UI / UX CASE STUDY

# CodeFlux
# Design Case Study

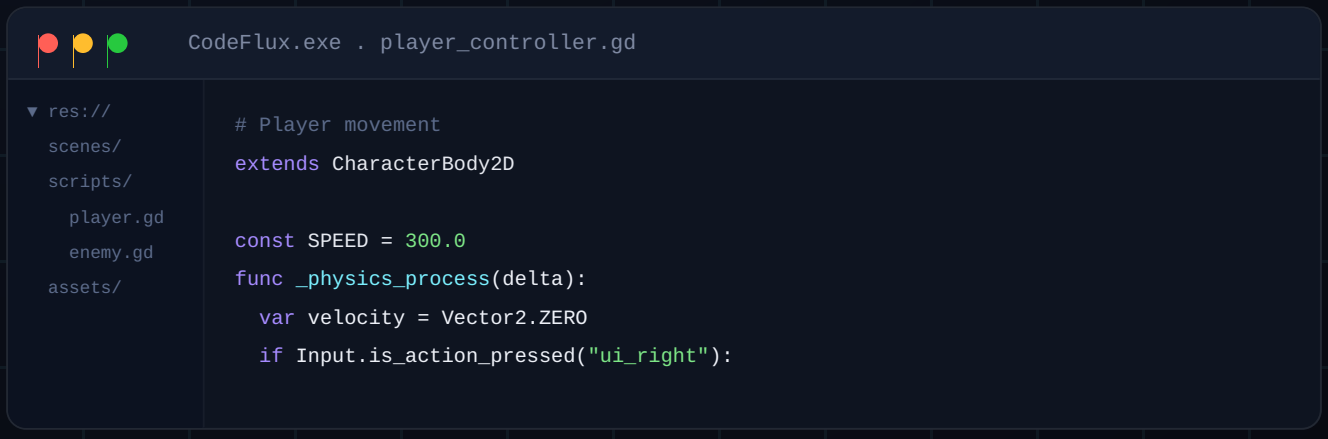
A precision built code editor for the Godot 4 workflow. This case study breaks down the interface decisions, the research behind them, and the reasoning for every major design choice in the product.

GDScript / C#

Godot 4 Integration

Dark Neon System

Desktop App




UI DESIGN, UX RESEARCH AND CASE STUDY BY

**Ali Ahmad Khan**

studio-flux.netlify.app
2026

01 . OVERVIEW

# Project Overview

CodeFlux is a desktop code editor built for game developers working inside the Godot 4 engine. It is one half of Flux Studio, a two app suite from FunLogic Studios that also includes DesignFlux, a companion visual design tool. CodeFlux focuses entirely on scripting in GDScript and C#, with an interface shaped around how Godot developers actually work.

| | |
|---|---|
| PRODUCT | CodeFlux, part of Flux Studio, by FunLogic Studios |
| PLATFORM | Native desktop application for Windows 10 and 11, 64 bit, version 1.0.0 |
| MY ROLE | UI design, UX research, and interaction design for the CodeFlux editor experience |
| USERS | Indie and studio developers scripting gameplay logic for Godot 4 projects |
| CORE TOOLS | Code editor surface, file explorer, FluxAI assistant panel, settings panel |

## Why CodeFlux Exists

General purpose code editors are built for every language and every workflow at once. That breadth adds friction for a developer whose entire day is spent inside one engine. CodeFlux removes that overhead by narrowing the tool to exactly two languages and one engine, so every panel, shortcut, and default is already tuned to the task at hand.

> Design goal: give a Godot developer an editor that opens fast, highlights what matters for GDScript and C#, and never asks them to configure their way into a workflow that should already be the default.

## Scope of This Case Study

This document covers the research that shaped the product, the visual and interaction system used across the editor, a breakdown of the core features and the reasoning behind each, and the overall outcome of the design.

02 . UX RESEARCH

# Understanding the Developer

Before any interface work began, the research focused on one question. What slows a Godot developer down when they are using a general editor for GDScript or C#, and what would a purpose built alternative need to fix.

## Key Observations

- **Context switching cost.** Developers moving between the Godot editor and a separate code editor lose track of node names, signal names, and scene structure, which leads to typos and repeated lookups.
- **Generic autocomplete.** Standard editors do not know about Godot's node tree or signal system, so suggestions are limited to language syntax rather than the actual project.
- **Visual fatigue.** Long scripting sessions in bright or high contrast themes cause eye strain, especially during late development crunch periods.
- **Tool fragmentation.** Developers often juggle a separate linter, snippet manager, and reference material, none of which are aware of each other.

## Design Implications

### Reduce Lookup Friction

Bring Godot 4 node and signal awareness directly into autocomplete, so the editor understands the project rather than only the language.

### Protect Focus

Use a low glare, high contrast dark theme by default, paired with a single editor centric layout with no unnecessary chrome.

