

Warsaw University of Technology

FACULTY OF ELECTRONICS AND
INFORMATION TECHNOLOGY



Bachelor's diploma thesis

in the field of study Computer Science
and specialization Computer Systems and Networks

Creating multiplatform match three game engine

numer pracy według wydziałowej ewidencji prac {liczba}

Krzysztof Rudnicki

numer albumu 307585

promotor
dr hab. inż. Tomasz Martyn

konsultacje
dr hab. inż. Tomasz Martyn

Warszawa 2023

Streszczenie pracy

Creating multiplatform match three game engine
{SLOWA KLUCZOWE}

Thesis abstract

Example of creation of multiplatform apps on basis of creating match-three game engine
{KEYWORDS}

Contents

1	Introduction	5
1.1	Background and Motivation	5
1.2	Scope of the Research	5
1.2.1	Multiplatform Development	6
1.2.2	Game Engine Focus: Match Three Genre	6
1.2.3	Tool and Library Integration	6
1.2.4	Theoretical and Practical Exploration	6
1.2.5	Limitations	6
1.3	Objectives of the Thesis	6
1.3.1	Design of a Cross-Platform Core Architecture	7
1.3.2	Effective Integration of Libraries	7
1.3.3	Development of Core Match Three Mechanics	7
1.3.4	Performance Benchmarking and Optimization	7
1.3.5	Documentation and Usability	7
1.3.6	Providing a Blueprint for Future Development	7
1.4	Choice of graphic rendering API	8
1.5	Choice of OpenGL Library	8
2	Literature Review	10
2.1	Evolution of Match Three Games	10
2.1.1	Origins and Early Forerunners	10
2.1.2	Mainstream Adoption and Innovations	10
2.1.3	Technological Influences and Platform Diversification	10
2.1.4	Enduring Appeal and Contemporary Significance	11
2.2	Overview of Multiplatform Game Development	11
2.3	Existing Game Engines and Their Limitations	12
3	Basics of Game Engine Design	14
3.1	Core Components of Game Engines	14
3.2	The Role of OpenGL in Game Development	15
3.3	Multiplatform Considerations	15
4	Toolchain and Libraries Overview	16
4.1	Introduction to GLFW	16
4.2	Role of GLAD in OpenGL Loading	16
4.3	Graphics Rendering with FreeType and stb_image	16
4.4	Introduction to ft2build and Text Rendering	16
5	Design and Architecture of the Match Three Engine	17
5.1	Game Loop and State Management	17
5.2	Asset Management and Rendering	17
5.3	Event Handling and User Input	17

6	Cross-Platform Development Challenges and Solutions	18
6.1	Operating System Variabilities	18
6.2	Addressing Hardware and Driver Differences	18
6.3	Testing and Quality Assurance Across Platforms	18
7	Implementation Details	19
7.1	Core Engine Implementation	19
7.2	Match Three Logic and Mechanics	19
7.3	Integration of Libraries and Tools	19
8	Performance Optimization and Benchmarking	20
8.1	Performance Metrics and KPIs	20
8.2	Optimization Techniques Employed	20
8.3	Benchmark Results Across Platforms	20
9	Case Study: Development of a Prototype Match Three Game	21
9.1	Game Concept and Design	21
9.2	Application of the Engine in Real-world Development	21
9.3	Feedback and Iterative Improvement	21
10	Future Work and Enhancements	22
10.1	Potential Extensions to the Engine	22
10.2	Integrating Newer Technologies and Libraries	22
10.3	Broader Applicability to Other Game Genres	22
11	Conclusion	23
11.1	Summary of Achievements	23
11.2	Contributions to the Field of Game Development	23
11.3	Reflection on the Development Process	23
12	References	24
13	Appendices	25

Chapter 1

Introduction

1.1 Background and Motivation

In the ever-evolving realm of digital entertainment, video games have solidified their position not merely as a pastime, but as a cultural phenomenon. Among various game genres, "Match Three" games, characterized by their intuitive mechanics and satisfying gameplay, have attracted millions of players around the globe. Players are drawn into the experience of aligning three or more similar game pieces, leading to cascading matches and gratifying chain reactions. This universal appeal is evident in the success of titles like "Candy Crush Saga" and "Bejeweled", which have not only dominated mobile gaming charts but also established a significant presence on desktop platforms.

