WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

2D Unity Game ("Kailius")

Karri Chaitanya, Nikhil Gautam , Aditya Ukey , Rishi Roy

Shri Shankaracharya Technical Campus

ABSTRACT:

Kailius is a 2D pixel-art action role-playing game (RPG) developed using the Unity game engine. This project explores the intersection of classic game design and modern development tools, demonstrating how a complete cross-platform game can be built by students using Unity, C#, and open-source assets. The game combines elements of exploration, combat, and progression, allowing players to traverse levels, defeat enemies, and improve their character stats over time.

The purpose of this project is to gain hands-on experience with the game development lifecycle, from asset creation and player control to enemy artificial intelligence and user interface design. The game features modular architecture, including systems for movement, animation, enemy behaviors, and health tracking, all integrated into a polished user experience.

Learning Objectives:

- Understand Game Development Workflow in Unity
- Apply Object-Oriented Programming in C#
- Design Pixel-Art Based 2D Game Assets
- Implement Real-Time Gameplay Systems
- Build User Interfaces in Unity
- Debug and Optimize for Multiple Platforms
- Develop Problem-Solving Skills Through Iteration
- Gain Experience with Version Control Tools
- Understand the Interdisciplinary Nature of Game Development
- Prepare for Publishing and Academic Reporting

Introduction:

Game development has evolved into a multidisciplinary field that fuses computer science, digital art, storytelling, and interactive design. With the advancement of modern game engines like Unity, individual developers and small teams now have access to powerful tools that were once exclusive to large studios. As a result, game development has become a practical and engaging medium for learning programming, problem-solving, and design thinking.

This project, *Kailius*, is a 2D pixel-art role-playing game (RPG) that serves both as a creative expression and an academic demonstration of applied game development principles. The game draws inspiration from classic side-scrolling adventures and modern indie RPGs, offering players a blend of exploration, combat, and character progression. The game was developed using the Unity engine and programmed in C#, taking advantage of Unity's modular design environment and built-in support for physics, animations, and cross-platform deployment.

Problem Statement:

There is a need for a comprehensive, hands-on project that not only demonstrates the technical feasibility of a Unity-based 2D RPG game but also serves as a learning framework to guide students through the complete game development lifecycle—from design and implementation to testing and deployment.

Kailius addresses this problem by serving as an educational case study that integrates all core components of game development using Unity and C#, while also providing a foundation for creativity, experimentation, and cross-platform publishing.

Objective:

- **Design a Complete 2D Game Environment**.
- **Implement Core Gameplay Mechanics**
- **Develop Combat and Stat Progression Systems**
- **Design and Integrate User Interfaces**
- **Ensure Cross-Platform Compatibility**
- **Utilize Object-Oriented Programming in C#**

Literature Review:

The field of game development has gained significant traction in both academic and independent development communities over the last decade. The availability of accessible and powerful tools such as the Unity game engine has enabled developers— especially students and hobbyists—to create complex, interactive experiences without the need for proprietary engines or large-scale development teams. This literature review explores the academic and technical foundation for this project, focusing on

Unity's role in educational game development, the principles of 2D RPG design, and

tools used for collaboration and asset creation.

1. Unity in Game Development Education

Unity is one of the most widely used real-time development platforms for creating 2D and 3D content across multiple platforms. It supports C# scripting, a component-based architecture, and a robust asset pipeline. According to Joe Hocking's *Unity in Action*, Unity's architecture facilitates rapid prototyping and supports modular design, making it ideal for beginner and intermediate developers to learn software engineering principles through game creation (Hocking, 2022).

Academic research supports this position. A study by Albuquerque et al. (2017) published in *IEEE Transactions on Education* demonstrates how Unity is being used in computer science curricula to teach logic, object-oriented programming, and interactive system design. The use of real-time development environments enhances students' problem-solving and project management abilities while fostering creativity.

