2407-BSE

Investigative Studio II – R&D Report

CS301.4

Alexander Legner  
William Tetzlaff

# Document Outline

## Table of Contents

[Document Outline 2](#_Toc182507786)

[Table of Contents 2](#_Toc182507787)

[Table of Figures 3](#_Toc182507788)

[Table of Tables 3](#_Toc182507789)

[Research and Development Report 5](#_Toc182507790)

[Executive Summary 5](#_Toc182507791)

[Problem Statement 6](#_Toc182507792)

[Research Objectives 6](#_Toc182507793)

[Methodology 7](#_Toc182507794)

[Week-by-Week Breakdown 7](#_Toc182507795)

[Week 1 7](#_Toc182507796)

[Week 2 8](#_Toc182507797)

[Week 3 8](#_Toc182507798)

[Week 4 9](#_Toc182507799)

[Week 5 10](#_Toc182507800)

[Week 6 10](#_Toc182507801)

[Week 7 11](#_Toc182507802)

[Week 8 11](#_Toc182507803)

[Problems and Solutions 12](#_Toc182507804)

[Problem 1: Godot Addons Not Native Being Native to C# 12](#_Toc182507805)

[Problem 2: Custom Build of Godot Editor 13](#_Toc182507806)

[Problem 3: Godot Failing Build Process Randomly 14](#_Toc182507807)

[Problem 4: Godot Removing Editor Set References 15](#_Toc182507808)

[Problem 5: Scenes Can’t Execute Scripts Code Within Subscenes Automatically 16](#_Toc182507809)

[Problem 6: Code Running When Loading Project 16](#_Toc182507810)

[Problem 7: Unable to Serialize C# Classes Which Inherit from Node 17](#_Toc182507811)

[Problem 8: SVG Images Saving Twice Within Scene 18](#_Toc182507812)

[User Testing 19](#_Toc182507813)

[Task 1: Create and Load Into a New World. 19](#_Toc182507814)

[Task 2: Open 2 Inventories and Move a Partial Stack of Items 19](#_Toc182507815)

[Task 3: Open Component Selection and Equip One into Hot Bar 20](#_Toc182507816)

[Task 4: Build Multiple Components and Multiple Vessels 20](#_Toc182507817)

[Task 5: Pause And Save The Game Then Exit to Main Menu 21](#_Toc182507818)

[Task 6: Delete Your World from The World Manager 21](#_Toc182507819)

[Results 22](#_Toc182507820)

[Recommendations 22](#_Toc182507821)

[Conclusion 23](#_Toc182507822)

[**References** 24](#_Toc182507823)

## Table of Figures

[Figure 1: Converting GDScript to C# Addon Example 12](#_Toc182507830)

[Figure 2: Godot Custom Editor 14](#_Toc182507831)

[*Figure 3: Godot .NET Random Build Error* 14](#_Toc182507832)

[Figure 4: Example for Checking a Variable Reference 15](#_Toc182507833)

[Figure 5: UI Scene Godot 16](#_Toc182507834)

[Figure 6: \_Ready - Is Editor Check 16](#_Toc182507835)

[Figure 7: Custom JSON Data type Example Vector3 17](#_Toc182507836)

[Figure 8: Example of Custom Interface Class 17](#_Toc182507837)

[Figure 9: SVG Textures Fix 18](#_Toc182507838)

[Figure 10: User Testing Results Graph 22](#_Toc182507839)

## Table of Tables

[Table 1: User Testing Task 1 Results 19](#_Toc182507824)

[Table 2: User Testing Task 2 Results 19](#_Toc182507825)

[Table 3: User Testing Task 3 Results 20](#_Toc182507826)

[Table 4: User Testing Task 4 Results 20](#_Toc182507827)

[Table 5: User Testing Task 5 Results 21](#_Toc182507828)

[Table 6: User Testing Task 6 Results 21](#_Toc182507829)

Intentionally Blank

# Research and Development Report

Executive Summary

**Project Title:** Architects in Void

**Project Overview:** Architects in Void will be a space exploration game that aims to immerse players in the deep void of space. Players will embark on a journey to create their colony in space while battling the dangers of being in space and other entities in the empty void around them.

**Objective:** The primary objective of Architects in Void is to deliver an immersive yet fun space exploration experience that balances realism with accessible gameplay mechanics. By utilizing realistic space physics simulations and intuitive building and flying mechanics, the game aims to immerse the players in realism of space while still engaging in an entertaining gameplay experience.

**Key Features:**

* **Realistic Physics Simulation:**

Players will experience realistic space physics including zero gravity physics and realistic thrust/flight dynamics. The realism enhances gameplay by challenging players to master the complexities of being in space.

