

CMPE 491

Performance Characterization of Open Source
Game Engines

Bilal Atım

Advisor:

Atay Özgövde

TABLE OF CONTENTS

1. INTRODUCTION	1
1.1. Broad Impact	1
1.2. Ethical Considerations	2
2. PROJECT DEFINITION AND PLANNING	3
2.1. Project Definition	3
2.2. Project Planning	4
2.2.1. Project Time and Resource Estimation	4
2.2.2. Success Criteria	5
2.2.3. Risk Analysis	5
3. RELATED WORK	6
3.1. Comparing 3D Performance of Godot and Unity Game Engine [1] . . .	6
3.2. Godot vs. Unity in 2D Physics [2]	7
3.2.1. Experiment	7
3.2.2. Results	7
3.3. Benchmark of Godot 4 in Minecraft Clone [3]	8
3.3.1. Experiment	8
3.3.2. Results	9
4. METHODOLOGY	10
5. REQUIREMENTS SPECIFICATION	11
5.1. User Requirements	11
5.2. System Requirements	11
6. DESIGN	12
6.1. Information Structure	12
6.2. Information Flow	12
6.3. System Design	13
7. IMPLEMENTATION AND TESTING	17
7.1. Implementation	17
7.2. Testing	17

7.3. Deployment	18
8. RESULTS	19
REFERENCES	22

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. [4] 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 .

¹More details:<https://www.statista.com/study/134732/media-report-games/>

²More 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
Researching the Godot Game Engine	6 hours
Preparing a presentation about Godot	2 hours
Research Godot Game Architecture	1 hours
Godot equivalents of Unity Game Loops	2 hours
Learn Godot native language GDscript	2 hours
Create basic Godot project	1 hours
Create basic Godot 3D physics scene	1 hours
Create basic car controller	2 hours
Create Flappy Bird in Godot	8 hours
Create Flappy Bird in Unity	5 hours
Android build of Flappy Bird (Unity and Godot)	4 hours
Size comparison for Unity and Godot	4 hours
Create coroutine 2D demo scene for Godot	4 hours
Calculate draw calls in Unity and Godot	4 hours
Sprite atlas vs Atlas Texture performance differences	3 hours
Create FPS screen for Unity and Godot	2 hours
Create NavMesh scenes in Unity and Godot	7 hours
Create animation in Godot	2 hours
Making agents walk on the roadside in the car game	2 hours
Create a maze environment for Navigation Experiment	2 hours
Testing NavMesh experiment with different parameters	2 hours
Creating performance charts	2 hours
Creating experiment scenes to test the physics engines	2 hours
Creating experimental scenes with rendering and object pooling	2 hours
Making a chase game with AI agent	2 hours
Report experiment results	2 hours
Researching related work	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

