Portfolio Aufgaben zu Programmieren 2

Vorlesungsbegleitender Übungssatz
Die Übungsaufgaben sind Teil der Prüfungsleistung (Portfolio)

Formales zur Abgabe:

- Die Abgabe erfolgt in Moodle.
- Abgabefrist ist der 30.09.2022
- Es muss eine Archiv-Datei hochgeladen werden.
  (nur folgende Archive sind gestattet: zip, rar, 7z)

- Legen Sie eine **VisualStudio Solution oder ein CMake Projekt** für die Portfolio Lösung an.

- Visual Studio Solutions müssen **ohne Debug und Release Verzeichnisse (ohne build Verzeichnisse)** abgebeben werden.

- Ihre Abgabe muss eine **schriftliche Abgabe** (pdf, word) beinhalten.

  Diese enthält:
  - In welchen Sourcen (Dateien), welche Aufgaben bearbeitet werden.
  - Die schriftliche Bearbeitung der Aufgabe 6

- Andere Arten von Abgaben sind nicht gestattet!

- Optional: Geben sie gebaute Windows binarys mit ab. (Diese müssen in einem separaten Ordner liegen und die schriftliche Abgabe muss auf sie verweisen)

Formales zur Abgabe – Anmerkungen:

- In den Aufgaben werden die Mindestanforderungen beschrieben, es kann darüber hinaus Bonuspunkte geben.

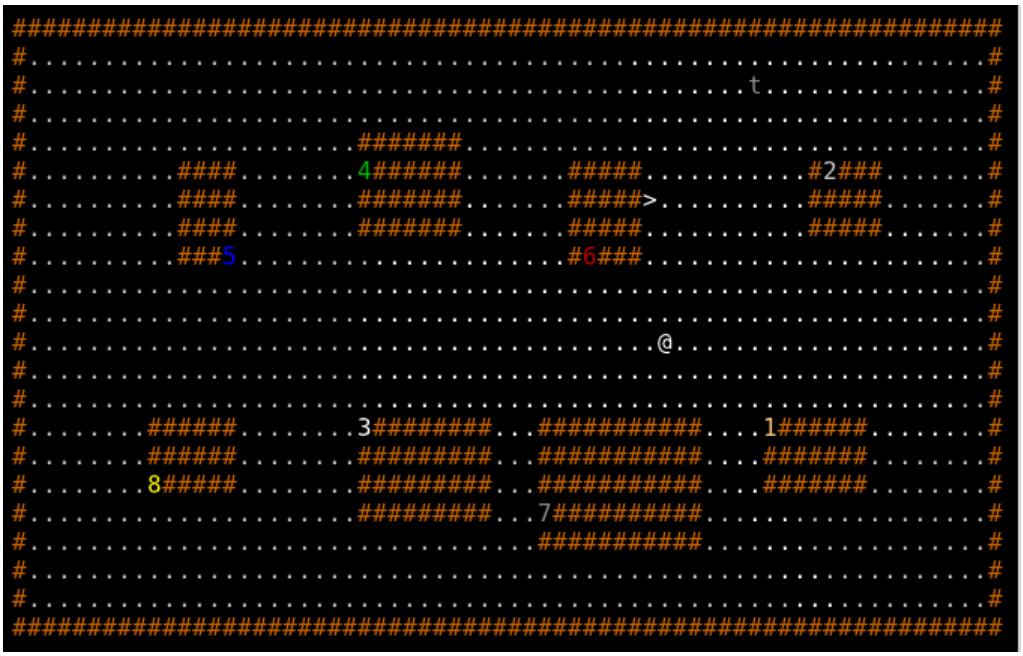
- Beispiele:
  2D-Version mit eigenen Grafiken statt Konsolen-Version. Mehr konkrete Kindklassen. Erweitertes Interface. Mehr Konfigurationsmöglichkeiten/Mehr Attribute.

- Es steht Ihnen frei englische oder deutsche Bezeichner zu verwenden

- Teilweise ist die Verwendung von Bibliotheken erlaubt. Sourcen immer referenzieren.

Basis der Portfolio Aufgaben soll ein rudimentäres Spiel sein.

