Introduction to Engineering, FENS

**Project: Blackjack in Godot (GDScript)**

INTERNATIONAL UNIVERSITY OF SARAJEVO

horizontal line

# 

# 

# 

Name: Amra Hozić

Student ID: 230302151

# Introduction

This report endeavors to examine the development processes associated with a specific project undertaken within the Godot game engine. Functioning as an open-source platform with a robust feature set, Godot is acknowledged for its role as a facilitator in the creative domain of game development.

The primary objective of this report is to conduct a retrospective analysis of the developmental trajectory of our project within the Godot framework. This exploration encompasses the engine's foundational strengths, distinctive features and its workflow, offering insight into the engine's impact on the creative process.

The narrative unfolds by elucidating key insights, challenges, and achievements encountered during the project's development, offering a nuanced perspective on the interplay between creative expression and the functionalities afforded by Godot.

# Godot

### 1.1 Generally about Godot

Godot, the acclaimed open-source game engine, emerged from the creative minds of Juan Linietsky and Ariel Manzur. This dynamic duo conceived Godot in 2007, and over the years, the project has evolved into a robust and community-driven platform for game development. One of Godot's defining features is its inclusive accessibility, attracting a diverse user base ranging from novice indie developers to seasoned professionals. Its versatility makes it an ideal choice for small indie studios, educational institutions, and even larger game development enterprises. The engine's appeal lies in its user-friendly interface, extensive documentation, and a welcoming community that fosters collaborative learning and knowledge sharing.

Godot boasts several standout features. Its node-based scene system facilitates modular game design, enabling developers to create and organize game elements intuitively. The engine supports both 2D and 3D game development, making it adaptable to a broad spectrum of projects. Godot's built-in GDScript, a Python-like scripting language, simplifies the coding process and accelerates development without compromising performance.

The engine's commitment to open-source principles ensures that developers can use Godot without worrying about licensing fees or royalties. This factor, coupled with its continuous development and responsive community, positions Godot as a dynamic force in the game development landscape.



### 1.2 GDScript

GDScript, the dedicated scripting language for the Godot game engine, simplifies the development landscape with its Python-like syntax. Ideal for both beginners and seasoned developers, GDScript promotes rapid prototyping and efficient coding. Its dynamic typing and automatic memory management contribute to a streamlined experience, allowing developers to focus on creativity rather than technical intricacies.

Designed to seamlessly integrate with Godot's node-based scene system, GDScript empowers developers to create modular and interactive game elements effortlessly. Its concise syntax reduces boilerplate code, facilitating the rapid translation of creative visions into functional game logic. In essence, GDScript is the creative medium that enables developers to bring their gaming ideas to life efficiently within the Godot ecosystem.

### 1.3 Node system

Godot's node system is a powerful framework that organizes game elements hierarchically, allowing developers to create intricate and dynamic game worlds. Nodes, representing scenes, graphics, and scripts, serve as modular entities for efficient and reusable design.

**Hierarchical Efficiency:**

Nodes establish a hierarchical structure where each node, whether representing a scene, object, or script, can be managed and manipulated with structured precision.

**Modular Reusability:**

Encapsulating functionality, nodes promote a design philosophy focused on reusability. Developers can create and reuse nodes across different game parts, contributing to an efficient workflow and an organized codebase.

**Versatile Entities:**

Nodes in Godot's system cover a spectrum of entities, from interactive characters to dynamic scripts, acting as components that shape the game's functionality.

**Extensibility for Innovation:**

The extensibility of the node system empowers developers to create custom nodes, fostering innovation and allowing for the implementation of unique functionalities within games.

**Seamless Script Integration:**

Associating nodes with scripts enables developers to define the behavior and logic of game elements, creating a seamless connection between visual representation and functionality.

In summary, Godot's node system is a robust foundation for game development, providing a dynamic, modular, and extensible framework. Its versatility, coupled with script integration, makes it an essential component for creating interactive and engaging game experiences.

### 1.4 Scene system

Godot's scene system is a vital component for structuring and organizing game elements efficiently. It empowers developers to create complex and interactive environments by composing scenes with multiple nodes.

**Efficient Organization:**

The scene system provides a structured approach to organize game elements, offering developers a way to manage and arrange nodes in a coherent manner. This efficiency is crucial for maintaining clarity in the development process.

