

**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

22.05.2021



**Kurs:** TINF18B5

**Dozent:**  Herr Maurice Müller

**Inhaltsverzeichnis**

[Abkürzungsverzeichnis / Glossar III](#_Toc72574793)

[Abbildungs- und Tabellenverzeichnis IV](#_Toc72574794)

[1 Projektbeschreibung 1](#_Toc72574795)

[1.1 Funktionsumfang 2](#_Toc72574796)

[1.2 Code-Struktur 3](#_Toc72574797)

[1.2.1 LOC 3](#_Toc72574798)

[1.3 Kompilieren, testen & ausführen 4](#_Toc72574799)

[2 Domain Driven Design 5](#_Toc72574800)

[2.1 Ubiquitous Language 5](#_Toc72574801)

[2.2 Repositories 7](#_Toc72574802)

[2.3 Aggregates 7](#_Toc72574803)

[2.4 Entities 7](#_Toc72574804)

[2.5 Value Objects 8](#_Toc72574805)

[3 Programming Principles 9](#_Toc72574806)

[3.1 SOLID 9](#_Toc72574807)

[3.2 GRASP 9](#_Toc72574808)

[3.3 DRY 9](#_Toc72574809)

[4 Entwurfsmuster 10](#_Toc72574810)

[5 Clean Architecture 11](#_Toc72574811)

[6 Legacy Code 12](#_Toc72574812)

[7 Refactoring 13](#_Toc72574813)

[8 Unit Tests 14](#_Toc72574814)

[8.1 Konzept 14](#_Toc72574815)

[8.2 ATRIP-Regeln 14](#_Toc72574816)

[8.3 Code Coverage 14](#_Toc72574817)

[9 Zusatz: API-Design 15](#_Toc72574818)

[9.1 API 15](#_Toc72574819)

[9.2 Analyse 15](#_Toc72574820)

# Abkürzungsverzeichnis / Glossar

**API** Application Programming Interface, Programmierschnittstelle

**UL** Ubiquitous Language

**LOC** Lines of Code, die genaue Anzahl von Zeilen im Quellcode. Niedrigere Werte sind besser.

**CLI** Command-line interface, Kommandozeile

# Abbildungs- und Tabellenverzeichnis

[Abbildung 1: Problemdomäne 4](#_Toc72570785)

[Tabelle 1: LOC vor Refactorings 3](#_Toc72574647)

[Tabelle 2: Getting Started 4](#_Toc72574648)

[Tabelle 3: Ubiquitous Language für die Kerndomäne 6](#_Toc72574649)

[Tabelle 4: Ubiquitous Language für die generische Domäne 7](#_Toc72574650)

# Projektbeschreibung

Das vorliegende Dokument ist die schriftliche technische Dokumentation zu dem Programmentwurf *Weeping Snake* im Rahmen von Advanced Software Engineering an der DHBW Karlsruhe.

Es wird die Programmiersprache C# mit dem .Net 5.0 Framework verwendet. Im Folgenden werden verschiedene Analysen durchgeführt, deren Ergebnisse aufgezeigt sowie daraus resultierende Änderungen begründet. Die Arbeit ist dabei nach verschiedenen Problembereichen untergliedert, welche getrennt voneinander betrachtet werden können.

Bei dem Spiel *Weeping Snake* steuern die Teilnehmer einen *Player* auf einem begrenzten Spielfeld (*Board*). Die Player scheiden aus dem Spiel aus, wenn sie das Spielfeld verlassen. Ziel des Spiels ist es, möglichst viele Punkte zu sammeln. Hierfür bekommen die Teilnehmer Punkte, wenn sie mit ihrem *Player* den *Player* eines gegnerischen Spielers schneiden. Um dies zu ermöglichen, wird die Navigation durch eine Richtungsänderung, Geschwindigkeitsänderungen und Sprünge ermöglicht (Aktionen können über mehrere Runden hinweg vorausgeplant werden). In dem Fall, dass ein *Player* sich selbst schneidet oder geschnitten wird, verliert er Punkte. Die Länge der „Schlange“, welche den *Player* repräsentiert, ist von der Geschwindigkeit abhängig, da der Pfad der letzten 5 Runden für Schnitte relevant ist.