Sie können ein konsolenbasiertes Spiel erstellen:



Oder ein Spiel mit 2d Assets:

Aufgabe 1a)

Erstellen Sie ein Programm, dass die folgenden Eigenschaften erfüllt:

1. Erstellten Sie ein Programm, dass eine 2d map generiert

   Die map soll die folgenden Eigenschaften haben:

   a. Sie sollen aus einzelnen tiles (tilesets) bestehen.
   b. Sie soll aus min. 15x15 tiles bestehen
      Hier ein Beispiel für eine Konsolen basierte map

      xxxxexxxxxxxbxx
      xxxxxxxxbbxxxxx
      xxxbbbbxxxxxxbx
      xbbxxbbxxxxxxxb
      xxxxxxxxbbxxxbb
      xxxxxbxbxbxxxxx
      bbbbxxxxxxbbbxx
      xxxxxbbxxxxxxbb
      xxxxxxxxbbxxxxx
      xxxxsxxxbbxxxxx

      s = start
      e = exit
      b = blocked (blocked path)
      x = traversable (path)

   c. Es muss einen passierbaren Pfad vom Start zum Ende geben.
   d. Start und Ende müssen erkennbar sein.
   e. Es muss min. 4 Arten von tiles geben
      Start
      Ende
      Blockiert
      Passierbar

Aufgabe 1b)

Erweitern Sie 1a wie folgt:

1. Eine map soll automatisch generiert werden.
   (In 1a könnt ihr dies auch von Hand/statisch machen können)

2. Die Generierung sollte zumindest zum Teil zufällig (und)oder prozedural generiert sein (also eines von beiden, beides oder ein mix).

3. Die map soll weiterhin alles erfüllen, was in 1a gilt.

Aufgabe 1c)

Erweitern Sie 1b wie folgt:

1. Erstellen Sie mindestens einen weiteren tile-Type „treasure chest“

2. Erstellen Sie eine Klassen ItemBase
   a. Diese stellt die Basisklasse für unseren Items dar.
   b. Diese Klasse muss ein Attribut Gewicht haben. Das Gewicht muss größer als 0 sein.
   c. Diese Klasse muss ein Attribut Namen haben.
   d. Diese Klasse muss ein Attribut Beschreibung haben.
   e. Diese Klasse muss ein Attribut Wert (Preis) haben.
   f. Erstellen Sie mindestens eine konkrete item Klasse (die von ItemBase abgeleitet ist)

3. Erweitert die map-Erzeugung so wie folgt:
   a. Auf tiles, die passierbar sind, können Items liegen.
   b. Auf jeder map min. 5 Items liegen (diese können, müssen aber nicht sichtbar sein).
   c. Die map soll weiterhin alles erfüllen, was in 1a und 1b gilt.

Aufgabe 2a)

Erstellt eure inventory Klasse (diese soll in Aufgabe 2b verwendet werden).

Das inventory Klasse muss:

a) eine template basierte Klassendeklaration haben. Der template Parameter soll an den item Container (siehe b) weitergegeben werden.

b) einen generischen Container (template basiert) für das Speichern der Items verwenden.
Es kann sich dabei um einen Container aus einer Bibliothek handeln.
Das inventory soll die ItemBase Klasse aus 1C, bzw. deren Kindklassen verwenden.

c) bei der Erstellung (Instanziierung) festlegen wie viele item Slots sie hat.

d) sicherstellen, dass die maximale Anzahl an Items nicht überschritten wird.

e) Methoden zum Setzen und Abrufen von Items haben (Es muss mindestens eine Methode geben, bei der man den Itemslot angeben muss).

Erweitern Sie die inventory Klasse so, dass Sie 3 dedizierte equipment slots hat.
Für diese Equipment slots gilt:

a) es kann pro slot nur eine ganz spezifische Art von item abgelegt werden.
Bsp. Ringe, Hosen, Waffen,…

b) sie verwenden 3 unterschiedliche Itemklassen - erweitern Sie ihr Programm (basierend auf Aufgabe 1c). Bsp. Slot 1 - 1Ringe , Slot 2 Hosen, Slot 3 Waffen.

c) Mindestanforderung: Die neuen Itemklassen sollen ein Attribut „+Stäke“ haben (mehr dazu in Aufgabe 2b).

Aufgabe 2b)

Erstellen Sie eine Klasse Char

1. Diese stellt die Basisklasse für unseren PC (player character) und NPC (non player character) dar.

    a. Die Klasse soll ein interface definieren, dass eine Traversierung der map (Siehe Aufgabe 1) ermöglicht.

