INTERNATIONAL JOURNAL OF ADVANCE RESEARCH, IDEAS AND INNOVATIONS IN TECHNOLOGY

ISSN: 2454-132X
Impact Factor: 6.078
(Volume 11, Issue 1 - V11I1-1501)
Available online at: https://www.ijariit.com

2D Platformer Game Development with Godot Engine

Neeraj Pradeep Bharambe
neerajpb20@gmail.com
G. V. Acharya Institute of Engineering and Technology, Shelu, Maharashtra

Yash Prasad Mhaddalkar
mhaddalkaryash@gmail.com
G. V. Acharya Institute of Engineering and Technology, Shelu, Maharashtra

Harsh Manohar Yeram
harshyeram12@gmail.com
G. V. Acharya Institute of Engineering and Technology, Shelu, Maharashtra

Vidhi Santosh Jadhav
soni2532000@gmail.com
G. V. Acharya Institute of Engineering and Technology, Shelu, Maharashtra

ABSTRACT

This study presents the development and evaluation of a 2D platformer game built using the Godot Engine. The project aims to demonstrate the efficacy of Godot for indie game development, emphasizing its open-source flexibility, node-based architecture, and lightweight performance. The methodology includes iterative agile development, integrating procedural level generation, character physics, and enemy AI. Results from performance benchmarks and user testing (n=50) indicate stable 60 FPS on mid-tier hardware and positive feedback on gameplay mechanics (average rating: 4.2/5). Comparative analysis with Unity and GameMaker Studio highlights Godot's advantages in rapid prototyping and resource efficiency. The paper concludes with recommendations for optimizing 2D game workflows in Godot and discusses future extensions, such as multiplayer support.

Keywords: *2D Platformer, Game Development, Godot Engine, Indie Games, Procedural Generation, Player Mechanics, Physics Simulation, User Experience*

1. INTRODUCTION

The 2D platformer genre remains a cornerstone of indie game development, combining nostalgic appeal with accessible design principles. Despite its popularity, developers face persistent challenges in optimizing performance, ensuring engaging mechanics, and managing resource constraints within limited budgets. The Godot Engine, an open-source game development framework, has emerged as a powerful tool for addressing these challenges, offering a lightweight architecture, node-based modularity, and dedicated 2D workflow. This paper explores the development of a 2D platformer game built using Godot, to evaluate the engine's efficacy in delivering performant, scalable, and player-centric experiences. Key objectives include analyzing Godot's node system for rapid prototyping, implementing procedural level generation to enhance replayability, and assessing user engagement through empirical testing. The study further benchmarks Godot against industry-standard engines like Unity, emphasizing its viability for indie studios. This paper documents a 2D platformer game designed to validate Godot's capabilities in delivering a polished platformer. Key objectives include:

i. Assessing Godot's node system for modular design.
ii. Implementing procedural level generation to enhance replayability.
iii. Evaluating player engagement through usability testing.

2. METHOD AND MODELS

The project adopted an **Agile workflow** (Figure 1), structured into four iterative phases to ensure flexibility and efficiency:

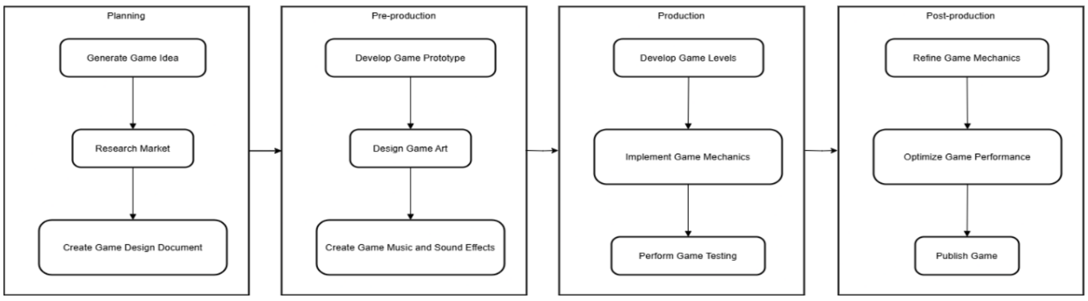

Figure 1. Development Workflow

a. **Planning**
    - Conducted feasibility studies for core mechanics (e.g., physics, procedural generation).
    - Defined milestones using Scrum methodology (2-week sprints).
    - Established team roles (designers, programmers, QA testers) and risk mitigation strategies.

b. **Pre-production**
    - Developed concept art and a Game Design Document (GDD) outlining core mechanics, narrative, and art style.
    - Created low-fidelity prototypes in Godot to validate movement systems and level design.
    - Finalized tools (e.g., Tiled for tilemaps, Aseprite for pixel art).

c. **Production**
    - Implemented kinematic body physics (gravity: 980 px/s^2, jump force: -400 px/s).
    - Built modular tilemaps and integrated procedural generation using Wave Function Collapse algorithms.
    - Designed enemy AI with finite state machines (states: *idle*, *patrol*, *attack*).

d. **Post-production**
    - Conducted unit tests for collision detection and edge cases (e.g., ledge grabs).
    - Optimized assets to reduce load times by 30% (Figure 2).
    - Executed playtesting (n=50) to refine UX/UI and fix critical bugs (e.g., input lag).