Yet, despite the evident popularity and longevity of the Match Three genre, there remains a challenge in the development realm: the creation of a robust, versatile, and multiplatform game engine. While numerous game engines cater to shooting, adventure, or role-playing genres, there's a notable lack of dedicated Match Three engines that can seamlessly function across multiple operating systems, from Windows to Mac to GNU/Linux. This gap is surprising given the genre's ubiquity and the increasing demand for multiplatform games in today's fragmented digital ecosystem.

The technological landscape of game development is dotted with myriad tools and libraries, each offering unique capabilities. OpenGL, for instance, has stood the test of time, delivering powerful graphics rendering capabilities across platforms. While it serves as a bedrock for various engines, its intricate details and vast scope can be daunting for developers looking to create a niche game engine. Coupled with other essential libraries like GLFW, GLAD, FreeType, stb_image, and ft2build, the potential to create a sophisticated Match Three game engine becomes palpable. However, harnessing these tools effectively requires a deep understanding and a clear vision.

This thesis emerges from a confluence of two primary motivations. Firstly, to fill the apparent void in dedicated, multiplatform Match Three game engines. And secondly, to showcase the integration of OpenGL with a suite of auxiliary libraries, demystifying the process and offering a blueprint for future developers. As games continue to break barriers of language, culture, and geography, there is immense value in fostering tools that simplify their creation, ensuring that the joy of gaming is accessible, regardless of the platform. The subsequent chapters will delve deep into the intricacies of designing such an engine, highlighting challenges, solutions, and innovations that arise in this exciting journey.

1.2 Scope of the Research

Within the broad spectrum of game development, it is crucial to delineate the boundaries of exploration and analysis to ensure focused research and actionable insights. This section elucidates the extent and limitations of the current research, offering clarity on the domains of study and potential applications of the developed engine.

1.2.1 Multiplatform Development

The heart of this research is anchored in multiplatform game development. While the realm of gaming spans myriad devices from mobile phones to gaming consoles, our attention will predominantly be on three major desktop operating systems: Windows, Mac, and GNU/Linux. The challenges associated with creating a uniform gaming experience across these platforms, considering their architectural, functional, and graphical differences, will be dissected. This research does not venture into the domain of mobile, console, or web-based game engine development.

1.2.2 Game Engine Focus: Match Three Genre

While game engines can cater to an array of genres, the essence of this research lies in the Match Three genre. This decision stems from its universal appeal and the nuanced challenges inherent in its mechanics. We will delve deep into the core logic, rendering techniques, and optimizations peculiar to Match Three games. Other genres, although intriguing in their own right, fall outside the purview of this study.

1.2.3 Tool and Library Integration

This research will meticulously delve into the integration of OpenGL with a selection of libraries: GLFW, GLAD, FreeType, stb_image, and ft2build. These tools were chosen for their pivotal roles in creating a holistic game engine—handling everything from window management to graphics rendering and text display. The study will detail the interoperability of these tools, their roles, and their synergies in the broader game engine architecture. Libraries and tools not mentioned remain outside the scope.

1.2.4 Theoretical and Practical Exploration

While a significant portion of this research is rooted in practical implementation, there is also a strong emphasis on theoretical concepts underpinning game engine development. The balance ensures that readers gain not only a hands-on understanding but also a conceptual foundation to innovate and expand upon the discussed methodologies.

1.2.5 Limitations

It's essential to acknowledge that, while the engine aims to be comprehensive and robust, it may not encapsulate every possible feature or optimization inherent to commercial-grade game engines. The primary intent is to lay a solid foundation upon which further enhancements can be made, rather than delivering an exhaustive commercial product.