**Node Composition:**

Scenes can consist of multiple nodes, allowing developers to build diverse and intricate game worlds. Each node within a scene contributes to the overall functionality, making it possible to create dynamic and engaging gameplay experiences.

**Dynamic Environments:**

The scene system facilitates the creation of dynamic environments by enabling developers to combine various nodes within a scene. This flexibility is particularly valuable for crafting game worlds that respond dynamically to player interactions.

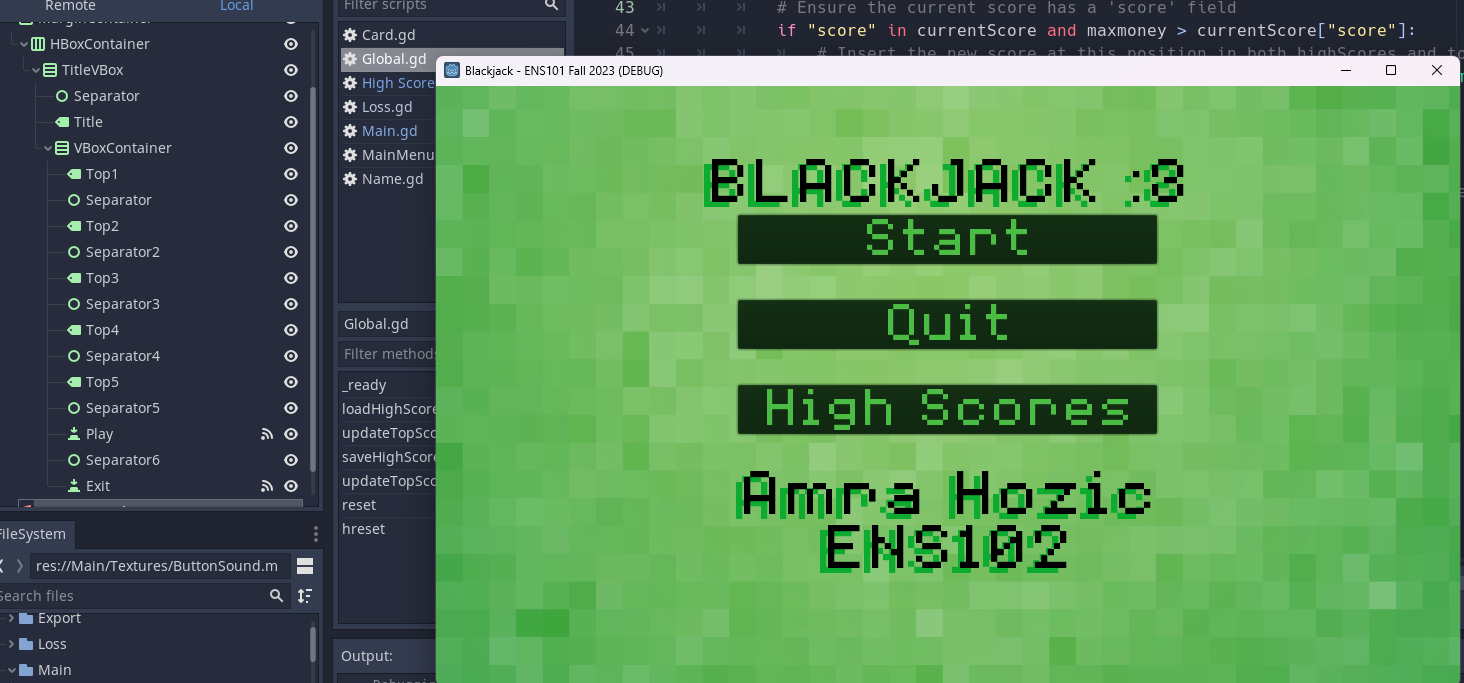
**Interactive Elements:**

Nodes within scenes can represent a wide range of interactive elements, from characters and objects to interactive scripts. This versatility allows developers to implement diverse gameplay mechanics and create immersive player experiences.

**Effortless Collaboration:**

The scene system streamlines collaboration among developers by providing a clear and organized structure. Developers can work on different aspects of a game within specific scenes, enhancing productivity and coordination.

We can use the Main Menu scene from the project as a great example of these traits.