3.1. Comparing 3D Performance of Godot and Unity Game Engine [1]

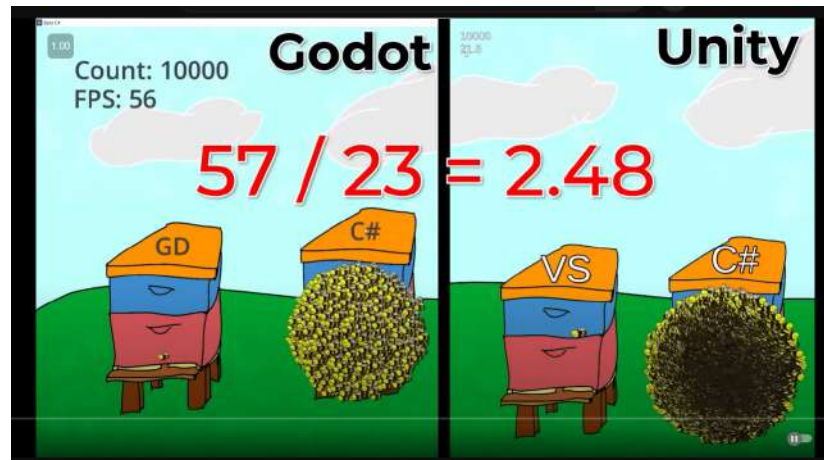


Figure 3.1. Object count test for Godot and Unity

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>

3.2. Godot vs. Unity in 2D Physics [2]

3.2.1. Experiment

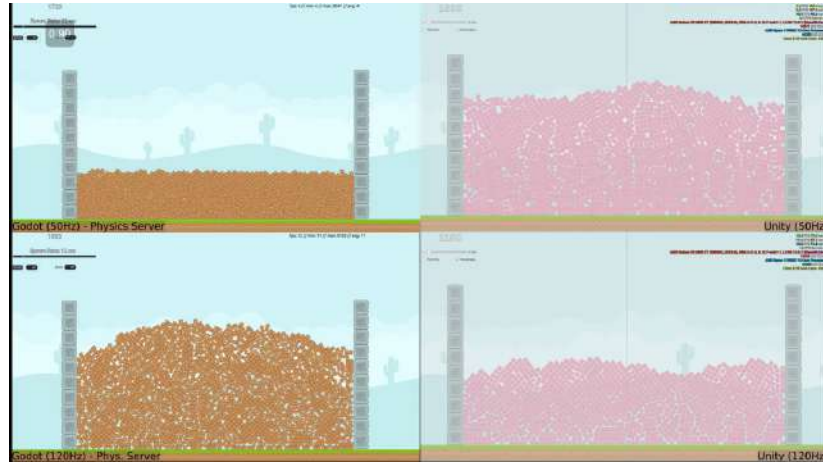


Figure 3.2. 2D Physics test for Godot and Unity

The YouTube channel called Mean Gene Hacks organized an experiment to compare the 2D physics performance of Godot and Unity. It allows us to see the change in FPS by changing the number of 2D physics objects. You can access the video from the link below. <https://www.youtube.com/watch?v=GPgkW0h4r1k&list=WL&index=3>

3.2.2. Results





		
2D Physics Engine	 GodotPhysics Engine	 Box 2d Physics Engine
50 Hz Performance	1729 objects	1895 objects
120 Hz Performance	1993 objects	1150 objects

Figure 3.3. 2D Physics results for Godot and Unity

3.3. Benchmark of Godot 4 in Minecraft Clone [3]

3.3.1. Experiment

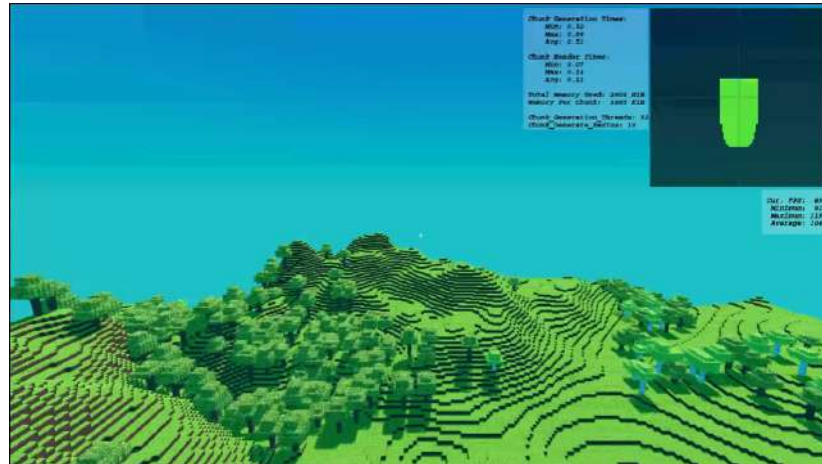


Figure 3.4. Benchmark Scene (Minecraft Clone)

The YouTube channel called SDG Games organized an experiment to compare Godot 4 and Godot 3.5 performances. He compares the performance of Godot 3.5 and Godot 4 by performing the following tests:

- Static chunk load test
- Dynamic chunk load test

Experiment Paramaters:

- Server
- Mesh
- GridMap
- MultiMesh

You can access the video from the link below. <https://www.youtube.com/watch?v=qb0GBc254QY&list=WL&index=6&t=918s>

3.3.2. Results

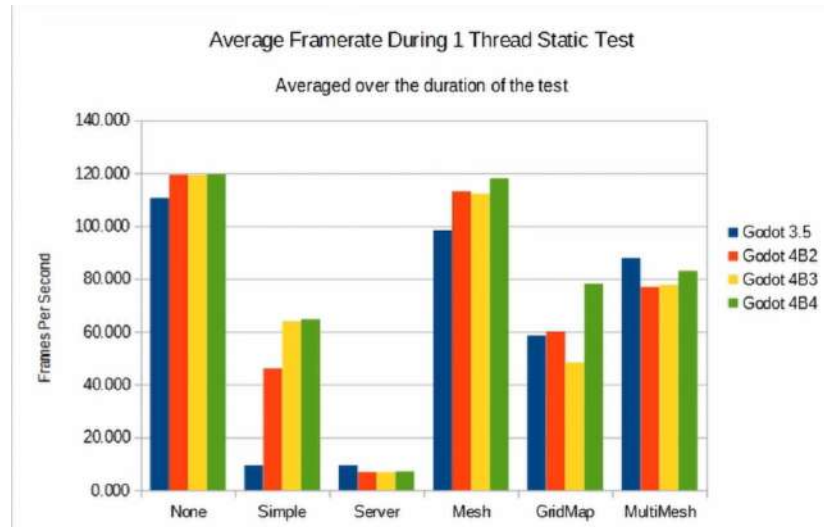


Figure 3.5. Average FPS During Static Test

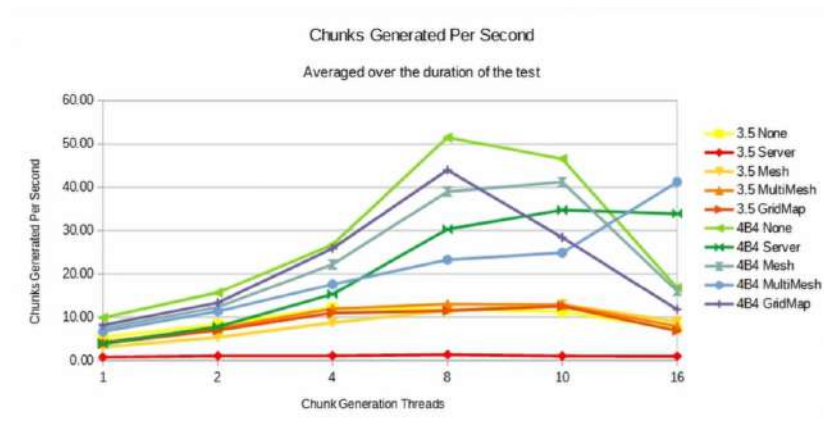


Figure 3.6. Chunks Per Second

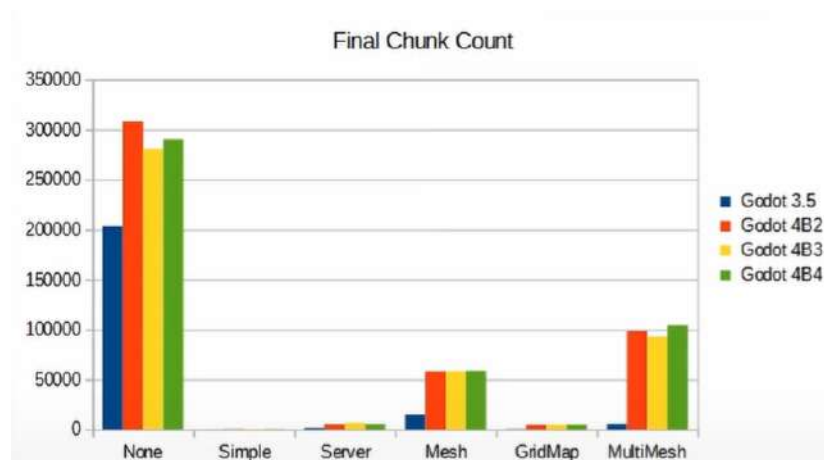


Figure 3.7. Final Chunk Counts

4. METHODOLOGY

1) Choosing technical fields:

- Coroutine
- NavMesh
- Draw Call
- Physics 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

5.1. User Requirements

- The user shall be able to run the exact same rendering scene in both game engines for rendering testing.
- The user shall be able to run the exact same physics scene in both game engines for physics testing.
- The user shall be able to see that the performance increases with object pool in the physics scene of both game engines.
- The user shall be able to see the effects of the number of interacting objects on the physics scenes in both game engines, by changing the number of box blocks.
- The user shall be able to run the exact same navigation path finding scenes in both game engines for AI navigation path finding testing.
- The user shall be able to see the effect of the number of agents on performance in navigation path finding scenes of game engines, by running scenes with different agent numbers.
- The user shall be able to see the FPS values on the screen in both game engines.

5.2. System Requirements

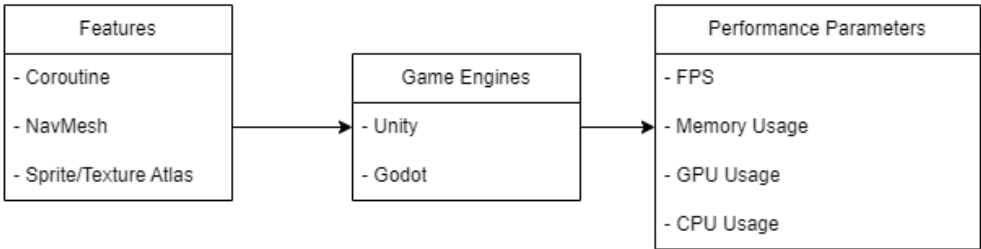
- The system shall meet the system requirements of Unity 2019.3.19 and Godot 4.2.1.

6. DESIGN

6.1. Information Structure

ER Diagrams:

□



□

Figure 6.1. Creating an experiment Flow

6.2. Information Flow

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

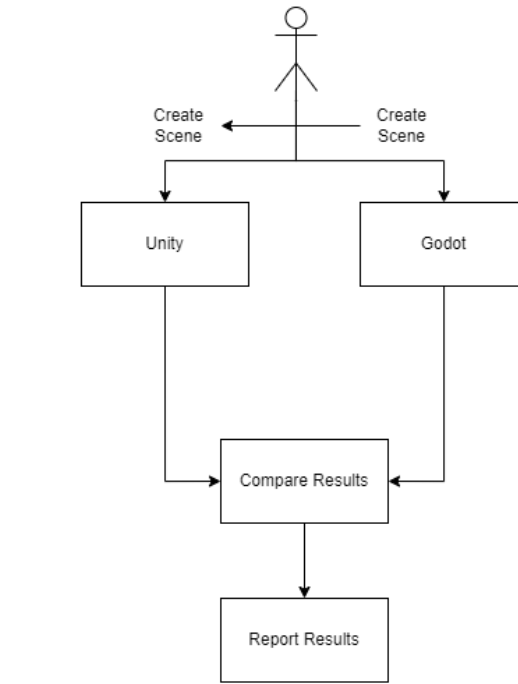


Figure 6.2. Creating an experiment Flow

6.3. System Design

Class diagrams, module diagrams.