In conclusion, this research endeavors to provide a comprehensive understanding of multiplatform Match Three game engine development. By setting clear boundaries, it aims to dive deep into specific challenges, tools, and solutions, providing valuable insights for budding game developers, researchers, and enthusiasts. The subsequent sections will further crystallize the objectives, methodologies, and findings that underpin this exciting exploration into the world of game development.

1.3 Objectives of the Thesis

In any research endeavor, clear objectives pave the way for structured investigation, targeted outcomes, and measurable accomplishments. As we journey through the intricate world of multiplatform Match Three game engine development, it's paramount to define the guiding lights – the objectives that underpin this thesis. Herein, we delineate the primary goals and anticipated contributions of this research.

1.3.1 Design of a Cross-Platform Core Architecture

The foremost objective is to design a core engine architecture that's inherently multiplatform. This demands an in-depth understanding of the nuances of different operating systems, from their system calls to their graphics pipelines. The engine should be modular and scalable, ensuring that the core mechanics and logic can be implemented seamlessly across Windows, Mac, and GNU/Linux platforms without redundancy.

1.3.2 Effective Integration of Libraries

With a host of libraries at our disposal, a significant goal is to weave them into the engine in a cohesive manner. This requires meticulous exploration of GLFW for window management, GLAD for OpenGL function pointers, FreeType and ft2build for text rendering, and stb_image for image handling. Beyond mere integration, the objective is to ensure that these libraries operate in harmony, complementing each other to augment the engine's overall performance and capability.

1.3.3 Development of Core Match Three Mechanics

While the architecture and library integrations form the skeleton, the Match Three mechanics are the heart of the engine. The aim is to devise efficient algorithms and techniques to handle typical Match Three functionalities – from matching detection to gravity-induced piece drops and chain reactions. The challenge lies not just in creating these mechanics, but in optimizing them for smooth gameplay and adaptability to various game designs.

1.3.4 Performance Benchmarking and Optimization

An essential facet of this research is gauging the engine's performance across platforms. The objective here is twofold: to establish benchmarking methodologies that accurately reflect the engine's capabilities and to implement optimization techniques that enhance its efficiency. Performance isn't merely about speed; it also encompasses reliability, consistency, and adaptability to various hardware specifications.

1.3.5 Documentation and Usability

Creating an engine is half the battle; ensuring it's accessible and usable for developers is equally crucial. An objective of this thesis is to produce comprehensive documentation, detailing the engine's architecture, functions, and integration points. This aids developers in understanding, modifying, or expanding upon the engine, ensuring its longevity and adaptability to various Match Three game concepts.

1.3.6 Providing a Blueprint for Future Development

While the immediate objective is the engine's development, a broader goal is to inspire and guide. By addressing challenges, sharing solutions, and showcasing the integration of diverse tools, this research hopes to offer a blueprint for budding developers. Whether they wish to enhance this engine, create new game engines, or explore further into the realm of multiplatform development, this thesis serves as a foundational guidepost.

In summation, the objectives of this thesis are ambitious yet grounded. They stem from a blend of technical aspirations and the broader goal of contributing meaningfully to the game development community. As we embark on subsequent chapters, these objectives will shape the narrative, guiding readers through the intricacies of design, development, testing, and application in the captivating world of Match Three games.

1.4 Choice of graphic rendering API

There are 3 main APIs for graphical rendering

- DirectX
- OpenGL
- Vulkan

DirectX developed by Microsoft focuses on Windows operating systems and Microsoft line of consoles Xbox, it is deemed as being harder with more low level programming and requiring better understanding of how underlying mechanisms work but in turn offers functionalities and better performance. It does not have free license.