* **Ship Customization and Construction:**

Players will be able to design and build their spaceships. This will allow players to create unique ships suited to their mission’s needs. Each design choice will impact performance, manoeuvrability, and functionality in the game environment.

* **Exploration and Discovery:**

The game offers a procedurally generated void with unique asteroid belts and resources. The universe will be primarily based on our solar system the Milky Way to keep with the realism but will allow players to explore our other planets like never before.

* **Resource Management and Survival:**

The player will be responsible for maintaining their spaceship and equipment. This will require the player to bring or find certain resources to help them stay alive. The unique resources will allow the player to upgrade/edit the ship's components.

* **Missions and Bases:**

The gameplay loop will be around completing certain missions (single-player campaign) which will involve making trips to existing bases or making new bases, upgrading ships, fighting enemies, which will allow the player to feel a purpose in the void. These bases will also be key for the player to resupply and start/end these missions.

**Conclusion:** the proposed space simulation game “Architects in Void” promises to deliver an engaging mix of realism, creativity, and exploration. By combining innovative gameplay mechanics with realistic physics, the game aims to inspire curiosity about space while offering an engaging and immersive gaming experience.

## Problem Statement

How can realistic space physics and mechanics be balanced with intuitive gameplay to create an immersive and enjoyable experience for players in a space exploration and spacecraft piloting game?

Explanation:

* **Realistic Space Physics:**   
  Having actual physics of space which includes orbital mechanics, gravitational forces, momentum, and many more is crucial in creating a believable and immersive experience in space. Research would look at how realistic physics can be incorporated into the game without overwhelming the player with complexity.
* **Intuitive Gameplay:**   
  While realism is important, it is also equally important to make the game accessible and enjoyable for players. The research would explore how to simplify the complexity of space physics into simple and intuitive gameplay and mechanics that players can learn easily which would allow them to focus on space exploration rather than being overwhelmed by technical details.
* **Immersion and Enjoyment:**   
  The end goal is to create a unique experience that is both realistic, engaging, and entertaining for players. Research would include how to create a good gameplay loop within a realistic space environment including the challenges and interactions within the game.
* **Realistic Space Distances**   
  Space has large distances between each planet. These distances if used directly 1:1 scale would be too large for the typical computer to run. Research would explore how we can make the game seem realistic to a player while still being able to run modern computers.

## Research Objectives

**Project Goals:**

* The project's primary goal is to deliver a functional prototype of the video game proposed ‘Architects in Void.’ This prototype will have all its features outlined in the SRS working and be bug-free.
* The secondary goal of this project is to answer the research question: “How can realistic space physics and mechanics be balanced with intuitive gameplay to create an immersive and enjoyable experience for players in a space exploration and spacecraft piloting game?”

**Project Objectives:**

* Learn innovative technologies and use them within the project.
* Overcome any limitations and problems that occur during development.
* Implement all features outlined in the SRS for the prototype.
* Conduct research to investigate the problem and answer the research question.
* Complete all project deliverables on time as outlined in Project Constraints.

**Executive Summary Objective:** The primary objective of Architects in Void is to deliver an immersive yet fun space exploration experience that balances realism with accessible gameplay mechanics. By utilising realistic space physics simulations and intuitive building and flying mechanics, the game aims to immerse players in the realism of space while still engaging in an entertaining gameplay experience.

## Methodology

**Approach to R&D:** The research and development process we are using is agile. This means our development process follows an iterative process with regular feedback, design improvements and development. We used scrum meetings every week, allowing us to have weekly sprints and discuss changes regularly.

**Iterative Development:** Every week in our scrum meetings we went over the outcome of the sprint, what was next to do in the sprint, any problems that occurred during the sprint and any changes that needed to occur to the project.

**Validation Strategy:** To validate our MVP developed by the R&D process, we will conduct user testing to ensure the MVP follows the plan from the IDD. Any changes to the IDD will be accounted for and will be part of the user testing process. We will also test all the functionality listed in the IDD and ensure the entire programs features match the features outlined in the IDD and work as expected.

Week-by-Week Breakdown

Week 1

*Objectives*

* Create initial project structure.
* Create tests around initial ideas.
* Ensure double precision works within chosen environment.
* Start creation of UI.

*Research Activities*

* Learn Godot game loop logic.
* Learn Godot tools.
* Learn about creation of own tools.
* Learn about custom builds of Godot.
* Testing of ideas around component creation.

