Praktikum 1: DungeonCrawler Basisgerüst

Lernziele

- Vererbung
- Polymorphie
- Zweidimensionale Strukturen

Aufgabenstellung

Im Praktikum zu PG2 werden Sie in diesem Semester einen so genannten „Dungeon Crawler“ implementieren. Dabei geht es darum, eine Spielfigur (den Helden/die Heldin) rundenweise durch eine vorgefertigte Welt zu führen, wobei sich die Figur pro Runde um genau ein Feld bewegen kann.

Die Figur kann dabei auf freundlich oder feindliche gesinnte Nicht-Spieler-Figuren treffen (Non-Player Characters oder NPCs), wobei es zu Interaktionen zwischen Spieler und NPC kommen kann.

In Praktikum 1 werden Sie ein Grundgerüst implementieren, welches im Laufe des Semester noch erweitert wird. Achten Sie daher auf eine saubere Arbeitsweise, da der Quellcode Sie noch einige Wochen begleiten wird!

Hinweis: In den meisten Fällen werden Zeiger (und dynamisch erzeugte Objekte) verwendet. Stellen Sie sicher, dass Sie dies tatsächlich so implementieren!

Hinweis: Achten Sie ebenfalls darauf, nicht von den Klassen- und Methodennamen abzuweichen.

Hinweis: Sie können bei Bedarf die angegebenen Klassen beliebig um Attribute und Methoden erweitern.

Das Spiel an sich ist aus der „Top-Down“ Perspektive und ist in quadratischen „Kacheln“ organisiert, welche die kleinste Einheit eines Levels darstellen.

In den ersten beiden Praktika findet die Ausgabe des Spielfelds über eine (gewohnte) Terminalausgabe statt. Ab Praktikum 3 werden Sie eine Benutzerschnittstelle in Qt implementieren. Achten Sie also beim Implementieren der ersten beiden Praktika darauf, keine Ein- und Ausgaben außerhalb der Klasse `TerminalUI` zu machen.

Klassenübersicht

`Tile` stellt eine Kachel dar, also ein Feld der Spielwelt. Die Klasse `Tile` soll als **abstrakte** Klasse implementiert werden. Von `Tile` erben aktuell drei konkrete Kacheltypen:

- `Floor` ist eine normale, betretbare, Bodenkachel
- `Wall` ist eine nicht betretbare Wand
- `Portal` „beamt“ eine Figur beim Betreten zu einem anderen Portal. Immer zwei Portale sind auf diese Art miteinander verbunden.

`Character` ist die Klasse für Spielfiguren. In diesem Praktikum werden Held/Heldin und NPCs noch nicht unterschieden. Auch verfügt eine Figur noch nicht über irgendwelche Attribute, diese kommen erst in späteren Praktika hinzu.

`Level` ist die Karte bzw. das Spielfeld des Dungeon Crawlers. Die Karte besteht im Wesentlichen aus einem zweidimensionalen Array aus Kacheln (bzw. Zeiger auf `Tile`) sowie einem Array aus Figuren (Zeiger auf `Character`).

`DungeonCrawler` stellt die „Managerklasse“ für das gesamte Spiel dar. Das Spiel soll so konzipiert sein, dass prinzipiell mehrere Levels existieren können, daher enthält `DungeonCrawler` ein Array aus `Level` Zeigern. Auch findet hier die Verarbeitung der Spielzüge statt. Prinzipiell werden hier die Mechaniken implementiert,

allerdings werden viele Verarbeitungsschritte in den anderen Klassen implementiert. So bleibt das Spiel an sich flexibel und erweiterbar. Implementieren Sie also nicht alles in dieser Klasse![1]

`AbstractUI` ist eine abstrakte Oberklasse für die verschiedenen Benutzerschnittstellen. Implementieren Sie zunächst als einzige Unterklasse `TerminalUI` mit Ein- und Ausgabe über `cin` bzw. `cout`. Die Trennung zwischen eigentlicher Spielimplementierung und Implementierung der Benutzerschnittstelle macht es einfach, später `TerminalUI` einfach durch eine andere Klasse wie zB. `QtGUI` zu ersetzen.

1 Klasse `Tile`

Für den Ablauf des Spiels werden so genannte „Events“ definiert. Die dazugehörigen Methoden beginnen per Konvention normalerweise mit on. Für die Tile Klassen sind bspw. `onEnter` und `onLeave` definiert. Eine Klasse kann „entscheiden“ auf diese Events zu „reagieren“ in dem die entsprechende Methode überschrieben wird.

1.1 Unterklassen von `Tile`

Floor ist eine normale, betretbare Bodenkachel ohne weitere Funktionalität. Ein Floor kann durch einen Character betreten und verlassen werden, ohne dass irgendetwas passiert.

Wall wiederum ist ein nicht betretbarer Kacheltyp, ein Character soll ein Feld mit dieser Kachel also nicht betreten dürfen.

Portal ist eine spezielle Art von Kachel. Portale gibt es immer in Pärchen, die untereinander durch Pointer verbunden sind. Sobald ein Character ein Portal betritt, wird dieser sofort an das andere, zu diesem Portal gehörige Portal „gebeamt“.