NavMesh Scene in Unity and Godot:

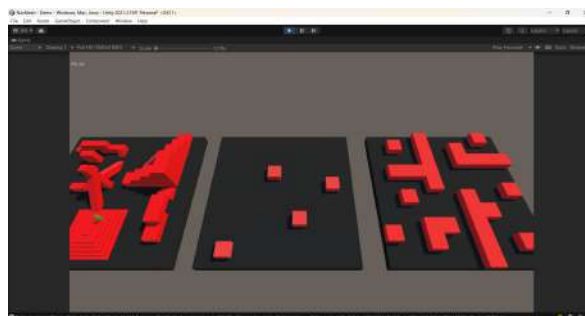


Figure 6.3. NavMesh Scene in Unity

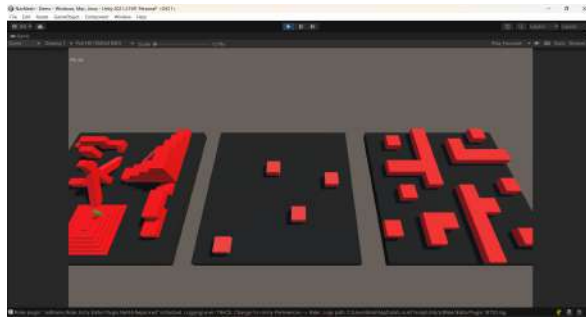


Figure 6.4. NavMesh Scene in Godot

Sprite Atlas Scene in Unity and Godot:

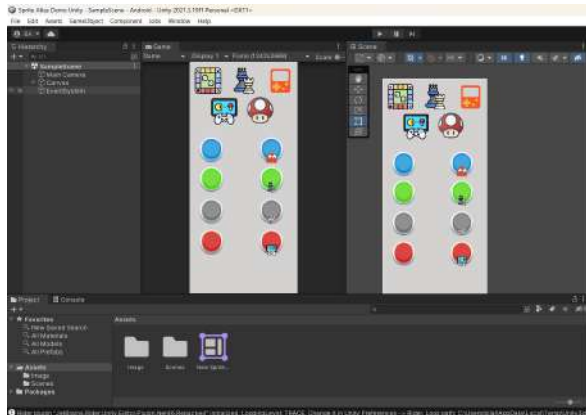


Figure 6.5. Sprite Atlas Scene in Unity

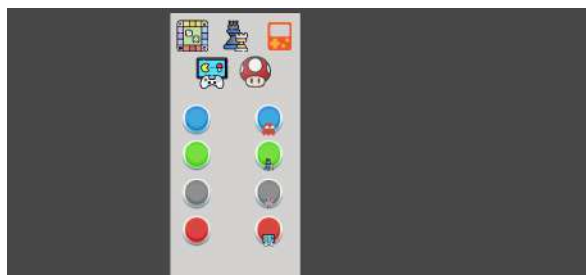


Figure 6.6. Sprite Atlas Scene in Godot

Flappy Bird Scene in Unity and Godot

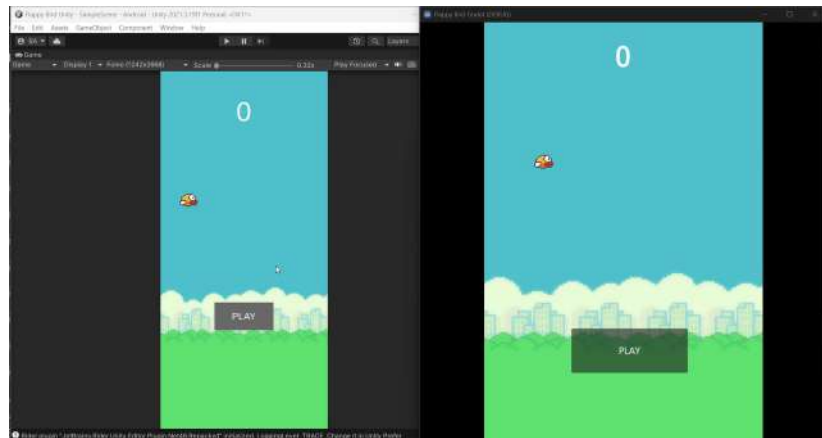


Figure 6.7. Flappy Bird Scene in Unity

NavMesh Scene in Unity and Godot at a Maze

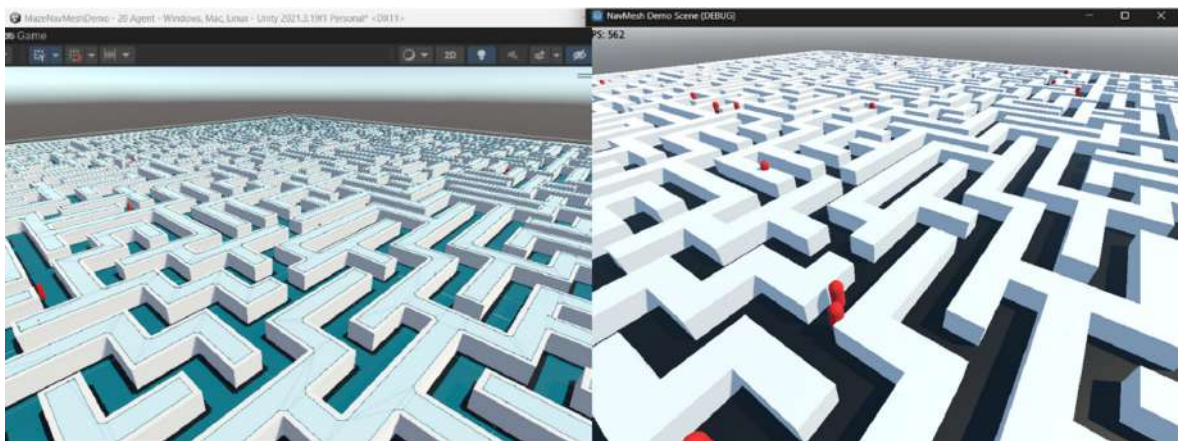


Figure 6.8. NavMesh Scene in Unity and Godot at a Maze

Physics Scene in Unity and Godot at a Maze

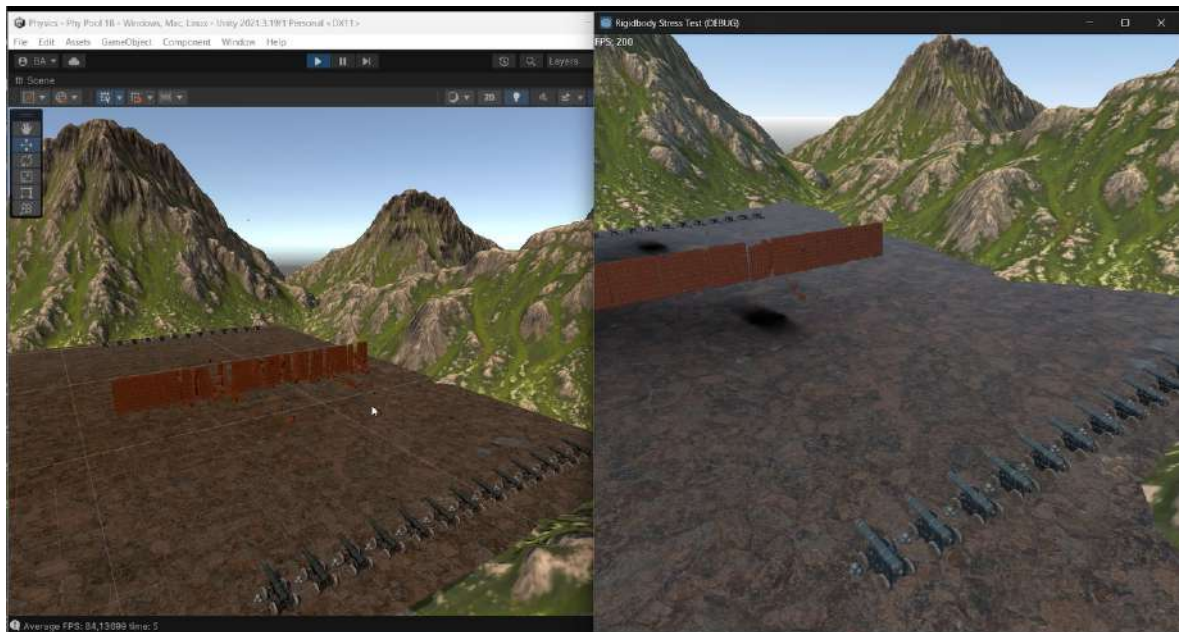


Figure 6.9. Physics Scene in Unity and Godot at a Maze

Car game in Godot



Figure 6.10. Car game 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 GDscript, 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