In the depicted Main Menu scene, various elements are discernible, contributing to the overall user interface. Notably, the presence of a signal icon adjacent to a specific button element suggests that it is intricately connected to a script. This signal-script linkage is a visual indicator of the button's functionality, implying that when interacted with, the button triggers predefined actions or behaviors orchestrated by the associated script. Such visual cues play a crucial role in conveying the interactive nature of elements within the scene, providing users with insights into the dynamic functionalities offered by the Main Menu interface.

In summary, Godot's scene system is a powerful tool for developers, offering efficient organization, dynamic environment creation, and seamless collaboration. Its flexibility in node composition empowers developers to bring their creative visions to life in the form of engaging and interactive game worlds.

### 1.5 Other features

There are a few more important Godot features that should be briefly mentioned, but we will not dive deep into them as they’re not very relevant to this specific project or they are straightforward.

**Visual Scripting:**

- Godot supports visual scripting, allowing developers to create game logic through a node-based graphical interface.

- Ideal for those who prefer a visual approach to coding.

**2D and 3D Engine:**

- Godot supports both 2D and 3D game development, providing a versatile environment for creating diverse projects.

**Animation System:**

- Godot's animation system enables the creation of complex character animations and scene transitions.

- Supports both skeletal and sprite-based animations.

**Physics Engine:**

- The built-in physics engine facilitates realistic movement and collision detection.

- Supports rigid bodies, areas, and joints for comprehensive physics simulations.

**Multiplatform Export:**

- Godot allows developers to export games to various platforms, including Windows, macOS, Linux, Android, iOS, and more.

- Simplifies the process of reaching a wide audience.

**Built-in Debugging Tools:** - Godot includes a range of debugging tools to aid developers in identifying and resolving issues during development.

- Offers real-time debugging features for efficient code inspection.

**Asset Import Pipeline:**

- Godot's asset import pipeline supports a variety of file formats, simplifying the integration of graphics, audio, and other assets.

**Community and Documentation:**

- The Godot community is an integral component, fostering collaboration, knowledge sharing, and support.

- Extensive documentation provides resources for developers of all skill levels.

These key components collectively contribute to Godot's reputation as a versatile, user-friendly, and community-driven game development engine.

# Project presentation

In my presentation about my Godot project, I'll break down the key elements that make it tick. We'll explore the most important parts of the project.

### 2.1 Introduction

To start things off, we've neatly organized our game files into four main folders: Main Menu, Loss Screen, Main Game, Name insert screen and High Scores screen. This systematic arrangement simplifies project management, allowing for easy access and organization of resources specific to each game phase. Each folder holds the resources needed only for that part of the game process.

### 2.2 Card Script

Next, we will talk about the Card script. This Godot GDScript defines a `Card` node, representing a playing card in a game. Here's a breakdown of the code:

