

**Praxisprojekt im Rahmen von  
Advanced Software Engineering**

**Technische Dokumentation zum Programmentwurf**

**Weeping Snake**

Duale Hochschule Baden-Württemberg Karlsruhe  
Fakultät Technik

Studiengang Informatik – Informatik

von

Silas Mario Schnurr

20.05.2021



**Kurs:** TINF18B5

**Dozent:**  Herr Maurice Müller

**Erklärung zur Eigenleistung**

Gemäß § 5 (3) der „Studien- und Prüfungsordnung DHBW Technik“ vom 22. September 2011.

Ich habe die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

----------------------------------- --------------------------

Ort Datum Unterschrift

**Inhaltsverzeichnis**

[Abkürzungsverzeichnis / Glossar IV](#_Toc72307016)

[Abbildungs- und Tabellenverzeichnis V](#_Toc72307017)

[1 Projektbeschreibung 1](#_Toc72307018)

[1.1 Funktionsumfang 1](#_Toc72307019)

[1.2 Code-Struktur 1](#_Toc72307020)

[2 Domain Driven Design 2](#_Toc72307021)

[2.1 Ubiquitous Language 2](#_Toc72307022)

[2.2 Analyse ?? 2](#_Toc72307023)

[3 Programming Principles 3](#_Toc72307024)

[3.1 SOLID 3](#_Toc72307025)

[3.2 GRASP 3](#_Toc72307026)

[3.3 DRY 3](#_Toc72307027)

[4 Entwurfsmuster 4](#_Toc72307028)

[5 Clean Architecture 5](#_Toc72307029)

[6 Legacy Code 6](#_Toc72307030)

[7 Refactoring 7](#_Toc72307031)

[8 Unit Tests 8](#_Toc72307032)

[8.1 Konzept 8](#_Toc72307033)

[8.2 ATRIP-Regeln 8](#_Toc72307034)

[8.3 Code Coverage 8](#_Toc72307035)

[9 Zusatz: API-Design 9](#_Toc72307036)

[9.1 API 9](#_Toc72307037)

[9.2 Analyse 9](#_Toc72307038)

# Abkürzungsverzeichnis / Glossar

**API** Application Programming Interface, Programmierschnittstelle

**UL** Ubiquitous Language

# Abbildungs- und Tabellenverzeichnis

**Es konnten keine Einträge für ein Abbildungsverzeichnis gefunden werden.**

**Es konnten keine Einträge für ein Abbildungsverzeichnis gefunden werden.**

# Projektbeschreibung

Das ist die schriftliche Dokumentation. Es wird die Objektorientierte Mainstream-Programmiersprache C# mit dem .Net 5.0 Framework verwendet.

enthält Ergebnisse der Analysen und Begründungen der Entscheidungen • Code-Struktur als UML-Diagramme notiert – Detailgrad an die Aussagen anpassen

Nötige Software:

apt-get install -y nuget mono-devel mono-xbuild

## Funktionsumfang

Der klar definierte sinnvolle Nutzen dieser Programmbibliothek ist…

* Spielen, gegner rammen (5 runden lang) gibt punkte
* Ältere spuren sind hindernisse / existieren nicht
* Aktionen vorbestimmen (links, links, rechts, rechts) -> eine aktion pro runde

Bestandteile: Backend des Spiels als Programmbibliothek (incl API), eine seperate Web-API, ein offline console client, der über die Programmbiblithek einzelspieler Spiele gegen Bots erlaubt.

## Code-Struktur

2k Zeilen Code > 20 Klassen

Code-Struktur ist als UML-Diagramm notiert – Detailgrad an die Aussagen anpassen

|  |  |  |
| --- | --- | --- |
| **Projekt** | **Funktion** | **Zeilen von Quellcode** |
| WeepingSnake.Game |  | 2200 |
| WeepingSnake.Game.Tests |  | 600 |
| WeepingSnake.WebService |  | 200 |
| WeepingSnake.ConsoleClient |  | 200 |
| WeepingSnake.ConsoleClient.Tests |  | 200 |
|  |  |  |

## Kompilieren, testen & ausführen

Das Programm kann begrenzt mit Docker ausgeführt werden, zum Erstellen & Testen und Ausführen mit Interkation ist dotnet (Lauffähig unter allen typischen Systemen) nötig.

Im Folgenden ist ein Auszug aus der README-Datei des Repositories, in welcher dieser Prozess beschrieben ist.

**Weeping Snake**

  

**Getting Started**

1. Make sure that dotnet is installed on your machine. You need it to build, test and run the code
   * To **install dotnet** for example with pacman you can use sudo pacman -S dotnet-sdk
   * It is also possible to use **Docker** via the included [Dockerfile](https://github.com/SilasDerProfi/weeping-snake/blob/main/src/WeepingSnake.Game/Dockerfile) to run the program, but no interaction is possible (e.g. controlling your own player)
2. You need the source code. Just load it via the git clone command
   * E.g. git clone https://github.com/SilasDerProfi/weeping-snake.git
   * Obviously you can specify a specific directory for the clone

**Build**

With every commit the code is compiled automatically. You can see if the build was successful by the badge in this readme.

