

NEON STRIKERS

2D Platformer Brawler

Comprehensive Project Documentation

Senior Capstone Project Florida International University

Development Team

Louis Flores — Technical Lead & Documentation

Brian Mesa Developer Lead

Fabienne Olibrice — Developer

Florida International University

Spring 2026

1. Executive Summary

Neon Strikers is a 2D platformer brawler developed in Unity for the Senior Capstone course at Florida International University. The game places players in a neon-lit arena environment where they compete in fast-paced combat using movement, platforming mechanics, and attack combos. The project was developed across multiple agile sprints by a three-person team and culminated in a public showcase demonstration.

This document provides complete technical and process documentation for the Neon Strikers project, covering everything from initial problem definition and system design to implementation details, testing, and lessons learned. It is intended to be thorough enough that another team could understand the system fully and continue development from where this team left off.

2. Problem Definition

2.1 Background and Motivation

The capstone project brief called for teams to design, develop, and demonstrate an original software product that showcases skills acquired throughout the Computer Science program. The team selected a 2D game because it offered rich opportunities to apply object-oriented design, physics simulation, state machine architecture, and user interface design all within a single cohesive product.

2D platformer brawlers represent a well-understood genre with clear success metrics: tight controls, readable game states, satisfying feedback loops, and competitive replayability. These constraints gave the team a concrete design target while leaving ample room for original creative and technical contributions.

2.2 Problem Statement

The core challenge was to design and implement a polished, playable 2D multiplayer brawler game from scratch using Unity, delivering a product that was stable enough for a live public demonstration and interesting enough to engage participants unfamiliar with the project.

2.3 Project Scope

The following features were in scope for the initial release:

- 2D platformer movement with responsive controls
- Combat system with basic attack combos
- Health and damage mechanics
- Arena-style level design with platform layouts
- Game state management (start screen, gameplay, game over)
- Neon visual aesthetic with particle effects
- Local multiplayer support

The following items were considered out of scope for the initial release but noted for future work:

- Online multiplayer / networking
- Character customization or unlock systems
- A campaign or story mode
- Mobile or console platform ports

2.4 Target Users

The primary users are casual to semi-competitive gamers familiar with the platformer brawler genre. A secondary audience includes other developers who may want to review or extend the codebase. The public showcase also exposed the game to players with no prior game background, which surfaced important usability feedback.

3. Team Structure and Project Management

3.1 Team Composition

The Neon Strikers development team consisted of the following members across the life of the project:

Name	Role	Responsibilities	Status
Louis Flores	Technical Lead & Documentation	All technical documentation, architecture, sprint records, testing artifacts	Active — Full Semester
Brian Mesa	Developer Lead	Core game mechanics, Unity implementation, physics, level design	Active — Full Semester
Fabienne Olibrice	Developer	Supporting development, UI elements, testing	Active — Full Semester
Mike Niedz	Original Project Lead / Design	Initial project vision, design direction, team coordination	Departed mid-semester

3.2 Agile Development Process

The team followed an agile Scrum-inspired workflow. Development was organized into sprints of approximately two weeks each, with the following recurring ceremonies:

- Sprint Planning — defining sprint goals and assigning tasks
- Daily Scrums — brief synchronous or asynchronous standups covering what was done, what is planned, and any blockers
- Sprint Reviews — demonstrating completed work at the end of each sprint
- Sprint Retrospectives — reflecting on process, identifying improvements for the next sprint
- Meeting Minutes — recorded and maintained by Louis Flores for all team sessions

3.3 Tools and Platforms

Tool	Purpose
Unity	Primary game engine and development environment
GitHub	Version control, issue tracking, bug list management
Discord	Team communication, async standups
Google Docs / Drive	Shared document drafting and sprint artifacts
Word / DOCX	Formal deliverable documentation