## Funktionsumfang

Der Funktionsumfang des Projekts geht über das eigentliche Spiel hinaus. Neben dem Backend des Spiels, welches als Programmbibliothek (mit API) implementiert wird, existiert eine separate Web-API sowie ein offline Client (CLI), der über die Programmbibliothek Einzelspieler Spiele gegen Bots (mehr oder weniger klug) erlaubt.

Über den CLI-Client ist es möglich, das Spiel zu spielen. Hierbei gibt es die Möglichkeit, als Gast zu spielen oder mit einem Benutzeraccount. Durch die Verwendung eines Benutzeraccounts besteht der Vorteil, dass die Errungenschaften in einer Highscore liste angezeigt werden. Dieser Benutzeraccount lässt sich verwalten, sodass die E-Mail-Adresse oder das Passwort geändert werden können. Die eigentliche Durchführung des Spiels erfolgt ebenfalls in einer Konsole. Hierfür müssen die Aktionen über die Tastatur eingegeben werden. Dieser CLI-Client dient jedoch nur der Demonstration der Grundfunktionalitäten. Das Backend ist so aufgebaut, dass die Spielparameter konfigurierbar sind (Über eine Konfigurationsdatei) und dadurch z. B. Richtungsänderungen nicht immer im 90° Radius erfolgen müssen. Dadurch können sich Spieler nicht nur vertikal und horizontal, sondern in jede beliebige Richtung, abhängig von dem gewählten Radius bewegen. Ein weiterer Bestandteil ist das Logging, welches es ermöglicht Spielinformationen in einer \*.log Datei zu persistieren. Dieses Logging ist standardmäßig deaktiviert und kann über die Datei weepingsnake.config mit dem Eintrag IsLoggingEnabled=true aktiviert werden.

## Code-Struktur

Die 27 Klassen (inklusive geschachtelter Klassen und klassenähnlicher Konstrukte[[1]](#footnote-1)) von dem *Game*-Backend lassen sich in sechs Aufgabenbereiche unterteilen. In dem folgenden UML-Klassen-Diagramm sind die Einteilung der Klassen sowie deren Beziehungen aufgeführt. Der Detailgrad des UML-Klassen-Diagramms fällt in dieser Übersicht jedoch geringer aus, da die grobe Struktur verdeutlicht werden soll. Zusätzlich zu den sechs Aufgabenbereichen gibt es einige unterstützende Klassen, welche unter Utility zusammengefasst werden können. Diese werden an mehreren Stellen verwendet, sodass eine explizite Darstellung der Assoziationen zu Unübersichtlichkeit führen würde.



Abbildung : Code-Struktur in UML Notation

### LOC

Die Anzahl der Quellcodezeilen (LOC – Lines of Code) wird von der GitHub Codemetrik mit ca. 4.000 angegeben. Die strengere Analyse mithilfe der IDE Visual Studio Enterprise 2019 ergibt die folgenden Ergebnisse für die Berechnung der Codemetrik der relevanten Projekte.

|  |  |  |
| --- | --- | --- |
| **Projekt** | **Funktion** | **LOC** |
| Game  WeepingSnake.Game | Backend für die Verwaltung und Durchführung von *Game*s | 2219 |
| WebService  WeepingSnake.WebService | Schnittstelle (REST) für den Zugriff über das Internet | 290 |
| ConsoleClient  WeepingSnake.ConsoleClient | Offline-Client (CLI), um im Einzelspielermodus auf das *Game*-Backend zuzugreifen | 526 |

Tabelle : LOC vor Refactorings

Die Angabe der LOC bezieht sich auf den Quellcodestand vor der Umsetzung der Refactorings, welche in diesem Dokument beschrieben werden. Dieser Quellcodestand ist auf dem Main-Branch als Tag „1.0“ einsehbar ([github.com/SilasDerProfi/weeping-snake/tree/1.0](https://github.com/SilasDerProfi/weeping-snake/tree/1.0)).

## 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.

|  |
| --- |
| **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 |

Tabelle : Getting Started

# 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)

Abbildung : Problemdomäne

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* |

Tabelle : Ubiquitous Language für die Kerndomäne

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. |

Tabelle : Ubiquitous Language für die generische Domäne

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

1. struct in C#: Wie eine Klasse, mit gewissen Einschränkungen (hat immer Standardwerte / ist nie null und kann nicht erben oder vererbt werden) [↑](#footnote-ref-1)