Save and Load Function

Phillip Schneider
University of Bremen
Bremen, Germany
phisch@uni-bremen.de

***Abstract*—This document will explain why and how we implemented a function to save and load mission related data. To avoid loading and interpreting the necessary mission data during each and every run of TraVis, we designed a routine that saves all the mission data TraVis needs in a binary file and loads this file if a mission is simulated again.**

***Index Terms*—Unreal Engine 4, C++, Cpp, SaveGame, Saving, Loading, Compressed, Binary**

I. Introduction

Travis is a masters project operated by *University of Bremen (UoB)* and *Marum*, which tries to re-simulate a deep dive mission in the Unreal Engine. Re-simulating in this case means that log data is provided by an Autonomous Underwater Vehicle (AUV) that needs to be read by a computer and displayed in a simulated environment. To read more about this mission specific data and how TraVis treats this data, read the *data-paper*. Before TraVis can start to simulate a mission, all mission data files need to be converted in a format that is readable for the unreal engine. Because we don't want **TRAVIS** to do this procedure every time , we developed a function, that saves all the data that TraVis needs to simulate a mission; recognizes if a mission was already simulated and then loads the previously saved data. This function is also capable of compressing the converted parts.

II. What advantages does a save and load function offer for a software

During develpoment we recognized the process of loading and interpreting the mission data made up the majority of the startup time of the program. Not to forget the recources this process takes every time. The main advantage is the time that can be saved due to skipping the process of converting and interpreting the mission data. The second advantage will be the amount of storage that can be saved by saving only the data that is realy used by the program and compressing it afterwards. In our testcase we had a input file of 478,353 KB (∼478 Megabyte). The saved file is about ∼35,000KB (∼35 Megabyte). If we use a compression algorithm, the now compressed savefile needs 16,205 KB (16.205 Megabyte) of storage. Summing up we expect three advantages from implementing a save- and load-data function.

- Reducing time to setup **TRAVIS** when loading previously saved missions
- Reducing the necessary storage[1] by saving only the data **TRAVIS** uses and
- applying a compression algorithem to this data to save evan more storage.

A. Which data does TRAVIS need to smimulate a mission?

First of all we have to outline which mission related data **TRAVIS** has to visualize and how this is accomplished [2]. There are two big files which **TRAVIS** gets as inputdata. A *CSV-file* which contains the data which was logged by the AUV [3] during a mission and a *ASCII-file* in formatted in the *Esri grid* [4] file format. The ASCII-File contains information which descripe a heightmap. We use this information to generate the deepsea landscape area. The algorithems which read and convert the ASCII-file automatically ignore all values which aren't used later. The same is done with the CSV-file. An optimised algorithm for loading and calculating the data from csv files rendered a saveload approach obsolete. Thus this paper does not consider the loading and saving of csv-files. The converted data from the ASCII-file is later used to generate a deap-sea landscape. This landscpae consists of the heightmap information and is implemented with a *procedural mesh component* (PMC) in C++ [5]. Each instance of the PMC class needs serveral arrays as input parameters to generate a three-dimensional graphic object. To generate these arrays another algorithem takes the converted data from the ASCII-file and creates the arrays. At this point in the TraVis environment we can implement an optimization by saving these arrays in a savefile and skipping the whole routine described abouve, when a mission was simulated before.

B. Which data does TRAVIS store in a save-file

As mentioned in the previous subsection, **TraVis** saves all needed information to generate a mesh component without having to re-read and calculate these information from the ASCII-file as mentioned earlier.

[1]we actually do not make use of this advantage because the ascii file will still remain in the same directory
[2]This is documented in the other papers for this project: datacomunicator
[3]for further information see https://www.marum.de/Entdecken/AUV-MARUM-SEAL.html
[4] https://en.wikipedia.org/wiki/Esri_grid and the *landscape* paper
[5]for more detail read the landscape paper

III. Saving and Loading - From ASCII to binary and vice versa

The already mentiont TraVis savefile is a binary file. All variables are serialized to a binary array by using the *Archive class* of UE4. The binary array is then saved to the hard disk or SSD. From a UE4 C++ standpoint *binary array* means *TArray<uint8>*. Most of the fileformates used in the UE4 can be converted to *TArray<uint8>* easily by buildin functions. With the *FileManager class TArray<uint8>* can be written to the hard disk or SSD. **TRAVIS** savefiles have the **.savetravis* extension.

