**Project: Neon Cyber Rush**

**Built with Java Swing • Designed by Bushra Ahmed**

**Overview**

**Neon Cyber Rush** is a glowing, retro-style brick breaker game built from scratch using Java Swing. Inspired by the classic arcade era, this project blends object-oriented programming with visual design and smooth gameplay to create a desktop experience that’s both fun and technically polished.

Every component — from the paddle physics and power-ups to the animated trail effects and level progression — was carefully implemented to showcase not only programming skills but also an eye for UI and interactivity. It’s designed to feel like a complete game, not just a student project.

**Why I Built It**

As a computer science student, I wanted to create a project that went beyond simple form inputs or textbook examples. I set out to build something playable, visually engaging, and complete — something I could proudly add to my portfolio and GitHub.

This game was a chance to apply real programming practices: user input handling, collision detection, rendering loops, event-driven behavior, file persistence, and modular architecture. But it was also a way to let my creativity show through design.

**Key Features**

* **Custom animated start screen** with neon-styled title and button
* **Glowing paddle** with animated gradients and real-time movement
* **Ball trail effects** using an in-game particle system
* **Multi-level gameplay** — brick grid grows tougher with each win
* **Brick types**: regular single-hit and advanced two-hit bricks
* **Random power-ups**: expand paddle, slow ball, and more
* **Game over & win screens** with glowing fonts and restart support
* **High score system** saved to disk for session persistence
* **Fully object-oriented structure** with reusable components
* **Packaged JAR file** with assets for cross-platform execution

**How I Built It**

1. **Core Game Loop**  
   I started by building the GamePanel class with a timer loop, using javax.swing.Timer to drive animation and input response. This class manages the paddle, ball, brick grid, power-ups, and game state transitions.
2. **Object-Oriented Design**  
   Classes like Paddle, Ball, Brick, AdvancedBrick, and EffectManager were all modular. Each handles its own rendering and behavior. Inheritance is used for variation (e.g. advanced bricks needing 2 hits).
3. **Rendering & Visual Effects**  
   All graphics are drawn using Graphics2D, with gradients, glows, and custom fonts from the Theme class. The ball trail is created using a simple list of coordinates updated over time.
4. **Levels & Game Flow**  
   After clearing a level, the player automatically progresses to the next one — each level adds more brick rows. The player can restart any time using the spacebar, and pause with P.
5. **Power-Ups**  
   Power-ups randomly spawn when bricks are destroyed and apply effects when caught by the paddle. These include paddle expansion and ball speed reduction.
6. **Persistence & Polish**  
   I used Scanner and PrintWriter to store high scores in a local highscore.txt. The game was tested across systems and exported using jar tooling with a proper manifest.txt and organized folder structure (src, assets, out).

**What I Learned**

This project helped me grow as both a developer and designer. I learned how to structure Java projects for scalability, how to manage rendering and input in a real-time loop, and how to debug complex interactions like ball collisions and power-up effects.

It also taught me the importance of **polish** — small visual choices like shadows, spacing, animation timing, and how screens transition make a huge difference in how finished a project feels.

Most importantly, I gained the confidence to build a complete game from scratch, and the discipline to refine it until it looked and felt right.

**Files Included**

* /src — All source code
* /assets — Backgrounds and images
* NeonCyberRush.jar — Runnable game file
* manifest.txt — Required for JAR packaging
* README.md — This file
* ProjectOverview.docx — Additional write-up for resume/portfolio