*Development Activities*

* Setup Godot project. - Create start of UI + main menu.
* Test double floating precision. - Add main font.
* Create player character. - Make window scalable.
* Thruster generation test. - Add settings UI.
* Setup file structure. - Add settings functionality.
* Added custom made plugin (inspector buttons). - Add armour creation test.

*Learnings and Insights*

* Godot allows custom builds + allows development with C# and double floating-point precision.
* UI in Godot is complex and requires code to link buttons to other code.
* C# builds randomly crash the editor. This is a known issue.
* Learnt the basics around the Godot editor and how to create objects/scripts within it.

*Outcomes*

* Strong start of the project with structured file system and custom tools to help development.
* Initial tests for component creation show strong future with the program if developed further.
* UI for main menu complete and settings complete (settings not functional).

Week 2

*Objectives*

* Continue creation of UI + Logic.
* Create customisable controls which load at runtime.
* Continue development of component creation.

*Research Activities*

* SVG textures in Godot.
* Documentation for controls/buttons in Godot.

*Development Activities*

* Pause menu UI + Logic. - Grid drawing.
* Added all customisable controls for project. - Create customisable controls.
* Debug object drawing. - Create box and thruster creation.

*Learnings and Insights*

* SVG textures natively in Godot do not update when scaled. There is a plugin which helps this but would require updating every single texture and re adding them into the UI structure.
* Controls/buttons are managed simply in Godot. Can’t be testing in editor – only at runtime.

*Outcomes*

* Pause menu UI complete.
* Customisable controls created for runtime purposes.
* Can create armour box and thrusters at runtime using mouse controls.
* Added debug objects into project.

Week 3

*Objectives*

* Continue creation of UI + Logic.
* Add backend hot bar for switching between components to create.
* Further development around component creation.

*Research Activities*

* None – focused on development.

*Development Activities*

* Add hot bar scenes temporarily.
* Create HUD UI.
* Further grid snapping for component creation.

*Learnings and Insights*

* Nothing to add.

*Outcomes*

* Added hot bar logic for switching between components to create.
* HUD visuals created.
* Further development around component creation.

Week 4

*Objectives*

* Convert all SVG textures to use improved plugin
* Test exporting of project
* Fix window not going to centre when launching or changing size
* Add / test sound within project

*Research Activities*

* Look at FMOD API for Godot.
* Godot documentation for custom builds and export templates.
* Godot documentation for windows.

*Development Activities*

* Start SVG Textures conversion
* Create generic helper class
* Create and Test export templates
* Fix window centre of screen when launching and changing size
* Add FMOD into project and test

*Learnings and Insights*

* There is a FMOD addon and API for Godot, however we would need to build it ourselves like creating a custom build of the Godot Engine.
* Display Server and Window are the same but with different functions.
* Export templates are needed to export the game to a specific client. Custom ones need to be built when dealing with custom editor options.

*Outcomes*

* Main menu, settings and world selector all converted to use new SVG plugin.
* Created export templates for Windows and Linux. Both working after test.
* Added FMOD into project and got a test sound working.
* Fixed bugs.

Week 5

*Objectives*

* Complete SVG textures migrations to plugin.
* Create inventory for vessels.
* Add more functionality around world management.

*Research Activities*

* None – focused on development.

*Development Activities*

* Add SVG Textures for component selection UI. - Save and load from pause menu.
* Ability to delete worlds from world manager. - Create inventory UI and backend.
* Update and use secondary font. - Finish SVG conversion.

*Learnings and Insights*

* Nothing to add.

*Outcomes*

* All UI now uses improved SVG textures plugin.
* Inventory created for vessels with functioning UI.
* Added saving, loading and deleting of worlds.
* Fixed bugs and UI related issues.

Week 6

*Objectives*

* Create fixed sized blocks.
* Add functionality around selectable blocks.
* Save data to use JSON.

*Research Activities*

* Serialization and deserialization of C# classes to JSON.

*Development Activities*

* Fix thumbnails of components. - Stop player moving when in UI.
* Add a test world environment with gravity. - Add component menu UI.
* Create cockpit and interactable blocks. - Add JSON data converter.
* Save and load vessel data.

*Learnings and Insights*

* Node based classes cannot directly be serialized and deserialized with JSON. A blank class with limited data instead must be used to convert the data.

*Outcomes*

* Created a cockpit which is mountable by the player.
* Created a test world with gravity.
* Fixed UI related bugs.
* Added JSON serialization and deserialization to save / load data.
* Create a component selection UI.

Week 7

*Objectives*

* Implement sounds within game.
* Create documentation for MVP + R&D report.
* Fix any bugs and tidy up code.
* Make sure all code has up to date comments.

*Research Activities*

* User testing for UI in game.
* User testing for functionality of building features in game.

