Travis Launcher

Kristof Kipp
University of Bremen
Bremen, Germany
kkipp@informatik.uni-bremen.de

***Abstract*—This paper introduces the Travis launcher that generates and alternates a config file for a specific mission in Travis.**

***Index Terms*—unreal engine, unreal, trajectory, Travis**

I. Introduction

Travis is able to dynamically read a mission provided by the user. It also dynamically creates and renders a deep sea landscape based on another file, called the asc file. These files need to be changed when different missions should be simulated. As the unreal engine does not internally provide users with a FilePicker and the creation of such a thing would exceed the scope of the Travis project, the team decided to outsource the changing of parameters to another program outside the unreal engine. This is done with a simple WPF application written in C# and called *Travis Launcher*.

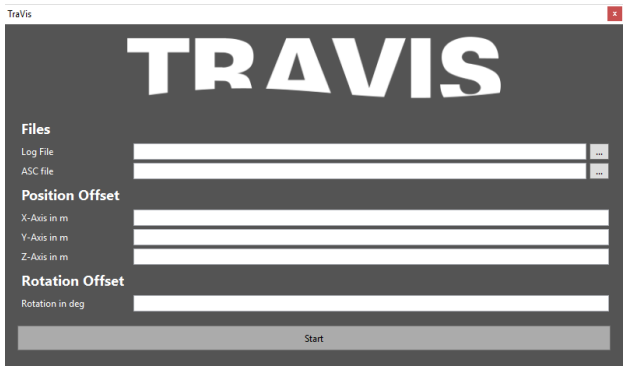

Fig. 1. the launcher

II. Main Program

Upon starting the program the directory in which the launcher is started is traversed. If the main launcher of Travis (e.g., *Travis.exe*) is found, the launcher is started, otherwise a message is shown and the launcher closed again.

```
if (Directory.GetFiles(Environment.CurrentDirectory)
    .Any(x => x == "Travis.exe"))
{
    this.Executable = Path.Combine(Environment.
        CurrentDirectory, "Travis.exe");
}
else
{
    MessageBox.Show("could not find Travis.exe - 
        make sure it exists in the current directory
        ", "the tallest error on earth",
        MessageBoxButton.OK, MessageBoxImage.Error);
    //Close();
}
```

When a file with the name of the main executable is found, the launcher knows it's in the correct directory and tries to read the config file that holds all the neccessary data for the simulation.

```
// config?
FileStream cfg = File.Open("Travis.cfg", FileMode.
    OpenOrCreate, FileAccess.Read);
StreamReader sr = new StreamReader(cfg);
try
{
    Args = sr.ReadLine().Split(' ').ToList(); //
        Convert into space seperated list
    txtFindAscfile.Text = Args[0]; // Asc File
        Location
    txtFindLogfile.Text = Args[1]; // Logfile
        Location
    txtPosX.Text = Args[2]; // Positional Offset X
    txtPosY.Text = Args[3]; // Positional Offset Y
    txtPosZ.Text = Args[4]; // Positional Offset Z
    txtRot.Text = Args[5]; // Rotation Offset
}
catch (Exception)
{
    Args = new List<string>(); // If there was an
        error with loading the config file, we
        create a new one.
}
sr.Close();
cfg.Close(); // Close the files and their respective
     handles
}
```

This data consists of different values:

A. Asc File

The asc file is the file that is read to generate the bathymetric landscape. It resembles a $N \times M$ matrix that holds integer values of the depth at a given point.

B. Log File

The log file is the file that holds the data logged by the AUV while being on a mission.

C. Positional Offset

The positional offset is applied to the position of the landscape-mesh after calculating and rendering it.

D. Rotational Offset

The rotational offset is applied to the landscape-mesh after calculating and rendering it.

