SWE 599 Final Project - Fall 2023-2024

Deniz Baran ASLAN - 2021719183

Topic: Game Development

# Project Report

| Task | Deadline |
|---|---|
| ~~Submit plan~~ | ~~October 22, 2023, 23:59~~ |
| ~~Search for relevant publications and documentation, design prototype~~ | ~~November 5, 2023~~ |
| ~~Build and deploy prototype~~ | ~~November 16, 2023~~ |
| ~~Submit progress report~~ | ~~November 26, 2023, 23:59~~ |
| ~~Buil pre-release version, test and analyze~~ | ~~December 7, 2023~~ |
| ~~Finish writing documentation, write report, add bibliography, proofread~~ | ~~December 20, 2023~~ |
| ~~Submit final report, make presentation and demo~~ | ~~January 3, 2024, 23:59~~ |

## Table of Contents

## 1. Introduction

In the evolving field of game development, the landscape is marked by the dominance of established engines such as Unreal, and the emergence of up-and-coming contenders. Godot and Unity, two prominent names in the sphere of game engines, are good examples of this split. While Unity has held strong as a good choice for developers, Godot has steadily been gaining traction as a compelling alternative.

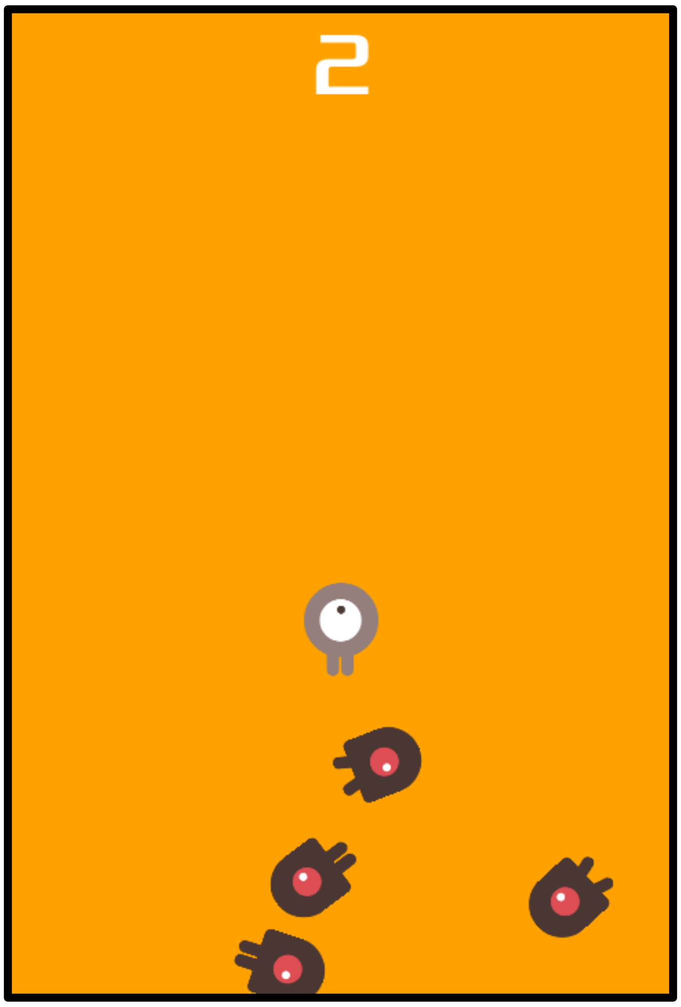
Unity's legacy as a versatile and powerful engine has earned it widespread popularity, establishing itself as a cornerstone in the game development industry. Its comprehensive feature set, robust tools, and expansive ecosystem positioned it as a go-to solution for developers across diverse projects and skill levels. However, recent controversies surrounding Unity sparked considerable upheaval within the development community. Unity's controversial decision to introduce developer charges per download stirred an uproar among studios and developers, igniting debates and concerns over the potential financial impact on smaller studios and indie developers. A subsequent retraction of this decision attempted to assuage the community, yet the damage was done, prompting some developers and studios to seek alternatives.

In this shifting landscape, Godot emerges as a notable contender, with its open-source nature, user-friendly interface, and growing community support. The controversy surrounding Unity's pricing strategy has spurred a surge in interest toward Godot as a viable replacement. Its rise as a popular choice stems not only from its accessibility and robust feature set but also from the community's advocacy for a more transparent game development environment. This project aims to delve into game development with the Godot Engine, explore its capabilities, and contrast its state with that of Unity Engine.

