Python Dokumentation

**V Modell:**

Architektur:

* Vorüberlegungen über Vorraussetzungen des Projektes
* Planung der Dateistruktur mithilfe eines Klassendiagrammes gemeinsam
* Besprechen der Übergabe und Rückgabeparameter
* Richtlinien festlegen für Kommentare, Variablen-/ Funktionsbenennung
* Zuordnung der einzelnen Komponenten zu den Personen

Komponenten Spezifikation:

* Einzeln Spezifizieren und ausarbeiten anhand der ausgearbeiteten Richtlinien
* Bei Unklarheiten Rücksprache halten
* Überprüfen ob gestellte Anforderungen so umsetzbar
* Ständige Absprache über geänderte Parameter
* (Einbindung verschiedener Libraries)

Implementierung:

* Erneutes planen, wenn beim Programmieren auffällt, das etwas nicht so funktioniert, wie gedacht
* Räumliches Zusammentreffen, um Austausch zu verbessern
* Testen der Funktionen nach Fertigstellung im Einzelnen
* Testen des Programmes nach Fertigstellung als Gesamtes

(evtl Entwicklungsbespiel?)

(Entwicklung der Tests noch einfügen und Testabdeckung)

**Klassendiagramm:**

(Einfügen aus Git wenn fertig)

**Aktivitätsdiagramme:**

Wir haben für jede Funktion mit mehr als 5 Zeilen ein dazu gehöriges Aktivitätsdiagramm mit draw.io erstellt, dabei haben wir immer die Funktionen eines Teilprogramms auch in einem Diagramm festgehalten. Aus Gründen der Übersichtlichkeit haben wir darauf verzichtet die Diagramme alle einzeln in die Dokumentation ein zu binden. Die Diagramme finden sich in dem Order mit dem Programmcode, gleichnamig wie die dazu gehörigen Teilprogramme.

Öffnen lassen sich die Diagramme in der Desktopversion von draw.io oder aus dem Browser mit der Option „Vorhandenes Diagramm öffnen“.

Die Funktionen, welche wir nicht als Aktivitätsdiagramm gezeichnet haben, weisen nicht die nötige Komplexität auf um mit einem Aktivitätsdiagramm etwas zu veranschaulichen.

**Software:**

Verwendete Bibliotheken:

Random: Wir haben die Random Bibliothek eingebunden, um es dem Computer zu ermöglichen auf ein beliebiges Feld zu schießen und seine Schiffe beliebig zu platzieren.

Unittests: Die Unittests haben wir eingebunden um Unittests ausführen zu können

termcolor/colorama: Mit diesen Bibliotheken haben wir die Ausgaben/Boards farblich gestaltet

json: Json beutzen wir, um einen aktuellen Spielstand zwischenspeichern zu können, sollte das Programm mitten im Spiel geschlossen werden und ggf. einen gespeicherten Spielstand weiter spielen zu können.

os: Die OS Bibliothek verwenden wir um den Bildschirm zu leeren, damit man nicht sieht, wo der andere Spieler seine Schiffe platziert hat. Dazu benutzen wir die OS Bibliothek zur Ermittlung des aktuellen Pfades

**Spiel**

Beschreibung Dateiformat zur Speicherung:

(json beschreiben)

Beschreibung des Nutzerinterfaces:

(einfach Konsole?)

Beschreibung der CPU

Der Computer verteilt, sofern der ein Spieler Modus gewählt wurde, die Schiffe zufällig im Feld. Wenn der Computer an der Reihe ist, schießt er auf ein zufälliges Feld. Sollte dieses schon beschossen worden sein, dann schießt der Computer erneut auf ein zufälliges Feld.

Wenn der Computer einen Treffer landet, dann wird das Feld gespeichert und in eine Himmelsrichtung weiter geschossen, sollte ein Wassertreffer folgen wird in eine andere Richtung weiter geschossen. Wenn die Richtung herausgefunden wurde, wird diese gespeichert, bis entweder das Schiff versenkt wurde oder ein Wassertreffer erzielt wird.

Wird das Schiff versenkt, so wird das Feld, welches gespeichert war, sowie die Richtung gelöscht. Wenn ein Wassertreffer nach herausfinden der Richtung, aber vor versenken des Schiffes erzielt, wird die Richtung um 180° gedreht und ab dem gespeicherten ersten Trefferfeld aus weiter geschossen. Der Computer hat keine zusätzliche Intelligenz, um nicht auf die Felder direkt neben einem versenkten Schiff zu schießen.

Dokumentation Spielablauf:

Einzelspieler:

Zweispieler

**Unittests:**

(Logs Tests/Coverage)

**Fehlersicherheit:**

(Wie wurde mit Fehlern umgegangen, wie abgefangen)

**(Pylint)**gehört maybe zu Fehlersciherheit?