4. System Design

4.1 Architecture Overview

Neon Strikers is built on Unity's component-based architecture. Game objects represent entities in the scene, and behavior is attached via C# MonoBehaviour scripts. The game is structured around the following primary systems:

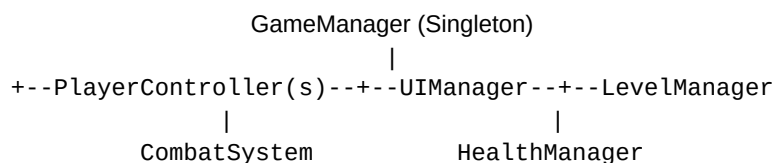
- Game Manager — singleton managing global state, scene transitions, and game flow
- Player Controller — input handling, movement physics, and animation state
- Combat System — attack detection, hitbox/hurtbox management, and damage application
- Health Manager — per-entity health tracking, death events, and respawn logic
- Level Manager — platform layout, spawn point management, and environmental hazards
- UI Manager — HUD elements including health bars, timers, and score display

4.2 Architecture Diagram

[Architecture Diagram Placeholder — see Figure 4.2.1]

The diagram below illustrates the high-level dependency relationships between the major systems. Arrows indicate data flow or event subscriptions. All systems communicate through a central event bus where appropriate to reduce tight coupling.

[Figure 4.2.1: System Architecture Diagram]



4.3 Player Controller Design

The PlayerController script is the most complex component in the codebase. It manages:

- Horizontal movement with configurable speed and acceleration
- Jumping with variable height based on hold duration
- Coyote time — a brief grace period allowing jumping slightly after leaving a platform
- Jump buffering — queuing a jump input while still airborne
- Wall sliding and wall jumping
- Attack state machine with light, heavy, and aerial attacks
- Animation state transitions driven by velocity and action flags

4.4 Combat System Design

Combat in Neon Strikers uses a hitbox/hurtbox model. Each character has a set of collider-based hitboxes that activate during attack animations. When a hitbox overlaps an opponent's hurtbox, the CombatSystem resolves the interaction:

1. The attacking PlayerController triggers an attack animation.
2. The animation fires an event at the designated frame to enable the hitbox collider.
3. If an overlap with a valid hurtbox is detected, CombatSystem.ResolveHit() is called.
4. Damage, knockback direction, and hitstun frames are applied to the target.
5. HealthManager deducts the damage value and checks for knockout condition.
6. The hitbox collider is disabled at the end of the attack window.

4.5 Game State Machine

The GameManager maintains a finite state machine governing the game lifecycle:

State	Description	Transitions To
MainMenu	Start screen with title, player prompts, and settings access	CharacterSelect, Options
CharacterSelect	Players choose characters before a match (if implemented)	Gameplay
Gameplay	Active match in progress; physics, input, and combat all active	Paused, RoundEnd
Paused	All gameplay frozen; menu overlay displayed	Gameplay, MainMenu
RoundEnd	Short delay after a knockout; displays result overlay	Gameplay (next round), MainMenu
GameOver	Final result screen after all rounds completed	MainMenu

4.6 Level Design

The primary arena is designed as a single-screen layout with multiple platform tiers. Design principles followed:

- Vertical variation — platforms at different heights reward aerial combat
- No dead ends — all platform positions offer an exit route
- Hazard zones — blast zones at screen edges for KO conditions
- Visual clarity — platforms are clearly silhouetted against the neon background
- Spawn symmetry — player spawns are mirrored to ensure fairness at match start

5. Implementation Details

5.1 Development Environment

Component	Version / Detail
Engine	Unity 2022 LTS (2D mode)
Language	C# (.NET Standard 2.1)
IDE	Visual Studio / VS Code with Unity extension
Version Control	Git via GitHub
Target Platform	Windows PC (standalone build)
Physics	Unity 2D Physics (Rigidbody2D, Collider2D)
Rendering	Universal Render Pipeline (URP) 2D

5.2 Project Structure

The Unity project follows a standard folder organization:

Assets/

- Scripts/ — All C# MonoBehaviour and utility scripts
 - Player/ — PlayerController, InputHandler, AnimationController
 - Combat/ — CombatSystem, Hitbox, Hurtbox, DamageData
 - Managers/ — GameManager, UIManager, LevelManager, HealthManager
- Scenes/ — MainMenu, Arena01, GameOver scenes
- Sprites/ — Character sprites, tilesets, UI elements
- Animations/ — Animator controllers, animation clips
- Audio/ — Sound effects, background music
- Prefabs/ — Player, projectiles, UI overlays

5.3 Key Scripts

5.3.1 PlayerController.cs

Handles all player input, movement physics, and state. Key fields include:

- moveSpeed: float — base horizontal movement speed
- jumpForce: float — vertical impulse applied on jump
- coyoteTime: float — grace period for late jumps (default 0.15s)
- jumpBufferTime: float — input buffer window (default 0.1s)
- attackDamage: float — base damage value per light attack

5.3.2 CombatSystem.cs

Manages the resolution of combat interactions. Implements a layer-based collision filter so that hitboxes only register against opposing team hurtboxes, preventing self-damage.

5.3.3 GameManager.cs

A persistent singleton using DontDestroyOnLoad. Manages the state machine, round counters, score tracking, and scene transitions. Exposes static events that other systems subscribe to rather than holding direct references.

5.3.4 HealthManager.cs

Attached to every damageable entity. Tracks current health, exposes TakeDamage(float) and Heal(float) methods, and fires OnDeath and OnHealthChanged events consumed by UIManager and GameManager respectively.

5.4 Physics Configuration

Several physics values were tuned iteratively during development and showcase testing. The following table reflects the final shipped values:

Parameter	Value	Notes
Player Gravity Scale	3.5	Higher than Unity default for snappier feel
Jump Force	14.0	Allows approximately 3 platform heights clearance
Move Speed	8.0	Balanced for arena width
Max Fall Speed	-20.0	Terminal velocity cap prevents tunneling
Friction (platform)	0.0	Zero friction; deceleration handled in script
Attack Hitstun (light)	0.15s	Short but noticeable stagger
Knockback Force	6.0	Scales with damage percentage in future builds

6. Sprint History and Progress

6.1 Sprint Overview

Development proceeded across five primary sprints. The following provides a condensed record of goals, completions, and notable events per sprint.

Sprint	Focus	Key Deliverables	Status
Sprint 1	Foundation	Project setup, Unity scene, basic player movement, GitHub repo initialized	Complete
Sprint 2	Core Mechanics	Jump physics, platformer feel, initial combat hitboxes, basic health system	Complete
Sprint 3	Game Loop	Game state machine, round management, HUD, arena layout v1	Complete
Sprint 4	Polish & Features	Visual effects, neon aesthetic, audio integration, bug fixes from internal testing	Complete
Sprint 5	Showcase Prep	Stability hardening, build delivery, documentation finalization, showcase presentation	Complete

6.2 Notable Mid-Project Changes

During Sprint 3, the team experienced a significant structural change when the original project lead, Mike Niedz, departed from the project due to scheduling conflicts with the presentation requirements. This transition is documented in detail in the accompanying Shortcomings and Challenges document. In summary:

- Brian Mesa assumed the role of Developer Lead, taking primary ownership of the Unity implementation.
- Louis Flores assumed full responsibility for all technical documentation, sprint records, and meeting minutes.
- The team re-evaluated and reorganized the project roadmap during Sprint 3 retrospective to account for reduced capacity.
- Certain scope items were deferred to the wishlist, and the team refocused on delivering a stable, playable core experience.