OpenGL is developed by Khronos Group and offers good compatibility, especially if using OpenGL ES subset which works on Windows, Linux, Mac OS, Android, iOS and all major consoles. It is widely recognized as easiest of APIs and most popular choice for writing first game engine. On the other hand it lacks some of more advanced features which have to be written manually. It uses open source license similar to BSD

Vulkan is also developed by Khronos Group and as such is deemed as a spiritual successor of OpenGL with focus on using modern C++ features and fixing issues created by OpenGL 30 years old development time. Out of these three it is recognized as the hardest one as it is both complicated and newest. Similarly as OpenGL it uses open source license, namely Apache License 2.0.

Considering all of those characteristics I decided to go with OpenGL API, specifically OpenGL ES subset with its focus on compatibility as making a multiplatform application is one of the focuses of this thesis. I decided that since this will be my first attempt at game engine development I need something that is relatively easy and has a lot of resources online. I would most likely not use advanced features of Vulkan and DirectX and therefore finish my thesis before approaching problems where OpenGL does not deliver more complicated architecture. From my own private preferences I also prefer software with open source license.

1.5 Choice of OpenGL Library

There are 4 basic OpenGL libraries that I considered:

- freeGLUT
- SDL
- SFML
- GLFW

freeGLUT was created as opensource alternative to GLUT, is considered to be the worst out of all 4, written in archaic way, using C or very old C++, which in turn results in unexpected "buggy" behaviour, it is also not really popular with lack of online guides

SDL - Simple DirectMedia Layer has big userbase, it is not designed to be used as a standalone library and requires additional libraries to do networking or to create more complex applications.

SFML is the library with most features out of all 4, it supports networking, audio and has system features by default. It uses modern object oriented C++. Main problem with SFML is that it is not very popular API, therefore troubleshooting problems with SFML is quite hard and it has only few use guides online

GLFW is an library that is both the most popular and with fewest features by default. It forces users to use additional libraries for networking, sound, physic calculations and so on but in turn is also quite small and flexible. It has biggest community and a lot of guides, like one hosted at learnopengl.com

or one created by programming youtuber Chernob

I decided to use GLFW library. I wanted something that is relatively easy to troubleshoot and has abundance of learning materials online.

Chapter 2

Literature Review

2.1 Evolution of Match Three Games

The world of video games is vast and varied, encompassing everything from sprawling multi-universe role-playing games to concise mobile puzzles. Amidst this variety, Match Three games have carved a unique niche for themselves, offering players a blend of strategy, pattern recognition, and instant gratification. To appreciate the significance of our endeavor in creating a multiplatform Match Three game engine, it's essential to trace the genre's journey, understanding its roots, its evolution, and its enduring appeal.

2.1.1 Origins and Early Forerunners

Match Three games owe their lineage to the broader genre of tile-matching video games. The earliest entrants in this space were games like "Tetris" in the 1980s, where players manipulated falling blocks to create complete lines. Although not a Match Three game in the traditional sense, "Tetris" laid the groundwork by emphasizing spatial reasoning and pattern recognition.

The late 1980s and 1990s saw games like "Columns" and "Dr. Mario" come to the fore. These games introduced the mechanic of matching three or more identical items, but they were still largely rooted in the falling-block paradigm. The true prototype of modern Match Three mechanics can be credited to "Shariki," a 1994 Russian game where players swapped adjacent pieces to form chains of three or more identical items.

2.1.2 Mainstream Adoption and Innovations

The early 2000s marked a turning point for Match Three games, with the release of "Bejeweled." Simplifying and refining the mechanics of "Shariki", "Bejeweled" brought the genre to the limelight. Its intuitive gameplay, combined with visually appealing graphics, set the benchmark for numerous successors.

The success of "Bejeweled" catalyzed a wave of innovation. Developers experimented with various twists on the core mechanics. Games introduced power-ups, barriers, objectives, and narrative elements. "Candy Crush Saga", released in 2012, integrated a level-based progression system, further elevating the genre's complexity and depth.

2.1.3 Technological Influences and Platform Diversification

The evolution of Match Three games cannot be dissociated from the technological shifts in the gaming industry. Initially tethered to PCs and consoles, the rise of mobile devices opened a new frontier for the genre. The touch interface of smartphones and tablets proved to be an ideal medium for the drag-and-swap mechanics of Match Three games.