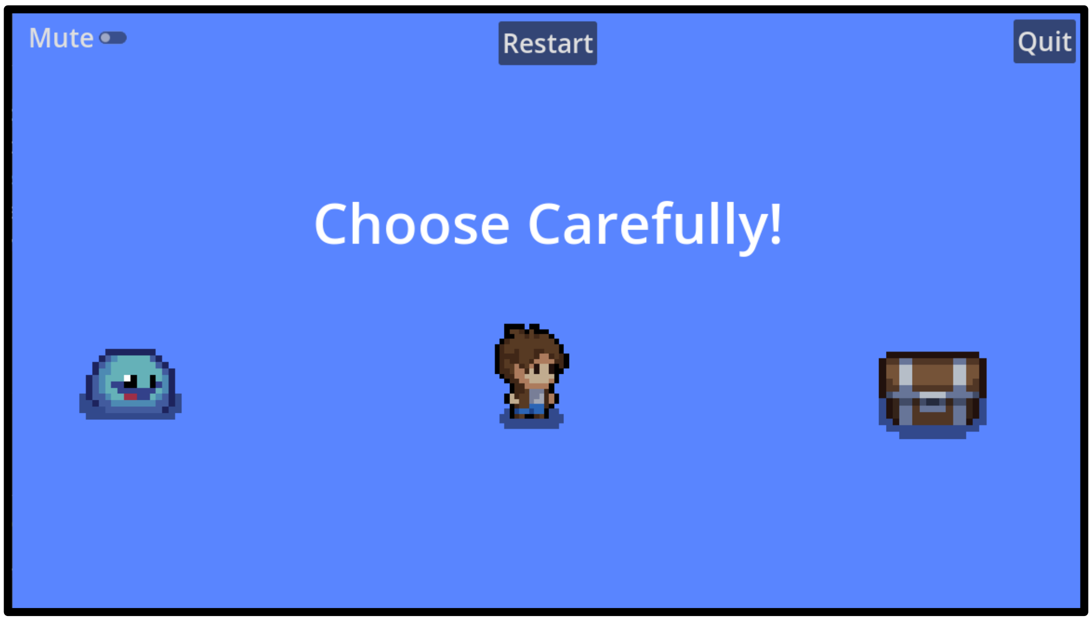
## 2. Methodology

The methodology adopted for this comparative analysis is comprised of a deliberate selection of features, using  open-source assets, utilizing performance metrics, and keeping simplicity in mind during development. Instead of developing full-fledged minigames, the focus was on creating small, targeted scenes or specific features within each engine. This approach allowed for granular testing and analysis of various functionalities, including physics engine capabilities, asset management, scripting and monitoring.

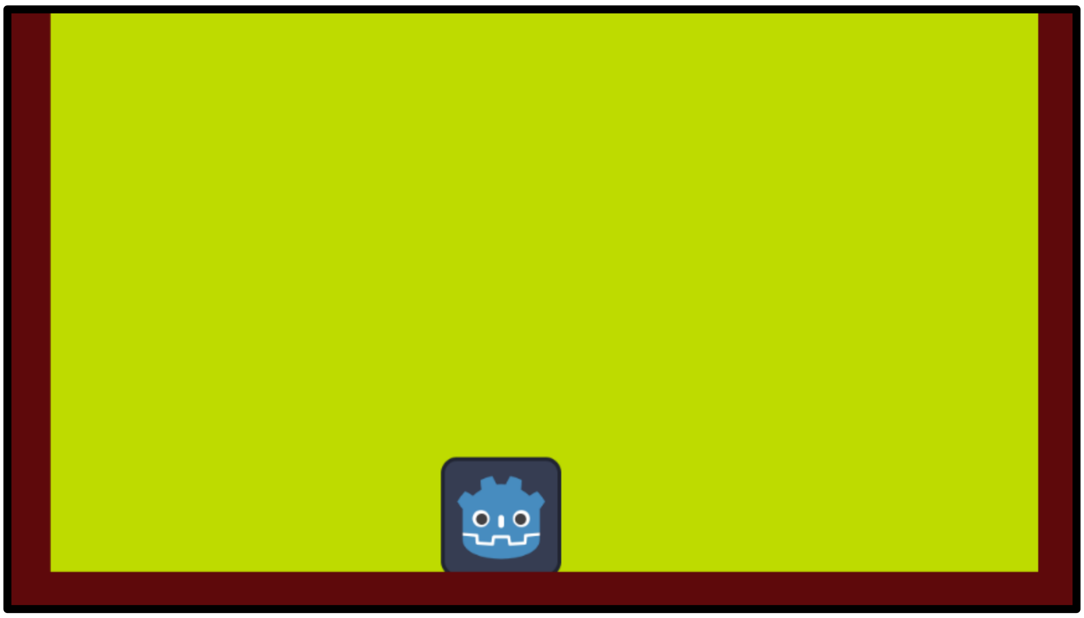
To maintain a standardized approach, free open-source assets were utilized wherever applicable. These assets were sourced from community-driven repositories. The emphasis was on functionality and performance rather than completeness. Scenes were intentionally kept small, encapsulating specific aspects of game development to facilitate focused experimentation and analysis.



