Team Content Workflow Improvement

Filippo Maria Leonardi

Saxion University of Applied Sciences

Student Number
459231

Company Name
VSTEP Simulation

Internship Coach
Yvens Rebouças Serpa

Company Coach
Thomas Breekveldt

Abstract

The following graduation thesis describes the internship work carried out at VSTEP Simulation with the goal of simplifying and speeding up the workflow of the Team Content. The research started with defining what processes of the development were slowing the Team Content down and then it examines each of them, reporting what solutions have been applied to solve those problems and describing the benefits that those brought to the team performance and usability.

Table Of Contents

Chapter 1

Introduction

1.1 Background

> *“Competitiveness is defined as the ability of companies to compete while maintaining or improving the average standard of living. If you are cutting wages to become more competitive, that’s not really more competitive. It’s raising the skill and the efficiency of those workers so that they can support and sustain that higher wage."*

This quote from Michael Porter (Davidson, 2015), a professor at Harvard Business School, explains clearly that the effectiveness of the workers in a company is critical in order to be competitive in the market.

Over the last couple of decades (Bang & Markeset, 2011), companies have been dealing with an environment of intense competition because of the economic globalization (Aničić & Nestorović, 2020). In this scenario, the ability to have an efficient company that delivers the products to its customers in a quick way, can be considered one of the most important competitive advantages to have.
A good level of optimization is not just a reduction of the costs for the company, but it would also improve the customer satisfaction, which is seen as a major differentiator of the company in the market, and increasingly has become an important element of business strategy (Gitman & McDaniel, 2005).

1.2 Research Overview

The following graduation thesis describes the internship work carried out at VSTEP Simulation (section 1.3) in Rotterdam, covering a semester of full-time work in 2022.

During this internship, I have been assigned to the Team Simulation (subsection 1.4.3) and the main activities performed were:

- Analysis of the user's workflow.
- User's feedback implementation.
- Creation of new features/workflow automation.
- Maintenance of the existing code.
- Code reviewing feedback analysis from other programmers.

The implementation of tasks base been mostly done autonomously, but I received support from the Company Coach or other colleagues when needed.
I worked at VSTEP every week 40 hours: the work schedule was quite flexible as the company allows each employee to choose its working hours between 07:00 and 19:00 and it allows choose if working from home or the office. Because of that, during the internship I worked on average two days per week at the office and 3 days from home.

1.3 Company Outline

VSTEP Simulation is a Dutch company founded in 2001 which develops simulators and virtual training software that allow the training for multiple simulated environments in a practical, cost-effective, and sustainable way. The company has around 60 employees from more than 20 different nationalities and the office is based in Rotterdam.
The current company's core product lines are:

- **NAUTIS:** a maritime simulator used as training for naval academy students, where the controls and the physics of this software try to reflect reality as much as possible in order to offer a realistic simulation (VSTEP, 2022).
- **Response Simulator (RS):** a full-suite virtual training platform built in collaboration with safety and security experts (VSTEP, 2021). It immerses users in a realistic 3D environment in which detailed incidents can be simulated.

Nautis and RS can be controlled by 2 different roles: the instructor, which can control the simulation of the scenario (create new objects, move existing objects, change the parameters of the scenario, etc), and the trainee which can control only a single character or ship of the scenario per time and has to perform the training.

1.4 Company Oganization

The company is divided into 4 departments. The company organization chart (Figure 1.1) has been recreated based on the chart available on Personio (Personio, 2022), the HR management website that VSTEP uses, and shows the different company departments and teams.

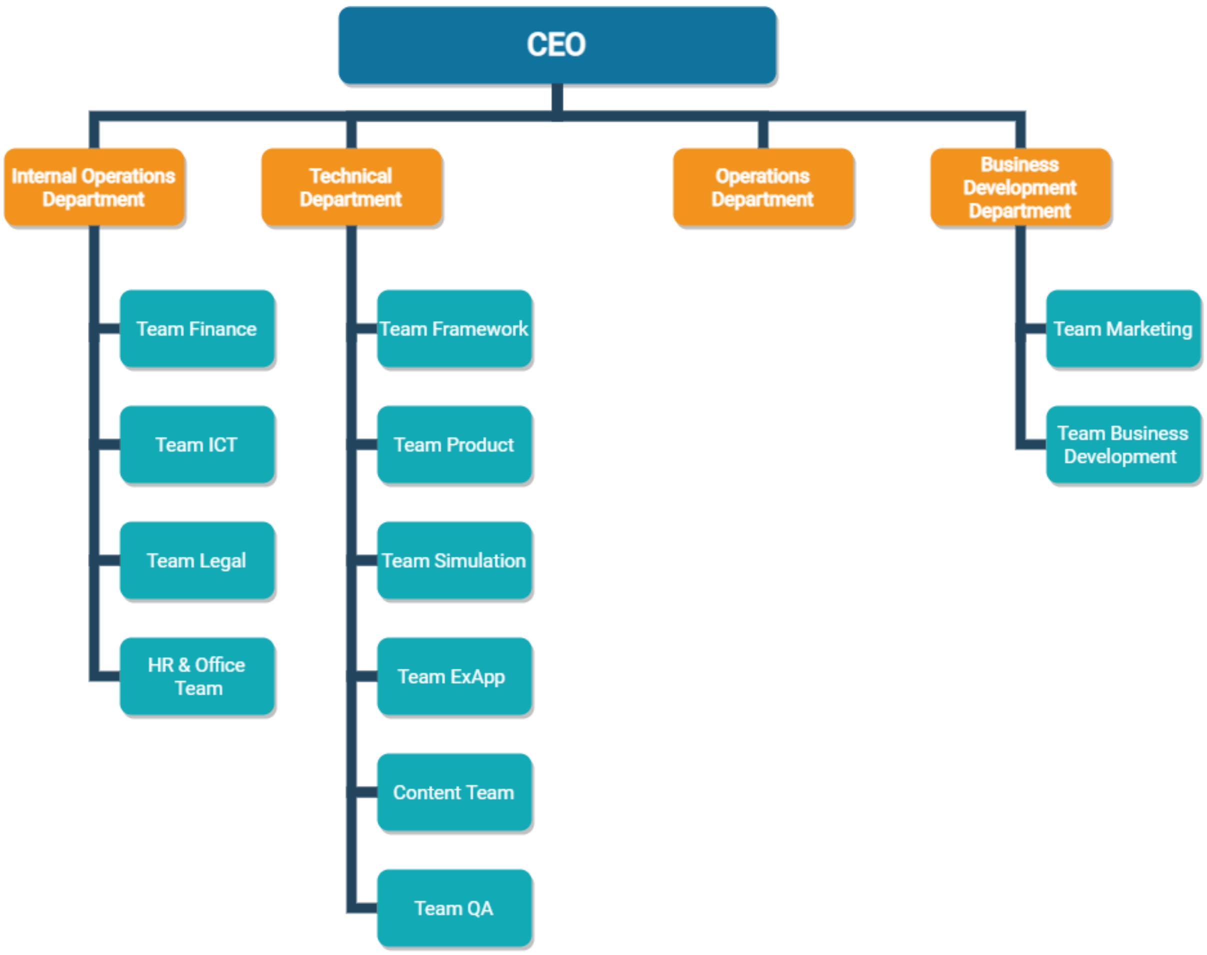


Figure 1.1: VSTEP organization chart.

1.4.1 Business Development Department

The business development department is divided into 2 teams. The marketing team currently consists of people working to make VSTEP known to a bigger public and create all the commercial content for the company. The business development team works to make sure that NAUTIS and RS always fully meet customer's needs

1.4.2 Operations Department

The operations department makes sure that the company's products get ready for the customers and that they receive support in case of need. This department also assists with the preparations, installation, and validation of the simulators at the customer's facilities located all over the world (Figure 1.2).

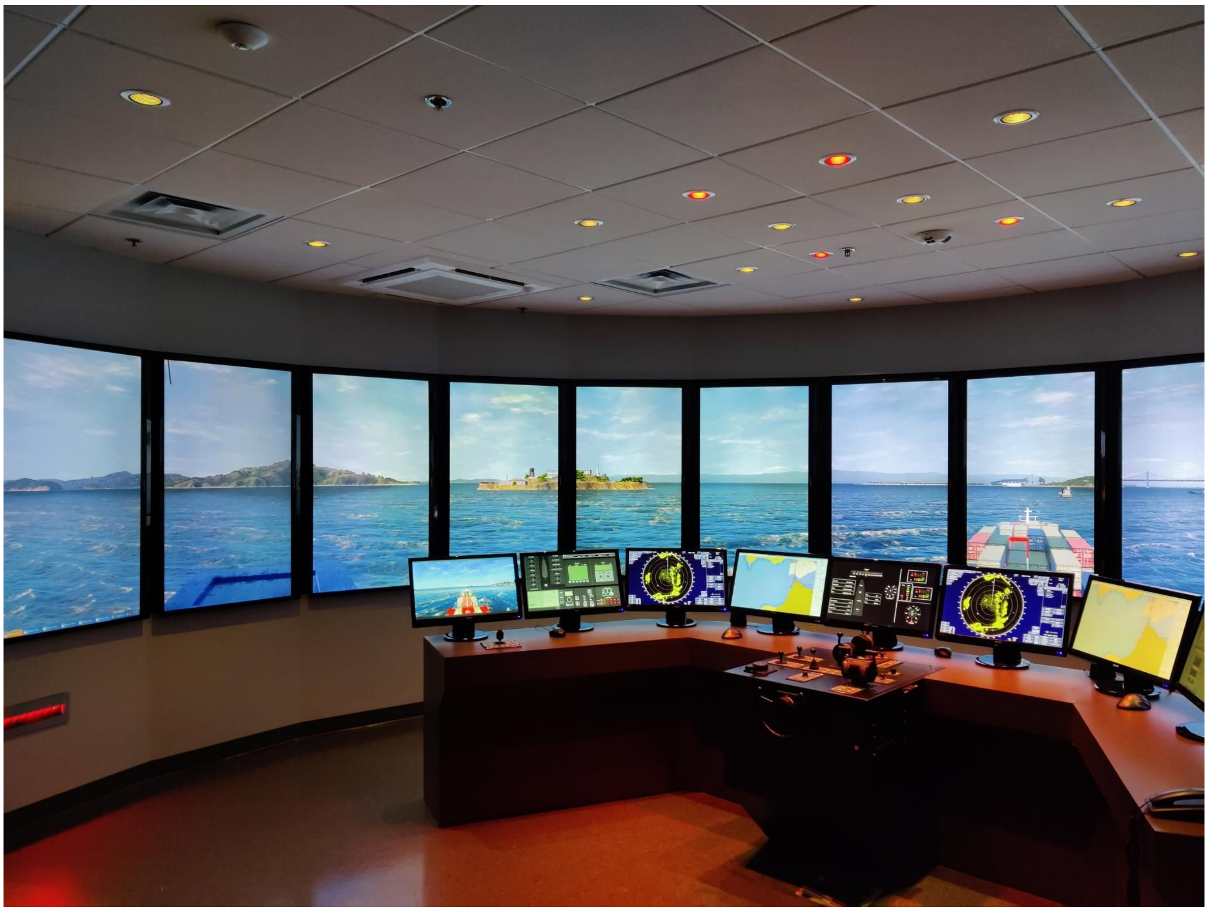


Figure 1.2: VSTEP installation example.

1.4.3 Technical Department

The Technical Department is the largest of VSTEP. The members of this department have their different specialties and together they create the company's products.

- **Team Framework:** works on the software and simulation platform and engine.

- **Team Product:** works on the simulation products with an emphasis on tooling (instructor station, procedure, scenario).
- **Team Simulation:** works on the simulation products with an emphasis on simulation systems (hydrodynamics, physics).
- **Team ExtApp:** works on the software and tools that run side-by-side with the simulation software (panels, assessment tool).
- **Team Content:** works on the creation of all the visual elements (3D models, materials, textures, particles, environments) and on the setup of the new prefabs used in the simulations.
- **Team QA:** works on testing the products to ensure that they work correctly.

1.4.4 Internal Operations Department

There are 4 different teams within the Internal Operations Department: Finance, ICT, Legal, and HR & Office. All teams work closely together to provide and maintain a great work environment within the company.

1.5 Company Meetings

In VSTEP there are regular meetings that employees join to give updates on their work and to discuss their tasks During this internship, the following regular meetings were joined:

- **Team Simulation:** weekly update where everyone from the Team Simulation gives updates to the team leader, and reports if there are problems and if support is needed on a certain task.
- **Townhall:** monthly update where VSTEP the managers were giving an update about the company status and its future development to the other employees through a PowerPoint presentation.
- **Content Workflow Improvement update:** Bi-weekly meeting where I was giving an update about my progress to Team Content and the Company Coach. This was the most important meeting of the internship where I could ask questions, get feedback and get new inputs about things that could be improved or issues that were compromising Team Content's effectiveness. Based on the input received, at the end of this meeting I was organizing the list of tasks to work on, with the help of the Company Coach, and ranked them by their priority and impact on the development.

1.6 Problem Analysis

The problem that this research tried to solve arose from the desire of VSTEP's management to simplify and speed up the work of the Team Content.
Due to a lack of time and resources, VSTEP could not work on solutions that would have increased the efficiency of the team.
Time is a common obstacle for companies: in fact, employees from small/middle-sized companies are unlikely to work on optimization, because instead they usually use their time to work on active projects.

As communicated by VSTEP, the purpose of this research would not have been only about creating new tools, but also about improving existing features that would have made them more efficient and easy to use by the end-user.

Team Content have been reporting development issues that affected their productivity. These problems were discussed during the Content Workflow Improvement Update meeting (section 1.5) and the list of issues reported was sorted by their importance, with the goal of giving priority to the processes that impacted negatively the most.
The list of problems reported is composed of the following elements and they will be examined further in the report (section 3.1):

- Duplicating prefabs and checking if their links are set up correctly required too much time.
- Nautis/RS needed to be restarted when a prefab asset was edited.
- The camera didn't focus on the prefab components.
- The Selection highlight visibility was low and could have been improved.
- Flag line items spacing could not be changed dynamically.
- Team Content could not import/export mesh files from 3D Studio Max 2020 and 2022.
- The time for some environments was invalid.
- Some environments were crashing when loaded in the Scenario Editor.
- Switching GIT branches required too much time to be performed.