Simultaneously, browser-based games, powered by technologies like Flash, allowed players to engage with Match Three puzzles without heavy downloads or installations. As technology progressed, the need for engines that could cater to diverse platforms – mobile, browser, and desktop – became apparent.

2.1.4 Enduring Appeal and Contemporary Significance

Despite the plethora of game genres available today, Match Three games remain popular. Their success can be attributed to their 'easy to learn, hard to master' ethos. The games offer immediate rewards – the satisfaction of creating a match, the visual delight of pieces disappearing, and the strategic depth of planning several moves ahead. Additionally, the modular nature of their design allows for easy adaptation, whether it's incorporating a narrative, integrating with popular cultures, or tailoring to specific demographics.

In conclusion, the trajectory of Match Three games is a testament to their adaptability and universal charm. From humble beginnings in the tile-matching universe to reigning as one of the most recognizable genres in mobile and desktop gaming, they continue to captivate audiences. This historical perspective underscores the significance of developing tools and engines tailored to Match Three games, ensuring that the genre thrives and evolves in the ever-changing gaming landscape.

2.2 Overview of Multiplatform Game Development

In today's digitally-connected landscape, games are no longer confined to a singular platform. Developers strive to reach wider audiences by ensuring that their creations are accessible across a variety of operating systems. Multiplatform game development has evolved as a solution to bridge the divide between different user bases. This section delves into the intricacies, challenges, and methodologies inherent in multiplatform game development.

2.2.1 The Need for Multiplatform Development

The proliferation of computing devices, each with its unique operating system, has diversified the audience base. With Windows, Mac, and GNU/Linux ruling the desktop world, the potential to reach a more expansive audience multiplies when a game is made available across these platforms. Moreover, by catering to multiplatform users, developers can harness a more comprehensive feedback loop, potentially driving improvements and innovations.

2.2.2 Challenges in Multiplatform Development

While the idea of developing once and deploying everywhere sounds enticing, it comes with its set of challenges:

Hardware Disparity: Different platforms might have varying hardware specifications, which can influence the game's performance. **OS Specific Limitations:** Each operating system has its nuances, from system calls to user interface guidelines. **Toolchain Variability:** Development tools and libraries might have platform-specific versions or may not be available for all platforms. **Testing Complexities:** Ensuring consistent gameplay experience requires rigorous testing across all target platforms.

2.2.3 Tools and Libraries for Cross-Platform Development

Over the years, developers have leaned on tools and libraries to mitigate multiplatform development challenges. Libraries like OpenGL offer a unified graphics rendering solution, while GLFW facilitates consistent window management and input handling. The inclusion of GLAD, FreeType, stb_image, and ft2build in our game engine project exemplifies the quest for libraries that offer consistent behavior across platforms.

2.2.4 Strategies for Effective Multiplatform Development

Modular Design: By compartmentalizing the game engine into distinct modules, developers can isolate platform-specific code, ensuring that core logic remains untainted by platform dependencies. **Middleware Utilization:** Middleware solutions can bridge the gap between different platforms, offering a common interface for game development. **Continuous Integration and Testing:** Automated testing across platforms can identify inconsistencies, ensuring that the game offers a uniform experience. **Community Engagement:** Leveraging the developer community can unearth platform-specific insights, solutions, and optimizations.

2.2.5 Future Trajectory of Multiplatform Development

With the advent of cloud gaming, web-based games, and increasingly powerful mobile devices, multiplatform development's scope is set to expand. Developers will grapple with newer platforms,

more varied devices, and increasingly discerning audiences. The essence will remain the same: ensuring that games offer a consistent, engaging, and seamless experience, irrespective of the platform.

Concluding, multiplatform game development stands at the intersection of technological prowess and user-centric design. By understanding its nuances, challenges, and methodologies, developers are better equipped to navigate the diverse landscape of modern gaming. This foundation will prove invaluable as we journey further into the development of our multiplatform Match Three game engine.

