Entwicklungslog

22.07.2020

-Idee ein Entwicklungslog zu erstellen um Prozess zu beschreiben und vor allem Umsetzungs/Feature-Ideen festzuhalten

-Die ersten 2 Klassen Game und MyConsole sind implementiert. User kann mit Game interagieren. Main methode funktioniert gut.

-Früherer Gedanke (im Klassendiagramm bereits vorhanden): Es soll eine Fight Klasse geben, die einen Kampf zwischen dem Player und einem Enemy kontrolliert.

-Das Objekt dieser Klasse soll innerhalb der main von Game erzeugt (und auch wieder zerstört) werden, wenn der Player die „attack <Enemy>“ eingibt
-> Fight hat selbst auch eine main, die ähnlich ist zu der main methode von game. Diese wird innerhalb der main von Game, gleich nach der Instanziierung aufgerufen und kriegt das MyConsole Objekt als Parameter übergeben. Dann gibt es ein Attribut „isFightEnd“ (wieder analog zu „isGameEnd“ von Game), dass sobald einer der Charaktere stirbt (oder sonstige abbruchbedingungen) auf true gesetzt und somit der main-Methodenaufruf in der „zweiten Ebene“ verlassen wird. Falls der Spieler stirbt, kann das abgefangen werden (entweder über den player object verweis in main oder durch einen observer) und darauf reagiert werden (isGameEnd = true)

<u>Der nächste Schritt</u>

Allerdings möchte ich mich noch nicht der Fight Klasse zuwenden, da ich gerne erstmal „chronologisch“ weitermachen möchte.
Beim Erstellen des Game-Objekts sollen aus einer JSON-Datei Informationen zu dem Anfangszustand der verschiedenen Räume und den darin befindlichen Charakteren geholt werden, und mit diesen dann die benötigten Objekte instanziiert und verknüpft werden.
Deshalb beginne ich mit der UML Planung der dafür benötigten Methoden von der Game Klasse.

24.07.2020

Bevor ich die Planung des JSON-Lade Prozesses beschreibe, lege ich mich darauf fest, dass ich die Anwendung über http-server starte und teste, somit ist fetch möglich.
Desweiteren ist mir aufgefallen, dass die Klasse Fight und Game viele Gemeinsamkeiten haben und die Natur der main-Abläufe einer Sequenz ähneln -> deshalb neue abstrakte Klasse Sequence mit der abstrakten Methode main().

So lassen sich auch weitere Sequenzen beschreiben, und „ineinander verschachteln“. Z.b. Im Fight kann der Player sein inventory öffnen -> InventorySequence Objekt wird erzeugt und dessen main aufgerufen bis User irgendetwas benutzt oder es wieder schließt.

Interfaces müssen/sollten eingebaut werden für das geladene JSON Objekt, um die typsicherheit zu garantieren und nicht variablen von den Klassentypen zu haben, da die JSONObjekte (Room properties) sonst die Funktionen auch implementieren sollten.

Bei der Planung der JSON Laden Funktion und der Instanziierung aller geladeten Räume etc. fällt mir auf, dass man vielleicht garkeine gameMap braucht, vielleicht reicht eine statische Variable innerhalb der Room Klasse aus. Aber dies entscheide ich zu einem späteren Zeitpunkt, wenn mehr Logik reinkommt.

-Planung der LoadJSON Aktivität fertiggestellt.
-Hinzufügen von einigen neuen Klassen und Interfaces (vor allem JSON-spezifische)

Mir fällt auf, dass die Planung der Charaktere und die Vererbung ein wenig komisch und wahrsch. noch nicht gut genug ist, dies muss noch angepasst werden in Zukunft.

Nun gilt es das Konzept umzusetzen.

Die Initialisierung eines Spiels soll so ablaufen, dass erst die JSON geladen wird und dann über die verschiedenen Array-Properties iteriert wird, falls eine property aftritt, die ein Array ist, beispielsweise die items property innerhalb eines Raumes, (dass es geht wird über die typsicheren Interfaces der jeweiligen properties sichergestellt), wird wiederum innerhalb des Konstruktors in einem „tiefergelegenen“ (Konstruktor-) Methodenaufruf über den Array iteriert und wieder für jeden Eintrag ein Objekt erstellt und dem Konstruktor die aktuellen für ihn relevanten Daten mitgegeben.
So wird die Initialisierung bzw Instanziierung rekursiv vollführt.