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.
- Rendering Performance Test: To measure the rendering performance of the two game engines, the same scene was rendered in both game engines. FPS values were measured depending on the number of objects. In these scenes, code-based

calculations and the physics engines are not used. The effect of increasing the number of objects on the FPS was observed by varying the object count rendered to the camera.

- **Physics Performance Test:** To measure the physical computation performances of the two game engines, the same scene was rendered in both game engines. Average FPS values were measured depending on the number of objects colliding with each other. Rendering also affected performance in initial measurements. For this reason, the experiments were repeated with the number of objects drawn in the frame being 0 to avoid any effect of rendering. By increasing the number of objects, the effect of the increase in the number of objects drawn on the camera on FPS was observed.
- **AI Path Finding Performance Test:** To measure the path finding performances of the two game engines, the same scene was created in both game engines. The effect of the change in the number of agents on average FPS was observed. Average FPS values of a number of agents that would push the limits of the systems were measured under the path finding algorithm. Later, the maze scene was created in both game engines to make path finding more difficult. Again, the change in average FPS depending on the number of different agents was reported. All demo scenes created for Unity and Godot are in the Github repository below.

<https://github.com/bilalatim/CMPE491>

My Project Video Link:

<https://www.youtube.com/watch?v=j66ojU6q570&t=11s>

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

We see that the build sizes of the those in which only unnecessary packages are deleted are very close to each other.

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.

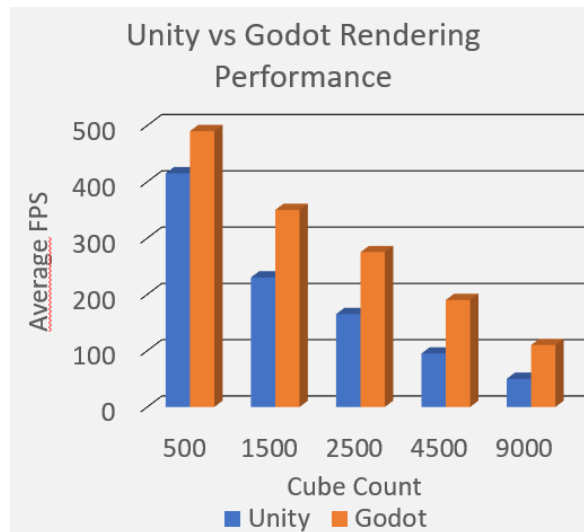


Figure 8.1. Average FPS results for rendering in Unity and Godot

Above, we see the FPS values in a scene where physics is not used in both game engines. When we examine the average FPS values, we see that Godot's FPS value is higher for all object numbers. However, the FPS difference here can be balanced or changed with various optimizations in both game engines.

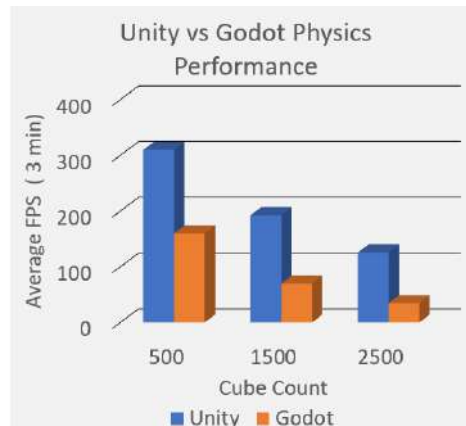


Figure 8.2. Average FPS results for physics in Unity and Godot

Above, we see the average FPS values for both game engines in an experimental scene where the 3D Physics engine is used extensively. Godot gave lower average FPS results than Unity in the experimental scene where physical calculations were intensively calculated. We see an exponential FPS decrease depending on the increasing number of objects in Godot. We see that Unity's physics engine performs better in the experiment scene where the physics engine is used extensively.

