CMPE 491

Performance Characterization of Open Source Game Engines

Bilal Atım

Advisor:

Atay Özgövde

TABLE OF CONTENTS

1. INTRODUCTION

1.1. Broad Impact

The gaming industry has a very large market. There are many game engines in this large industry. The most popular of these game engines are Unity and UnrealEngine. Game engines enable game developers to develop 2D and 3D games from start to finish. Thanks to their cross-platform nature, they allow game developers to develop games for different platforms at the same time.

The size of the sector also brings with it a very large economy. "The Games market is the second biggest market within Media. The worldwide revenue was $396.2 billion in 2022."[1] For this reason, the price policies offered by game engines direct game developers to research other free or open source game engines. One of the most popular among these game engines is Godot.

"Godot Engine is a feature-packed, cross-platform game engine to create 2D and 3D games from a unified interface. It provides a comprehensive set of common tools, so that users can focus on making games without having to reinvent the wheel. Games can be exported with one click to a number of platforms, including the major desktop platforms (Linux, macOS, Windows), mobile platforms (Android, iOS), as well as Web-based platforms and consoles.

Godot is completely free and open source under the permissive MIT license. [1] No strings attached, no royalties, nothing. Users' games are theirs, down to the last line of engine code. Godot's development is fully independent and community-driven, empowering users to help shape their engine to match their expectations."[2] .

[1]More details:`https://www.statista.com/study/134732/media-report-games/`
[2]More details: `https://docs.godotengine.org/en/stable/about/introduction.html`

The results of the performance review will provide game developers with more insight into open source game engines. By drawing attention to Godot's shortcomings and weak points, it can enable improvements to be made in those areas.

1.2. Ethical Considerations

Methodological Ethics:

- The materials and code examples to be used in demo projects must be completely open source and necessary references will be included.
- Performance comparisons of game engines will be evaluated with the same criteria under the same conditions.
- Performance metrics will be the metrics used to evaluate the performance of a game.
- Except for subjective issues such as the user experience of the game engine, the results in performance comparisons will be created from objective data.
- The intention is not to create a conflict of interest between Unity and Godot.

2. PROJECT DEFINITION AND PLANNING

2.1. Project Definition

To compare the Godot and Unity game engines in terms of their features and capabilities, to observe and report the changes in their performance by producing environments and scenes that will challenge the game engines' systems. Listing the pros and cons of Godot and Unity that ensure developer satisfaction and ease of use from a game developer's perspective. Determining the limits of game engines with edge cases. Reporting of these limits on different devices.

2.2. Project Planning

2.2.1. Project Time and Resource Estimation

Table 2.1. Weekly Works

	Work Definition	Time
Week 1	**Researching the Godot Game Engine**	**6 hours**
Week 1	**Preparing a presentation about Godot**	**2 hours**
Week 2	**Research Godot Game Architecture**	**1 hours**
Week 2	**Godot equivalents of Unity Game Loops**	**2 hours**
Week 2	**Learn Godot native language GDscript**	**2 hours**
Week 3	**Create basic Godot project**	**1 hours**
Week 3	**Create basic Godot 3D physics scene**	**1 hours**
Week 3	**Create basic car controller**	**2 hours**
Week 4	**Create Flappy Bird in Godot**	**8 hours**
Week 4	**Create Flappy Bird in Unity**	**5 hours**
Week 4	**Android build of Flappy Bird (Unity and Godot)**	**4 hours**
Week 4	**Size comparison for Unity and Godot**	**4 hours**
Week 5	**Create coroutine 2D demo scene for Godot**	**4 hours**
Week 5	**Calculate draw calls in Unity and Godot**	**4 hours**
Week 5	**Sprite atlas vs Atlas Texture performance differences**	**3 hours**
Week 6	**Create FPS screen for Unity and Godot**	**2 hours**
Week 6	**Create NavMesh scenes in Unity and Godot**	**7 hours**
Week 6	**Create animation in Godot**	**2 hours**

2.2.2. Success Criteria

- The extra features and missing features of Godot and Unity should be listed.
- Scenes that will push the limits of the Unity and Godot game engines should be created and compared in terms of features such as FPS, GPU, CPU and RAM usage in these scenes.