Chapter 2

Theory

The following chapter is needed to explain the basic concepts that are useful to have a better understanding of the report.

2.1 Unigine

NAUTIS and RS are built on top of Unigine (*Unigine*, 2022), a 3D real-time engine that is appropriate for simulation products because of its photorealistic rendering of objects and environments, which enhances the user's immersion.

NAUTIS and RS use C++ and UnigineScript as programming languages.
The Unigine Script is a proprietary scripting language, similar to C++ in syntax, which instead does not need to be compiled, offering a faster development.

NAUTIS and RS have several C++ modules that are accessible from UnigineScript through the extension of the Unigine API. In fact, these modules can communicate through the creation of functions accessible via the Unigine Plugin that after the compilation of the C++ code, become available to be called from UnigineScript.

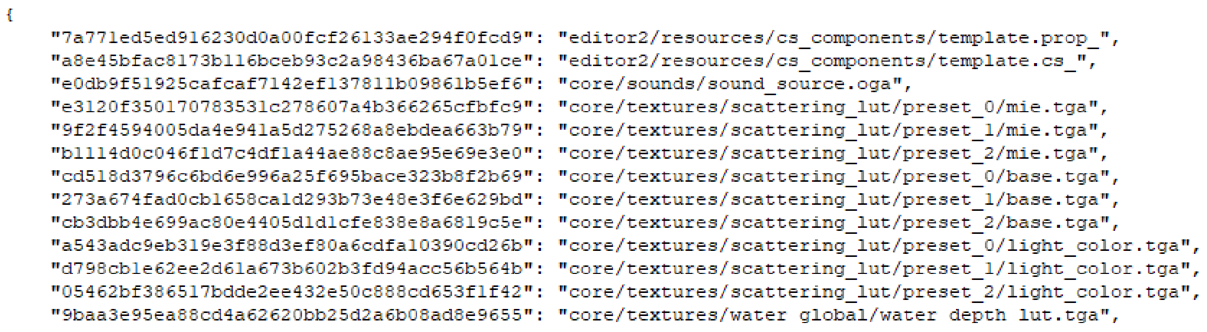
2.1.1 Nodes

In Unigine all the objects added into the world are called nodes.
Unigine provides different types of nodes, which differ in their visual representation and behavior. Nodes are created and stored in the world and their data are saved into a .world file and into an external .node file, which allows them to be imported into the world when necessary.

2.1.2 GUIDs

A GUID (Globally Unique Identifier) is a 40-character hexadecimal string generated using the SHA-1 hash algorithm. Unigine automatically generates GUIDs for all the assets, by defining a virtual path for each of them. All GUIDs are stored in *data/guids.db* (Figure 2.1), which is a JSON file that contains a list of pairs composed of the GUID and the file path relative to the data folder of Unigine.

Using GUIDs provides a more flexible file management solution: it is possible to abstract from file names, which means that a path to the file can be changed while keeping the same GUID link (*File System - Documentation - Unigine Developer*, 2022).
For example, if the file name of an asset that was being used by several nodes needed to be changed, it was not needed to change every single reference to that asset, but it would have been possible to just only change the path of that GUID, in the *guids.db* file.



```
{
    "7a771ed5ed916230d0a00fcf26133ae294f0fcd9": "editor2/resources/cs_components/template.prop_",
    "a8e45bfac8173b116bceb93c2a98436ba67a01ce": "editor2/resources/cs_components/template.cs_",
    "e0db9f51925cafcaf7142ef137811b09861b5ef6": "core/sounds/sound_source.oga",
    "e3120f350170783531c278607a4b366265cfbfc9": "core/textures/scattering_lut/preset_0/mie.tga",
    "9f2f4594005da4e941a5d275268a8ebdea663b79": "core/textures/scattering_lut/preset_1/mie.tga",
    "b1114d0c046f1d7c4df1a44ae88c8ae95e69e3e0": "core/textures/scattering_lut/preset_2/mie.tga",
    "cd518d3796c6bd6e996a25f695bace323b8f2b69": "core/textures/scattering_lut/preset_0/base.tga",
    "273a674fad0cb1658ca1d293b73e48e3f6e629bd": "core/textures/scattering_lut/preset_1/base.tga",
    "cb3dbb4e699ac80e4405d1d1cfe838e8a6819c5e": "core/textures/scattering_lut/preset_2/base.tga",
    "a543adc9eb319e3f88d3ef80a6cdfa10390cd26b": "core/textures/scattering_lut/preset_0/light_color.tga",
    "d798cb1e62ee2d61a673b602b3fd94acc56b564b": "core/textures/scattering_lut/preset_1/light_color.tga",
    "05462bf386517bdde2ee432e50c888cd653f1f42": "core/textures/scattering_lut/preset_2/light_color.tga",
    "9baa3e95ea88cd4a62620bb25d2a6b08ad8e9655": "core/textures/water global/water depth lut.tga",
```

Figure 2.1: guids.db file.

2.1.3 Source Code

VSTEP owns a license that allows its employees to access the Unigine source code: this it's a big advantage for developers because it allows them to have a better understanding of how the engine works, as with that it's possible to read the code and debug it.
For instance, to give an idea of how the source code is important, the lack of it would have compromised the development of most of the tasks carried out during the internship, because there would not have been enough resources to understand and tackle some issues.

2.2 Prefabs

Prefabs are compound objects loaded from the disk that can be licensed by VSTEP to be available only by the clients that own the license. Nautis/RS have a different type of prefab for each object used in the simulation (vehicles, buildings, ships, etc). Each prefab is identified with a prefab code, and its data is stored in a folder named as its prefab code (/data/Prefabs/<type>/<prefab-code>).

The prefab folder contains the following folders which include a category of files:

- **Cfg:** VSTEP configuration files for the prefab components.
- **Nodes:** nodes from which the prefab is composed.
- **Materials:** materials used by the prefab mesh surfaces.
- **Meshes:** meshes used by the prefab nodes.
- **Textures:** textures used by that prefab materials.

Lastly, the root folder of a prefab contains the rest of the data which does not fall in one of the above categories, such as generic prefab information, thumbnails, etc.

2.3 Editors

Team Development used 4 editors to implement their changes into the framework:

- **Objects Editor:** to customize the object properties and its components.
- **Environment Editor:** to create and edit the environments used in the simulation.
- **Scenario Editor:** to create scenarios for the training of the trainees.
- **Dynamics Editor:** to configure the physics parameters of a ship.

All the cited editors, with the exception of the Scenario Editor, were only used internally in the company, but VSTEP was planning to extend them also to their clients in order to provide them the freedom to create and customize content on their own.

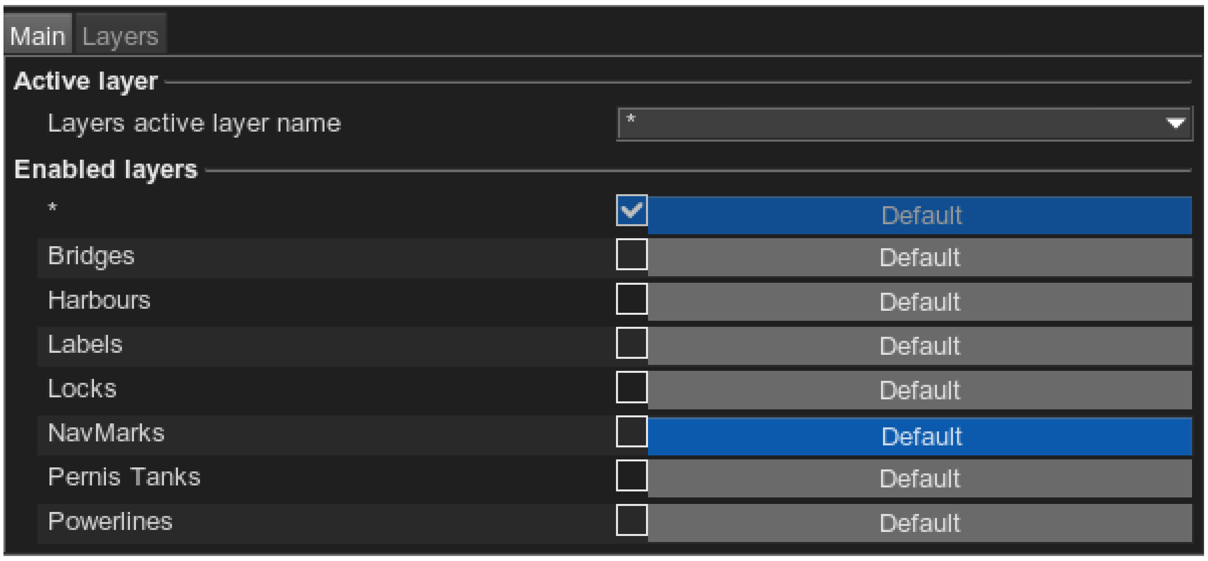
2.4 Environments And Scenarios

2.4.1 Environment

An environment is the representation of an area used for the simulation, which is composed of one or more layers that can contain objects.

A layer can be enabled/disabled in the Environment Editor when necessary if its objects need to be displayed/hidden.

When an environment is loaded in the Environment Editor only the default layer (*) is enabled, while when a scenario based on an environment is created all the layers that have the default button active, are enabled (Figure 2.2).



Active layer

Layers active layer name	*

Enabled layers

*	✓	Default
Bridges		Default
Harbours		Default
Labels		Default
Locks		Default
NavMarks		Default
Pernis Tanks		Default
Powerlines		Default

Figure 2.2: Environment layers window.

2.4.2 Scenario

A scenario is an assignment created by VSTEP or an instructor. It is based on an environment and it's being used by the trainees to practice an exercise.

2.5 Selection Bubble

A selection bubble is a shape that gets displayed when a group components is selected in the Property Window of a prefab.
A selection bubble can render a primitive shape(cube, sphere, cylinder) or a custom mesh (Figure 2.3).
Nautis and RS have a Configuration Menu which allows to customize the program settings based on the user's needs, including options to customize some of the selection bubbles parameters.

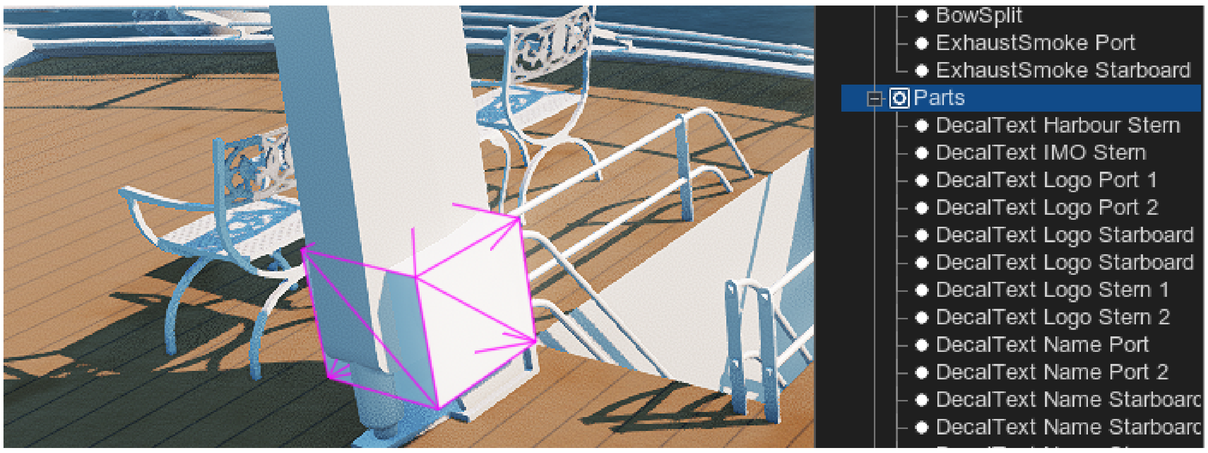


Figure 2.3: Example of selection bubble.

2.6 Azure DevOps

VSTEP uses Azure DevOps to organize its project, which is a cloud-based platform that provides development services for allowing teams to plan work, collaborate on code development, and build and deploy applications (*Azure DevOps Services*, 2022). Azure DevOps is a flexible tool, as it is not necessary to use all the services offered, but it is possible to adopt each of them independently.
The services used during the graduation were 3:

- Azure Boards
- Azure Repos
- Azure Pipelines

2.6.1 Azure Boards

Azure Boards provide tools for Agile planning and monitoring of work items with activity backlog reporting. The agile method allows working with small iterations, each of them corresponds to an independent work item that will be merged into the final product. This makes the development much more flexible as it allows quick implementation of changes to the project, providing benefits for the company and its customers.

2.6.2 Azure Repos

Azure Repos provides the hosting and the management of a GIT repository hosted on the cloud. GIT is a distributed version control system designed for tracking changes in files. VSTEP uses GITFlow as a branching strategy to manage the repository. The GITFlow approach uses two main branches to manage version control of the project: the first is the master which keeps all the releases, and the second is the develop branch, which is fundamental for the development of the next versions and it's used as the basis for future integrations. In addition, the GITFlow approach requires that each new feature should have its own branch, which must also be present on origin in order to allow other developers to collaborate. The new feature branches need to be created from develop and when the feature development is completed, the features branch gets merged back to the development branch.

Pull Request

The Pull Request is a feature that allows somebody to propose its changes to a branch and request somebody else to review those changes. At VSTEP is mandatory to create a Pull Request to merge the changes made for a task into another branch, to provide a higher quality of the code, and to try to avoid merging commits that could contain issues.

2.6.3 Azure Pipelines

Azure Pipelines allow to automatically build and test code projects. In order to complete a Pull Request, Azure Pipelines test services had to be run since by policy it was mandatory to pass the C++ and Unigine script compilation without errors. This feature helps the company to avoid merging broken code in the stable branches, through an automatic execution that reduces the effort of the developers by a lot

Chapter 3

Research

3.1 List of issues

This section of the report contains the list of issues reported by the Team Content in the Content Workflow Improvement Update meeting (section 1.5).

