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.

25.07.2020

Da die JSON Datei nun richtig geladen wird und das Spiel auch erfolgreich initialisiert wird, gilt es nun die verschiedenen Eingabe- Optionen abzufangen und je nach Eingabe das gewünschte Verhalten zu triggern.
Überlegung:
Sequenzen sollen eine main haben (wie schon im Klassendiagram angedeutet), sollen die aktion vom Spieler holen (stehen also in Verbindung zur Konsole) und sollen sich selbst „vorstellen“

Beispiel: Raum wird gewechselt -> Raumbeschreibung wird von der Sequenz selber ausgegeben
oder: Kampf beginnt: Kampfsequenz stellt beteiligte vor
In Jeder Sequenzvorstellung sollen die in der jetzigen Sequenz erlaubten Optionen dargestellt werden.
Jede Sequenzinstanz ist selbst verantwortlich den Input vom User über die Konsole zu holen.

Die Verifizierung der Eingabe, das Ausfindig machen eventueller Ziele (bsp: Kampf initiieren) und generell spezifische Methoden erfolgen über einen Controller.
Auch das erstellen einer „Sub“-Sequenz soll über die Controller geschehen, die selber noch auf ihre parent Sequenz verweisen und so Informationen darüber haben, was für Objekte sich gerade in der Sequenz befinden.
Bsp: GameSequence besitzt currentRoom, User möchte Ziel angreifen „attack Gnom“, Controller übernimmt die Verifizierung und muss dann herausfinden, ob in dem characters-Array des

currentRoom ein Enemy Objekt existiert, dessen Name „Gnom“ ist, erst dann soll eine neue Sub-Fight-Sequenz instanziiert werden.

Die Überprüfung der Eingabe kann mit mehreren Ebenen von try/catch Blöcken realisiert werden, was es möglich macht dem User spezifischere Fehlermeldungen mitzuteilen (die – was ich später auch vor hab – in einer enum gespeichert werden sollen)

Planung in Form von Aktivitätsdiagrammen beginnt (mit kleinen tests nebenbei).

Bei der Planung fällt auf, dass die Überprüfung nicht ganz trivial ist:

-User soll a oder attack eingeben können
-Wenn user attack eingibt, muss er ein ziel mit angeben und attack und das ziel sollen mit leerzeichen getrennt sein -> Problem: Namen sollen bzw können auch leerzeichen enthalten

Diese und weitere Schwierigkeiten gibt es, allerdings plane ich es erstmal für den einfachsten Fall, dass der Name keine Leerzeichen enthält.

Eine Unterscheidung zwischen „einfacher Aktion“ und „Aktion mit Ziel“ soll trotzdem vorgenommen werden, allerdings auch für einen „einfachen“ Fall:

[AKTION] [LEERTASTE] [ZIELNAME]

Wobei dann in der nächsten Ebene unterschieden wird, um welche Aktion es sich genau handelt.

Hier denke ich sind Regular Expressions sehr hilfreich.

Neue Erweiterungsidee: Weitere Subklassen von Room, SpawnRoom, in dem neue Enemies spawnen können, LockedRoom, der nur durch einen Schlüssel geöffnet werden kann

Dafür muss das RoomData Interface angepasst werden mit einem typ-Attribut und eventuell einem optionalen Attribut „neededItem“

26.07.2020

Während der Konzeption und Planung der Controller, stellt sich mir die Frage ob ich es nicht hinbekomme nur eine Controller Klasse zu machen und bei der Erstellung innerhalb der main methoden der jeweiligen Sequenz, die commands in den Konstruktor zu geben, um so nicht für jede Sequenz einen eigenen Controller zu haben, da ein Controller eigentlich immer das selbe tun soll, sich nur von den commands unterscheidet, die allerdings immer die selbe logik verfolgen.

Vielleicht ist eine Command Klasse die sich in SimpleCommand (inventory, quit, commands) und TargetCommand(fight with <target>, look at <target>, take the <item>, drop the <item>) aufteilt sinnvoll.