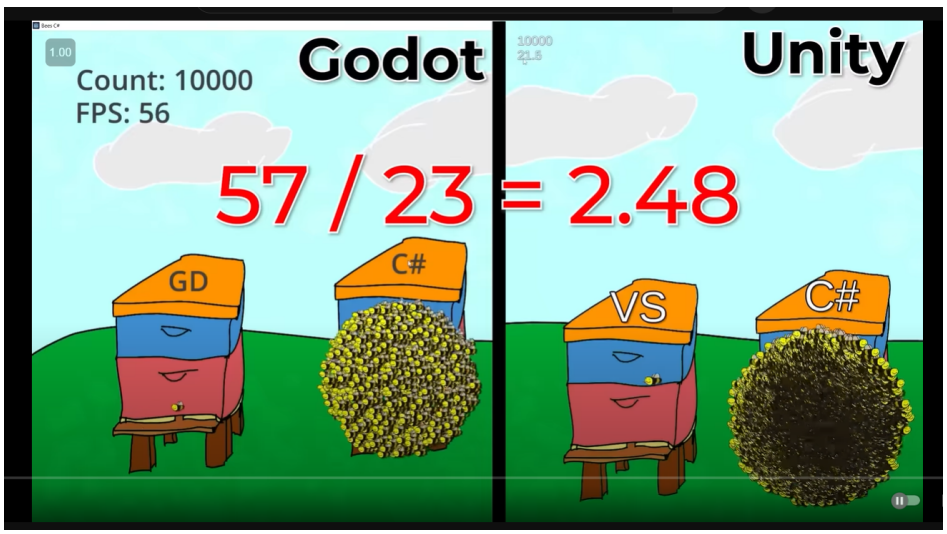
- Aspects that make it easy to use or make it difficult to use subjectively from the user's perspective should be reported for both game engines.
- The results obtained should be supported with graphics and visuals.

2.2.3. Risk Analysis

Some features of Unity do not have an exact equivalent in Godot, making it difficult to make comparisons on these issues. The fact that Godot is a newer game engine compared to Unity, which is open source and still under development, makes it difficult to obtain sufficient data on some issues.

3. RELATED WORK

Comparing 3D Performance of Godot and Unity Game Engine



The YouTube channel called Smart Penguins organized an experiment to compare the performance of Godot and Unity. It allows us to see the change in FPS by changing the number of objects. You can access the video from the link below.
`https://www.youtube.com/watch?v=dKNv5dQ5W4c`

4. METHODOLOGY

1) Choosing technical fields:

- Coroutine
- NavMesh
- Draw Call
- Addressables
- Physics System
- Particle System
- Animations System

2) Producing identical game scenes in Unity and Godot

3) Comparison of the performances of the created scenes

4) List results with numerical data and graphs and visual support

5) Report the ease of use of game engines and the conveniences and difficulties they provide to the user with a subjective evaluation.

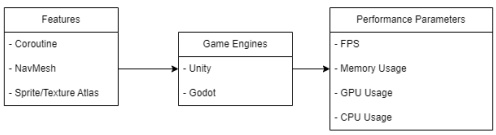
5. REQUIREMENTS SPECIFICATION

Establishing experimental scenes that will tire and strain the system from different aspects.