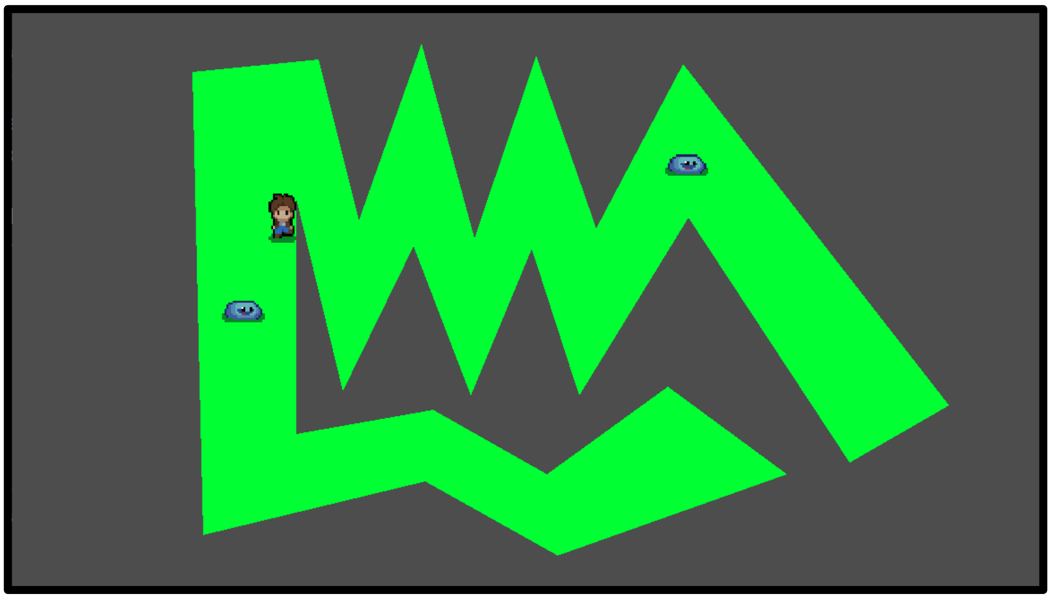
The first project is the default 2D tutorial prepared by the Godot team. This tutorial is meant to teach newcomers about the basic workflow of the engine, while introducing them to the nodes and signals system. Nodes are like building blocks that represent different parts of a game, and they can connect to make things work. Signals are like messages that these blocks send to each other. They help these blocks interact without getting too mixed up. This makes it easier to organize and control different parts of the game, making the game-making process smoother and less confusing. Compared to first-time development in Unity, Godot's system is much more straightforward, and allows the developer to dive straight in to implementing gameplay mechanics.



The second project is an attempt to make an actual minigame, testing features such as animation management, buttons, audio management and movement. A single script controls the flow of the game, the win and lose states, and the gameplay loop. Sound and image assets can easily be imported into the project and used in the relevant scene. The resource management is much like Unity's, with a straightforward file manager window.