7. Testing and Evaluation

7.1 Testing Strategy

Testing for Neon Strikers spanned informal internal testing during development, peer testing sessions, and a live public showcase. The three primary testing modes were:

- Developer Testing — ongoing manual testing by team members during active development
- Internal Peer Testing — structured sessions where team members tested each other's implementations against defined acceptance criteria
- Showcase Testing — uncontrolled live testing by members of the public and fellow students at the capstone showcase event

7.2 Test Cases

The following functional test cases were defined and verified prior to the showcase:

Test ID	Description	Expected Result	Outcome
TC-01	Player can move left and right on flat ground	Smooth horizontal movement at correct speed	Pass
TC-02	Player can jump from ground	Character leaves ground and reaches expected apex	Pass
TC-03	Jump does not trigger in mid-air (no double jump)	Second jump press ignored while airborne	Pass
TC-04	Coyote time allows late jump off platform edge	Jump registers within 0.15s of leaving edge	Pass
TC-05	Light attack deals damage to opponent	Target health decreases by correct amount	Pass
TC-06	Attack does not damage self	Hitbox filtered; no self-damage	Pass
TC-07	Player death triggers round end	RoundEnd state entered; overlay displayed	Pass
TC-08	Score increments correctly after round	Winning player score increases by 1	Pass
TC-09	Game returns to main menu after final round	GameOver state entered; main menu loads	Pass
TC-10	Both players can play simultaneously	No input conflict; independent character control	Pass

7.3 Showcase Testing Observations

The capstone showcase provided the most valuable real-world testing data. Participants unfamiliar with the game were invited to play, and team members observed interactions and noted issues in real time. Key observations:

- Many participants were unfamiliar with the controls and needed verbal instruction — indicating a need for an in-game tutorial or control display overlay.
- Several bugs emerged under conditions that internal testing had not covered, including edge cases with simultaneous attack inputs and platform boundary interactions.
- Physics values that felt correct during development felt floaty or overly aggressive to some participants — pointing to a need for configurable physics presets or tuning options.
- The lack of a clear on-screen control mapping confused first-time players repeatedly.
- Multiplayer sessions during the showcase surfaced synchronization-adjacent issues in input response that were not observed in isolated testing.

A full list of bugs identified during and after the showcase has been tracked on the project GitHub repository under the Issues tab. The Wishlist and Future Work document details a proposed structured testing event to systematically address these.

7.4 Known Bugs at Time of Submission

The following bugs were identified and documented on GitHub but not resolved prior to the final submission:

- BUG-01: Player occasionally clips through thin platforms at high fall speeds
- BUG-02: Hitbox briefly active for one extra frame after attack animation ends, causing late hit registration
- BUG-03: Health bar UI does not animate smoothly on rapid successive hits
- BUG-04: Simultaneous death of both players causes undefined round result
- BUG-05: Wall jump direction occasionally inverted when pressing jump at the exact frame of wall contact
- BUG-06: Game over screen sometimes fails to load on first trigger (second trigger succeeds)
- BUG-07: Audio continues briefly after scene transitions due to missing AudioSource cleanup

All bugs are logged with reproduction steps on the GitHub repository and are candidates for the proposed Live Testing Event described in the Wishlist document.

8. Results

8.1 Final Deliverables

The following deliverables were completed and submitted as part of the capstone project:

- A playable standalone Windows build of Neon Strikers
- Full source code hosted on GitHub with commit history
- Sprint review, retrospective, and meeting minutes documentation for all sprints
- This comprehensive project documentation package
- Shortcomings and Challenges document
- Wishlist and Future Work document

8.2 Showcase Reception

The game was demonstrated live at the capstone showcase to faculty, fellow students, and guests. Reception was generally positive, with participants enjoying the fast-paced combat and neon visual style. The showcase validated the core game loop and confirmed that the project delivered a functional, playable product. It also surfaced the bug list and usability feedback that will drive future improvement work.

8.3 Performance Metrics

The final build maintained a consistent 60 frames per second on standard development hardware with no observed hitching during normal gameplay. Build size was within acceptable range for a prototype-scale Unity project.