3.1.1 Prefab Duplication

When Team Content needed to create a new prefab for Nautis/RS that was similar to an existing prefab, it was more convenient in terms of time to clone and adapt the existing prefab, rather than creating a new prefab from scratch.
After cloning the prefab, Team Content had to manually replace all the old assets links of the duplicated prefab. To do that, it was possible to use any replacement tools (i.e. NotePad++), since all the assets of a prefab were linked with path-based links.
However, Unigine changed the way assets were linked with each other in version 2.7.0.3, replacing path-based links with GUID links. This change made the prefabs duplication procedure longer as the replacement of all the assets had to be made manually on every single link on the prefab. Furthermore, this procedure was also not safe because it could have happened that one member of Team Content forgot to replace one or more links, which could have caused a crash of the engine.

3.1.2 Duplicated Prefabs Links

As already mentioned previously in section 3.1.1, when creating new prefabs, Team Content was used to duplicate existing prefabs that were similar to the ones that needed to be created. In this way, it could be avoided to create a new prefab from scratch, which would require more time. However, this procedure may have generated further problems in case all the links with the old prefab were not being replaced correctly in the duplicated prefab.

If for example Team Content created a prefab called "PrefabB", by duplicating a prefab called "PrefabA" and they didn't replace all the links correctly, it could have happened that:

- **Good case scenario:** a client has a license to use both "PrefabA" and "PrefabB". No problems are visible in this case.
- **Bad case scenario:** a client has a license to only use "PrefabB". This would be a problem since the assets of the "PrefabB" are linked to "PrefabA" and the assets of "PrefabA" would not have been available for the client. This means that "PrefabB" could have not been loaded correctly.

3.1.3 Prefabs Assets Runtime Reload

To test if a prefab didn't have any visualization issues, Team Content had to load it on Nautis/RS to check if all its assets were displayed correctly. When it wasn't, Team Content had to restart the engine to test the prefab after an asset file was modified, having to wait up to ten minutes for each restart.

3.1.4 Camera Prefabs Components Focus Support

Nautis and RS allow the user to focus on one or more selected objects in the editor. When the camera focuses on an object, it accelerates toward its position until a certain distance is reached.
A problem with the camera focus discussed with Team Content was that it was possible to focus prefabs, but not their components. This could have been useful especially when it was needed to know in a fast way where a small component was positioned in a big prefab, as the component would not be easy to spot.

3.1.5 Selection highlight Visibility

Team Content reported 3 problems with the highlight of the selected component of a prefab:

- When selecting a component of a prefab, the way the components of the prefabs were highlighted when selected, could have been improved as its visibility was low, therefore it's not clear for the users what component they selected.
- The selection bubbles of some components had a color that differed from the color set in the Configuration Menu.
- The surfaces of some selection bubbles were not visible.

3.1.6 Flag Line Items Spacing

To indicate that a ship is carrying dangerous cargo, a ship can have one to three (depending on the danger of the carried cargo) blue cones in a vertical line. Team Content could not add these items because the distance between each cone was set too high in the code, making it impossible for some ships to fit under some bridges (Figure 3.1).

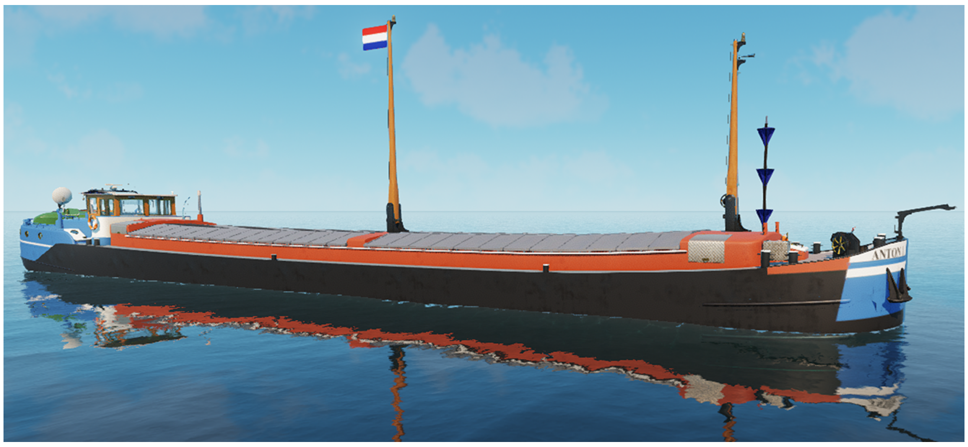


Figure 3.1: Distance between blue cones.

3.1.7 3D Studio Max plugin

3D Studio Max is a 3D modeling, animation and rendering program developed by Autodesk (*Autodesk*, 2021), which is used by VSTEP to make the 3D content for Nautis/RS.
VSTEP used a 3D Studio Max plugin made by Unigine, to import and export mesh files. However, Unigine decided to don't provide plugins for 3D Studio Max anymore, since they recommended using FBX. Therefore, the last plugin offered by Unigine was for 3D Studio Max 2017.
VSTEP owned only 1 license for 3D Studio Max 2017, meaning that only one employee from Team Content could use 3DS Max 2017 to import/export a mesh file per time. This was inefficient, in fact Team Content reported that it frequently happened that an employee had to wait that another finished using 3D Studio Max 2017, before being able to use it.

3.1.8 Dark Environment Bug

Team Content reported that some environments were loaded at night in the Scenario Editor because of an unknown reason, and the date displayed was set on the 1st of January 1970.

3.1.9 Environment layers crash

Team Content incurred a crash when loading an environment after a specific ship was added to the default layer. To avoid the editor from crashing, Team Content found a workaround that consisted of first enabling all the layers, before adding the ship. The reason why this workaround was working was unknown, furthermore, not all the environments were having this issue.

3.1.10 Slow GIT branches checkout

Team Content reported that switching from one branch to another required too much time on their computers, due to the fact that their hardware was less powerful compared to the computer of the other teams in the Development department.
Switching branches was needed each time employees had to work on a different task from the one that they were working on, or they had to test the work performed by another employee. Switching branches is a process that happened frequently at VSTEP, especially by Team Leaders that often need to check the work of other employees.

3.2 Methodology

This thesis used qualitative and quantitative research as research methods, focusing on understanding what processes of the development were slowing down the target audience and prevented them to work more efficiently, and by how much they were affected.
The data were gathered during the Content Workflow Improvement Update meeting (section 1.5) and the private consultations with the Company Coach and the Team Content members.
Furthermore, more quantitative research has been done for the Slow GIT Checkout branch issue (section 3.1.10), by collecting measurements of the time needed to perform several actions. These data have collected ten times for each action to provide a reliable result, and the average value of those measurements was taken into account.
Lastly, during the research phase for each task, it was almost always necessary to conduct research in VSTEP source code, the Unigine source code, and its documentation to collect technical data for tasks of the assignment.

3.3 Research Questions

3.3.1 Main Research Question

How can the workflow of the Team Content be improved in order to reduce VSTEP's development cost and the effort among the employees of the team?

3.3.2 Sub Research Question

- How can the duplication of the prefabs be improved to let the employees perform it faster?
- How can the prefabs links be checked in a faster way to let the employees know if they are not set up correctly?
- How can prefabs assets be reloaded without needing to restart Nautis/RS?
- How can the camera focus show where a prefab component is located?
- How can the selection highlight be improved to increase the visibility for the user?
- How can Team Content customize the distance between each flag line item to allow the ship to also pass under low bridges?
- How can Unigine mesh files be imported/exported on a newer version of 3D Studio Max 2017?
- Why do some scenarios load at night and display the wrong date, and how can this be fixed?
- Why do some environments make RS/Nautis crash when loaded in the Scenario Editor, and how can this be fixed?
- How can switching GIT branches be more efficient in terms of time?

3.4 Scope

This assignment only applies to adding improvements to the Team Content, but this doesn't exclude that other teams could also take advantage of those benefits. Furthermore, the work was limited to the tasks discussed with the Team Content and the Company Coach during the Content Workflow Improvement Update meeting (section 1.5).

3.5 Testing

All the features developed have been tested by myself and afterward Team Content tested them to give remarks and feedback. If the testing passed without any problems, then a Pull Request was created to have the code checked by other programmers. Every time an issue was found during the previous testing phases, the process started all over again, and the code was adapted to implement the feedback.

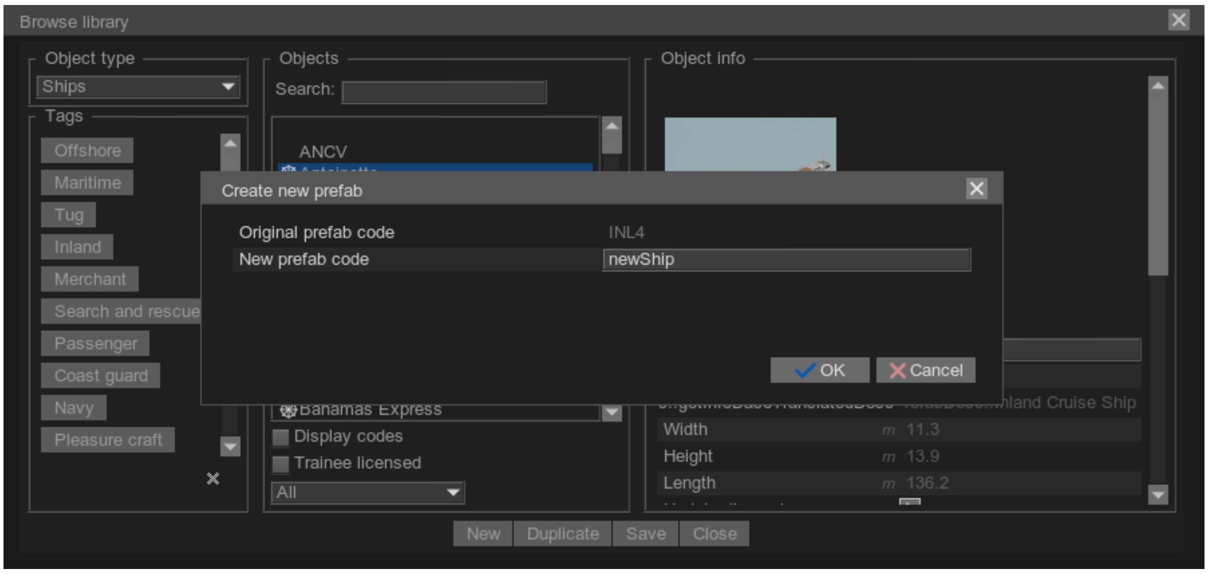
Chapter 4

Professional Products

This chapter describes the main tasks conducted during this research.
All the solutions proposed have the goal to solve the problems cited in section 3.1 and were discussed together with the Company Coach and Team Content during the Content Workflow Improvement Update meeting (section 1.5).

4.1 Prefabs Duplicator Tool

The solution for this problem consisted of adding a new window (Figure 4.1) in the Object Editor to automatically clone chosen by the user a prefab and replace all the links in its assets that pointed to the original prefab folder.



Object type: Ships

Create new prefab

Original prefab code	INL4
New prefab code	newShip

Width	m	11.3
Height	m	13.9
Length	m	136.2

Figure 4.1: *Prefabs Duplicator Tool window.*

This window allows to insert the code of the prefab and to check that it doesn't already exist. The window is openable from the Duplicate button in the Browser

Library which is accessible from the Object Editor.
All the GUI elements in Unigine are generated from .ui files, which are in the XML format. These files describe containers and widgets provided by Unigine. Each of them is described by an XML tag in the UI file. The duplicate button was added from a UI file, and a callback function was specified for when the button was clicked.
This tool has been intended only for internal use, so the button to open the window has been made visible only to developers. The callback click creates a new prefab window, which is a container with UI widgets inside it.
During the development, time had to be spent researching the documentation and the source code of Unigine, to find a way for creating GUID links for the assets of the duplicated prefab.

4.1.1 Unigine Assets Files

For the Unigine asset files, new GUIDs had to be generated. In order to do that, the approach chosen was to process each node and then recursively create all the new links for each asset used by that node (materials, textures, and meshes).

4.1.2 VSTEP Assets Files

The files from VSTEP (Components Files) were not linked through GUID links, but by using XML path links. Because of that, the links could have been replaced the same way as it was done before Unigine introduced the GUID system, but instead with an automated PowerShell script.
The reason why a PowerShell script was created instead of using Unigine Scripting, was because PowerShell allowed faster development and testing compared to writing Unigine Scripting code: in fact, PowerShell scripts can be run almost instantly, while in order to test Unigine Scripting code, it is needed to wait that Unigine starts, which required on average around 2 minutes of time.

The PowerShell script could be called from Unigine by using the function *run* from the *Console* class, allowing to call Windows run commands from Unigine. One con of PowerShell scripts is that they are supported only by Windows, however this script is currently being used internally, and Team Development used Windows OS only.
The functioning of the duplicate algorithm is viewable from the image in the appendix (Figure 7.1).