*Development Activities*

* Add FMOD wrapper class to interface with the API.
* Create documents related to MVP + R&D report

*Learnings and Insights*

* UI was functional and clear.
* Some controls were not obvious to user unless they were already familiar with similar control schemes from other related games.
* Users generally liked the component creations process and the customisation that came with it.
* User would also like the option to have only fixed sized objects as well as the drag drop building.

*Outcomes*

* Created a FMOD sounds wrapper class in C#.
* Added sounds into game.
* Created all documentation for project.
* Updated all scripts to following coding standards.

Week 8

*Objectives*

* Get MVP ready for presentation.
* Present MVP to stakeholders.

*Research Activities*

* None – focused on finalising project.

*Development Activities*

* Presentation
* Bug Fixing
* User Testing
* Final touch ups for demo

*Learnings and Insights*

* Nothing to add.

*Outcomes*

* Fixed any bugs that arose from user testing.
* Finalised MVP ready for demo in presentation.
* Create presentation for project.
* Presented project to stakeholder.

## Problems and Solutions

### Problem 1: Godot Addons Not Native Being Native to C#

* **Problem Statement:** Most addons made by anyone using Godot have used GDScript as the programming language for the addon with no native support to C#.
* **Background:** Godot’s primary language is GDScript, and C# is just a supported option with limited support around it. GDScript remains the popular choice for other developers when creating addons for projects within Godot.
* **Impact:** This requires us to either rewrite the entire addon for C# or create a wrapper class to help call the GDScript functions.

#### Proposed Solution (or Implemented Solution):

* Where possible we have left addons used to continue to use GDScript if we don’t need to interface with them. Otherwise, we have written our own converted version of the plugin to support C# scripting.

A screen shot of a computer program

Description automatically generated

Figure 1: Converting GDScript to C# Addon Example

#### Implementation Process:

* Once an addon was identified as needing converted to support C#, we went through the entire addons scripts and made a C# equivalent of addon. The addon was then tested to make sure it worked the same way as expected.

#### Actual Outcome:

* We only had to convert one addon which was the FMOD addon by “utopia-rise” on GitHub.
* The implementation had many functions and scripts to convert but overall works as expected and exposes the addon to C# to work with the rest of our code.

### Problem 2: Custom Build of Godot Editor

* **Problem Statement:** To get Godot working with C# and double precision, we had to build our own custom version of the editor. This means others can’t open our project without creating their own build using the same parameters.
* **Background:** Godot offers a C#/.NET build of the engine directly from their GitHub and Website but not directly for C#/.NET AND double precision.
* **Impact:** This requires us to create a build of the engine for each developer in their own development environment. We also learnt while developing the project we had to build each plugin and addon using this custom build of the engine.

#### Proposed Solution (or Implemented Solution):

* Build the engine for our needs by following the documentation / guide Godot provides.

*Implementation Process (Windows):*

* Pulling the latest stable version of the editor from GitHub.
  + git clone https://github.com/godotengine/godot.git -b 4.3-stable
* Run “scons” to create a build based on your specified instructions
  + scons platform=windows precision=double module\_mono\_enabled=yes arch=x86\_64
  + scons platform=windows precision=double module\_mono\_enabled=yes target=template\_debug arch=x86\_64
  + scons platform=windows precision=double module\_mono\_enabled=yes target=template\_release arch=x86\_64
* Generate the C# “glue”
  + "bin/godot.windows.editor.double.x86\_64.mono.exe" --headless --generate-mono-glue modules/mono/glue
* Run .Net CLI command to configure NuGet
  + dotnet nuget add source C:\Users\<username>\MyLocalNugetSource --name MyLocalNugetSource
* Run python file with NuGet source
  + "modules/mono/build\_scripts/build\_assemblies.py" --godot-output-dir ./bin --push-nupkgs-local "C:\Users\<username>\MyLocalNugetSource" --precision=double
* Test output is correct version

*Implementation Process (Linux):*

* Pulling the latest stable version of the editor from GitHub.
  + git clone https://github.com/godotengine/godot.git -b 4.3-stable
* Run “scons” to create a build based on your specified instructions
  + scons platform=linux precision=double module\_mono\_enabled=yes arch=x86\_64
  + scons platform=linux precision=double module\_mono\_enabled=yes target=template\_debug arch=x86\_64
  + scons platform=linux precision=double module\_mono\_enabled=yes target=template\_release arch=x86\_64
* Generate the C# “glue”
  + bin/godot.linuxbsd.editor.double.x86\_64.mono --headless --generate-mono-glue modules/mono/glue
* Run .Net CLI command to configure NuGet
  + dotnet nuget add source ~/MyLocalNugetSource --name MyLocalNugetSource
* Run python file with NuGet source
  + modules/mono/build\_scripts/build\_assemblies.py --godot-output-dir ./bin --push-nupkgs-local " ~/MyLocalNugetSource--precision=double
* Test output is correct version