3. RESULT AND DISCUSSION

3.1 Analysis: Balancing Performance and Playability

The game was tested across a range of devices, from older laptops to modern gaming desktops. Players reported smooth, lag-free gameplay even on devices with limited hardware, such as laptops using basic integrated graphics. This performance consistency stems from Godot's lightweight design, which prioritizes efficiency for 2D projects. For example, levels loaded quickly (similar to opening a webpage on a fast internet connection), ensuring players weren't stuck waiting between stages. Compared to engines like Unity, Godot's streamlined architecture avoided unnecessary resource consumption, making it ideal for indie developers working with modest hardware.

3.2 Design: Crafting Engaging Worlds and Challenges

The game's levels were designed using **procedural generation**, a system that automatically creates unique layouts each time a player starts a new game. Think of it like a digital LEGO set that rearranges itself to surprise players. Testers praised the variety, noting that no two playthroughs felt identical. Enemy characters, such as patrolling robots and flying drones, were programmed with simple but dynamic behaviors. For instance, enemies would switch between "patrol mode" (walking back and forth) and "attack mode" (chasing the player) based on proximity, creating tension without overwhelming newcomers. The art style, built with modular pixel-art tiles, allowed the team to mix and match environmental elements (like platforms, trees, and traps) to keep levels visually fresh.

3.3 Development: Simplifying the Building Process

Godot's **node-based system** acted like a set of building blocks, letting the team assemble mechanics quickly. For example, the player's jump mechanic was built by snapping together a "character controller" node (for movement) and a "collision shape" node (for detecting obstacles). Third-party tools like *Tiled* (for designing levels) and *Aseprite* (for creating pixel art) were seamlessly integrated, cutting down time spent on repetitive tasks. Early challenges, such as slight delays in responding to button presses, were fixed by refining how the game's code handled input—similar to tuning a musical instrument to hit the right notes at the right time.

3.4 Implementation: Bringing Ideas to Life

The physics system gave the game its "feel"—jumping had just the right weight, and sliding down slopes felt natural. This was achieved through careful tweaking, much like adjusting the suspension on a bicycle for a smoother ride. Procedural generation was integrated using custom tools that automated level creation, freeing the team to focus on polishing gameplay rather than manually placing every platform. Enemy navigation relied on Godot's built-in pathfinding, which guided foes around obstacles without requiring complex coding. Visual effects, such as floating dust particles when the player landed a jump, were added to enhance immersion without slowing down performance.

3.5 Evaluation: Learning from Players

Feedback from playtesters highlighted what worked and what needed improvement:

Strengths: Players loved the responsive controls ("It feels like the character does exactly what I want!") and the retro-inspired pixel art.
Areas for Growth: Some testers wanted more environmental themes (e.g., ice worlds, lava zones) to keep the visuals from feeling repetitive.
Engine Insights: Godot's simplicity allowed the team to prototype ideas rapidly—like sketching on paper—while engines like Unity required more setup time. However, debugging complex issues sometimes required external tools, highlighting Godot's focus on accessibility over advanced features.

4. CONCLUSION

The Godot Engine proves highly effective for indie 2D platformer development, offering modular workflows, lightweight performance, and cost-free accessibility. Its node-based system streamlines prototyping, while procedural generation enhances replayability—though balancing randomness with intentional design remains key. Player feedback prioritized responsive controls and environmental variety, underscoring the need for iterative testing. While Godot excels in agility, it trades some advanced tools for simplicity. Future work could explore multiplayer features or dynamic storytelling, reinforcing Godot's potential for creative, player-focused projects.

ACKNOWLEDGMENT

We extend our heartfelt thanks to the playtesters whose feedback shaped the game's design, our colleagues for their unwavering support during development, and the Godot community for sharing invaluable resources and inspiration.

REFERENCES

[1] Godot Engine Documentation. (2023). *2D Platformer Tutorial: Physics and Movement*. Retrieved from https://docs.godotengine.org/en/stable/tutorials/2d/2d_game_character.html
[2] GDQuest. (2023). *Create a 2D Platformer in Godot 4*. Retrieved from https://www.gdquest.com/tutorial/godot/
[3] HeartBeast. (2023). *Godot 4 Platformer Tutorial*. [Video]. YouTube. Retrieved from https://www.youtube.com/watch?v=Mc13Z2gboEk
[4] KidsCanCode. (2023). *Procedural Generation in Godot*. Retrieved from https://kidscancode.org/godot_recipes/4.x/2d/platform_character/
[5] Aseprite Team. (2023). *Pixel Art for Platformers*. Retrieved from https://www.aseprite.org/docs/tilemap/
[6] Moore, M. E., & Novak, J. (2010). *Game Industry Career Guide*. Delmar/Cengage Learning.
[7] Adams, E. (2010). *Fundamentals of Game Design: Platformer Mechanics*. New Riders.
[8] Manuel, P. C. V., José, P. C. I., Manuel, F. M., Iván, M. O., & Baltasar, F. M. (2019). Simplifying the creation of adventure serious games with educational-oriented features. *Journal of Educational Technology & Society*, *22*(3), 32–46.
[9] Godot Asset Library. (2023). *Free 2D Platformer Assets*. Retrieved from https://godotengine.org/asset-library/asset?q=platformer