A. When should be loaded and saved

When **TRAVIS** is started a config file is read. This config file contains the path to the CSV- and ASCII-file. **TRAVIS** then automaticly checks if there already exists a savefile for the ASCII-file. From this point on, there are two trails:

- **There is no binary savefile:**
  - Create an **ASCReader** object and start reading the ascii-file and converting of the data
  - create a **HeightMesh** object and generate the arrays which are necessary for the procedural mesh component
  - create a **SaveLoadMission** object and start serializing the data
  - start compressing the serialized data
  - save the generated binary file
  - draw the deapsea landscape
- **There is already a binary savefile:**
  - create a **SaveLoadMission** object and load the binary file , decompress it and deserialize the data
  - create a **HeightMesh** object and use the data from the binary file for the procedural mesh component
  - draw the procedural mesh component

One can easily recognize that the saveload function definitely speeds up TraVis.

B. How does the save and load process work?

As mentioned already above the main routine of the save and load process starts as soon as the process of generating a deap-sea landscape with a PMC is initiated. Right before the PMC instance is created, an instance of the *HeightMesh* class calls either the *LoadGameDataFromFile()* or *SaveGameDataToFile()* function. These both functions ar members of the *SaveLoadMission* class and do exactly one thing each: save data to the hard disk / SSD or load data from hard disk / SSD. Both functions use exactly the same list of parameters. All of these individual parameters are *passed by reference*. With this decision made, we save up some space in the ram [6] and can modify all arguments directly. The LoadGameDataFromFile() and SaveGameDataToFile() work in a similar way but in an opposite order. The order for **SaveGameDataToFile()** is:

[6]Some of these parameters contain several hundred MB and TraVis already uses an average of 6 GB ram or more

1. call the SaveLoadData() function and write all parameters to an binary array
2. start compressing this binary array
3. write the compressed array to the hard disk / SSD

The order for **LoadGameDataFromFile()** is:

1. read the savefile from hard disk / SSD
2. decompress the savefile which results in a binary array
3. call the SaveLoadData() function and read all parameters from the binary array

Figure 1 showes the different sequences of the save and load process.

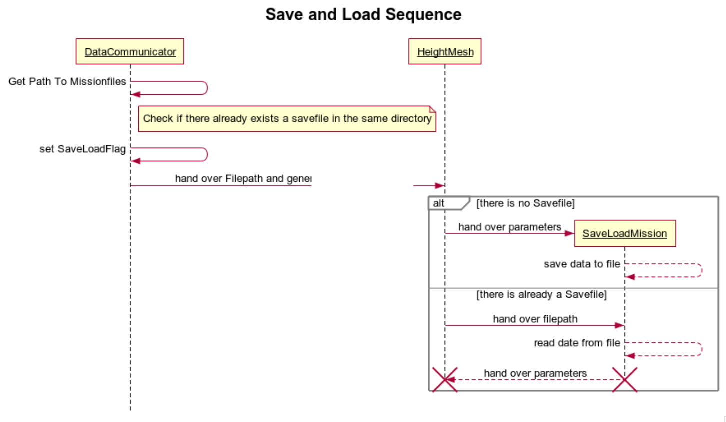

Fig. 1. Sequence diagram of the save and load function

Because every argument is passed by reference this whole process can be included and excluded very easily by just commenting out the SaveGameDataToFile() and LoadGameDataFromFile() function calls. This was helpfull durin development.

The excerpt from the TraVis source code below shows the SaveLoadMission() helpfer function. It makes use of the overloaded << C++ operator. The << operator is context sensitive which means *it can either get data out of an archive and put it into a variable or put data from the variable into the archived binary format*[1].

```
void USaveLoadMission::SaveLoadData(
FArchive& Ar,
FString& SaveDataString,
TArray<FVector>& SaveDataVector,
TArray<int32>& SaveDataInt32,....)
{
   Ar << SaveDataString;
   Ar << SaveDataVector;
   Ar << SaveDataInt32;
   Ar << Savenormals;
   Ar << SaveUV0;
   Ar << SavevertexColors;
}
```

Conclusion

As already shown above TraVis benefits from a save and load function. In future versions of TraVis an overview of all previously simulated missions could be a good extension of the software. It would be nice to see if also the log-data from the CSV-file could be put in the savefile and how much more proceedings could be accelerated. Users then would only need the TraVis savefiles to simulate missions and could share them among themselves instead of keeping gigabytes of mission related data.

References

[1] Rama, "Save System, Read & Write Any Data to Compressed Binary Files," in Unreal Wiki, https://wiki.unrealengine.com/Save_System,_Read_%26_Write_Any_Data_to_Compressed_Binary_Files

[2] Epic Games, "Saving Your Game with C++," in Unreal Wiki, https://docs.unrealengine.com/en-us/Gameplay/SaveGame/Code