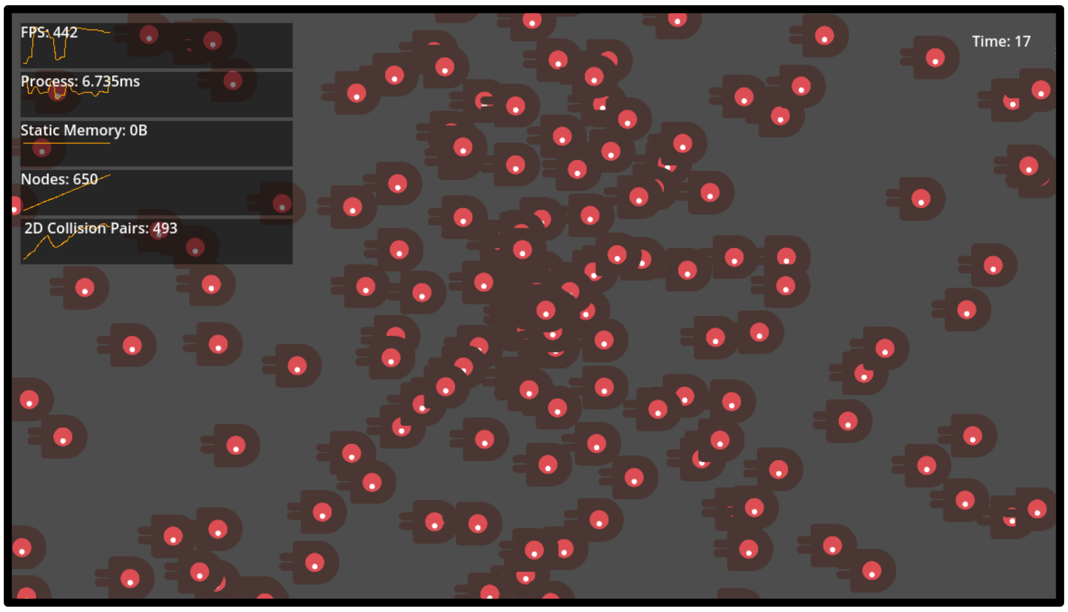


The third project is an attempt to test out gravity, movement and collision. These features were easy to implement, thanks to the built-in functions. All the developer needs to do is to call the right functions in a script, attached to the relevant node. The details an be worked out through the inspector, rather than relying on code for everything.



The next project is a simple scene with a player and enemy nodes, placed in a small labyrinth. The player character follows the mouse pointer, figuring out the optimal path to reach the pointer. Pathfinding is not too difficult to implement, the developer simply needs to determine the target and the player's movement pattern in a script, and the engine takes care of the rest. The player character moves towards the mouse while colliding with the walls and being obstructed by the enemies.

The final project is a tool to stress-test the engine while getting performance metrics directly from the system, using a monitoring asset. Colliding mobs keep spawning from the center, and they keep changing direction as they collide with the walls and with one another. As time goes on, the engine has to account for an overwhelming amount of collision pairs, leading to lowered performance and greater RAM usage. The FPS value starts above 400 and keeps decreasing over time, eventually reaching single digits. The primary goal was to simulate and assess the engines' resource utilization, looking at Frames Per Second (FPS), Central Processing Unit (CPU) usage, Random Access Memory (RAM) usage, and load times.



Simplicity remained a guiding principle throughout the development process. Instead of creating comprehensive minigames, the focus was on discrete features or small scenes that showcased specific functionalities. This approach aimed to streamline the testing process, enabling a more thorough examination of individual features and their impact on system resources and performance. The methodology prioritized experimentation, adding and removing various features until arriving at the desired outcome.

## 3. Discussion

The development process in both Godot and Unity was tailored specifically for 2D game creation in this comparative analysis, although both engines possess robust capabilities for both 2D and 3D game development. The assessment delved into the intricacies of 2D design and implementation to discern the engines' performance within this domain.

Within Godot, its unique node-based scene system served as the backbone for organizing game elements. Nodes, representing specific functionalities, interconnect to form a modular approach to development. Signals facilitate seamless communication between nodes, enabling them to collaborate without creating tight dependencies. This structure streamlines development, encourages modularity, and assists in efficient game logic implementation.

The comparison between Godot and Unity unveiled several disparities in performance and features. Godot's resource-efficiency is evident, making it suitable for projects mindful of resource consumption. In contrast, Unity demonstrated superior performance metrics, albeit with higher resource demands. Godot's node and signal system contributed significantly to an organized and modular development process, showcasing its advantage over Unity's approach. Godot is very lightweight, it can be downloaded, launched immediately with no installation, and a new project can be created in little time. Projects can also be exported, closed, and reopened with very short wait times. On the other hand, Unity has to be downloaded and installed slowly through a dedicated hub, projects take long to initialize and export, and the editor itself has to load numerous libraries and resources before launching fully.

However, Unity is still ahead of Godot in a few aspects. Unity has a better set of built-in tools for 3D development, whereas Godot needs to rely on a few external tools to achieve the same results. This might change in the future as Godot is constantly improving its 3D capabilities. Another difference is the size of available assets and online documentation, along with tutorials. Unity has a much more diverse range of dedicated assets compared to Godot, but the difference can be made up through the use of open-source, platform-independent assets. Owing to its popularity, Unity also has a greater number of dedicated tutorials, guides, and online knowledge bases. Once again, these differences may eventually be addresses thanks to Godot’s growing popularity.

These observations emphasize Godot's allure due to its resource efficiency and structured workflow. Yet, Unity's performance metrics hint at its potential for graphically intensive projects.

## 4. Conclusion

In summary, Godot and Unity offer distinct advantages. Godot's resource efficiency and organized workflow make it suitable for resource-sensitive projects, while Unity's superior performance metrics cater to graphically demanding endeavors. Developers must weigh trade-offs between resource efficiency and performance capabilities when selecting engines aligned with project requirements.

The present study is limited in scope, only exploring general workflow and a few specific functionalities. Future work might involve real-time performance measurements to validate findings and the testing of further functionalities to refine comparisons between Godot and Unity. With the recent controversies on Unity’s side and the rising popularity of Godot, the future of the two engines remains to be seen.

My work is available online in the following GitHub repository:
https://github.com/dbaslan/SWE599