Die Controller Klasse soll doch nur eine Ausprägung haben und einen Commands-Array besitzen. Bei der Instanziierung von einem Controller, sollen aus dem String Array, der von der spezifischen Sequence kommt und alle legalen Commands aus seiner spezifischen Enumeration holt, die Instanzen der Commands erstellt und in den Array gepusht werden.
Dann findet die Validierung, wie sie bis jetzt schon geplant ist (per regex) statt und der erste Entry des zurückkommenden String Arrays (durch .match(regex)) entspricht dann dem commandName.

In meiner ersten Überlegung wollte ich gleich mit der Regex nur die Inputs abfangen, die auch den commandName's der erlaubten Commands entsprechen, allerdings müsste ich dann wenn ich nicht für jede Sequenz einen eigenen Controller haben wollte, anhand der viableCommands die als String von der Sequenz zur Verfügung gestellt werden würden eine relativ aufwendige und fehleranfällige Methode schreiben müssen, die eine regex, erstellt, die für die spezifischen commands angepasst ist, um den Controller dynamisch zu halten.

Dies ist mit der neuen Methodik nicht mehr notwendig. Das Regex wird nurnoch genutzt um zu unterscheiden ob es sich um ein SimpleCommand oder ein TargetCommand handelt.

Dann kann folgend überprüft werden:

Falls der matchArray an der Stelle [2] undefined ist, heisst das, dass es sich um ein SimpleCommand handeln muss, also wird kein target gesucht und einfach der viableCommands Array des Controllers durchsucht nach einem Objekt, dessen commandName dem matchArray index 1 entspricht, und - falls eins gefunden wurde - dessen executeCommand-Methode aufgerufen (ansonsten throw(„Input is invalid“).

Falls der matchArray an der Stelle [2] nicht undefined ist, heisst das, der User hat wahrscheinlich irgendwas in der Form „[wort] [abstand] [wort] ([abstand] [wort] [abstand]…)“ eingegeben.
Minimum anforderung für ein match ist „[buchstabe] [abstand] [buchstabe]“. Wobei die erste Match gruppe (also der eitnrag index 1 im match array) „[buchstabe/wort] [abstand]“ beinhält.
Dann wird erstmal nach dem commandName gesucht (siehe oben).
Falls ein treffer erzielt wird, darf die executeCommand-Methode noch nicht aufgerufen werden, da zuerst noch das CommandTarget ausfindig gemacht werden muss.

Per .find() wird das Objekt aus dem Entities-Array geholt, welcher einfach lokal mit dem spread operator aus dem characters, items und adjacentRooms array zusammengestellt wird.
Falls nichts gefunden wird -> throw(„That is not a legal target!“)

Auch wenn diese Gedanken bis hierhin recht gut enstanden sind und ich für die Probleme die sich mir gestellt haben, relativ schnell Lösungsansätzen gefunden hab, bin ich schon seit 20 Minuten am überlegen, wie ich das Problem löse, dass ich während der Entwicklungszeit nicht weiss, welchen Typ im Endeffekt das Target hat.
Da ich den Controller sehr allgemein halten möchte, und ihm nicht die Logik zumuten will, dass er wissen muss, was er bei welchem input zu tun hat, gibt es ein Problem.
Wenn der Controller die executeCommand Funktionen implementieren würde, dann könnte man erst das Command ausfindig machen und dann gleich den Typ des Targets nach der

Fallunterscheidung wissen und somit nurnoch den relevanten Array durchsuchen.
Die Schwierigkeit bei meiner Herangehensweise ist, dass ein Raum und ein Character beides legale targets sind und somit beide Arrays (characters und rooms) der parentSequence durchsucht werden müssen.
Nur weil die beiden Typen im allgemeinen legale targets sind, heisst es jedoch nicht, dass sie für jeden TargetCommand legale Ziele sind – so soll zum Beispiel ein Room nicht ein legales Ziel von „fight with „ sein.
Eine Lösung wäre, den Commands bei der instanziierung den jeweiligen relevanten Array mitzugeben und dann innerhalb des Command Objekts nochmal das Target in dem Array zu suchen, was allerdings nicht sehr performant wäre.

-Regex angepasst für den dynamischen Fall. (vorher waren die command Namen innerhalb des regex wortwörtlich drin, aber siehe o. warum das ein Problem war).