1.2 Attribute

Ein Objekt vom Datentyp `Tile` benötigt die folgenden Attribute:

`texture` ist ein String, in welchem gespeichert wird, welche **Textur** für die Kachel verwendet werden soll. In der aktuellen Ausbaustufe ohne Qt GUI kann dieser String verwendet werden, um in `TerminalUI` die Kacheltypen zu unterscheiden und ggfs. ein bestimmtes Zeichen für jeden Tile Typ auszugeben.

`character` ist ein **Pointer** auf den `Character`, der aktuell auf dieser Kachel steht, bzw. `nullptr`, falls dort kein `Character` steht. Eine Kachel „weiß“ also über diesen Pointer, ob und welcher Character gerade auf ihr steht. (Der Character wiederum besitzt ebenfalls einen `Tile*` und weiß daher, auf welcher Kachel er steht).

`row` und `column` sind die Position der Kachel auf dem Level. Diese verändern sich nicht über die Lebenszeit eines `Tile` Objekts und sollen daher `const` sein.

Während `Wall` und `Floor` keine weiteren Attribute benötigen, wird in `Portal` ein Zeiger `Tile* destination` auf das Zielportal benötigt. Sie können bei Bedarf weitere Attribute hinzufügen.

1.3 Methoden

Schreiben Sie getter- und setter Methoden, wo Sie sie sinnvoll finden bzw. benötigen. Denken Sie daran, dass sich Kacheltyp und Position (Reihe/Spalte) einer Kachel nicht ändern sollen (setter Methoden sind hier also nicht zielführend).

[1] Beispielsweise implementiert `DungeonCrawler` zwar prinzipiell die Bewegung einer Figur, das „Beamen“ der Portale ist allerdings in `Portal` implementiert.

Weiterhin werden folgende Methoden benötigt:

`string getTexture()` liefert den Texturnamen, mit dem diese Kachel in der Anzeige dargestellt wird. Eine Kachel kann in der `TerminalUI` durch ein Zeichen repräsentiert werden, bspw:

- Floor: `"."`
- Wall: `"#"`
- Portal: `"O"`

Steht allerdings ein `Character` auf dieser Kachel, so wird die Textur des Characters zurückgegeben (bspw. der String `"X"`).

`bool hasCharacter()` soll `true` zurückliefern, wenn sich ein Character auf dieser Kachel befindet, ansonsten `false`.

bool moveTo(Tile* destTile, Character* who) implementiert die Bewegung des Characters who von einer Kachel zu einer anderen. Dabei wird der Character von der aktuellen Kachel (`this`) auf die Zielkachel `destTile` bewegt. Die Methode liefert `true`, falls die Bewegung erfolgreich war. Mehr zum Bewegungsablauf in Teilaufgabe 1.4.

`Tile* onEnter(Tile* fromTile, Character* who)` implementiert, was beim **Betreten** einer Kachel passieren soll. Die Kachel, die versucht wird zu betreten ist `this`, die Kachel wo die Bewegung gestartet wurde ist `fromTile`. Ist die aktuelle Kachel (in dieser Situation) nicht betretbar, gibt die Methode `nullptr` zurück. Ansonsten wird die Kachel zurückgegeben, welche betreten wird. Diese kann sich von `this` unterscheiden, bspw. bei einem Portal.

`Tile* onLeave(Tile* destTile, Character* who)` implementiert analog zu `onEnter` was beim **Verlassen** der Kachel geschehen soll. Auch hier ist die Kachel, welche verlassen wird, `this`. Die Methode liefert `nullptr`, wenn die Kachel nicht verlassen werden kann.

Prinzipiell werden Sie weitere Methoden kennenlernen, die bei **Events** oder **Aktivitäten** aufgerufen werden, um alle an der Aktivität beteiligten Instanzen über die Aktivität zu „informieren“. Alle diese Methoden werde ich mit **on** beginnen, um klarzumachen, welchem Zweck diese dienen.

1.4 Bewegungsablauf

Um die Bewegung eines `Character` durchzuführen, erfüllen die drei Methoden

- `bool moveTo(Tile* destTile, Character* who)`
- `Tile* onEnter(Tile* fromTile, Character* who)`
- `Tile* onLeave(Tile* destTile, Character* who)`

folgende Aufgaben:

- Von der Managerklasse `DungeonCrawler` aus wird für einen Character die Bewegungsrichtung ermittelt, bspw. durch Tastatureingabe.
- Die Managerklasse ermittelt den Pointer auf die aktuelle Kachel des Characters
  - Diese kann vom Character bezogen werden, da dieser einen Pointer auf seine Kachel hat
- `DungeonCrawler` ermittelt den Pointer auf die Zielkachel und ruft anschließend `moveTo` auf der Characterkachel mit der Zielkachel als Parameter auf.
- In `moveTo` ermittelt die Kachel zunächst, ob sie verlassen werden kann, indem `onLeave` aufgerufen wird. Ist das Ergebnis `nullptr`, so kann `moveTo` mit `false` beendet werden. Zusätzlich speichern Sie das Ergebnis von `onLeave`, da dies die tatsächlich verlassene Kachel ist.