**1. Variable Declarations:**

**- `cNumb`:** String variable to store the rank of the card (e.g., "A", "2", "K").

**- `cSign`:** String variable to store the suit of the card (e.g., "Hearts", "Diamonds").

**- `value`:** Integer variable to store the numeric value of the card.

**- `facedown`:** Boolean variable indicating whether the card is face-down.

**Functions:**

**- `get\_value()`:** Returns the numeric value of the card.

**- `getinfo()`:** Returns the concatenation of the suit and rank, forming a string representation of the card.

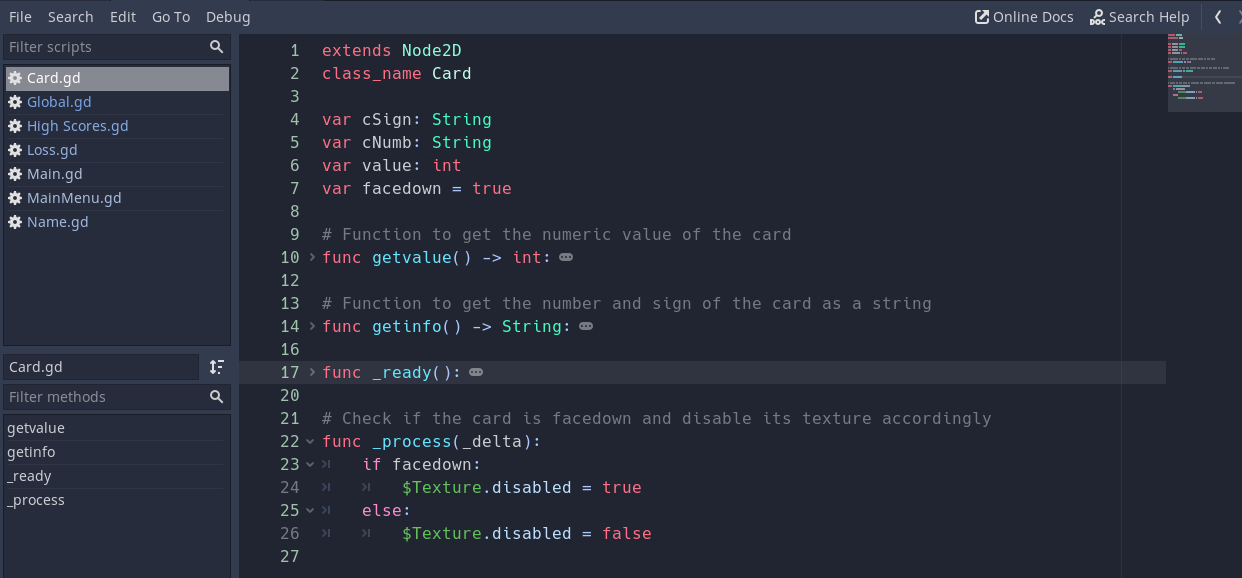
**\_ready() Function:**

- Loads the appropriate card texture based on the provided `suit` and `rank`. The texture file is assumed to be in the "Main/Textures" directory.

**\_process() Function:**

-Checks the `facedown` variable and adjusts the visibility of the card's texture accordingly. If the card is facedown, the texture is disabled; otherwise, it is enabled.

In summary, this script encapsulates the behavior and appearance of a playing card, allowing for easy integration into a larger card game project in Godot. The `\_ready()` function initializes the card's texture, and the `\_process()` function handles the visual state based on whether the card is face-up or face-down.



### 2.3 Global Script

Furthermore, we can talk about the Global.gd script. This code appears to be a part of a game script or system in the Godot game engine, written in GDScript. It defines a class that extends the `Node` class, indicating that it can be used as a node in the Godot scene tree. This script seems to serve as a singleton, meaning there is only one instance of it in the entire game. Singletons are commonly used to manage global game state or provide services that need to be accessible from various parts of the game. Variables and functions from Global.gd can be accessed from other scripts simply by using the Global.[NAME] prefix.

Here's a breakdown of the code:

**1. Variable Initialization:**

- `PlayerName`, `money`, `maxmoney`, `Cheat`, `count`, and `Dealer\_count` are initialized to default values.

- `topScoreString1` to `topScoreString5` are initialized as empty strings.

- `highScores` and `top5ScoresValues` are initialized as empty arrays.

**2. \_ready() Function:**

- This function is called automatically when the script instance is ready.

- It calls `loadHighScores()` to load high scores from a file.

**3. loadHighScores() Function:**

- Opens and reads a file (`high\_scores.txt`) containing high scores in JSON format.

- Parses each line of the file as JSON and appends it to the `highScores` array.

**4. updateTopScores() Function:**

- Updates the top scores by inserting the player's score in the correct position.

- Ensures only the top 5 scores are kept.

- Calls `updateTopScoreStrings()` to update string representations of the top scores.

- Calls `saveHighScores()` to save the updated scores to the file.

**5. saveHighScores() Function:**

- Opens the high scores file and writes the scores in JSON format.

**6. updateTopScoreStrings() Function:**

- Updates string representations of the top scores for display.

- Uses a `match` statement to assign values to `topScoreString1` to `topScoreString5`.

**7. reset() Function:**

- Resets counters related to the game.

**8. hreset() Function:**

- Calls `reset()`.

- Resets player-related variables and sets default values.

### 2.4 Main Script

The main script, designated as the `Main` class in the Godot game engine, serves as the central control for a card game. The script initializes crucial variables such as the `Card` scene, arrays for card numbers and signs, and various game-related parameters. The `\_ready()` function is responsible for setting up the initial user interface state, shuffling cards, creating the cards, and making certain UI elements visible or hidden.

The script contains functions for handling card manipulation, UI updates, player actions, and game outcomes. Notably, the script incorporates features like card shuffling, creating cards, updating card values based on game logic, and managing player actions such as hitting, passing, and doubling down.