4.1.3 Testing

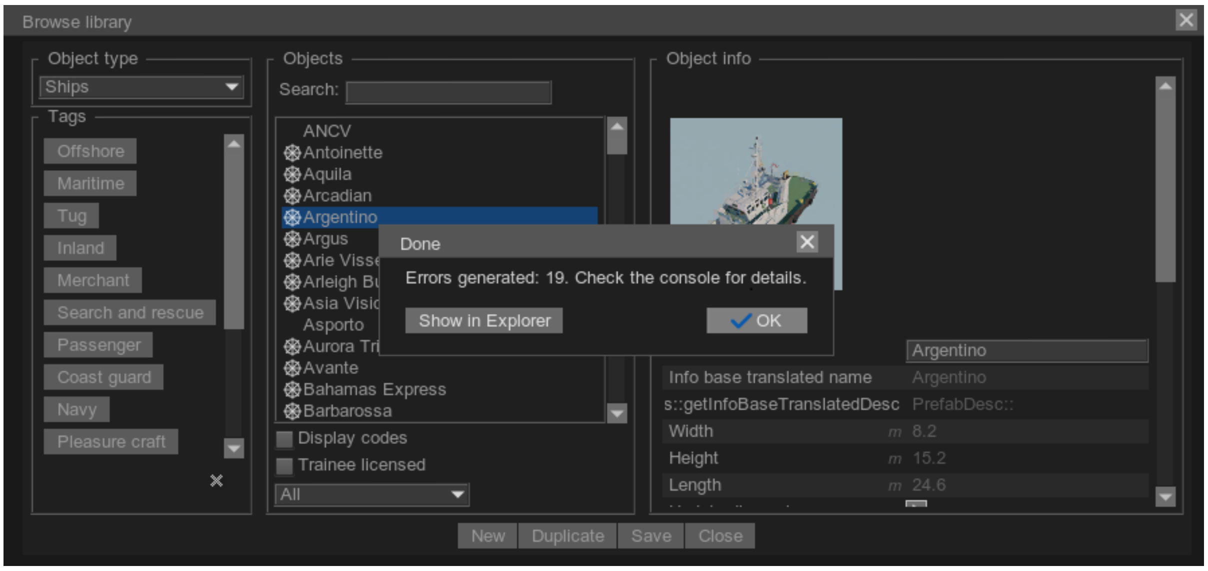
The prefab duplicator tool has been tested by one member of the Team Content and the code was reviewed by two programmers in the Pull Request. For this task, before starting writing code, a proposal document was sent to a member of Team Content and the Company Coach. The document contained an explanation of what the final user would have used the Prefab Duplicator Tool, by providing a high-fidelity image of how the GUI of the Duplicator Tool window would have looked like. This was needed to get feedback and to know if they agreed with the solution proposed.
When the product was further developed, new testing has been performed by Team Content, which consisted in trying to duplicate different kinds of prefabs to see if the duplicated prefab was correctly visualized in Nautis/RS, and to see that all the links of the prefab were correctly generated and not pointing to the original prefab. During the testing a member of Team Content reported that a prefab was not being duplicated correctly as one part of it was not visible in Nautis.
After some further investigation, it has been discovered that this was caused by an issue in the recursive function which was used to process the elements that were children of a node. The problem of that was that the root node was not being processed, so if a node had no children, it would have been exported as an empty node.
The fix this, the solution applied was to process the root node first instead of the first child of the root node, so that the node could have been handled correctly.

4.1.4 Results

The Prefab Duplicator Tool allowed saving Team Content a lot of time. According to Team Content, it was needed up to three hours to manually duplicate a prefab, while with the Prefabs Duplicator Tool, only a few seconds were needed.
Furthermore, the tool could be useful to also check if an existing prefab was set up correctly (Figure 4.2). In fact, during the duplication, errors will be printed in the console if the name of a file is not correct, if a file doesn't exist or if a prefab is linked to another prefab.

4.2 Prefabs Links Checker Tool

To make Team Content aware of which prefabs contained links pointing to other prefabs without needing to manually check each of them, an automated tool has been developed by creating a PowerShell script. The reason for that was that writing code in PowerShell allows faster development, a faster usage by the end-user (Nautis/RS don't need to be launched) and it's also safer as it's separated from the code used for Nautis/RS.

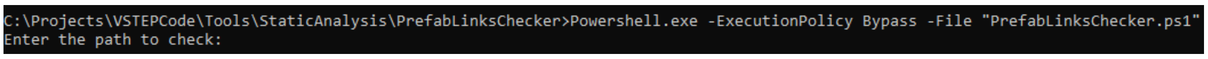


Errors generated: 19. Check the console for details.

Figure 4.2: Prefab duplicator tool result window.

Furthermore, a PowerShell script can be integrated with Azure Pipelines, allowing for example to automatically check the links of a prefab every time a Pull Request that contains changes to a prefab is created.
The tool can be launched from a .bat file in two ways:

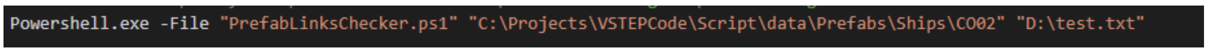
- **No arguments:** the path to scan will need to be inserted by the user and the log output file will be created in the logs folder (Figure 4.3).



```
C:\Projects\VSTEPCode\Tools\StaticAnalysis\PrefabLinksChecker>Powershell.exe -ExecutionPolicy Bypass -File "PrefabLinksChecker.ps1"
Enter the path to check:
```

Figure 4.3: Prefab Duplicator Tool when no arguments are passed.

- **With arguments:** the path to scan and the log output file path can be specified from the arguments (Figure 4.4). This mode has been created to give the script the compability to be supported by Azure Pipelines.



```
Powershell.exe -File "PrefabLinksChecker.ps1" "C:\Projects\VSTEPCode\Script\data\Prefabs\Ships\C002" "D:\test.txt"
```

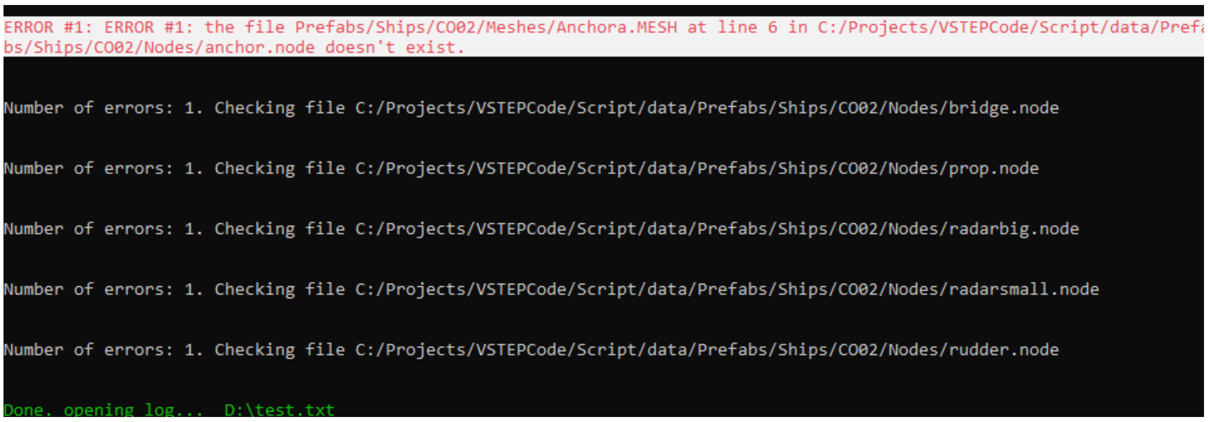
Figure 4.4: Prefab Duplicator Tool arguments.

4.2.1 Tool Functioning

The theory needed for providing a solution for Team Content about this issue was already known from the research performed for the assets links for the Prefabs Duplicator Tool (section 4.1). A flow chart that shows more in detail the operations that this tool performs has been added to the appendix of this paper (Figure 7.2), showing that the tool stores all the GUIDs contained in the *guids.db* file in an array map, and then it checks all the lines of the XML files contained in the prefab folders to find a link. When a GUID link is found, the tool gets the file path relative to that GUID and checks if the file exists or if it's located in another prefab folder. The tool shows an error when the following statements are true:

- A prefab uses assets from another prefab folder.
- The file does not exist on the drive.

When an error is shown to the console (Figure 4.5) it also displays the line number and the name of the file which contained the error, allowing the user to know exactly where the problem is located. Furthermore, when an error is shown, a log file is created, displaying all the errors found during a prefab file/folder links check. This allowed the user to access the list of errors also after the tool was closed.



```
ERROR #1: ERROR #1: the file Prefabs/Ships/C002/Meshes/Anchora.MESH at line 6 in C:/Projects/VSTEPCode/Script/data/Prefa
bs/Ships/C002/Nodes/anchor.node doesn't exist.


Number of errors: 1. Checking file C:/Projects/VSTEPCode/Script/data/Prefabs/Ships/C002/Nodes/bridge.node


Number of errors: 1. Checking file C:/Projects/VSTEPCode/Script/data/Prefabs/Ships/C002/Nodes/prop.node


Number of errors: 1. Checking file C:/Projects/VSTEPCode/Script/data/Prefabs/Ships/C002/Nodes/radarbig.node


Number of errors: 1. Checking file C:/Projects/VSTEPCode/Script/data/Prefabs/Ships/C002/Nodes/radarsmall.node


Number of errors: 1. Checking file C:/Projects/VSTEPCode/Script/data/Prefabs/Ships/C002/Nodes/rudder.node


Done. opening log...   D:\test.txt
```

Figure 4.5: Prefab duplicator tool result window

4.2.2 Testings

The tool has been tested by one member of Team Content, which reported one main problem:

- It has been requested to make the tool more user-friendly when it was launched with no parameters.

To improve the usability of the tool, the code was adapted by adding output messages to the user, checking if the input provided by the user was valid, and providing examples in the console window of inputs that the user could have inserted. The prefab duplicator has been tested by myself, to ensure that all the assets were reloaded correctly and that the node's cache was still working as it was before my code changes.
After Team Content approved the changes, the code has been tested by one programmer, but no relevant issues were reported.

4.3 Prefabs Assets Runtime Reload Tool

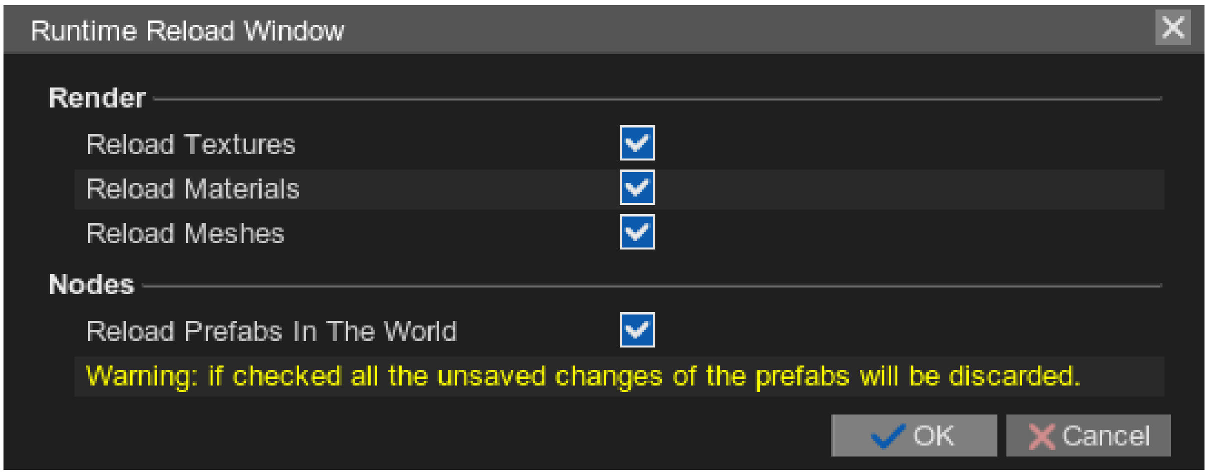
To offer a faster development, the solution to allow Team Content to test assets quickly, was to offer the possibility to reload the prefabs assets while Nautis and RS were running.
The assets of a prefab that needed to be reloaded were:

- Textures
- Materials
- Meshes
- Nodes

The Prefabs Assets Runtime Reload Tool has been made accessible from the dev menu and when clicked, it displays a window with 4 checkboxes that could be selected to decide what kind of assets to reload (Figure 4.6). By default, when the window is opened, all the checkboxes are always checked but in case it's not needed to reload some kinds of assets, those could be unselected from the window. This can be useful, especially when working with a lot of elements, and deselecting one or more kinds of assets, could result in faster reloading of the necessary assets.

Unigine offers a list of commands usable from its console (*Console - Documentation - Unigine Developer*, 2022) which could have been also called from Ungine Scripting code.
While researching in Unigine documentation for options to perform the runtime reload, it was found that the console offered a few commands to reload assets used in the scene while Unigine was running. The commands were:

- *render_streaming_reload texture* to reload all loaded textures.
- *render_streaming_reload mesh* to reload all loaded meshes.
- *materials_reload* to reload all loaded materials.


Runtime Reload Window

Render

Reload Textures ✓

Reload Materials ✓

Reload Meshes ✓

Nodes

Reload Prefabs In The World ✓

Warning: if checked all the unsaved changes of the prefabs will be discarded.

Figure 4.6: Runtime reload tool window.

4.3.1 Reloading Materials

Even though the Unigine console offered a command to perform the materials reloading during runtime, this was not working properly on Nautis/RS. The reason was that they manage the materials in a slightly different way from how Unigine does: in fact, the materials loaded in the scene are used as shared materials, and for each viewport mask, a new material that derives from the shared one is created, and the viewport mask is set to it.
The reason why the console command *materials_reload* could not be used was that by analyzing the Unigine source code, it was discovered the command called the function *clearMaterialInherit*, which set the material parent to null. This means that all the materials used for the viewport masks would have lost their reference to the shared materials and they would have become invalid.

The solution for this issue was to reload each material used in the scene by calling the function *load* of Unigine and setting as a parameter the file path of that material. The function *load* was not invoking the function *clearMaterialInherit*, so it was as expected to allow a runtime reload for the materials.

4.3.2 Reloading Nodes

In Nautis/RS, all the nodes that are loaded from the disk, are cached in a map array. This means that if a node instantiates in the scene gets deleted and then recreated, the prefab nodes would not be loaded from the disk, as the cached node will be loaded instead (Figure 4.7).
For the nodes, there was no console command that allowed to reload them during runtime so to perform the node reloading it was needed to first delete the node element from the cache, delete the prefab, and re-spawn it in the scene.