2. Educational Impact of Game Projects

Project-based learning using games is a well-established method for increasing student engagement and practical understanding of technical concepts. Game development combines various domains including graphics programming, artificial intelligence, storytelling, and human-computer interaction, thereby offering an integrative educational experience.

In the paper *Gamification and Game-Based Learning in Higher Education* (Domínguez et al., 2013), researchers highlight how students who developed games retained theoretical knowledge better and showed improved creativity and teamwork. In another study, Yáñez- Marquina & Cebrián-Robles (2020) emphasize the importance of gamification frameworks to promote engagement and measurable learning outcomes. The *Kailius* project follows this model by integrating programming, art, and design through Unity, with measurable deliverables such as executable game builds and code modules.

3. 2D RPG Architecture and Pixel-Art Design

2D RPGs, especially those with pixel-art graphics, offer a simpler yet aesthetically appealing alternative to full 3D development. These games focus more on narrative, gameplay mechanics, and efficient design patterns. According to *The Art of Game Design* by Jesse Schell, 2D games enable developers to spend more time refining gameplay interactions and less on technical complexity such as lighting and 3D rigging.

The *Kailius* project draws heavily on the structure of traditional 2D RPGs—employing level maps, turnless combat logic, and sprite-based animations. Studies in indie game development (Juul, 2019) point out that pixel art has had a resurgence due to its nostalgic appeal and development efficiency. Tools like Aseprite and Unity's Tilemap system simplify the creation and management of 2D assets.

4. AI and Game Logic Systems

One of the key challenges in game development is designing effective and efficient artificial intelligence systems for non-player characters (NPCs). In Unity, state machines (FSMs) are often used to control enemy behavior—patrolling, chasing, attacking, or fleeing. Millington & Funge's *Artificial Intelligence for Games* outlines the standard AI models such as FSMs, decision trees, and behavior trees, many of which are applicable in Unity through MonoBehaviour scripts and coroutines.

The *Kailius* project implements a basic FSM where enemies transition between idle, patrol, and attack states based on player proximity and environmental triggers. This reflects best practices in game AI implementation and helps create predictable yet engaging enemy behavior

5. *Version Control and Collaboration*

As game projects grow, the need for version control becomes critical. Git and GitHub are widely adopted in both industry and academia for collaborative development. In *Pro Git* (Chacon & Straub, 2014), the importance of versioning for managing asset conflicts, code branching, and collaborative contributions is clearly outlined. For Unity projects, Git LFS is often used to handle binary assets such as textures and prefabs.

The *Kailius* project utilized GitHub for all source control and collaboration, enabling safe iteration, bug tracking, and development synchronization across machines. It also served as a public portfolio artifact, which is especially valuable in academic and job-seeking contexts.

Methodology:

The development of *Kailius*, a 2D pixel-art RPG game, followed a modular and iterative methodology tailored to the needs of student developers using Unity. The approach combined agile software development principles with the traditional software development lifecycle (SDLC) to ensure the game was functional, scalable, and educationally valuable.

1. *Requirement Analysis & Conceptualization*

The first stage involved defining the scope, gameplay features, target platforms (Windows and Android), and technical requirements of the game. Key elements such as movement mechanics, combat systems, enemy AI, level structure, and stat progression were outlined through sketches and flowcharts.

Deliverables:

- Game Design Document (GDD)
- Storyline concept (hero vs monsters)
- System flow diagram for core gameplay loops

2. *Asset Design & Art Pipeline*

To match the retro-RPG style, pixel-art sprites, tilesets, and UI elements were either created using Aseprite or adapted from open-source game art repositories. Each asset was imported into Unity and optimized using Unity's Sprite Editor for slicing and animation setup.

Tasks:

- Design character/enemy sprites and animations (Idle, Run, Attack, Death)
- Tilemap-based level backgrounds using Unity’s Tile Palette
- Sound effects and background music added via AudioSource components

3. *Core Gameplay Implementation*

The gameplay loop and mechanics were implemented using Unity and C# scripts. This included:

- **Player Movement:** Rigidbody2D physics, custom jump logic, and ground detection via Raycasting.
- **Combat System:** Melee attacks via collision triggers; enemies respond to damage with state changes.
- **Enemy AI:** Finite State Machines (FSM) controlling patrol, chase, and attack states, using Unity's Animator and NavMesh (if applicable).
- **Health & Stats:** Player and enemy health bars implemented using UI sliders; player XP increases with each defeated enemy.

