**Allgemeine Konventionen**

1. **Projektstruktur:** Strukturieren Sie das Projekt logisch in Packages, zum Beispiel: **game.controllers**, **game.entities**, **game.scenes**, um die verschiedenen Aspekte des Spiels zu organisieren.
2. **Namensgebung:** Verwenden Sie aussagekräftige und konsistente Namen für Klassen, Methoden und Variablen. CamelCase für Methoden und Variablen, PascalCase für Klassen.
3. **Kommentare:** Kommentieren Sie komplexe Algorithmen, ungewöhnliche Lösungen oder Bereiche, die eine zusätzliche Erklärung erfordern, um anderen Entwicklern die Codeverständlichkeit zu erleichtern.

**Spielkomponenten**

1. **Entity-Namen:** Benennen Sie Entities (Spielobjekte) nach ihrem Zweck oder ihrer Funktion im Spiel, um ihre Verwendung zu verdeutlichen, z. B. **PlayerEntity**, **EnemyEntity**.
2. **Komponenten-Klassen:** Strukturieren Sie die Spielobjekte in Komponenten, wie **PlayerControlComponent**, **CollisionComponent**, um die Verantwortlichkeiten klar zu trennen.

**FXGL-spezifische Konventionen**

1. **GameApplication-Klasse:** Nutzen Sie eine eigene Klasse, die von **GameApplication** erbt, um die Initialisierung und Konfiguration des Spiels zu handhaben. Diese sollte die **initGame**- und **initUI**-Methoden enthalten.
2. **Config-Dateien:** Trennen Sie die Konfigurationseinstellungen, wie Fenstergröße, Ressourcenpfade usw., in separaten Dateien, um eine bessere Lesbarkeit zu gewährleisten.

**Event-Handling**

1. **Verwendung von FXGL-Events:** Nutzen Sie die eingebauten FXGL-Events, um auf Spielsituationen zu reagieren, z. B. CollisionEvents für Kollisionen oder TimeEvents für zeitbasierte Aktionen.
2. **Event-Handler:** Strukturieren Sie Event-Handler konsistent und benennen Sie sie präzise, um das verarbeitete Ereignis und die Aktion zu beschreiben.

**Komponententests**

1. **Unit-Tests:** Schreiben Sie Tests für einzelne Komponenten, um sicherzustellen, dass sie wie erwartet funktionieren. Verwenden Sie dafür gängige Testframeworks wie JUnit.
2. **Mocking:** Verwenden Sie Mocking-Frameworks wie Mockito, um externe Abhängigkeiten zu isolieren und effektiv Unit-Tests durchzuführen.

**Dokumentation**

1. **JavaDoc:** Verwenden Sie JavaDoc, um Klassen, Methoden und Variablen zu dokumentieren und so automatisch generierte API-Dokumentation zu erzeugen.
2. **Readme-Datei:** Erstellen Sie eine Readme-Datei für das Projekt, die Informationen zur Installation, Verwendung und Konfiguration enthält, um anderen Entwicklern den Einstieg zu erleichtern.

**Versionskontrolle**

1. **Git-Flow:** Verwenden Sie einen branchbasierten Ansatz wie Git-Flow für eine organisierte und effiziente Zusammenarbeit im Team.
2. **Aussagekräftige Commits:** Verwenden Sie aussagekräftige Commit-Nachrichten, die klar beschreiben, welche Änderungen vorgenommen wurden.