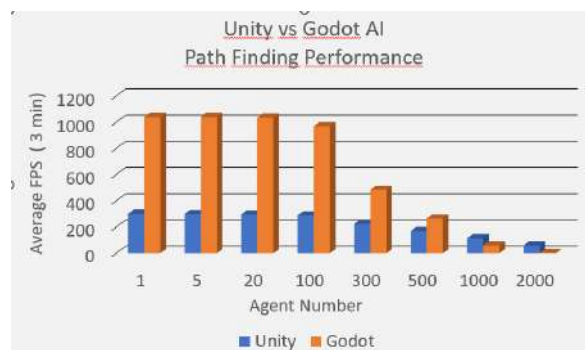


Figure 8.3. Results from game development process in Unity and Godot

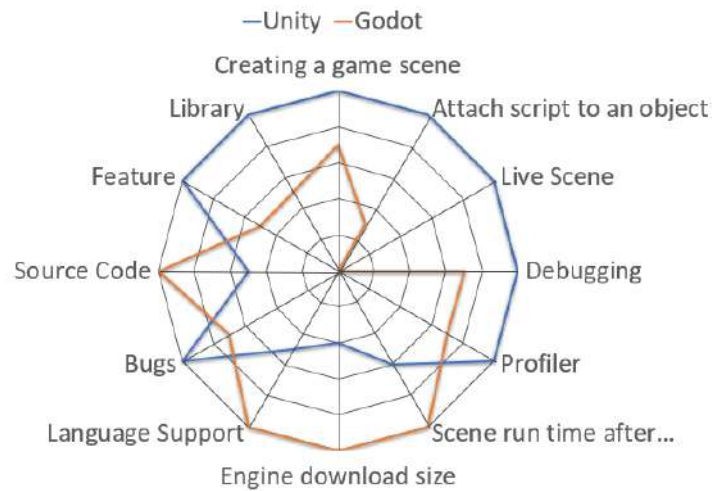


Figure 8.4. User experience results in Unity and Godot

- Creating a game scene: Creating a game scene in Unity is much easier and more practical than Godot.
- Attach script to an object: In Godot, only 1 script can be attached to each object. There is no such limitation in Unity.
- Live Scene: There is no live scene in godot. Live scene is very important feature for solving bugs and having control over game.
- Debugging: Debugging in Godot is more harder than unity
- Profiler: Unity profiler is more comprehensive and detailed than Godot profiler.
- Scene run time after script editing: Testing the scene after editing the script is incredibly fast on goddot compared to unity.
- Engine download size: Godot game engine download size is about 56 MB but Unity's download size is about 5GB.
- Language Support: Godot supports C#, C++ and GDScript languages. However, Unity only supports C.
- Bugs: Although every game engine has bugs and errors, Godot has much more problems than Unity.
- Source Code: Godot is open source and community driven but unity is not.
- Feature: Unity has much more features than Godot.
- Library: Unity has a very rich code library compared to Godot.

REFERENCES

1. Penguins, S., “Comparing 3D Performance of Godot and Unity Game Engine”, <https://www.youtube.com/watch?v=dKNv5dQ5W4c>.
2. Hacks, M. G., “Godot vs. Unity in 2D: Who will win?”, <https://www.youtube.com/watch?v=GPgkW0h4r1k&list=WL>.
3. Games, S., “How much faster is Godot 4? (Feat. My Minecraft Clone)”, <https://www.youtube.com/watch?v=qb0GBc254QY&list=WL>.
4. Juan Linietsky, A. M., “Complying with licenses”, https://docs.godotengine.org/en/stable/about/complying_with_licenses.html.