertiblePrefabChild` struct**.

7 (Not) Tackling Polymorphism

In the final JSON files generated by UPDC of `ConvertiblePrefab_SO`'s, you will notice the presence of **two fields for colliders: one for a boxCollider and another for a sphereCollider**. This is due to the fact that C# does not support polymorphism of struct data types.

This combined with the fact that Unity's JSONUtility is used by UPDC to generate JSON files rather than a custom build parser results in the inability to represent polymorphism in the JSON files, hence the presence of both types of colliders simultaneously rather than only one collider of particular type or less.

8 Exporting Models

Unity uses a combination of **Meshes and Materials to represent 3D objects**. **PokEngine uses Models** instead, which combine the functionalities of both. CSC therefore does the work of reading the relevant data from these both data structures to create a serializable struct that corresponds to the format expected by the PokEngine.

9 Integrating with ProBuilder

ProBuilder was used in the designing of the prototype of the game on Unity. It is a very powerful tool for prototyping, however it does not easily expose the meshes it generates.

While it is possible to retrieve them and save them to a specific location in the Unity's project from where they can be accessible by other tools, doing so links the ProBuilder scripts located in a particular scene to those specific meshes. This could lead to version control conflicts.
As a workaround, **CSC** instead **generates copies of the meshes used by ProBuilder** for it's own use, leaving ProBuilder scripts intact.
This does come at the cost of extra space being used on disk.

10 Potential Improvements

- The prefab exportation had been done in a rush, the current implementation has a lot of room for improvement:
  - There are way too many data structures defined to separate between different types of prefab objects, there must be a more elegant way to represent them.
  - For each instance of a given prefab in a Unity scene, a new `ConvertiblePrefab_SO` .asset is generated. This defeats the whole purpose or having prefabs in the first place. A system would need to be implemented to prevent the generation of identical convertible prefabs.
- While exportation of ProBuilder meshes is functional, it is implemented in a way as to duplicate already existing meshes. This is a behaviour that could be improved.
- As a consequence of UPDC's implementation, polymorphism is not handled. A redesign of UPDC might make the implementation of collider exportation more elegant.

11 Summary

The CSC tool provides the user with an easy way to convert a Unity scene and prefabs located inside into formats readable by PokEngine's parser via the use of UPDC.

While implementation transparency of the C# language and Unity's own systems may be beneficial when it comes to superficial tasks, in this case, it has proven to be a hurdle instead.
Polymorphism, data structures abstractions and data serialization is a problematic mix I have been confronted with for the first time during this project and the creation of this tool has shown me **key areas of knowledge that I need to be more mindful of**.