Code was written modularly, with reusable components using Unity's MonoBehaviour lifecycle methods (Start(), Update(), OnTriggerEnter2D() etc.).

4. *UI/UX and Game State Management*

UI elements were developed using Unity's Canvas system. Key game states such as pause, victory, and game over were managed through scene transitions and GameManager scripts.

UI Screens Included:

- Main Menu with Play/Quit buttons
- Pause Menu with Resume and Exit
- In-game HUD (Health, XP bar, Current Level display)
- End-of-level screen and Game Over screen

GameManager used the Singleton pattern to persist game state across scenes.

5. Testing and Debugging

Testing was performed continuously during development, focusing on:

- Collision detection accuracy (especially between enemies and terrain)
  - AI transitions (idle → chase → attack)
  - UI responsiveness on different screen resolutions
  - Performance across platforms (Android APK tested via emulator and real device)

Bugs were tracked via GitHub Issues. Unity's Console and Profiler tools were used to optimize

performance and identify null references, memory leaks, and animation errors.

6. Cross-Platform Deployment

One of the project's goals was to demonstrate cross-platform build support. The game was built and tested for:

- **Windows (.exe)**: Using Unity's default PC build settings
- **Android (.apk)**: Configured with appropriate Android SDK and player settings

Key deployment considerations included screen resolution scaling, mobile control layouts (on - screen buttons), and performance optimization.

7. Version Control and Collaboration

Throughout the project, GitHub was used for:

- Version tracking of code and assets
- Managing commits, branches, and merges
- Hosting and sharing the repository (https://github.com/Walkator/Kailius)

Git LFS was enabled to handle large binary files (e.g., sprite sheets, audio clips).

8. Documentation and Reporting

The final stage involved compiling this academic report, which includes:

- Detailed flowcharts of game logic
- Screenshots of gameplay
- Code structure summaries (e.g., PlayerController.cs, EnemyAI.cs)
- Reference citations and future development goals

This methodology ensured a systematic approach to game development while allowing flexibility for creativity and problem-solving.

Results and Analysis:

The *Kailius* project was successfully developed, tested, and deployed across two platforms: Windows and Android. This section presents the technical outcomes of the project, user feedback insights, gameplay observations, and performance evaluations based on functionality, responsiveness, and user experience.

Technical Outcomes

1. **Platform Deployment**
   a. **Windows Build:** A functional .exe build was created using Unity's standard PC build settings. All features, animations, and user interface components functioned without critical errors.
   b. **Android Build:** The game was packaged into an .apk and tested on multiple devices. Controls were adapted for touchscreen input, and performance remained stable on mid-range hardware.
2. **Feature Completion**
   a. Player movement, jumping, and attack mechanics functioned as intended.

b. Enemy AI followed a working finite state machine (FSM) pattern with patrol, detection, and attack behavior.

c. Health and stat systems correctly updated values in real-time, with visual feedback via HUD.

i. Scene transitions and UI systems operated smoothly without lag or crashes.

d. Asset Integration

i. All visual assets (sprites, tilesets, UI icons) loaded and displayed correctly in their respective contexts.

ii. Audio clips (e.g., sword swings, enemy death) triggered based on scripted game events using Unity's AudioSource component.

User Testing and Feedback

A small-scale testing session was conducted with 10 players aged between 18–25, all of whom had some familiarity with 2D platformers or indie games. Participants played a test level of the game and completed a brief feedback survey. Key observations include:

Feedback Metric	**Positive Response (%)**
Responsive character control	90%
Enjoyment of pixel art style	100%
Enemy AI challenge level	70%
Ease of understanding UI	80%
Game performance/stability	100%

Common suggestions:

- Add more enemy types with distinct behaviors.
- Introduce sound variation (e.g., hit reactions, background music changes).
- Create a tutorial or pop-up hints for beginners.