2.3 Existing Game Engines and Their Limitations

The gaming industry has seen the rise of various game engines, each offering a unique set of tools and functionalities tailored to different needs. While these engines have facilitated the creation of iconic games, they also come with certain limitations, especially concerning specific genres or platforms. This section will dissect prominent game engines and analyze their shortcomings, providing a backdrop against which our multiplatform Match Three game engine can be juxtaposed.

2.3.1 Unity: The All-Rounder with a Price

Unity is undoubtedly one of the most popular game engines available today. It's known for its versatility, supporting a wide range of platforms, from PCs to consoles to mobile devices.

Strengths: Robust community support, a vast asset store, and an intuitive interface make Unity a favorite among both beginners and professionals. Limitations: Licensing Costs: For developers who surpass a certain revenue threshold, Unity can become expensive. Overhead: Unity's all-purpose nature might introduce unnecessary bloat for simpler games, like Match Three titles, potentially impacting performance.

2.3.2 Unreal Engine: The Visual Powerhouse

Epic Games' Unreal Engine stands out for its cutting-edge graphical capabilities, often employed in AAA game titles.

Strengths: With its Blueprint visual scripting system and top-tier rendering capabilities, Unreal Engine is ideal for creating visually stunning games. Limitations: Learning Curve: Its advanced features can be overwhelming for beginners. Overkill for Simpler Genres: For a Match Three game, the engine might be too sophisticated, leading to longer load times and increased resource consumption.

2.3.3 Godot: The Open-Source Challenger

Godot has gained traction as a free, open-source alternative to mainstream engines.

Strengths: With no licensing fees and a growing community, Godot offers a range of tools for 2D and 3D game development. Limitations: Limited Documentation: Being relatively newer, Godot lacks the extensive documentation and tutorials that Unity or Unreal possess. Performance Issues: Some developers have reported performance hitches, especially for more complex games.

2.3.4 GameMaker Studio: The 2D Specialist

While GameMaker Studio is often the go-to for 2D game development, its capabilities extend beyond that.

Strengths: Its drag-and-drop system simplifies game development, making it accessible for novices. Limitations: Licensing Costs: Multiple versions with different price points can confuse and deter potential users. Limited 3D Capabilities: While primarily a 2D engine, its 3D capabilities are rudimentary compared to other engines.

2.3.5 Limitations: The Common Thread

While each game engine has its unique strengths and weaknesses, some common limitations emerge when considering the development of a specialized game like a Match Three title:

Overhead and Bloat: General-purpose engines can introduce unnecessary complexities for straightforward genres. Costs: Licensing fees can deter indie developers or those on a tight budget. Customizability: While these engines are adaptable, creating a game that deviates from standard templates can be challenging.

In summation, while existing game engines offer a plethora of tools and capabilities, there exists a niche for specialized engines tailored for specific genres. Recognizing the limitations of mainstream engines sets the stage for the development of our multiplatform Match Three game engine, which aims to address these gaps while providing a streamlined, optimized development experience.

Chapter 3

Basics of Game Engine Design

3.1 Core Components of Game Engines

3.1 Core Components of Game Engines

Game engines serve as the backbone for game development, providing an array of tools and features that streamline the process of creating a game from scratch. But at the heart of every game engine lie its core components, which dictate its capabilities, flexibility, and performance. This section delves deep into these foundational elements, shedding light on their roles and significance in the vast ecosystem of game development.

3.1.1 Rendering Engine

The rendering engine is responsible for all visual aspects within a game. It takes the game's data – including models, textures, and shaders – and converts them into pixels on the screen.

Role: Ensures all visual elements, from static backgrounds to dynamic character animations, are displayed with clarity, smoothness, and consistency. **Key Features:** Supports various rendering techniques such as ray tracing, rasterization, and shadow mapping to enhance visual fidelity.