## Defining Success

Success was defined as a developer being able to open a script, get contextually correct suggestions, catch errors before running the project, and export a clean file back into their Godot pipeline, all without leaving CodeFlux or adjusting default settings.

03 . DESIGN SYSTEM

# Visual and Interaction System

The interface uses a dark neon visual system shared across both CodeFlux and DesignFlux, so switching between the two tools feels like one continuous product rather than two separate apps.

## Why a Dark Neon Theme

- **Reduced eye strain.** A dark base with muted surface tones keeps long scripting sessions comfortable, which research showed was a real pain point for developers working late.
- **Higher code contrast.** Neon accent tones for keywords, functions, and strings sit clearly above a dark background, so scanning a script for structure is faster than on a light background.
- **Cultural fit.** Game development and editor tooling both lean toward dark, technical aesthetics, so the theme signals the right audience immediately.

## Layout Structure

The editor uses a three zone layout: a file explorer on the left for navigation, the code surface as the dominant central area, and a right side panel reserved for the FluxAI assistant. A status bar at the bottom keeps cursor position visible at all times without competing for attention.

> The layout deliberately keeps the code surface as the largest region on screen. Every supporting panel is collapsible or secondary, because the script itself, not the tooling around it, is what the developer is there to see.

## Typography and Color

A monospace typeface is used throughout the editor to preserve code alignment and readability. Syntax colors are limited to a small, consistent palette, purple for keywords, cyan for functions, and green for strings, so the eye learns the pattern quickly instead of parsing a wide, noisy color range.

04 . FEATURE BREAKDOWN

# What Is In the Product, and Why

Every feature in CodeFlux maps back to a specific friction point uncovered in research. The table below outlines the core building blocks of the editor and the reasoning behind each one.

| FEATURE | WHY IT WAS DESIGNED THIS WAY |
| --- | --- |
| **Language Switch** | A single top right dropdown toggles between GDScript and C#, so developers working across both languages never need a separate window or file type prompt. |
| **Godot 4 Autocomplete** | Node and signal aware suggestions were built directly into the editor because generic autocomplete was the single biggest lookup cost identified in research. |
| **Script Linter** | Errors surface while typing rather than at run time, which keeps developers inside their flow instead of bouncing back and forth to the Godot console. |
| **Snippet Library** | Common patterns like player movement or signal connections are one shortcut away, reducing repetitive typing for logic developers write on nearly every project. |
| **FluxAI Panel** | Placed as a persistent side panel rather than a popup, so a developer can ask a GDScript or Godot question without losing their place in the script. |
| **File Explorer** | A lightweight side panel mirrors the project folder structure, letting developers jump between related scripts without leaving the app. |
| **Editor Settings** | Font size, tab size, word wrap, and line numbers are exposed in one panel with sensible defaults, so most developers never need to open it at all. |
| **Export to .gd / .cs** | A single export step writes a clean file straight into the format Godot expects, keeping CodeFlux a companion tool rather than a replacement for the engine. |

05 . INTERACTION FLOW

# From Install to First Script

Onboarding was designed to get a developer from download to a working script in as few decisions as possible, since setup friction is often where a new tool gets abandoned.

1. Run the installer and follow the on screen setup prompts.
2. Open the install folder or use the desktop shortcut created automatically.
3. Launch CodeFlux directly from the desktop shortcut.
4. Point CodeFlux to an existing Godot 4 project folder on first launch.
5. Start scripting immediately with defaults already tuned for Godot work.

## Working Across the Suite

CodeFlux and DesignFlux share a common project format, so assets created in DesignFlux and scripts written in CodeFlux stay linked to the same project without manual file handling. A developer can use CodeFlux entirely on its own, or move between both tools as their project grows to include UI and asset work.

## Designing for Zero Cost Entry

CodeFlux is free with no account, subscription, or paywall. This shaped an important interface decision: there is no onboarding screen selling features or gating functionality behind a signup flow. The first screen a developer sees is the editor itself.

# Outcome

The final product is a focused, Godot native code editor that trades general purpose flexibility for depth in one workflow. Every interface decision, from the dark neon theme to the persistent FluxAI panel, ties back to a specific need surfaced during research rather than a stylistic default.

## What Worked

- The three zone layout kept the code surface dominant while still giving fast access to files and AI assistance.
- Godot aware autocomplete removed the most common source of friction identified during research.
- A shared visual language with DesignFlux made the two tools feel like one coherent studio rather than separate downloads.

## What I Would Explore Next

- Deeper customization of syntax color mapping for developers who prefer alternate accent palettes.
- Expanded snippet categories driven by real usage data once the product has a larger install base.

## Closing Note

CodeFlux is a case study in narrowing scope with intent. By designing for one engine and two languages instead of every possible workflow, the interface can make stronger, more confident decisions on behalf of the developer using it.