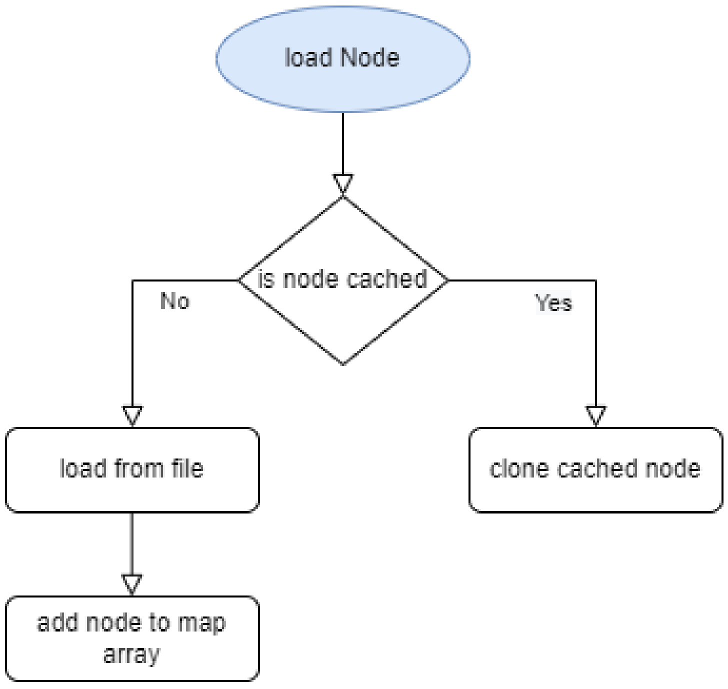


Figure 4.7: Nodes cache functioning flow chart.

4.3.3 Testing

The Prefabs Assets Runtime Reload Tool has been tested by myself, to ensure that all the assets were reloaded correctly and that the node's cache was still working as it was before my code changes.
This tool has been then tested by the Team Content and the code has been reviewed by other programmers in a Pull Request, but relevant issues have been reported by them.

4.3.4 Results

The Assets Runtime Reload Tool allows saving time that otherwise would have had to be spent waiting that Unigine was started (around 2 minutes) each time the assets of a prefab wanted to be tested in the Nautis/RS world.

4.4 Camera Prefab Component Focus Support

To extend the support of the camera focus to also the prefab components, it was needed to add an extra if statement to check if the selected object was a component (Figure 4.8).
After that, to make the camera focus at the right distance of the component, it

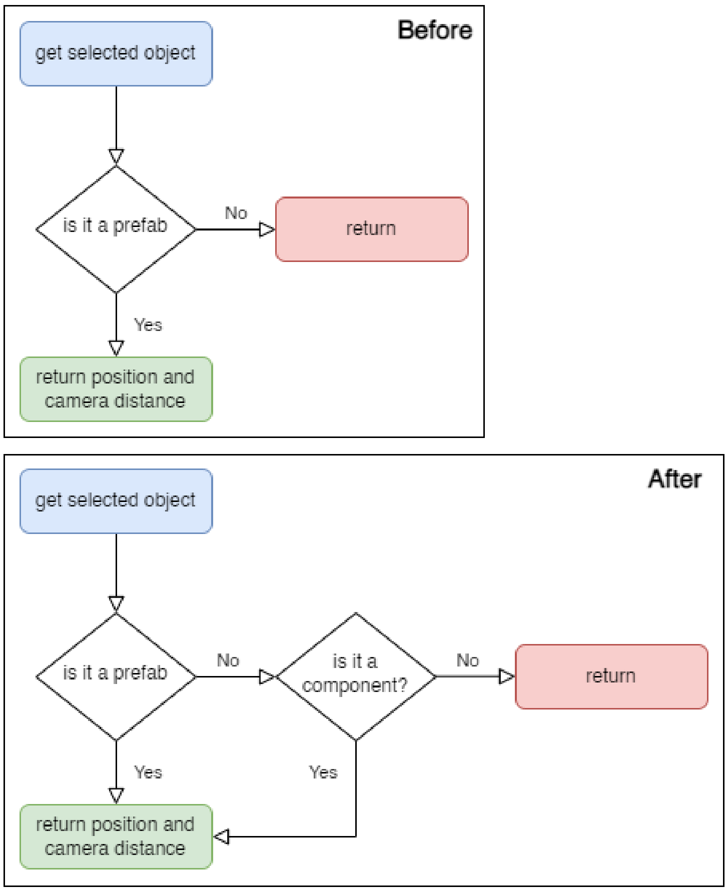


Figure 4.8: Camera focus prefab component support.

was needed to know the size of the component in the scene. To do that the solution adopted was to set the focus distance of the camera equal to the selection bubble radius of the selected component node, plus a fixed margin (Figure 4.9). In this way, the camera could have focused close to the selected component, making it easier for the user to understand where the component selected is located in the scene.

4.4.1 Camera Focus Crash

There were two ways to make the camera focus on one object:

- By pressing the F button on the keyboard.
- By clicking a button in the camera menu.

While working on the camera focus task, a crash was noticed when the camera menu button was being used. The camera focus could support the focus of one or multiple objects at the same time: the objects selection script sums all the

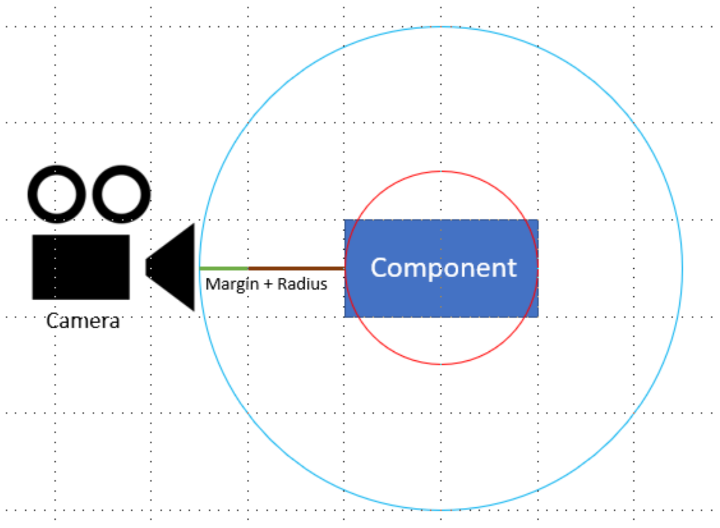


Figure 4.9: Camera focus distance.

positions of the selected objects and divides that by the number of the selected objects. The problem was that there was no if statement checking that the number of the selected objects was more than zero, so it could have happened that the code tried to divide a vector3 by zero, which is not possible. In conclusion, by checking if the number of selected objects was higher than zero, the problem was fixed and the crash was not reproducible anymore.

4.4.2 Testing

The solutions have been tested by a member of Team Content and no issues or feedback were reported.
The code was reviewed by two programmers are reported two main problems:

- The Configuration Menu of Nautis/RS contains a checkbox to disable/enable the selection bubbles. The code wasn't handling this, so if the selection bubbles were disabled and a user would have tried to focus on a component, the program would have crashed because the code would have tried to access the selection bubble variable, which would have been null.

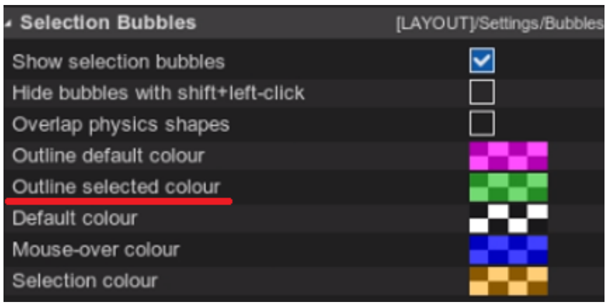
- Nautis has two views. One is the 3D view and the other is the chart view, used to show the simulation from a 2D top-down perspective. The chart view didn't support the selection bubbles, so the program would have crashed way as the other issue, because the bubble variable would have been null.

To fix these problems, the solution was that if the bubble variable would have been null, the focus would have worked without accessing the component selection bubble, letting the camera focus with the prefab size distance, which is the same way the distance would have been the same as before my changes.

4.5 Selection Highlight Visibility

4.5.1 Selected Component Highlight Color

To improve the selection highlight visibility, the solution was to make the selection bubble wireframe change color. A new option was also added to the Configuration Menu (Figure 4.10) to give the possibility to the user to change the color at will.



Selection Bubbles	[LAYOUT]/Settings/Bubbles
Show selection bubbles	☑
Hide bubbles with shift+left-click	☐
Overlap physics shapes	☐
Outline default colour	🟪
Outline selected colour	🟩
Default colour	⬜
Mouse-over colour	🟦
Selection colour	🟧

Figure 4.10: *Selection Bubble Configuation Menu.*

4.5.2 Color Difference

After asking among VSTEP employees, it has been discovered why the selection bubble colors were different for some components. The reason was that the colors of those objects had a meaning for the customers, and the meaning was explained in Nautis/RS manual. However, because the customers didn't have access to the Objects Editor, those colors weren't needed for that editor. Then, the solution for this problem was to ignore those custom colors when using the Object Editor, and use instead the color from the Configuration Menu.

4.5.3 Surfaces Not Visible With Custom Shapes

The surfaces weren't visible for the capsules, because these shapes were created in the code, by combining 2 spheres, and a cylinder. The problem with that was that the code made by VSTEP to create the custom capsule was not taking into account the surfaces, but was considering only the vertices. The solution was to instead use a function provided by Unigine called *createSurface* which allowed to combine together the vertices and the surfaces from different meshes.

4.5.4 Color Trasparency

During the development, it has been also noticed that it was not possible for the user to edit the transparency, as the alpha channel was set to either zero or one. To fix this, the code has been changed to support the transparency blending by using the alpha value provided in the Configuration Menu.

4.6 Flag Line Items Spacing

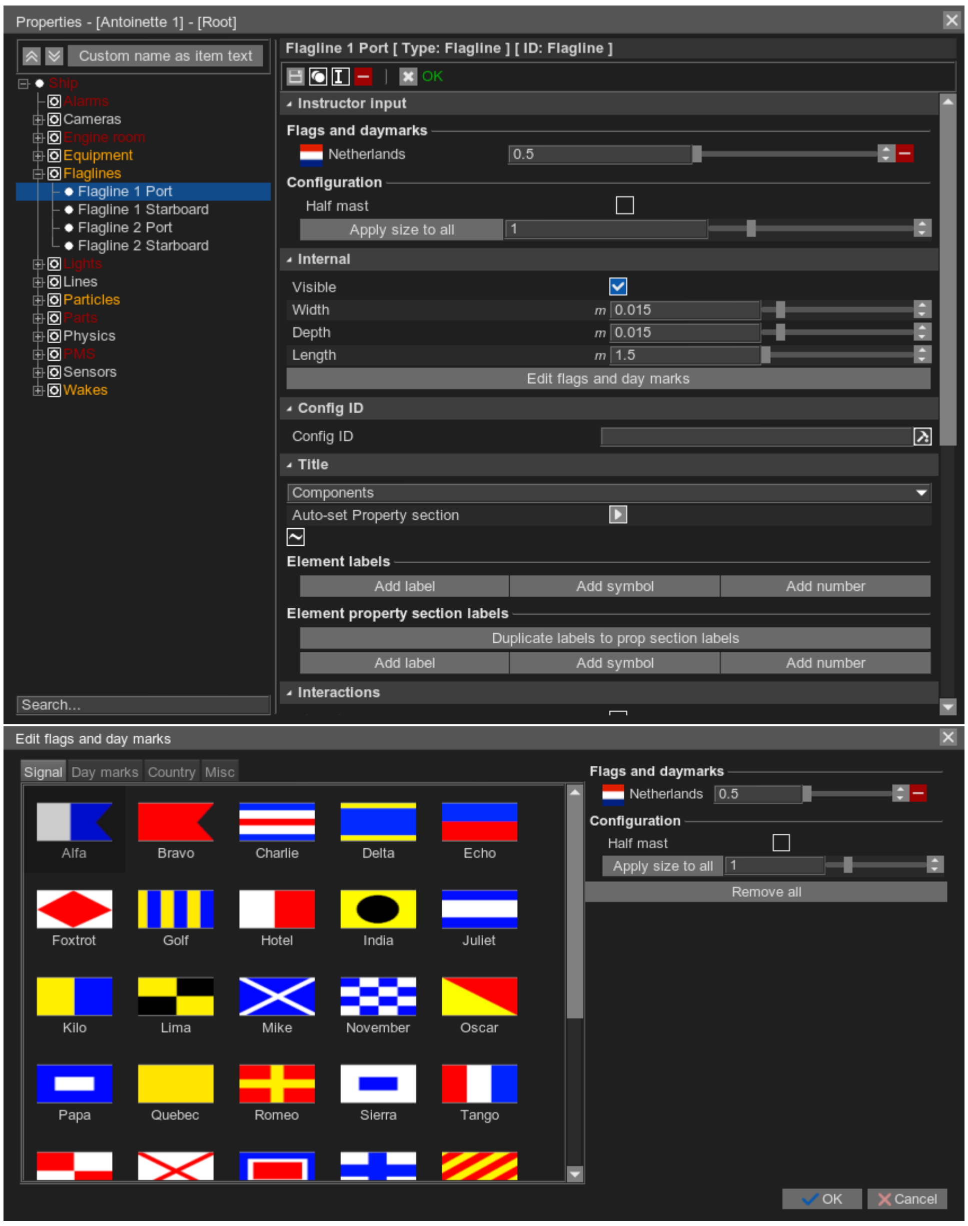
The vertical flag lines could be added on a ship from the Prefab Property Window in the Object Editor (Figure 4.11), where it was also possible to add the items to each flag lines.
Nautis had 4 kinds of flag line items:

- Signals
- Day Marks
- Country
- Misc

The solution applied for this problem was first discussed during the Content Workflow Improvement Update and consisted of adding a new slider to choose the space between the flag line items (Figure 4.12): the default space for the items was set the same as before my changes (0.1) and 0 for the Day Marks blue cones, in order to don't have any gap in between. Despite that, it would have been always possible to customize the space in between at will through the slider.