A screenshot of a computer

Description automatically generated

Figure 2: Godot Custom Editor

#### Actual Outcome:

* This created our custom working engine we could use in our development environment. This helped meet our requirements for a game engine which is ultimately why we chose this game engine.
* By following the implementation process we had a working Godot game engine with double precison and .NET/C# scripting. This also includes custom export templates which allow us to export the game for anyone to play on their own environment/computer.
* The process is also very similar for when needing to build addons using this custom version as well. We have documented our commands so that we can easily re create them in future.
* We have to had to upgrade already from Godot 4.2 to Godot 4.3 which required us to use the same commands.

Problem 3: Godot Failing Build Process Randomly

* **Problem Statement:** Describe the first problem clearly. What is happening, and why is it a problem? Include any relevant data or examples that help clarify the issue.
* **Background:** It is a known issue in Godot that the build process sometimes fails with Dot Net, but has yet to be fixed: <https://github.com/godotengine/godot/issues/78513>
* **Impact:** The impact is minimal just requiring us to restart and build again which works as normal.

*Proposed Solution (or Implemented Solution):*

* Currently we no proposed solution.

*Implementation Process:*

* At this stage we will just restart the godot editor as needed when this error occurs.

*Actual Outcome:*

* Sometimes get an error when building the project. Works as normal after a restart.
* Will continue to monitor the GitHub issue for a solution to the problem.

A screen shot of a computer

Description automatically generated

*Figure 3: Godot .NET Random Build Error*

### Problem 4: Godot Removing Editor Set References

* **Problem Statement:** Some variables which were set in the editor were randomly deleted when launching the project. This was typically after a merge of the project and meant we couldn’t test the merge until the variables were re added.
* **Background:** It is currently a known issue with Godot but has yet to be fixed. There is a discussion of it on GitHub: <https://github.com/godotengine/godot-proposals/issues/10694>
* **Impact:** When loading the project, sometimes references/variables are lost and it is not obvious sometimes until after you have made significant changes which can’t be reverted.

#### Proposed Solution (or Implemented Solution):

* Each script when starting the game manually checks and fetches the variables/references it needs for every single variable.

A computer screen shot of text

Description automatically generated

Figure 4: Example for Checking a Variable Reference

#### Implementation Process:

* For each script, each variable was individually managed and checked for. This prevents the reference from being null at runtime.

#### Actual Outcome:

* Most scripts followed this implementation process. Once we implemented this process, we have had minimal issues related with this problem.
* Starting the project takes longer than without the checks.

### Problem 5: Scenes Can’t Execute Scripts Code Within Subscenes Automatically

* **Problem Statement:** When creating scenes for the UI, I tried to use sub scenes to manage the different UI windows. However, the code/scripts within the subscenes were not running automatically, only the top-level script would run. This created issues as the UI needed to run together with other scripts to work properly.
* **Background:** Scenes are godot “prefabs” or saved objects. They contain nodes which contain up to a single script. Scenes can contain subscenes. These subscenes were not executing code in their scripts which were located within the subscene.
* **Impact:** Code not running when it’s supposed to breaks the program and logic.

#### A screenshot of a computer Description automatically generatedProposed Solution (or Implemented Solution):

* Use subscenes where only a top-level script is needed to manage it.
* What this means for UI is to use one large scene containing all the UI elements.

#### Implementation Process:

* Make all scenes with sub scenes and scripts into one big scene.

#### Actual Outcome:

* The UI scene now functions and all the code in the scripts run when it is supposed to.
* The UI scene is now a very large scene and needs to be edited carefully as the entire game depends on this large asset.

Figure 5: UI Scene Godot

### Problem 6: Code Running When Loading Project

* **Problem Statement:** Code would execute when the project was initially loaded.
* **Background:** Using the [Tool] tag expands the control you have when coding in C#. Most of our scripts use this tag. This tag also makes the editor call the \_Ready function when loading the project. This is because the tag is also used to create plugins and addons for the editor.
* **Impact:** Code being executed when loading the project caused many errors to occur as some references were missing in the sub scenes that were open at the time.

#### Proposed Solution (or Implemented Solution):

* Briefly explain what solution was implemented to solve the problem. Even though you're focusing on the outcome, it's important to clarify what the solution was, especially if it involved specific technologies, methodologies, or strategies.
* You may include any relevant technical or design decisions that guided the implementation.