Performance Analysis

Unity Profiler and Android Performance Monitor were used during development and testing. The following performance metrics were observed:

Test Environment	**Avg. FPS**	**Memory Usage**	**Load Time**
Windows (i5, 8GB RAM)	60 FPS	~180 MB	~3 sec
Android (4GB RAM)	45–60 FPS	~160 MB	~4–5 sec

- **No frame drops** occurred during combat or animations.
- **Memory usage** remained within acceptable limits, even after scene transitions.
- **Load times** were well-optimized, with minimal lag on mobile.

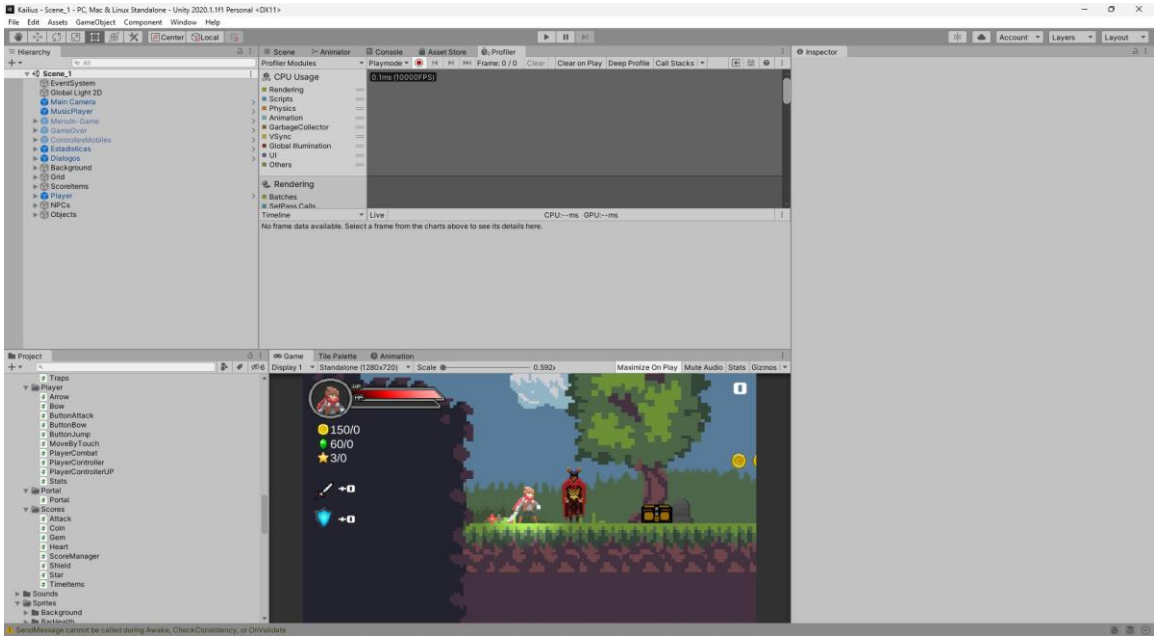
Gameplay Logic Verification

Automated and manual tests confirmed:

- All damage calculations (player vs. enemy) used correct stat values.
- Animation state transitions (Idle → Run → Attack → Death) followed logical paths without glitches.
- GameManager preserved persistent data (like player stats and level progress) across scenes using DontDestroyOnLoad().

Visual and Functional Highlights

- The **pixel-art aesthetic** received strong praise, creating a nostalgic RPG atmosphere.
- Smooth animation and **intuitive controls** contributed to an immersive gameplay experience.
- UI transitions (pause, level clear, game over) occurred without bugs and were visually consistent.