4.6.1 Wrong Flag Line Items Scaling

A problem found during the development was about the scaling of the flag line items. Since they were children of the flag line, when changing the scale of the ladder from the Configuration Menu, the scaling of the items was also affected (Figure 4.13). The function that created the flag line items was called "rebuild-flagLineItemVisuals" and was invoked each time the flag line items needed to be rebuilt (for instance when the flag line was loaded, when a flag line item was added, when a flag line item was removed, etc). What the function *rebuild-flagLineItemVisuals* did was remove all the items from the flag line and then re-adding them, calculating the correct vertical offset to use for each item. The *rebuildflagLineItemVisuals* function was already calculating the correct scale of the parent object by dividing 1.0f by the scale of the flag line, however this function was not being called when the flag line's scale was being modified.
The solution for this problem was then to call *rebuildflagLineItemVisuals* every



Flagline 1 Port [Type: Flagline] [ID: Flagline]

Instructor input

Flags and daymarks

Netherlands 0.5

Configuration

Half mast ☐

Apply size to all 1

Internal

Visible ✓

Width m 0.015

Depth m 0.015

Length m 1.5

Config ID

Config ID

Title

Components

Auto-set Property section

Element labels

Element property section labels

Interactions

Flags and daymarks

Netherlands 0.5

Configuration

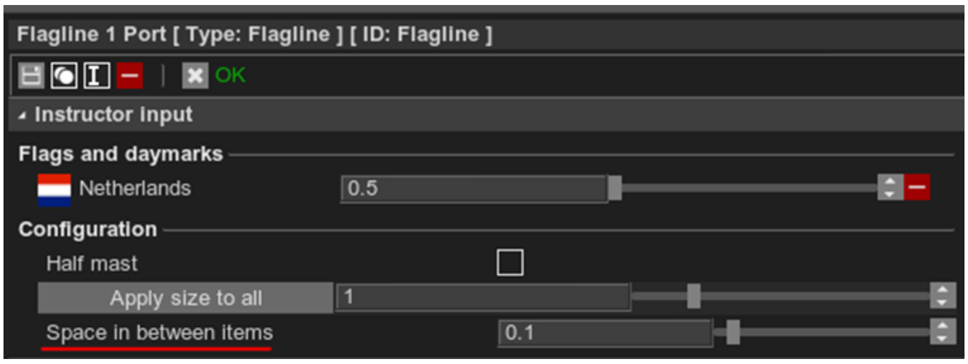
Half mast ☐

Apply size to all 1

Figure 4.11: Flag line Configuration Window.

time the scale of the parent was changed, in the update method, since there was already another function that was being called when the flag line's scale changed.

4.7 3D Studio Max Plugin


Flagline 1 Port [Type: Flagline] [ID: Flagline]

OK

Instructor input

Flags and daymarks

Netherlands 0.5

Configuration

Half mast

Apply size to all 1

Space in between items 0.1

Figure 4.12: Flag line items spacing slider.

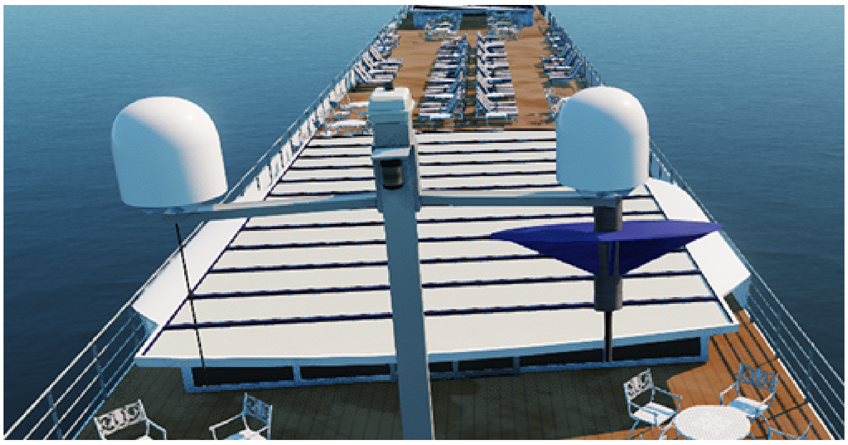


Figure 4.13: Cone wrong scaling.

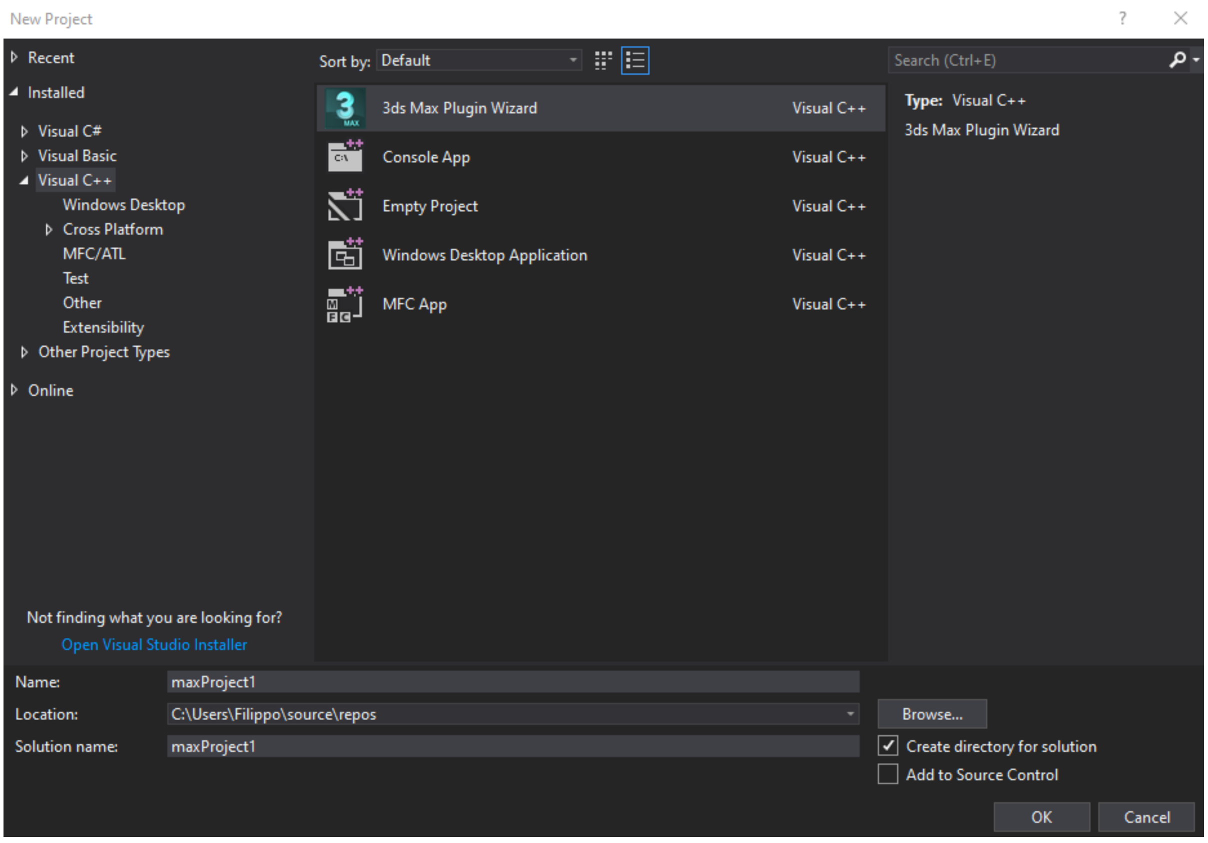
4.7.1 Unigine Source Code

The last version of Unigine where the source code of the 3D Studio Max plugin was found, dates back to May 31th 2017 with the version 2.5.
Unigine offered a .bat script which allowed to build the source code stored in a C++ file, into a 3D Studio Max plugin. However, several errors were showing up when trying to run that file, as the code provided was too outdated to work with the Unigine version that VSTEP was using. This meant that the plugin had to be recreated by using the existing plugin source code as a starting point and adapting it by analyzing the source code that the Unigine Engine used to import the meshes and the animations into the world.

4.7.2 3D Studio Max SDK

Autodesk provides and SDK for each version of 3D Studio Max, to let developers extend the core functionalities of 3D Studio Max.
The SDK contains also a set of files that when inserted into the template directory

folder of Visual Studio, a new option to the "New Project" window of Visual Studio is added (Figure 4.14). This new option allows creating a project that is already



Type: Visual C++
3ds Max Plugin Wizard

Name: maxProject1
Location: C:\Users\Filippo\source\repos
Solution name: maxProject1
[✓] Create directory for solution
[ ] Add to Source Control

Figure 4.14: 3D Studio Max Visual Studio Wizard.

set up to be run and debuggable on 3D Studio Max. In fact, when pressing the "Start" button on Visual Studio, after the code was compiled, the plugin file is created and automatically inserted in the "Plugins" folder, and 3D Studio Max was launched.

4.7.3 Mesh File Structure

To understand how to import and export .mesh files, it was first needed to know how the file format was structured. The documentation of Unigine described the content of mesh files as a sequence of data needed to describe a static mesh or an animation (*Mesh File Formats - Documentation - Unigine Developer*, 2022). A detailed overview of the mesh file composition can be found in (Figure Figure 4.15). After analyzing the plugin source code it was noticeable that the structure order of mesh files changed after version 2.5 and this needed to be

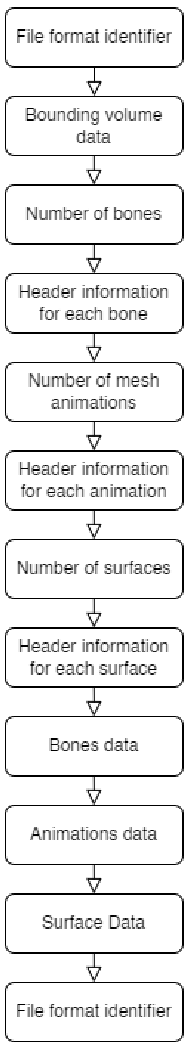


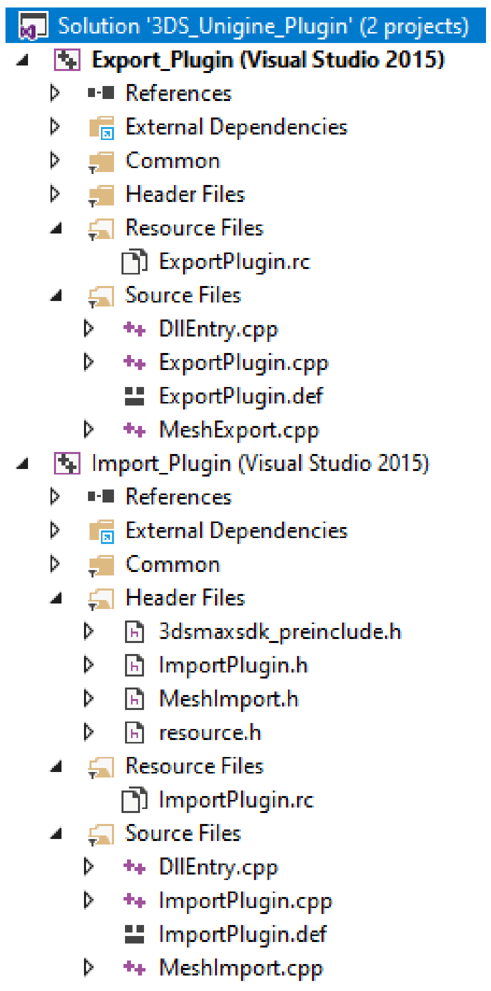
Figure 4.15: Mesh file data structure.

adapted for the new plugin to be compatible with the mesh files that VSTEP was currently using.

4.7.4 Implementation

As mentioned before, a new plugin had to be created. To do that a new empty Visual Studio solution was created, containing two projects created with the 3D Studio Max Plugin Wizard, one for importing and one for exporting (Figure 4.16). When the projects were created, a series of files are added to each of them by the Wizard Tool, including:

- **Plugin.rc** a file used to design the plugin windows interface.
- **Plugin.cpp** the class that contains the basic information of the plugin such



```
Solution '3DS_Unigine_Plugin' (2 projects)
  Export_Plugin (Visual Studio 2015)
    References
    External Dependencies
    Common
    Header Files
    Resource Files
      ExportPlugin.rc
    Source Files
      DllEntry.cpp
      ExportPlugin.cpp
      ExportPlugin.def
      MeshExport.cpp
  Import_Plugin (Visual Studio 2015)
    References
    External Dependencies
    Common
    Header Files
      3dsmaxsdk_preinclude.h
      ImportPlugin.h
      MeshImport.h
      resource.h
    Resource Files
      ImportPlugin.rc
    Source Files
      DllEntry.cpp
      ImportPlugin.cpp
      ImportPlugin.def
      MeshImport.cpp
```

Figure 4.16: Visual Studio Project Overview.

as name, description, extension supported, and a method called "doImport" or "doExport" (depending on the project), which are invoked when a file is imported/exported.

The way the plugin works is that when importing a mesh, a data buffer reads all the data of the file and parses them in 3D Studio Max objects, while for the exporting a mesh, a data buffer is used to write the data contained in the 3D Studio Max objects into a .mesh file.
When the compilation of the plugin was successful on Visual Studio, two files were created: a file with the extension .dli for the import plugin, and one with the extension .dle for the export plugin.

4.7.5 Testing

The plugin has been tested by three members of the Team Content who didn’t report any issue except for the UI of the plugin not being visible when exporting a mesh.
The reason why this was happening was that unlike with 3D Studio Max 2017, the plugins for 3D Studio Max 2020 and 2022, also required the localizable resources files in the plugin folder created during the project compilation.
After providing those files, Team Content was then able to see the dialog in the same way as it was visible with the old plugin.
This task allowed Team Content to work on Unigine .mesh files on 3D Studio Max, without worrying whether or not the license was being used by some other employee.

Chapter 5

Bug Fixing And Optimization

5.1 Dark Environment Bug

This bug was caused because the time was not handled correctly in the C++ code, and it was invalid.
A scenario in Nautis/RS contains two times:

- **The UTC time:** which stands for Coordinated Universal Time and represents time standard across the world.
- **The local time:** representing the local time in the scenario.

A scenario in Nautis can be based on an environment located in the world. The configuration file of an environment contains different information such as the longitude, the latitude, and the time zone. The ladder was being used to calculate the UTC time by doing:

$$UTC = localtime - timezone$$

The reason why the time was invalid for some environments was that the UTC time hour was less than zero or more than 23.
For instance, if the local time hour was nine and the time zone hour was 10, then the UTC time would have been -1, which is not valid.
To Fix this, the solution was to make the code use the function *mktime*, which is a built-in C++ function contained in *time.h* that takes as a parameter the pointer of the time data variable and adjusts it automatically in case the value is not correct.