#### Implementation Process:

* Add a check to stop the code from running when not in game.

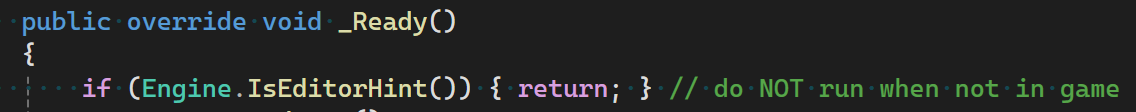


Figure 6: \_Ready - Is Editor Check

#### Actual Outcome:

* This simple line of code added the ready function prevented code from running in the editor when not running in game.

### Problem 7: Unable to Serialize C# Classes Which Inherit from Node

* **Problem Statement:** When trying to serialize a class that inherits from node to JSON, node has extra variables and values that can’t be converted.
* **Background:** C# serialization by default tries to convert all public variables. Some of these variables are custom to Godot and do not have the necessary code to define how to serialize and deserialize them. This code is not accessible to us at the engine level to make private.
* **Impact:** Cannot use serialization on Node based classes.

#### Proposed Solution (or Implemented Solution):

* Create an interface class which stores the necessary data and data types which know how to convert (serialize/deserialize) from C# objects to JSON.

#### Implementation Process:

* Create a class which holds the custom data types of already existing data types except without the extra data.

A screen shot of a computer program

Description automatically generated

Figure 7: Custom JSON Data type Example Vector3

* Each class which needs this feature will get a sub class containing custom variable types which are the same as the normal variable types but without the extra data can convert between the two types implicitly.

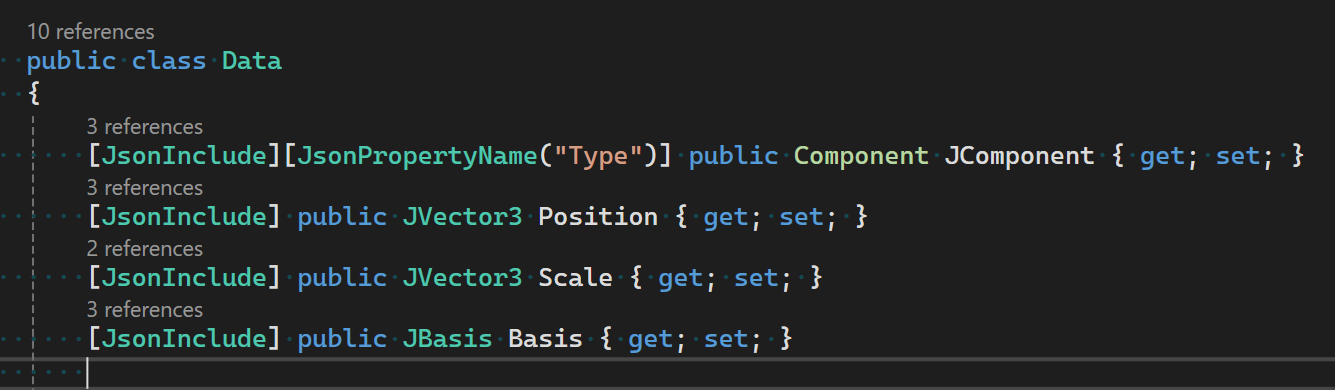


Figure 8: Example of Custom Interface Class

#### Actual Outcome:

* Serialization and deserialization work using the custom interface classes.

Problem 8: SVG Images Saving Twice Within Scene

* **Problem Statement:** With the custom plug we were using, SVG textures were trying to save the local data of their current size to the Scene.
* **Background:** SVG Textures plugin stores the current texture within memory and doesn’t remove it, only updates it at runtime.
* **Impact:** This created merge conflicts with the UI scene which was already a large enough scene due to Problem 5. SVG textures are already saved in their file and don’t need to be stored twice.

*Proposed Solution (or Implemented Solution):*

* Create a check before and after saving to remove all temporary textures from the scene.
* Let the scene save with no temporary data.
* Reload the SVG Textures to still show the UI in the editor.

*Implementation Process:*

A computer screen shot of text

Description automatically generated

Figure 9: SVG Textures Fix

Added this code to the top-level UI script which implements the proposed solution.

*Actual Outcome:*

* SVG textures are no longer saving their temporary data to the scene.
* Noticeable increased saving time when saving a scene involving UI.
* No impact when running the game.

## User Testing

Testers 1 -> 5 consist of 1 random person, 2 gamers and 2 other software developers to make sure the test results represent the general demographic of typical users. For anonymity, we won’t mention which group each tester would categorise as.