Erstellen Sie eine Klasse PlayerChar

2. Sie muss sich auf der map (Siehe Aufgabe 1) bewegen können.

3. Diese Klasse muss ein inventory besitzen (Siehe Aufgabe 2a).

4. Das inventory muss 10 slots für Items haben (+ eqipment slots).

5. Die Klasse muss (mindestens) ein Attribut Stärke besitzen.

6. Das Attribut Stärke soll das maximale Gewicht limitieren, dass ein player char mit sich führen kann.

    Bsp: 10 str -> max Gewicht = 20kg

7. Wird dieses Tragegewicht überschritten, soll sich die Spielfigur auf der map nicht mehr bewegen lassen.

8. Die Klasse muss ein Interface zum Aufheben und Ablegen von items haben.

9. Die Klasse muss ein Interface zum Ausrüsten von items haben (Verwendung der equipment slots)

10. Wenn ein item in einem inventory slot abgelegt wird, der das „+ Stärke“ Attribut hat (Siehe Aufgbae 2a), soll das Stäke Attribut der PlayerChar Objektes angepasst werden.

Aufgabe 3a)

Erstellen Sie folgende Sortieralgorithmen:

a) Sortieren der Items im inventory nach Gewicht (aufsteigend und/oder abfallend)

b) Sortieren der Items im inventory nach Name (alphabetisch aufsteigend und/oder abfallend)

c) Sortieren der Items im inventory nach Wert (Preis) (aufsteigend und/oder abfallend)

Aufgabe 3b)

Demonstrieren Sie die Funktionsweise Ihrer Sortieralgorithmen aus 3a) an künstlich generierten inventories.

Folgendes muss gelten:

1. Demonstrieren Sie alle Sortierfunktionen aus 3a).

2. Die inventories sollen zwischen 10 und 20 Elementen (Items) besitzen.

3. Die Items müssen so gewählt sein, dass sie die korrekte Funktion des Sortieralgorithmus erkenntlich machen (m.a.W. Wenn nach Gewicht sortiert wird und alle Items haben das Gewicht 1, dann wäre das hier nicht erfüllt)

4. Dem Benutzer muss erkennbar sein (z.B. durch eine Konsolenausgabe) was demonstriert wird. Zudem soll das unsortierte und sortierte inventory ausgegeben/visualisert werden.

Aufgabe 4a)

Erstellen bzw. integrieren (d.h. selbst programmieren oder eine vorhandene Implementierung einbauen) Sie einen der folgenden pathfinding Algorithmen, zur Bestimmung des kürzesten Pfades vom Start- zum Endpunkt der map (Basierend auf Aufgabe 1 und 2):

a) A*-Algorithmus
b) Dijkstra-Algorithmus

Schreiben Sie ein Demo-Programm, welches den kürzesten Pfad für eine nach Aufgabe 1b generierte map berechnet und dem Benutzer das Ergebnis ausgibt.

Die Verwendung von bestehenden Implementierungen (Bibliotheken) ist erlaubt.

Aufgabe 5a)

Diese Aufgabe verknüpft die Aufgaben 1-4! Somit gelten auch alle Beschränkungen, die dort definiert wurden. Traversierbare tiles, Gewichtsbeschränkung, inventory/Itembeschränkungen…

1. Erstellen Sie ein Programm, das einen PlayerChar (Siehe Aufgabe 2) auf dem Start tile einer map spawnt.

2. Zudem muss die map Generierung zufällig/proczedural sein (Anforderung identisch zu Aufgabe 1b).

3. Jede map soll min 3 unterschiedliche Items enthalten. Min 1 item muss ausrüstbar sein.

4. Das Programm kann rundenbasiert oder in Echtzeit sein.

5. Geben Sie dem Benutzer die Möglichkeit die map von Hand zu traversieren.

6. Geben Sie dem Benutzer die Möglichkeit die map mittels eines pathfinding Algorithmus automatisiert traversieren zu lassen.

7. Ziel ist es das Ende der map zu erreichen.

8. Der Benutzer soll die Möglichkeit haben sich sein inventory sortieren zu lassen (auf mindestens 2 Arten – aufsteigend/absteigend nach „X“ gilt als 2 Arten)

Traversierung von Hand:

1. Visualisieren Sie die Spielfigur auf der map.

2. Wenn ein Benutzer auf ein tile mit einem item läuft, soll er die Möglichkeit haben dieses aufzuheben und oder auszurüsten.

3. Der Benutzer soll jederzeit Zugriff auf sein inventory haben.

Traversierung mittels pathfinding Algorithmus:

1. Visualisieren Sie den Pfad, den die Spielfigur genommen hat.
   optionale Alternative - Bonuspunkt:
   Visualisieren Sie die automatische Traversierung der map durch die Spielfigur.

2. Wenn die Spielfigur ein tile traversiert (traversiert hat), welches ein item enthält, soll dieses automatisch aufgehoben werden.
   Dies gilt so lange:
   a. freie item slots verfügbar sind.
   b. die Spielfigur genug Stärke hat.