5.2 Environment Layers Crash

After time spent researching the cause of the crash, it was found that the reason for it was that 2 objects with the same id were loaded in the environment, and

this wasn't possible because each object's id must be unique.
In fact, when a layer gets enabled, the id of all the objects contained in that layer gets stored in an array map, which is being used for two reasons: to check that the ids are unique and to generate new ids.
The solution for this problem was to also store the ids of the objects in the disabled layers when the environment was loaded. In this way, the ID generated for all the new objects would have been unique and the crash fixed.
During the development, two further problems were found when loading a new environment.

- Environments with no layers were not being cleaned.
- The used IDs array map was not cleaned when loading a new environment.

5.2.1 Empty Environment Not Being Cleaned

When the Environment Editor was opened, an empty environment was loaded. However, this environment was special, because it didn't contain any layers. When a new environment was loaded, the objects in all the layers were being destroyed, but because the default environment didn't have any layers, if a new environment is loaded, all the objects added previously to the default environment weren't deleted. This could have also caused a crash if the new environment contained objects with the same ID as an object in the empty environment.
The solution for this bug was to destroy all the remaining objects in the environment after destroying all the enabled layers.

5.2.2 Used IDs Array Map Not Being Cleaned

The array map containing all the used IDs, used to calculate the IDs of the new objects, was never removing any element of it., was never removing any element of it. This means that for example if a ship was added to an environment, its ID would have been *ship0*. However, If then another environment was loaded, and a new ship of the same kind as the previous one was added to the environment, the ID would not have been ship0, but ship1.
The class managing the ID of the objects, was using 2 array maps to store the ids, one called *locked ids* and *used id*. The *Locked ids* array map elements were added and removed when an object was created/destroyed. However, it was not necessary to have 2 array maps at the same time, so *used id* could be deleted and replaced with *locked ids*.

5.2.3 Testing

The solutions have been tested by a member of Team Content and the code was reviewed by two programmers are reported two problems but no major issues or

feedback were reported.
The solutions have been also tested by a member of Team QA, who reported a crash while loading a recorded session. The crash was caused because the destroy functions called in the method *Layers::setDesc* for the task in section 5.2.1, were called while the recording was being loaded. So for example it was happening that while a ship was being created, then it was immediately destroyed while its components were being initialized.
The solution for this was to clear the objects in the environment is a function that was not being called when loading an environment, because when a session recording was loaded all the objects were already being destroyed in the function *Player::playFrame* (Figure 5.1).

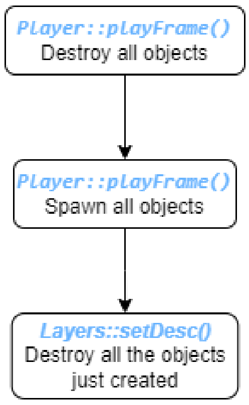


Figure 5.1: Recorded session loading.

5.3 Slow GIT Branches Checkout

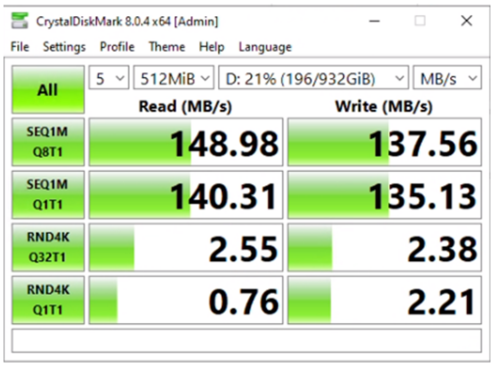
To understand why GIT was taking too long to switch branches on Team Content's computers, the research focused on the two factors that impacted the performance of GIT:

- The computer hardware.
- The antivirus scanning.

5.3.1 Device Storage

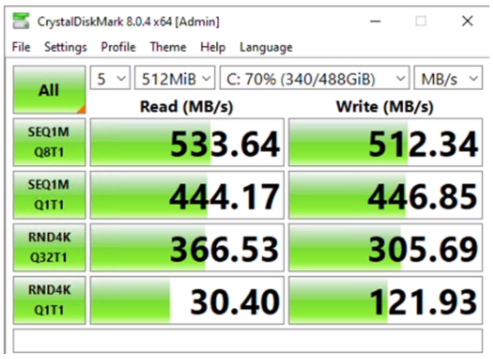
About the computer hardware, what impacted the most was the device storage type.
Especially when copying small files, an SSD is way more performant than an HDD.

That has been confirmed also by testing the speed of the SSD and the HDD on a test computer at the office (Figure 5.2 and Figure 5.3).


CrystalDiskMark 8.0.4 x64 [Admin]
File Settings Profile Theme Help Language
5
512MiB
D: 21% (196/932GiB)
MB/s

All	Read (MB/s)	Write (MB/s)
SEQ1M Q8T1	148.98	137.56
SEQ1M Q1T1	140.31	135.13
RND4K Q32T1	2.55	2.38
RND4K Q1T1	0.76	2.21

Figure 5.2: HDD Speed Test


CrystalDiskMark 8.0.4 x64 [Admin]
File Settings Profile Theme Help Language
5
512MiB
C: 70% (340/488GiB)
MB/s

All	Read (MB/s)	Write (MB/s)
SEQ1M Q8T1	533.64	512.34
SEQ1M Q1T1	444.17	446.85
RND4K Q32T1	366.53	305.69
RND4K Q1T1	30.40	121.93

Figure 5.3: SSD Speed Test

As it is noticeable, the SSD tested is faster than an HDD and can be up to 14 times faster than HDD when processing 4kb files.
The reason why the SSD could not be used by some employees was that the SSD in Team Development's computer was too small to contain the GIT repository, as the more changes were stored over time, the bigger the folder's size would have been.
To minimize this problem, two possible solutions were proposed:

- When the GIT repository started to take too much space, the repository had to be backed up on the HDD, the GIT repository had to be deleted from the SSD, and then re-downloaded. After that, the size would have been reduced and the repository can have been contained again on the SSD. It's important to point out that this solution could have been adopted only when all the important changes were pushed to the remote server. Unfortunately, even though research has been performed to find a GIT command that allowed to reduce the space of the GIT repository, the previously cited method was the most efficient discovered.

- Ask VSTEP to provide SSDs with higher disk space. Nowadays SSDs are relativity cheap and can bring benefits long term.

5.3.2 Antivirus

The impact of the antivirus depends on the operation performed (numbers of files, size of the files, etc). Some operations have been tested in the section below (section 5.3.3).
The reason why an Antivirus slows down the performance is that whenever a file

is being opened, saved, copied, or renamed, the antivirus first scans it and then allows Windows to perform that operation. This means that for example, if a file is copied to another folder, Windows cannot immediately copy the file, but first has to wait that the antivirus confirms that that file is safe to be copied, which requires more time.

5.3.3 Performance Impact

The performance impact has been tested with 3 operations, by comparing the time between SSD and HDD, and process whitelisted and process not whitelisted. All the times have been measured several times, and the value that you can read in the graphs represents the average of those measurements.

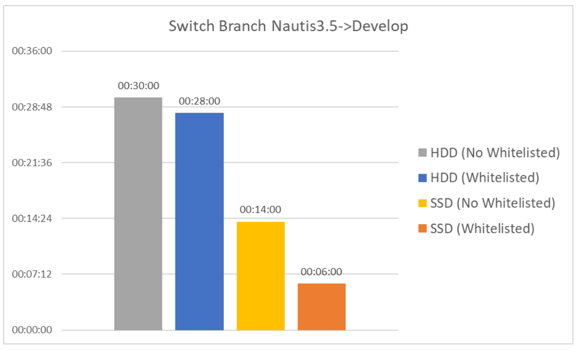


Figure 5.4: Switching GIT branch times.

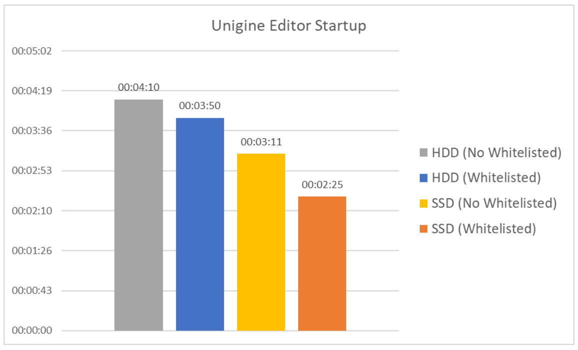


Figure 5.5: Launching Unigine Editor times.

For the Visual Studio Build measurement (Figure 5.6), Visual Studio mostly uses the CPU to compile the solution. Moreover, the files created have a big size, so they are fast to be saved also on an HDD. For this reason, compiling on an HDD or an SSD didn't show any relevant differences in time.

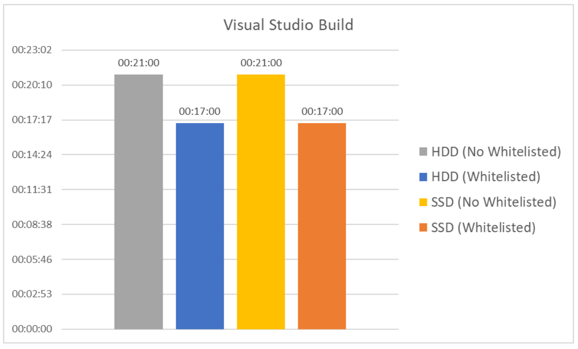


Figure 5.6: Visual Studio compilation times.

5.3.4 Results

After proving that whitelisting process would make VSTEP save time during development, several processes have been whitelisted by working together with Team ICT, to provide an improvement on the performance, without requiring any expenses by VSTEP. When Team ICT updated the process whitelisting, all the antivirus installed in the employees were automatically updated, so I was not needed to update every computer used by Team Development.

Chapter 6

Conclusion And Reflection

This thesis aimed to improve the workflow of Team Content. This has been achieved by researching and developing the following solutions for the research questions.

How can the duplication of the prefabs be improved to let the employees perform it faster?

A tool has been created, which allows duplicating prefabs in a faster and safer way, reducing the time needed for that from up to 3 hours to a few seconds per prefab. The tool carries out the following operations:

- Creates new links by generating new GUIDs for all the Unigine files.
- Changes the .prefabinfo file, to provide a new name to the new prefab.
- Creates new links for VSTEP files (i.e. Cfg folder and root folder files). To do that a PowerShell script has been used which allows faster development and testing than creating code with the Unigine Scripting language.
- Generates errors if during the duplication the name of a file is not correct, if a file doesn't exist or if a prefab is linked to another prefab.

How can the prefabs links be checked in a faster way to let the employees know if they are not set up correctly?

An automated tool has been developed to make Team Content aware of which prefabs contained links pointing to other prefabs.
The tool shows an error the following conditions are true:

- A prefab uses assets that are contained in another prefab folder.
- An asset file used in a prefab does not exist on the drive.

This tool allowed VSTEP to save up to half a day per object, as in this way issues could be caught earlier and new tickets/branches didn't need to be made afterward.

How can prefabs assets be reloaded without needing to restart Nautis/RS?

The runtime reload was archived by creating a tool that could be used by the user to decide what type of assets to reload runtime among textures, materials, meshes and nodes. The tool allowed to save up to 10 minutes per reload, depending on how much time the computer needed to restart Nautis/RS.

How can the camera focus show where a prefab component is located?

The support for the camera focus has been extended also to support prefabs components and the focus distance was based on the size of its selection bubble shape.

How can the selection highlight be improved to increase the visibility for the user?

The selection highlight of the selection bubbles components improved by applying the following changes:

- A new color for when a component was selected was added to Configuration Menu and applied to the selection bubble outline.
- Support to the selection bubbles opacity was added.
- When using the Object Editor, all the selection bubbles have been changed to have the same color.
- The mesh surfaces have been added to the custom capsule shapes.

How can Team Content customize the distance between each flag line item to allow the ship to also pass under low bridges?

A new slider to choose the space between the flag line items was added to the Configuration Menu: the default space for the items was set as the same as before the changes (0.1) and 0 for the Day Marks blue cones, in order to don't provide any gap in between. Despite that, it would have been always possible to customize the space in between at will through the slider.

How can Unigine mesh files be imported/exported on a newer version of 3D Studio Max 2017?

Unigine meshes could be imported/exported on new versions of 3D Studio Max by creating a new Visual Studio solution containing two projects added through the 3D Studio Max SDK wizard. The project has been created by using the plugin source code as a starting point and adapting it by analyzing the source code that Unigine Engine 2.13 used to import the meshes and the animations into the engine.

Why do some scenarios load at night and display the wrong date, and how can this be fixed?

The reason for this issue was that the time calculated in the C++ code was not valid. The solution was to use a built-in function called *mktime* that adjusted the time used by an environment when this is not correct.

Why do some environments make RS/Nautis crash when loaded in the Scenario Editor, and how can this be fixed?

The crash was caused by more than one object having the same ID.
The solution for this was to store the ids of the objects in the disabled layers when the environment was loaded. In this way, the ID generated for all the new objects would have been unique and the crash fixed.

How can switching GIT branches be more efficient in terms of time?

Switching GIT branches could be made faster by whitelisting the antivirus the GIT clients used by VSTEP's employees, and by having the checkout folder on the SSD instead of the HDD.

The changes that all the tasks above made to Nautis/RS allowed Team Content to have tools that were more efficient in terms of time and usability. In this way, VSTEP can be more competitive in the market as it can develop content in a faster way than how it was before this research.

This internship was an excellent opportunity to practice the subjects learned at school and to experience more about what a job in a real company looks like.
Working in the Simulation Team at VSTEP in Rotterdam has been definitely very stimulating, and I have felt like being part of a company composed of employees with a very high level of expertise that were always available to help or give suggestions when needed.

I retain myself satisfied with the overall internship experience, as I managed to complete all the tasks discussed with Team Content, including a problem which I was not sure it was possible to solve, like the 3D Studio Max plugins problem, as there was not much documentation provided by Unigine which explained how the plugin worked, or how to compile the plugin for a newer version of 3D Studio Max.
Furthermore, the communication between me and the Team Content led to good

results, as I was able to communicate with its members during the meetings, at the office and on Microsoft Teams, allowing me to get constant feedback on my work to deliver products that satisfied their needs in terms of user experience and performance.

