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:**

(auch in draw.io einsehbar, name: classDiagram.drawio)

**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. Also um zufällige Zahlen zu ermitteln an welche Stelle im Board etwas plaziert/ geschossen werden soll.

Unittests: Die Unittest-Bibliothek haben wir eingebunden um Unittests ausführen zu können.

termcolor/colorama: Mit diesen Bibliotheken haben wir die Ausgaben und 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, indem man diesen aus der Json-Datei lädt.

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, auf dem das Programm ausgeführt wird.

**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 lige Himmelsrichtung weiter geschossen, sollte ein Wassertreffer folgen wird in die entgegengesetzte Richtung weiter geschossen. Falls in dieser Richtung auch ein direkt ein Wassertreffer fällt, wird die Dimension gewechselt und die in die zwei noch verbleibenden Richtungen geschossen. Wenn alle Richtungen ausprobiert wurden, wird das Schiff vom Spieler versenkt sein. Die noch auszuprobierenden Richtungen werden im Programm von der shootingIq vorgegeben. Diese wird nach einem Wassertreffer um eins erhöht und gespeichert, sodass wenn die cpu wieder dran ist eine andere Richtung wählt, in die geschossen wird. Durch eine while Schleife wird solange in eine Himmelsrichtung geschossen, bis ein Wassertreffer fällt. Die erste Schießposition wird mithilfe der random-Bibliothek zufällig bestimmt und auch gespeichert, da man beim wechseln der Himmelsrichtung von der Ursprungskoordinate aus weiterschießen muss. Die erste Himmersrichtung in die geschossen wird wird auch zufällig in der Funktion firstDirection festgelegt und anhand dieser werden die nachfolgenden Richtungen auch angepasst.

Beim ändern der Richtung gibt es Fälle bei denen die cpu aus dem Feld hinaus schießt und somit ans andere Ende des Feldes schießt. Dies wird nicht abgefangen. Auch nachdem das Schiff versenkt wurde, schießt die cpu noch ein weiteres mal in die vorherige Himmelsrichtung, was auch nicht optimal ist, allerdings fehlte uns die nötige Zeit um dies auch noch umzusetzten. Die stärke der cpu reicht allerdings aus, um auch mal gegen den menschlichen Gegener zu gewinnen.

Dokumentation Spielablauf:

Bevor das Spiel wird der Nutzer gefragt, ob er einen vorherigen Spielstand hat, welchen er laden möchte. Sollte dass der Fall sein springt der Nutzer sofort in sein gespeichertes Spiel zurück, welches über eine json-Datei gespeichert und geladen wird, hierfür wird die Funktion loadrequest aus selectOperations verwendet.

Sollte es keinen gespeicherten Spielstand geben, oder will der Nutzer den alten Spielstand nicht mehr fortsetzten, so wird ein neues Spiel gestartet.

Hierfür wird der Nutzer gefragt, ob er im Einspieler Modus oder im Zweispieler Modus spielen möchte. Dies passiert mit Hilfe der selectStartingPlayer Funktion aus selectoperations. Ist eine Wahl getroffen, so startet die jeweilige Sequenz.

Einzelspieler:

Im Einspieler Modus wird erst der Name des Computergegners auf „Der Computer“ gesetzt und der Spieler nach seinem Namen gefragt. Anschließend wird der Spieler aufgefordert seine Schiffe zu platzieren, was in der Funktion classPlaceShip aus shipManager passiert. Der Ablauf des Schiffe platzierens sieht vor, dass man ein Feld für die Spitze des Schiffes mit Hilfe der Tastatur auswählt und anschließend die Richtung in welche das Schiff laufen soll angeben muss(mit w,a,s,d). Hierbei gilt es zu beachten, dass die platzierten Schiffe sich nicht berühren dürfen und auch nicht über das Eck zusammenstoßen dürfen, sollte dass der Fall sein, so muss man das Schiff erneut auf einen freien Platz des Boards platzieren.

Der Computer platziert anschließend heimlich seine Schiffe in classCpuPlaceShip ebenfalls aus der Datei shipManager. Er sucht sich immer zufällige Felder und eine zufällige Richtung aus und überprüft im Anschluss, ob es Regelkonform ist, dass Schiff dort zu platzieren.

Ist die CPU fertig damit ihre Schiffe zu platzieren, dann wird zufällig bestimmt, welcher Spieler das Spiel beginnen darf, dies geschiet in der Funktion selectStartingPLayer.

Anschließend beginnt das tatsächliche Spiel, in shooting aus shootingfunction darf der ausgeloste Startspieler beginnen ein Feld zu beschießen.

Wir nehmen an, dass die CPU beginnen darf.

Die CPU beginnt, indem sie zu beginn auf ein zufällig ausgewähltes Feld schießt. Dieses Feld darf von der CPU zuvor noch nicht beschossen worden sein. Sollte die CPU einen Treffer landen, so setzt sie Logik ein und versucht dass ganze Schiff zu versenken(siehe Beschreibung CPU). Bei einem Teffer darf erneut geschossen werden. Nach jedem Schuss wird das aktualisierte Board auf der Benutzeroberfläche ausgegeben. Trifft die CPU jedoch Wasser, so ist der Spieler an der Reihe, der Wechsel funktioniert über die Funktion nextPlayer aus shootingfunction.

Der Spieler bekommt das board ausgegeben und darf über Tastatureingabe wählen, wohin er schießen möchte. Das aktualisierte Board wird ausgegeben. Bei einem Treffer darf der Spieler erneut schießen, bei einem Wasser Treffer wird erneut die Funktion nextPlayer aus shootingfunction aufgerufen und die CPU ist wieder an der Reihe.

Das ganze wiederholt sich so lange, bis einer der beiden alle Schiffe des anderen zerstört hat. Dies wird im playerManager aus der Datei shootingfunction ermittelt.

Sollte einer der Beiden alle Schiffe des anderen zerstört haben, so wird die Nummer dieses Spielers an das Hauptprogramm zurück gegeben, wo dann die Funktion battleEnd des outputmanagers eine finale Ausgabe tätigt, wer das Spiel gewonnen hat. Danach beendet sich das Programm und man kann es zum wiederholten spielen erneut starten.

Zweispieler

**Unittests:**

(Logs Tests/Coverage)

**Fehlersicherheit:**

(Wie wurde mit Fehlern umgegangen, wie abgefangen)

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