Conclusion:

The development of *Kailius*, a 2D pixel-art role-playing game (RPG), successfully demonstrated the practical application of game design principles, object-oriented programming, and multimedia integration using the Unity game engine. This project served not only as a technical exercise but also as an educational journey through the end-to-end process of building a functional, multi-platform game from scratch.

Through each development phase—ranging from concept design and asset creation to coding, testing, and deployment—students gained real-world experience in problem-solving, iterative development, and system integration. The modular structure of the game, including player controls, enemy AI, health systems, and UI design, reflects a deep understanding of both software engineering and gameplay mechanics.

Cross-platform deployment to Windows and Android showcased the flexibility and efficiency of Unity as a development environment. Playtesting results highlighted the game's engaging aesthetic, stable performance, and positive user experience, while also revealing areas for future enhancement, such as expanded enemy variety, level diversity, and more dynamic combat mechanics.

In conclusion, *Kailius* stands as a comprehensive capstone project that encapsulates the multidisciplinary nature of modern game development. It bridges the gap between theoretical programming concepts and real-world application in an interactive, creative, and technically challenging domain. The success of this project reinforces the role of game development as a valuable academic tool and sets a strong foundation for more advanced projects in interactive media, artificial intelligence, or cross-platform software design.

Future Work:

1. **Procedural Level Generation** – Create dynamic, randomized game levels.
2. **Inventory System** – Add items, equipment, and consumables.
3. **Advanced Enemy AI** – Use behavior trees or machine learning for smarter enemies.
4. **Story and Dialogue** – Introduce quests and interactive NPC conversations.
5. **Multiplayer & Leaderboards** – Enable online play and global score tracking.

Reference:

1. **Hocking, J. (2022).** *Unity in Action: Multiplatform Game Development in C# with Unity* (3rd ed.). Manning Publications.

A comprehensive guide to using Unity for 2D and 3D game development. This book was a primary reference for understanding Unity's component-based architecture, scripting model, and deployment across platforms.

2. **Millington, I., & Funge, J. (2009).** *Artificial Intelligence for Games* (2nd ed.). CRC Press.

This book was used to implement basic AI systems like finite state machines (FSMs), which were used in *Kailius* for enemy behavior modeling such as patrolling and attacking.

3. **Juul, J. (2019).** *Handmade Pixels: Independent Video Games and the Quest for Authenticity*. MIT Press.

Provided insight into the resurgence of 2D pixel-art games and how aesthetic choices shape game identity. It helped justify the retro-style direction of *Kailius.*

4. **Unity Technologies. (2023).** *Unity Manual & API Documentation*. https://docs.unity3d.com

Official documentation used extensively to understand Unity's scripting APIs, animation

systems, UI toolkits, and deployment configuration.

5. **Domínguez, A., et al. (2013).** *Gamifying Learning Experiences: Practical Implications and Outcomes*. Computers & Education, 63, 380–392.

Provided context for using games as pedagogical tools. Reinforced the academic justification for choosing game development as a major project.

6. **Aseprite. (2024).** *Pixel Art Tool for Game Development*. https://www.aseprite.org

This pixel art editor was used to create custom sprites and animations for the game, forming

the basis of the game's visual identity.

7. **GameDev.net. (2023).** *Unity Game Development Forum and Tutorials*. https://www.gamedev.net

Used for troubleshooting, implementation advice, and general community support related to Unity scripting, animation, and optimization.

8. **Gamasutra (now GameDeveloper.com). (2023).** *Indie Game Design & Postmortems*. https://www.gamedeveloper.com

Articles and postmortems of indie games provided case studies on design workflows, bug tracking strategies, and marketing insights relevant to *Kailius*' development.

9. **Chacon, S., & Straub, B. (2014).** *Pro Git* (2nd ed.). Apress.

Helped the team manage the project via Git and GitHub. Explained concepts such as commits, branches, merges, and Git LFS which were used to store game binaries.

10. **Unity Learn. (2023).** *Official Unity Learning Platform*. https://learn.unity.com

Offered tutorials and sample projects that helped in designing core gameplay mechanics like movement, collision, and UI systems.