Project Documentation: Doodle Jump

# Desing Philosophy and Structure

The structure is based on the Model-View-Controller (MVC) desing pattern:

- Model: World, Player, BGTile, Score, and Camera encapsulate game logic, state and mechanics.  
 - View: GameView renders all visuals based on model updates via observer interfaces.  
 - Controller: Game manages the main loop and ties together input, model updates, and rendering.

By separating these responsibilities, the codebase remains readable, extendable, and testable. This modularity allows for future features such as pause menus, sound, or new tile types with minimal impact on existing components.

# Object-oriented Desing Principles

The project adheres to key SOLID principles:  
   
- Single Responsibility Principle: Each class has one job—Score tracks score logic, Camera normalizes coordinates, Player handles movement, etc.  
 - Open/Closed Principle: Via polymorphism (e.g. BGTile, BonusTile), new tile types can be added without modifying base classes.  
 - Liskov Substitution Principle: All derived tiles override virtual methods correctly, and can be used wherever a BGTile\* is expected.  
 - Interface Segregation: Observer interfaces (ScoreObserver, CameraObserver, etc.) are narrowly defined and easy to implement.  
 - Dependency Inversion: World depends on the abstract BGTileFactory, allowing flexibility in how tiles are constructed.  
   
Smart pointers (std::unique\_ptr and std::shared\_ptr) are used consistently to manage dynamic memory safely, eliminating leaks and dangling pointers. Observer patterns are implemented using weak\_ptr to prevent circular dependencies.

# Code Quality & Maintainability

- All classes and functions follow consistent naming conventions and are cleanly separated into .cpp and .h files.  
 - A .clang-format configuration ensures automatic and consistent formatting across all source files.  
 - [[nodiscard]], noexcept, inline, const and constexpr are used where appropriate to improve safety, performance and intent clarity.  
 - Each method is short, purposeful and includes concise API documentation (Doxygen style).  
 - The World::update() method has been split into five logical parts, increasing readability and making the code easier to debug and test.

# Error Handling & Robustness

- All textures and fonts are loaded with robust error checking using exceptions. Any failure is clearly reported and stops execution to avoid undefined behavior.  
 - Camera game-over logic is cleanly handled via screen-relative checks.  
 - Spawning and collision logic accounts for screen bounds and ensures tiles are placed meaningfully.  
   
All this contributes to a stable game loop with minimal runtime issues.

# Clarity, Extensibility and Testing

The game logic is designed with extensibility in mind:  
   
- Adding new platform types only requires a new class and optional entry in the factory.  
 - Observers decouple visual rendering from logic, enabling future implementations..  
   
To aid testing and debugging:  
 - Logic is separated from rendering.  
 - Camera and score observers can be replaced with mocks.  
 - Console logging is available in error scenarios (e.g., texture load failures).