III. Error Handling

Values may be invalid. Most likely when a textbox is empty. As this is just a simple evaluation method, there is no need for a complex algorithm, so the following is a simple check if any of the text boxes in the application is empty:

```
foreach (UIElement ele in grdMain.Children) //
    Iterate all Children of the main Grid (this one
    contains all the TextBoxes, labels, etc.)
{
    if (ele.GetType() == typeof(TextBox)) // We only
        care about the TextBoxes
    {
        if (string.IsNullOrEmpty((((TextBox)ele).
            Text))) // TextBox is Empty?
        {
            err = true;
            ((TextBox)ele).Background = new
                SolidColorBrush(Color.FromRgb(255,
                128, 128)); // Red for missing value
        }
        else
        {
            ((TextBox)ele).Background = new
                SolidColorBrush(Color.FromRgb(255,
                255, 255)); // All fine here
        }
    }
}
```

grdMain is the main grid that holds all textboxes and all the other controls in the application. So to keep the check simple all children are iterated and typechecked. If the type is textbox, the object is casted and the underlying value is checked (.IsNullOrEmpty()). If any of the values of the textboxes is empty, a boolean value is set to *true* and the background of the control is set to a red-ish color. If a value is fine, the background color is set to a standard white.

IV. File Dialogs

The Log and Asc-Files need to be found with the a file picker provided by the windows api. For this the Class *OpenFileDialog* in the namespace *Microsoft.Win32* is used. Each file has a ... button that operates the file picker. The files shown inside the picker is modified in the code file, one for each file type, both are shown here:

```
// Configure open file dialog box
Microsoft.Win32.OpenFileDialog dlg = new Microsoft.
    Win32.OpenFileDialog();
dlg.FileName = ""; // Default file name
dlg.DefaultExt = ".csv"; // Default file extension
dlg.Filter = "CSV␣Logfiles|*.csv"; // Filter files
    by extension
if (dlg.ShowDialog().Value)
{
    string nm = dlg.FileName;
    txtFindLogfile.Text = nm;
}
```

This is the logfile file picker. Log files are provided in csv file format. A fairly similar solution is used for the asc file picker.

```
// Configure open file dialog box
Microsoft.Win32.OpenFileDialog dlg = new Microsoft.
    Win32.OpenFileDialog();
dlg.FileName = ""; // Default file name
dlg.DefaultExt = ".asc"; // Default file extension
dlg.Filter = "ASC␣Bathymetry|*.asc"; // Filter files
    by extension
if (dlg.ShowDialog().Value)
{
    string nm = dlg.FileName;
    txtFindAscfile.Text = nm;
}
```

V. Saving

When the simulated is to be started, the state of the contents of the launcher need to be stored in a file read by the simulation itself. The launcher simply takes all the contents of the textboxes, puts them in a List of strings and then simply joins them to create a space seperated list of data.

```
if (!err) // No missing values?
{
    // Store all the values
    Args.AddRange(new string[] {
        txtFindAscfile.Text,
        txtFindLogfile.Text,
        txtPosX.Text,
        txtPosY.Text,
        txtPosZ.Text,
        txtRot.Text,
    });

    // Save them
    var str = string.Join("␣", Args.ToArray());
    FileStream cfg = File.Open("Travis.cfg",
        FileMode.Truncate, FileAccess.Write);
    StreamWriter sw = new StreamWriter(cfg);
    sw.WriteLine(str);
    sw.Close();
    cfg.Close();

        // Start the Simulation and close the
            Launcher
        Process.Start(this.Executable);
        Process.GetCurrentProcess().Kill();
    }
}
```

To store the data, the string of arguments is written in a file called *Travis.cfg*. Even when no changes are made, the file is truncated and re-written to make sure the config-data is correct.

VI. Outlook

Theoretically this whole program SHOULD be deprecated by creating a filepicker in the unreal engine, which should be possible when tackled correctly. If not, this program could be improved by automatically searching for valid log and asc-files in a given root directory. Additionally the values could be checked on plausability. Plus as an extra it should be possible to adapt the level of detail with which the heightmesh is generated. This could be done by adding a slider to the launcher that, while increasing the percentage (the heigher the better) lowers the skip value of the heightmesh[1], the latter is a first priority issue actually.

[1] see according paper