### Task 1: Create and Load Into a New World.

Table 1: User Testing Task 1 Results

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Results | Completed  Y/N | Task time  minutes | Ease of Use /5 | Feature Satisfaction /5 | Speed of Program /5 | Error / Bug Frequency |
| Tester 1 | Yes | > 1 min | 3/5 | 4/5 | 5/5 | None |
| Tester 2 | Yes | > 1 min | 4/5 | 5/5 | 5/5 | None |
| Tester 3 | Yes | > 1 min | 3/5 | 4/5 | 5/5 | None |
| Tester 4 | Yes | 2 min | 2/5 | 4/5 | 5/5 | 1 \*See Below |
| Tester 5 | Yes | > 1 min | 3/5 | 4/5 | 5/5 | None |

Any additional features or functionality testers would like to see:

* When creating a new game, load into that game automatically.

Any errors or bugs the tester encountered completing this task:

* Tester 4 entered a special character which windows can’t have in its file name which caused an error to occur.

### Task 2: Open 2 Inventories and Move a Partial Stack of Items

Table 2: User Testing Task 2 Results

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Results | Completed  Y/N | Task time  minutes | Ease of Use /5 | Feature Satisfaction /5 | Speed of Program /5 | Error / Bug Frequency |
| Tester 1 | Y | 2 min | 3/5 | 4/5 | 5/5 | None |
| Tester 2 | Y | 1 min | 4/5 | 4/5 | 5/5 | None |
| Tester 3 | Y | 3 min | 3/5 | 4/5 | 5/5 | 1\* See Below |
| Tester 4 | Y | 3 min | 2/5 | 3/5 | 5/5 | 1 \*See Below |
| Tester 5 | Y | 2 min | 3/5 | 4/5 | 5/5 | None |

Any additional features or functionality testers would like to see:

* A toggle which changed inventory windows behaviour. Closing inventory list should close all inventory windows unless toggled or “pinned”.

Any errors or bugs the tester encountered completing this task:

* Both tester 3 and 4 encountered the same error were trying to split the stack on a item in a window that is not currently selected ignores the input. It appears splitting only works when the window is already selected.

### Task 3: Open Component Selection and Equip One into Hot Bar

Table 3: User Testing Task 3 Results

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Results | Completed  Y/N | Task time  minutes | Ease of Use /5 | Feature Satisfaction /5 | Speed of Program /5 | Error / Bug Frequency |
| Tester 1 | Y | < 1 min | 5/5 | 5/5 | 5/5 | None |
| Tester 2 | Y | < 1 min | 4/5 | 5/5 | 5/5 | None |
| Tester 3 | Y | 1 min | 4/5 | 5/5 | 5/5 | 1 \*See below |
| Tester 4 | Y | 1 min | 3/5 | 5/5 | 5/5 | None |
| Tester 5 | Y | < 1min | 4/5 | 5/5 | 5/5 | None |

Any additional features or functionality testers would like to see:

* More graphically correct component, currently it only shows a scene from Godot.

Any errors or bugs the tester encountered completing this task:

* When having a equip slot already selected, updating the component in that slot doesn’t update what is currently selected until you press the control to equip that slot.

### Task 4: Build Multiple Components and Multiple Vessels

Table 4: User Testing Task 4 Results

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Results | Completed  Y/N | Task time  minutes | Ease of Use /5 | Feature Satisfaction /5 | Speed of Program /5 | Error / Bug Frequency |
| Tester 1 | Y | < 1 min | 5/5 | 3/5 | 5/5 | None |
| Tester 2 | Y | < 1 min | 5/5 | 4/5 | 4/5 | None |
| Tester 3 | Y | < 1 min | 5/5 | 3/5 | 5/5 | None |
| Tester 4 | Y | < 1 min | 5/5 | 3/5 | 3/5 | 1 \*See below |
| Tester 5 | Y | < 1 min | 5/5 | 3/5 | 3/5 | 1 \*See below |

Any additional features or functionality testers would like to see:

* All the testers wanted to see more unique components, at the time for the MVP there are only 3 unique components to create.
* Some of the testers wanted the option to only create fixed sized blocks for the armour just like the cockpit is done.
* One of the testers wanted the rotation of the components to have a snapping option.

Any errors or bugs the tester encountered completing this task:

* Tester 5 created a large component and when looking at it causes the game to lag noticeable.
* Tester 4 tried to merge 2 vessels but couldn’t be due to the game.