6. DESIGN

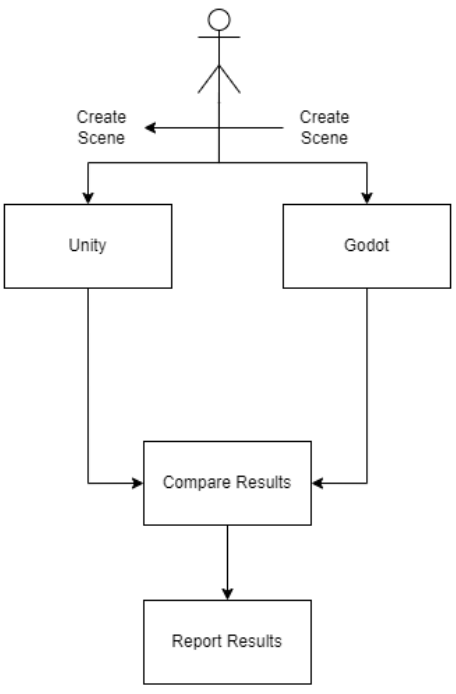
6.1. Information Structure

ER Diagrams.



6.2. Information Flow

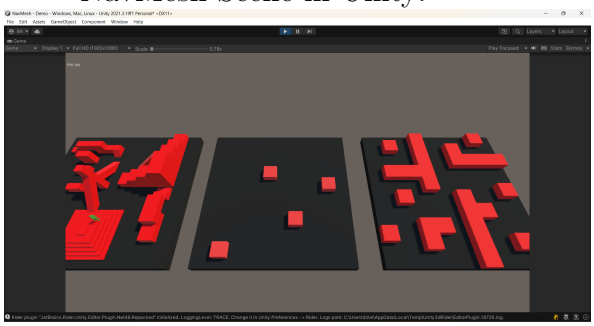
Activity diagrams, sequence diagrams, Business Process Modeling Notation. Create an experiment scene:



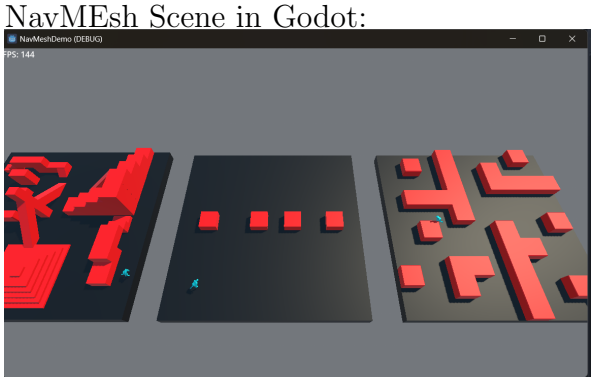
6.3. System Design

Class diagrams, module diagrams.

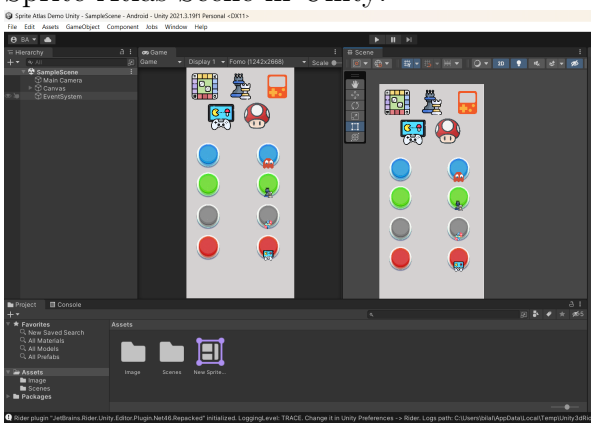
NavMesh Scene in Unity:



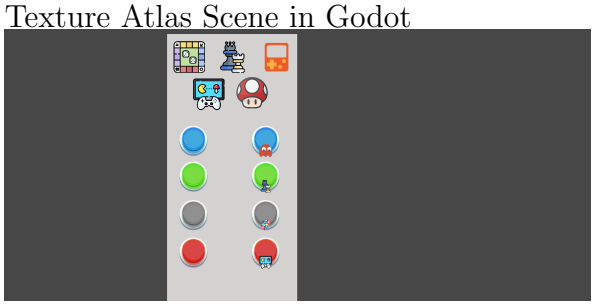
NavMEsh Scene in Godot:



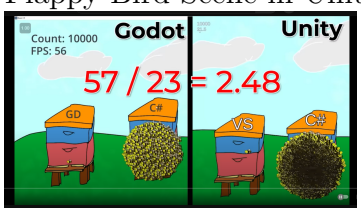
Sprite Atlas Scene in Unity:



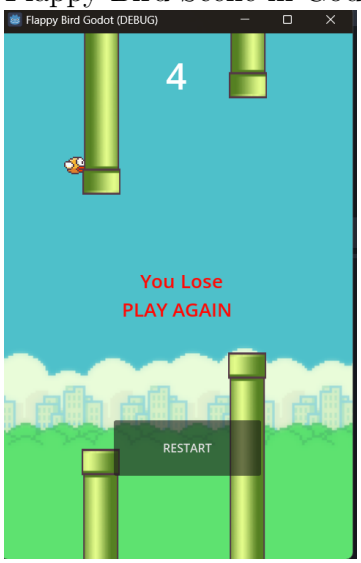
Texture Atlas Scene in Godot



Flappy Bird Scene in Unity



Flappy Bird Scene in Godot



7. IMPLEMENTATION AND TESTING

7.1. Implementation

Projects created in Unity were written in C# language. You can access the references of open source models and tools from the Asset Store and other external sources via the Github repository.

The projects used in Godot were written in GDscipt, Godot's native language. You can access the open source packages used to transfer Unity projects to the Godot project from the links below.

`https://github.com/V-Sekai/unidot_importer`

`https://github.com/prefrontalcortex/UnityGLTF`

`https://github.com/barcoderdev/unitypackage_godot`

All demo scenes created for Unity and Godot are in the Github repository below.

`https://github.com/bilalatim/CMPE491`

7.2. Testing

- Build Size Test: Flappy Bird was built and tested. An attempt was made to reduce the build size by making various optimizations in Unity and Godot. Godot optimization could not be performed due to an error encountered while optimizing Godot on the local computer.
- Draw Call Number Test: A scene consisting of 2D images was created in both Unity and Godot. The amount of decrease in the number of draw calls was measured using Sprite Atlas in Unity, but since the draw call counter section in Godot 2D was not yet working, the difference in draw call numbers before and after using Godot Texture Atlas could not be calculated.
- FPS Test: The same scenes were set up to measure the difference between Unity and Godotta Navigation Mesh systems. Average FPS amounts in Unity and Godot were measured.

7.3. Deployment

The created projects were run in locale. You can clone the repository by clicking on the project repository link above. You can test the projects locally by downloading 2021.3.19 or higher version for Unity and Godot 4.1.3 for Godot projects.

8. RESULTS

Table 8.1. Build Size Test Results

	Unity Android Build Size	**Godot Android Build Size**
Default	**21.4 MB**	**26.7 MB**
+ Editor Optimization	**20.4 MB**	**20.0 MB**
+ Build Optimization	**12.4 MB**	**ERROR**
+ Mobile Template	**10.4 MB**	**ERROR**

Godot optimization could not be performed due to an error encountered while optimizing Godot on the local computer.

Table 8.2. Draw Call Number Test Results

	Unity Draw Call Number	**Godot Draw Call Number**
Default	**25**	**-**
Sprite/Texture Atlas	**6**	**-**

The draw call counter section in Godot 2D was not yet working, the difference in draw call numbers before and after using Godot Texture Atlas could not be calculated.

Table 8.3. FPS Test Results

	Average FPS in Unity	**Average FPS in Godot**
Editor	**700**	**144**

Since Godot may have set a maximum FPS limit in the editor, the test will be repeated on different devices and made more extreme.

REFERENCES

1. Juan Linietsky, A. M., "Complying with licenses", , 2014, https://docs.godotengine.org/en/stable/about/complying_with_licenses.html.

APPENDIX A: SAMPLE APPENDIX

Contents of the appendix.