To build the code, you must run dotnet build src in the directory of your clone

**Test**

With every commit the code is tested automatically. You can see if the test were successful by the badge in this readme.

To build the code, you must run dotnet test src in the directory of your clone

**Run**

To run the code, you need to run dotnet run --project src/WeepingSnake.ConsoleClient in the directory of your clone

# Domain Driven Design

## Ubiquitous Language

Die Betrachtung der Ubiquitous Language (UL) erfolgt anhand der folgenden Gestaltung der Problemdomäne.

**Accountverwaltung**

(Generische Domäne)

**Highscore liste**

(Unterstützende Domäne)

Die Einteilung in die drei Subdomänen dient der Eingrenzung der jeweiligen zu betrachtenden Problemdomäne.

Die Kerndomäne „Spiel“ beinhaltet die eigentliche Durchführung eines Spiels, an welchem Spieler teilnehmen. In der Benutzeroberfläche sowie im Quellcode werden daher für diese Subdomäne spezielle Begriffe verwendet. Diese Begriffe sind vor allem auch von Begriffen der anderen Subdomänen abzugrenzen.

|  |  |
| --- | --- |
| Begriff | Beschreibung / Zusammenhang / Abgrenzung |
| Game | Eine Instanz eines Spieles, an welchem Teilnehmer gegeneinander spielen. |
| Player | Eine Spielfigur, welche von einem Menschen oder einem Computer auf dem *Board* gesteuert wird, mit dem Ziel das Spiel zu gewinnen.   * Zusammenhang mit: *Action*, *PlayerAction* und *Board* * Abgrenzung zu: *Person* * Ist Bestandteil von: *Game* |
| Board | Die Fläche auf welcher ein Spiel stattfindet. Spieler bewegen sich auf dieser Fläche.   * Ist Bestandteil von: *Game* |
| Points | Eine Bewertung innerhalb eines *Game*s, um den Erfolg der *Player* zu messen.   * Ist Bestandteil von: *Player* |
| Action | Eine mögliche Aktion, welche ein *Player* in einem *Game* tätigen kann. (turn left, turn right, change nothing, speed up, slow down, jump)   * Abzugrenzen zu: *PlayerAction* * Ist bestandteil von: *Game* |
| PlayerAction | Eine von einem *Player* getätigte *Action*, welche in der Zukunft ausgeführt wird oder bereits ausgeführt wurde.   * Abzugrenzen zu: *Action* * Ist bestandteil von: *Player* |

Die Subdomäne „Accountverwaltung“ ist eine generische Subdomäne, da diese neben dem eigentlichen Spiel essenziell für den Mehrspielermodus ist. Ohne diese Accountverwaltung ist eine Steuerung des Mehrspielermodus nicht möglich. Im Rahmen der Accountverwaltung ist im Rahmen der UL die Definition und Abgrenzung der folgenden Begriffe erforderlich.

|  |  |
| --- | --- |
| Begriff | Beschreibung / Zusammenhang / Abgrenzung |
| Person | Ein Mensch, welcher einen Account besitzt. Er kann an mehreren Games mit jeweils einem Player teilnehmen.   * Abgrenzung zu: *Player* |
| Username | Ein von der *Person* gewähltes Pseudonym, welches in der Highscore-Liste verwendet wird. |

Die Subdomäne „Highscore-Liste“ umfasst die statistische Übersicht der errungenen *Points* der *Person*s. Während eines *Game*s wissen die *Person*s nicht, von wem die gegnerischen *Player* gesteuert werden. Daher werden diese Informationen über die Highscore-Liste zusammengeführt.

Bounded Context…

## Repositories

Analyse und Begründung

## Aggregates

Analyse und Begründung

## Entities

Analyse und Begründung

## Value Objects

Analyse und Begründung

# Programming Principles

## SOLID

Analyse und Begründung

## GRASP

Analyse und Begründung

(insbes. Kopplung und Kohäsion)

## DRY

Analyse und Begründung

# Entwurfsmuster

* >= 1 Entwurfsmuster einsetzen und begründen
* UML (vorher/)nachher

# Clean Architecture

* >= 1 Umsetzung einer Klasse der Adapterschicht
* Begründung

# Legacy Code

* >= 2 Abhängigkeiten mit Techniken aus der Vorlesung brechen
* ausgewählte Stellen und Techniken begründen

# Refactoring

* Code Smells identifizieren
* >= 2 Refactorings anwenden und begründen
* für 2 Stelle vorher/nacher UML

# Unit Tests

## Konzept

* >= 10 Unit Tests
* Einsatz von Fake-/MockObjekten (ohne Frameworks wie z.B. Mockito)
* Analyse und Begründung für
  + Umfang der Tests
  + Einsatz der Fake/MockObjekte

## ATRIP-Regeln

## Code Coverage

# Zusatz: API-Design

Freiwillig bei b5

## API

* Bereitstellung einer sinnvollen API
  + Programmiersprachenebene oder http
  + >= 10 Methoden / HTTP Endpunkte

## Analyse

* Analyse der API anhand der Qualitätsmerkmale und Begründung für das Des