3.1.2 Physics Engine

A game's realism often hinges on its ability to mimic the real world's physical interactions. Here, the physics engine plays a pivotal role.

Role: Handles the simulation of physical systems. It determines how entities move, interact, and react to forces and collisions. **Key Features:** Simulates gravity, collisions, rigid and soft body dynamics, and other physical phenomena.

3.1.3 Sound Engine

An immersive gaming experience is not just visual; it's auditory. The sound engine is the component responsible for this auditory dimension.

Role: Manages the playback of sound effects, background music, and voice-overs, adjusting them based on game events and player actions. **Key Features:** Supports 3D sound positioning, sound attenuation, and pitch shifting, among other audio effects.

3.1.4 Input Handling

Games are interactive by nature, and this interaction is mediated by the engine's input handling system.

Role: Processes user inputs from various sources, be it a keyboard, mouse, gamepad, or touch screen. Translates these inputs into in-game actions or commands. **Key Features:** Detects multiple simultaneous key presses, supports touch gestures, and allows for input remapping.

3.1.5 AI Module

While not all games leverage AI, those that do rely on the AI module to simulate intelligent behaviors in non-player characters (NPCs) or to optimize gameplay.

Role: Drives the behavior of NPCs, ensuring they react realistically and dynamically to in-game situations. **Key Features:** Pathfinding algorithms, decision trees, and neural networks to simulate diverse and adaptive NPC behaviors.

3.1.6 Networking

In the age of online gaming and multiplayer experiences, the networking component of game engines has gained paramount importance.

Role: Manages all online interactions, from player-to-player communication to updating game states across different devices. Key Features: Supports real-time data transmission, handles lag compensation, and manages player matchmaking.

3.1.7 Scripting System

The scripting system allows developers to define game logic, character behaviors, and event responses without delving deep into engine code.

Role: Provides an interface for developers to input custom game logic, ensuring flexibility and adaptability in game design. Key Features: Support for popular scripting languages, real-time code modification, and debugging tools.

In essence, the core components of a game engine are akin to the various systems of the human body. Each plays a distinct yet interconnected role, ensuring the game runs seamlessly, offers an immersive experience, and responds aptly to user interactions. Understanding these components is crucial as they form the foundation upon which our multiplatform Match Three game engine will be built, optimized, and refined.

3.2 The Role of OpenGL in Game Development

3.3 Multiplatform Considerations

Chapter 4

Toolchain and Libraries Overview

4.1 Introduction to GLFW

4.2 Role of GLAD in OpenGL Loading

4.3 Graphics Rendering with FreeType and stb_image

4.4 Introduction to ft2build and Text Rendering

Chapter 5

Design and Architecture of the Match Three Engine

5.1 Game Loop and State Management

5.2 Asset Management and Rendering

5.3 Event Handling and User Input

Chapter 6

Cross-Platform Development Challenges and Solutions

6.1 Operating System Variabilities

6.2 Addressing Hardware and Driver Differences

6.3 Testing and Quality Assurance Across Platforms

Chapter 7

Implementation Details

7.1 Core Engine Implementation

7.2 Match Three Logic and Mechanics

7.3 Integration of Libraries and Tools

Chapter 8

Performance Optimization and Benchmarking

8.1 Performance Metrics and KPIs

8.2 Optimization Techniques Employed

8.3 Benchmark Results Across Platforms

Chapter 9

Case Study: Development of a Prototype Match Three Game

9.1 Game Concept and Design

9.2 Application of the Engine in Real-world Development

9.3 Feedback and Iterative Improvement

Chapter 10

Future Work and Enhancements

10.1 Potential Extensions to the Engine

10.2 Integrating Newer Technologies and Libraries

10.3 Broader Applicability to Other Game Genres

Chapter 11

Conclusion

11.1 Summary of Achievements

11.2 Contributions to the Field of Game Development

11.3 Reflection on the Development Process

Chapter 12

References

Chapter 13

Appendices