On the other hand, something that I feel I could have done differently was the time estimation for each task I worked on, which most of the time was too low compared to the time that I actually spent on it. The reason for that was that often there were unexpected issues that I didn't take into account during the estimation.
Estimating how much time a task will require, it's not something easy to predict, especially when somebody it's not experienced in doing so. Something that I would do in the future is to be more pessimist when estimating time, because it often happens that in a task something comes up that requires extra time work. Something else that I noticed that helped me to give a more accurate time estimation, is dividing a task into smaller tasks because of three main reasons:

- It's easier to give a time estimation on small tasks than on big tasks
- They create a timeline for the task, so that a team leader has more clear what the progress is
- It helps have a more clear overview of the steps that are needed to complete the main task.

In conclusion with this internship experience I gained new knowledge, skills and worked with new people, giving me new insights to start my career after graduation.

References

Aničić, D., & Nestorović, O. (2020). Globalization's influence on the competitiveness of national economy. *Journal of Process Management. New Technologies*, *8*(1), 12–17. doi: 10.5937/jouproman8-24404

Autodesk. (2021, 10). Retrieved from `https://www.autodesk.eu/`

Azure DevOps Services. (2022). Retrieved from `https://azure.microsoft.com/en-us/services/devops/`

Bang, K., & Markeset, T. (2011, 09). *Impact of Globalization on Model of Competition and Companies' Competitive Situation* (Tech. Rep.).

Console - Documentation - Unigine Developer. (2022). Retrieved from `https://developer.unigine.com/docs/latest/code/console/`

Davidson, P. U. T. (2015, 11). *Harvard professor calls trade deal watershed, says economy lags.* Retrieved from `https://eu.usatoday.com/story/money/2015/11/16/michael-porter-harvard-interview-trade-deal-economy/75599090/`

File System - Documentation - Unigine Developer. (2022). Retrieved from `https://developer.unigine.com/en/docs/2.15.1/principles/filesystem`

Gitman, L., & McDaniel, C. (2005). *The Future of Business: The Essentials (with InfoTrac) (Available Titles CengageNOW)* (2nd ed.). Cengage Learning.

Mesh File Formats - Documentation - Unigine Developer. (2022). Retrieved from `https://developer.unigine.com/en/docs/latest/code/formats/file_formats`

Personio. (2022, 04). *Personio. The People Operating System.* Retrieved from `https://www.personio.com/`

Unigine. (2022). Retrieved from `https://unigine.com/`

VSTEP. (2021, 09). *Response Simulator.* Retrieved from `https://www.vstepsimulation.com/response-simulator/`

VSTEP. (2022, 03). *NAUTIS Maritime Simulator.* Retrieved from `https://www.vstepsimulation.com/nautis-simulator/nautis-maritime-simulator/`

Chapter 7

Appendix

7.1 Reflection

7.1.1 Technical Research And Analysis

Technical research was needed to find the answers to the problems incurred while working on the tasks that aimed to improve the Team Content workflow. This has been done by reading the source code of RS/Nautis, analyzing the Unigine source code and documentation, and gathering information from the internet through websites, articles, videos and forums.

7.1.2 Designing And Prototyping

For this research, high fidelity and live data prototypes have been created to get feedback by providing the model of a solution to a problem.
High fidelity prototypes have been created to give a preview of how the UI would have looked like for those tasks where it was needed, while live data prototypes videos were shown to Team Content members to give them an overview of how the final product would have worked, by implementing only the basic functionalities of the code. In this way, it was possible to implement their feedback easily in the product by changing the way the product looked or worked.

7.1.3 Testing And Rolling Out

All the features developed have been tested by myself and rolled out when they didn't contain any issues. Furthermore, the features were tested by the stakeholders and if they didn't have any remarks the code was checked by other programmers. Every time an issue was found during these phases, the process started all over again, and the code was adapted to fix the issue.

7.1.4 Investigating And Analyzing

This thesis used qualitative and quantitative research as research methods, focusing on understanding what processes of the development were slowing down Team Content and preventing them to work more efficiently, and by how much.
The data were gathered during the Content Workflow Improvement Update meeting (section 1.5) and during the private consultations with the Company Coach and the Team Content members.
Furthermore, more quantitative research has been done for the Slow GIT Checkout branch issue, by collecting measurements of the time needed to perform several actions (Section 4.10.3). These data were collected ten times for each action to provide a reliable result, and the average value of those measurements was taken into account.
During the research phase for each task, it was almost always necessary to conduct research in VSTEP's source code, the Unigine source code, and its documentation to collect technical data for tasks of the assignment.

7.1.5 Conceptualizing

The problems of the research have been discussed during the Content Workflow Improvement Update meeting, with the company coach and Team Content, and for some tasks, also with other employees of the company.

7.1.6 Designings

The products developed were in line with the other company products: for example, the GUI created had the same style as other GUIs in Nautis/RS, and the code has been written with the purpose of having the same structure as the code written by other employees.

7.1.7 Communication

The report aims to explain in a clear way the internship process through a simple structure and easy language usage, and by giving an overview of the choices made and explaining why they were made. Furthermore, the paper uses several references that are referenced by using the APA citation guideline.

7.1.8 Learning Ability And Reflectivity

I worked on a new big project that has a complex structure and functioning. I spent time learning how to use it and understanding how it was structured by watching tutorials and analyzing the samples provided in the SDK. I did my research on the internet when it was necessary and I asked for feedback from colleagues from Team Development to know their opinions about my approach to my tasks. All the tasks that I have worked on allowed me to learn a lot and looking back I feel like I gained so much experience compared to when I started the internship. I mainly worked on 10 tasks during this internship, which are described in more depth in Chapter 4.

7.1.9 Responsibility

During the internship, I tried to behave professionally as much as possible, trying to act like a normal worker in the meetings and in the attitude while working. I tried to work independently by spending time on research and analysis in order to not increase the workload of my colleagues, but asking for help when it was necessary and I was blocked by something. In that case, I tried to ask for help from the person that was the most familiar with that problem, to receive support in an efficient way.

7.2 Prefabs Duplicator Tool Algorightm Flow Chart

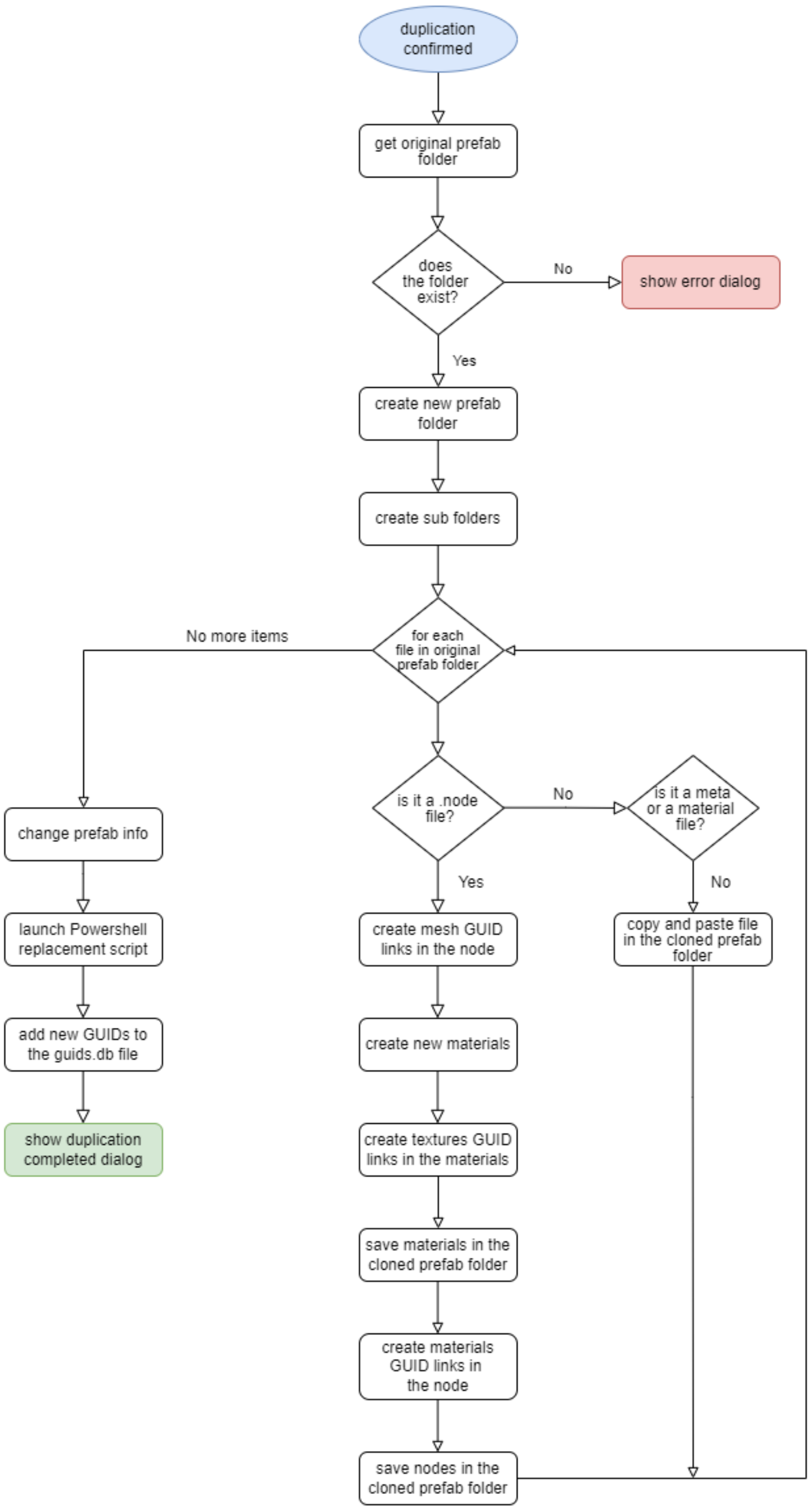

duplication confirmed
get original prefab folder
does the folder exist?
No
show error dialog
Yes
create new prefab folder
create sub folders
No more items
for each file in original prefab folder
is it a .node file?
No
is it a meta or a material file?
Yes
No
change prefab info
launch Powershell replacement script
add new GUIDs to the guids.db file
show duplication completed dialog
create mesh GUID links in the node
create new materials
create textures GUID links in the materials
save materials in the cloned prefab folder
create materials GUID links in the node
save nodes in the cloned prefab folder
copy and paste file in the cloned prefab folder

Figure 7.1: Prefabs Duplicator Tool Algorightm Flow Chart.

7.3 Prefabs Links Checker Tool Algorightm Flow Chart

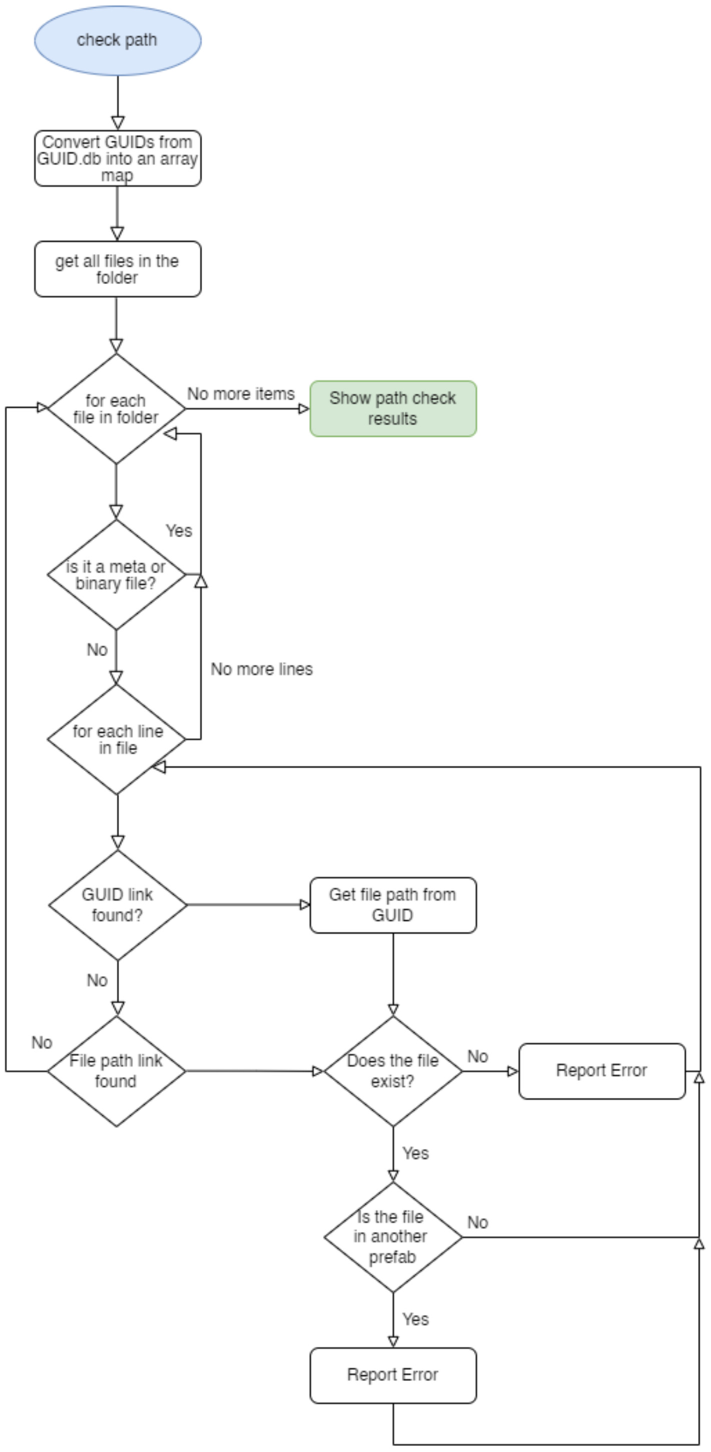


Figure 7.2: Prefabs Links Checker Tool Algorightm Flow Chart.

7.4 Professional Products Visuals

The link below allows accessing an online folder that contains videos and images as a result of the professional products developed during the internship.

`https://shorturl.at/bkpD1`