### Task 5: Pause And Save The Game Then Exit to Main Menu

Table 5: User Testing Task 5 Results

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Results | Completed  Y/N | Task time  minutes | Ease of Use /5 | Feature Satisfaction /5 | Speed of Program /5 | Error / Bug Frequency |
| Tester 1 | Y | 1 min | 5/5 | 5/5 | 5/5 | None |
| Tester 2 | Y | 2 min | 5/5 | 5/5 | 5/5 | None |
| Tester 3 | Y | 2 min | 5/5 | 5/5 | 5/5 | None |
| Tester 4 | Y | 2 min | 5/5 | 5/5 | 5/5 | None |
| Tester 5 | Y | 1 min | 5/5 | 5/5 | 5/5 | None |

Any additional features or functionality testers would like to see:

* None

Any errors or bugs the tester encountered completing this task:

* None

### Task 6: Delete Your World from The World Manager

Table 6: User Testing Task 6 Results

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Results | Completed  Y/N | Task time  minutes | Ease of Use /5 | Feature Satisfaction /5 | Speed of Program /5 | Error / Bug Frequency |
| Tester 1 | Y | < 1min | 5/5 | 5/5 | 5/5 | None |
| Tester 2 | Y | < 1min | 5/5 | 5/5 | 5/5 | None |
| Tester 3 | Y | 1 min | 5/5 | 5/5 | 5/5 | None |
| Tester 4 | Y | 1min | 5/5 | 5/5 | 5/5 | None |
| Tester 5 | Y | < 1min | 5/5 | 5/5 | 5/5 | None |

Any additional features or functionality testers would like to see:

* None

Any errors or bugs the tester encountered completing this task:

* None

## Results

Figure 10: User Testing Results Graph

These results show the average testers score. Scores show that on average, the ease of use was above 3/5. The speed of the program on average was above 4/5, being 5/5 most of the time. The feature satisfaction on average around 4/5. The bugs/errors encountered on average was less than 1 per on average of all the testers for all the tasks combined. Overall, the testing indicates a positive experience for the testers with room for us to improve upon the MVP.

## Recommendations

At this stage the MVP is ready for a demo to the stakeholder but is not ready for deployment for people to play. There were some bug fixes that came from user testing, but all those specific bugs have been fixed as of the end of the project. Next steps for the MVP include:

* Fixing component and vessel creation.
* Making all save data JSON format.
* Add more components for users to build with.
* Fix UI issues related to HUD.
* Fix new worlds input from allowing special characters.
* Add more detailed 3D models for each component.
* Add planets into the game.
* Add special effects and particle systems.

Long-term Vision:

* Add multiplayer support.
* Add custom modding / scripting support.
* Create trailer for game.
* Publish / sell game.

## Conclusion

We set out to answer the following question: “How can realistic space physics and mechanics be balanced with intuitive gameplay to create an immersive and enjoyable experience for players in a space exploration and spacecraft piloting game?”

So far, we have created an MVP of a video game using the Godot game engine using the programming C#. This prototype contains all the primary and secondary features outlined in the IDD, however with some user testing / feedback, we found some bugs and have further recommendations on what to improve on for the MVP.

To answer the research question, our space simulation has realistic physics and is immersive and enjoyable as demonstrated by our user testing results. The game consists of space exploration and spacecraft piloting along with many additional features to ensure the game allows creativity from the players.

Over the 8 weeks of research and development we encountered multiple problems and solved them by doing research and testing. We have learnt how to use an open-source game engine we had never used before this project and created a working MVP. Based on the user testing feedback and our own journey of developing the MVP, the team agree the project has been an overall success.

From here we recommend finishing up the MVP prototype and adding upon it to create a complete video game which we can publish. If we had to pick an area of focus, we would primarily recommend focusing on the multiplayer aspect of the game as it is new technology for us to learn and implement and would be a great addition to the video game.

**References**

*Building from source*. (n.d.). Godot Engine Documentation. https://docs.godotengine.org/en/stable/contributing/development/compiling/

Engine, G. (n.d.). *Download for Windows - Godot Engine*. Godot Engine. https://godotengine.org/download/windows/

Godotengine. (n.d.-a). *Prevent loss of user-entered data in the Godot Editor · Issue #10694 · godotengine/godot-proposals*. GitHub. https://github.com/godotengine/godot-proposals/issues/10694

Godotengine. (n.d.-b). *Release 4.3-stable · godotengine/godot*. GitHub. https://github.com/godotengine/godot/releases/tag/4.3-stable

Utopia-Rise. (n.d.). *GitHub - utopia-rise/fmod-gdextension: FMOD Studio GDExtension bindings for the Godot game engine*. GitHub. https://github.com/utopia-rise/fmod-gdextension