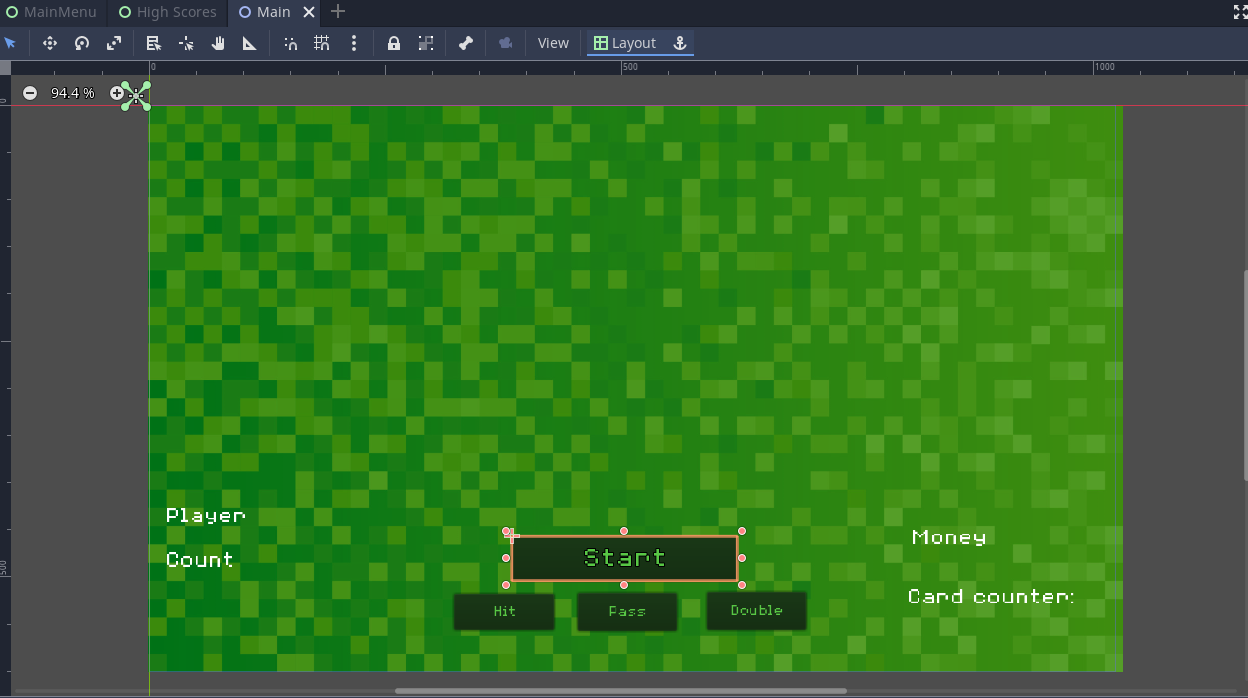
Gameplay-related functions, such as `addstart()`, `addplayer()`, `addDealer()`, and associated actions like checking for blackjack or determining the game outcome, are implemented. Additionally, the script contains functions for handling cheat logic and updating the card count.

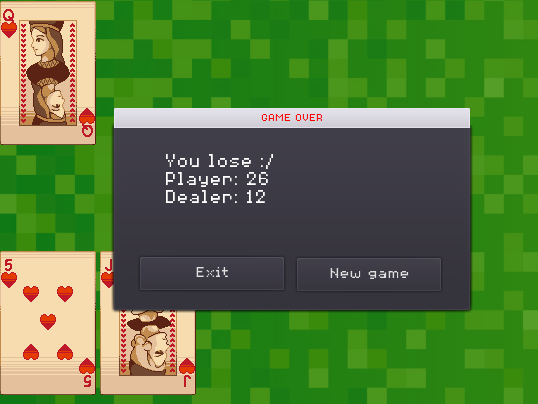
User interface elements, such as buttons and labels, are manipulated through functions like `\_on\_\*\_pressed()` and `button()`. These functions control the visibility and interactivity of UI elements based on game progression.

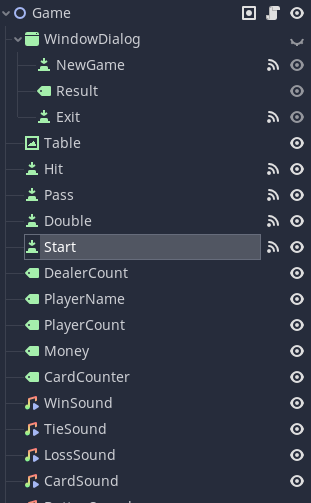
The script also handles scenarios where the game is won, lost, or tied, updating the player's money accordingly. The `bjcheck()` and `bjcheckdealer()` functions specifically check for blackjack conditions, influencing the game flow.