3. Wenn die Spielfigur ein tile traversiert, welches ein item enthält, welches ihm einen Stärke Bonus gibt (der besser ist, als der den er derzeit bekommt – bezogen auf einen spezifischen equipment slot), soll er dieses ausrüsten.

4. Dem Benutzer soll am Ende noch die Möglichkeit haben sich sein inventory anzeigen und sortieren (Punkt 8 Aufgabe 5) zu lassen.

Aufgabe 6

a) Bestimmen Sie für folgenden Algorithmus, wie oft jede Zeile in Abhängigkeit von N aufgerufen wird (N ist die Eingabemenge)
Betrachten Sie den worst-case

Algorithmus	Kosten	Anzahl
void selSort(int array[], int size) {	c_1	1
for (int step = 0; step < size - 1; step++) {//closes c7	c_2	n
int min_idx = step;	c_3	n-1
for (int i = step + 1; i < size; i++) { //closes c6	c_4	
if (array[i] < array[min_idx]){	c_5	
min_idx = i;}}	c_6	
swap(&array[min_idx], &array[step]);}}	c_7	n-1
	c_8	
	c_9	
	c_{10}	
	c_{11}	
	c_{12}	
	c_{13}	
	c_{14}	
	c_{15}	

b) Leiten Sie aus a die O(n) Notation für diesen Algorithmus ab, *begründen Sie ihre Antwort!*

$O(N^2)$

Aufgabe 7

Aufgabenteil Software-Design Pattern

Entwurfsmuster (*design patterns*) sind allgemeine Lösungsvorlagen für wiederkehrende Fragestellungen die beim Entwurf und der Entwicklung von Software auftreten.

Die klassischen Entwurfsmuster wurden durch das Autorenquartett Gang of Four (GoF) erstmals strukturiert beschrieben (sogenannte GoF Pattern) und in nachfolgende Kategorien eingeteilt:

- **Erzeugungsmuster (*Creational Patterns*)**
- **Strukturmuster (*Structural Patterns*)**
- **Verhaltensmuster (*Behavioral Patterns*)**
- **Muster für objektrelationale Abbildung**
- **Nachrichtenübermittlungsmuster (*Messaging Patterns*)**

Viele der klassischen Patterns sind heutzutage direkt als Paradigmen in Programmiersprachen und Programmierframeworks eingearbeitet. So leiten sich z.B. Eventsysteme, wie sie z.B. in C# realisiert sind unmittelbar aus dem Gedankengut des Observer Pattern ab.

Viele der klassischen Softwarepattern, aber auch andere verbreitete Pattern eignen sich auch für die Game Programmierung. Mehr hierzu findet Ihr z.B. im kostenfrei verfügbaren Buch von Robert Nystrom: https://gameprogrammingpatterns.com

Aufgabe:

Vorstellung eines Entwurfsmusters anhand eines Game-relevanten Codebeispiels

1. Suche Dir ein Pattern aus der nachfolgenden Liste aus. Jedes Pattern kann nur 1x gewählt werden. Daher bitte mit Namen „reservieren“.

   https://docs.google.com/spreadsheets/d/16Z8RJfVakE2vhNIFCrdFkvhPh0fiow2b7hNWZXMSQgk/edit?usp=sharing

2. Bereite eine 15-20-minütige Präsentation vor. Die Präsentationstermine finden ab September statt und werden am ersten Vorlesungstermin gemeinsam geplant. In der Präsentation erläuterst Du Deinen Kommilitonen das Pattern in Theorie und an einem Codebeispiel (siehe unten).

3. Bereite ein vollständig lauffähiges Codebeispie (C#, oder C++)l vor, welches das Pattern nutzt. Dieses Codebeispiel kannst Du ganz, oder in Teilen für Deine Präsentation nutzen.

4. Gebe die Präsentationsunterlagen (z.B. Powerpoint) und das Codebeispiel in Moodle im Rahmen der Portfolioaufgabe ab.

Bewertungsschema für diese Aufgabe:

Die Präsentation muss gehalten, wird aber nicht separat benotet. Benotet wird zu gleichen Teilen:
- Die qualitative Umsetzung des Patterns: Korrekte Implementierung und korrekter Einsatz des Patterns, allgemeine Codequalität.
- Präsentationsunterlagen (z.B. Powerpoint): Vollständigkeit, Verständlichkeit und Nachvollziehbarkeit der Unterlagen

Zeitlicher Umfang:

Die Aufgabe sollte im Durchschnitt in 2-5 Stunden komplett bearbeitbar sein. D.h. Einarbeitung in das Thema, Erstellung der Präsentationsunterlagen und des Codebeispiels.