9. Lessons Learned

9.1 Technical Lessons

- Physics tuning is highly iterative and should be done with real playtesters, not just developers, as early as possible.
- Hitbox timing requires careful frame-level control; a single frame of extra hitbox activity can cause significant gameplay feel issues.
- Coyote time and jump buffering are essential for a polished platformer feel and should be implemented from the beginning, not added later.
- Unity's event system and singleton GameManager pattern worked well at this scale but would require refactoring for a larger project.

9.2 Process Lessons

- Losing a team member mid-project has outsized impact on momentum and morale. A transition plan should be part of every project's risk registry.
- Design ownership needs to be explicit and distributed — when design direction lives in one person's head and that person leaves, the team faces both a productivity loss and a creative direction vacuum.
- Public showcase testing is irreplaceable for surfacing bugs. Internal testing cannot replicate the variety of inputs and behaviors that real users introduce.
- Meeting minutes and sprint documentation pay dividends when team structure changes — the records allowed the remaining team to reconstruct intent and priorities quickly.

9.3 Team Lessons

- Clear role definitions enable better division of responsibility but also create risk if a role becomes vacant. Roles should have at least partial redundancy.
- Frequent communication cadence (daily scrums) was effective for staying aligned on a distributed async schedule.
- Agile retrospectives helped the team surface and address friction points before they became blocking issues.

10. Guide for Future Teams

10.1 Getting Started

To continue development of Neon Strikers, a new team should take the following steps:

7. Clone the repository from GitHub and open the project in Unity 2022 LTS.
8. Review this documentation package in full, paying particular attention to the architecture overview (Section 4) and the physics configuration table (Section 5.4).
9. Review the GitHub Issues tab for the current bug backlog.
10. Play the game for a minimum of one hour to build intuition for the current game feel before making changes.
11. Read the Wishlist document to understand the intended direction for future work.

10.2 Recommended First Priorities

- Resolve BUG-04 (simultaneous death undefined behavior) as it is a critical game logic bug.
- Implement an in-game control display overlay to reduce onboarding friction for new players.
- Add a formal automated test suite using Unity Test Framework for the core systems.
- Conduct the Live Testing Event described in the Wishlist document and use results to prioritize bug fixes.

10.3 Code Conventions

- All scripts use PascalCase for class names and methods, camelCase for private fields.
- Public fields use [SerializeField] for Inspector exposure rather than direct public access.
- All events follow the C# EventHandler<T> pattern.
- Scene management must go through GameManager.LoadScene() — never call SceneManager.LoadScene() directly.

11. Appendix

11.1 Glossary

Term	Definition
Hitbox	A collision volume that, when active, can deal damage to overlapping hurtboxes
Hurtbox	A collision volume representing a character's vulnerable area
Coyote Time	A grace period allowing a jump to register shortly after walking off a platform edge
Jump Buffer	A window during which a jump input is remembered and fires when the player lands
Hitstun	A brief period after taking damage during which a character cannot act
KO / Knockout	Condition where a player's health reaches zero, ending the current round
Blast Zone	Invisible boundary at the screen edges; exiting it counts as a KO in some configurations
DontDestroyOnLoad	Unity lifecycle method that persists a GameObject across scene loads
URP	Universal Render Pipeline — Unity's scriptable render pipeline used in this project
MonoBehaviour	Unity's base class for all component scripts attached to GameObjects

11.2 References

- Unity Documentation: <https://docs.unity3d.com>
- Unity 2D Physics Reference: <https://docs.unity3d.com/Manual/Physics2DReference.html>
- Neon Strikers GitHub Repository: www.github.com/l0fl0/neon-strikers

11.3 Version History

Version	Date	Author	Changes
1.0	Spring 2026	Louis Flores	Initial complete documentation draft
1.1	Spring 2026	Louis Flores	Added showcase testing observations and bug list
1.2	Spring 2026	Louis Flores	Finalized all sections for submission