Overall, the main script encapsulates a comprehensive set of functions that collectively orchestrate the game's logic, user interface, and outcomes. Its role as a singleton allows for seamless integration with other scripts and facilitates a cohesive gaming experience.

### 2.5 UI of the Main scene

In the Main scene, the game begins with a minimalist display, where all elements, except for the "Start" button, remain hidden. This deliberate design choice ensures a clean and uncluttered interface for the initial phase. As the player progresses in the game, dynamic visibility changes occur. Once the game is in progress, key elements such as the player's money, card counter value, player name, player count, dealer count, and control buttons (double, pass, and hit) become visible. Additionally, the textures of the playing cards dynamically appear on the screen, providing a visually immersive experience as the game unfolds. This approach enhances the user interface, gradually revealing essential information and interactive elements to guide the player through different stages of gameplay. 

Upon the conclusion of a game, triggered by actions such as pressing "Pass" or the player or dealer busting, a window dialog emerges, offering players a post-game experience. Within this dialog, players are presented with options to either play again or exit the game. The window also displays pertinent information, including the final player count and dealer count, allowing players to reflect on the outcome. Additionally, the dialog reveals details about the winner, be it the player, the dealer, or a tie, providing a clear and informative summary of the concluded game. This thoughtful inclusion enhances the overall player experience by offering a seamless transition between game rounds and enabling players to make informed decisions for their next move. The picture below shows members of the scene UI and it includes various premade templates for UI elements such as the WindowDialog, buttons, sounds and labels.

The Godot game engine simplifies UI design through its intuitive scripting language, GDScript, and robust scene system. The elements present in the Main scene contribute to the ease of designing a user interface in Godot:

**1. Visibility Management:**

- Godot's scene system allows for straightforward management of element visibility. In the Main scene, hiding and revealing UI elements dynamically guides the player through different phases of the game, ensuring a clean and focused interface.

**2. Modular Functions:**

- The script's modular functions, such as `initialize\_ui()` and `ui\_visib()`, encapsulate specific UI-related tasks. This modular approach makes it easy to understand and maintain the code, facilitating iterative improvements and adjustments to the user interface.

**4. Dynamic Textures:**

- Godot's flexibility in handling textures and 2D elements allows for the dynamic appearance of playing cards during gameplay. This dynamic updating of textures enhances the visual appeal of the UI, providing a responsive and immersive gaming experience.

**5. Window Dialog for Game Outcomes:**

- The use of a window dialog to display game outcomes, providing options to play again or exit, simplifies the overall user experience. This centralized approach ensures consistency and clarity in presenting crucial information about the game, promoting ease of understanding and interaction.

**6. Event-Driven UI Updates:**

- The UI updates in response to specific game events, such as busting or pressing certain buttons. This event-driven approach streamlines the connection between game logic and UI presentation, simplifying the overall development process.

**7. Integrated Sound Effects:**

- Incorporating sound effects directly within the script, as seen in the `addstart()` and result functions, adds an auditory dimension to the UI. Godot's seamless integration of audio elements contributes to a more engaging and polished user interface.

In summary, the Godot game engine's features, such as the scene system, modular scripting, and dynamic texture handling, collectively contribute to making UI design in Godot easy and efficient. These elements empower developers to create responsive, visually appealing, and user-friendly interfaces for their games.

# Conclusion

In conclusion, the development of this project using the Godot game engine has demonstrated the platform's versatility and efficiency in creating a compelling card game experience. The use of autoloaded singleton scripts has facilitated seamless communication between different scenes and components, allowing for a cohesive and interconnected gameplay flow. The clear visibility management, modular design, and dynamic texture handling have significantly simplified the UI development process. Additionally, the incorporation of a window dialog for game outcomes enhances player engagement and provides a polished conclusion to each gaming session. Overall, the project underscores the strengths of the Godot engine in empowering developers to build interactive and visually appealing games with relative ease.

# Resources

<https://cazwolf.itch.io/pixel-fantasy-cards> - For card textures

<https://stock.adobe.com> - For images for backgrounds

<https://freesound.org> - For Sound Effects

<https://docs.godotengine.org/en/stable/tutorials/scripting/index.html> - Documentation for Godot and GDScript

ChatGPT - For help with code and error explanation