- Anschließend wird `onEnter` mit der tatsächlich verlassenen Kachel als Parameter aufgerufen, und das Ergebnis ebenfalls gespeichert. Dies ist die tatsächlich betretene Kachel. Ist das Ergebnis `nullptr`, so kann die Kachel nicht betreten werden und `moveTo` wird mit `false` beendet.
- Wenn sowohl `onLeave` als auch `onEnter` erfolgreich waren, muss nun `moveTo` den `Character*` von der tatsächlich verlassenen Kachel auf die tatsächlich betretene Kachel setzen.
- Weiterhin muss auch der `Tile*` in `Character` aktualisiert werden.

Das oben beschrieben Vorgehen erscheint Ihnen vermutlich zunächst als unnötig kompliziert, allerdings liefert es viele Freiheiten um zukünftig weitere Kacheltypen zu implementieren. Der Zeiger who wird bspw. hier noch gar nicht verwendet, vorstellbar wäre allerdings eine Tür die nur dann betretbar ist, wenn der Character ein bestimmtes Item im Inventar hat.

Für das aktuelle Praktikum können die Methoden wie folgt implementiert werden:

- `onLeave` gibt immer `this` zurück.
- `onEnter` liefert:
  - `this` bei einer „Floor“ Kachel
  - `nullptr` bei einer „Wall“ Kachel
  - Den Zeiger auf das verbundene Portal bei einer „Portal“ Kachel
- `fromTile` in `onEnter` sowie `destTile` in `onLeave` spielen aktuell noch keine Rolle.
- who spielt ebenfalls noch keine Rolle

2 Klasse `Level`

Die Spielwelt besteht aktuell aus einem zweidimensionalen Array aus `Tile`-Zeigern. Durch die Position im Array (Reihe/Spalte) ist die Position der Kachel in der Spielwelt definiert.

Hinweis: Sie können prinzipiell eine beliebige Datenstruktur verwenden, bspw.

- Ein zweidimensionales Array aus `std::array<std::array<Tile*, n>, m>` (bei einem Spielfeld mit n$\times$m Feldern)
- Ein zweidimensionales Array aus `std::vector<std::vector<Tile*>>`
- Ein dynamisch erzeugtes zweidimensionales C-Array

Sehen Sie eine weitere Datenstruktur (z.B. `std::vector`) vor, um die in diesem Level vorkommenden Figuren (`Character`) zu speichern.

2.1 Attribute

Speichern Sie, zusätzlich zur den Datenstrukturen für die `Tile*` und `Character*`, die Höhe und Breite des Levels in zwei **Konstanten** ab.

2.2 Konstruktoren & Destruktor

Sorgen Sie dafür, dass sämtliche dynamisch erzeugten Ressourcen (Tiles, Character, usw.) im Destruktor der Level-Klasse korrekt zerstört werden.

Auch sollte es fehlerfrei möglich sein, ein `Level` Objekt zu kopieren.

Erzeugen Sie zunächst im **Standardkonstruktor** eine kleine Karte mit mindestens 4x4 Felder sowie mindestens einem Pärchen von Portalen zu Testzwecken. Setzen Sie weiterhin auf ein beliebiges Feld eine Figur.

In nachfolgenden Praktika werden wir mit einer Leveldatei arbeiten, welche geladen wird.

2.3 Methoden

Implementieren Sie folgende Methoden:

`Tile *getTile(int row, int col);` liefert den Pointer auf das `Tile` Objekt an der angegeben Koordinate zurück.

`const Tile *getTile(int row, int col) const;` ist die `const` Variante

`void placeCharacter(Character *c, int row, int col);` setzt den `Character` auf das `Tile` Objekt an der angegebenen Koordinate. Dies soll ohne Aufruf der `onEnter` Methode geschehen und dient hauptsächlich dem Initialisieren des Levels.

3 Klasse `Character`

Diese Klasse ist (in der aktuellen Ausbaustufe) sehr einfach gehalten. Sie enthält ein Attribut vom Typ `string`, welches die für die Figur verwendete Textur enthält. Das Attribut soll (ausschließlich) über einen Konstruktor gesetzt werden, ebenfalls soll eine `string getTexture()` Methode existieren (ähnlich wie schon bei `Tile`).

Als weiteres Attribut soll es noch einen `Tile*` geben, der auf die Kachel verweist auf welcher der Character steht. Hierfür sollte ein setter existieren, da der `Tile*` bei einer Bewegung ebenfalls verändert werden muss.

Implementieren Sie eine Methode `int move()`, in der der Benutzer nach einer Bewegungsrichtung gefragt wird. Mögliche Richtungen sind die Zahlen 1 bis 9 wobei die Richtungen mit dem Ziffernblock auf der Tastatur übereinstimmen (8 bedeutet „nach oben“, 1 bedeutet „links unten“ usw.). Durch Auswahl der 5 kann eine Spielfigur in dieser Runde stehen bleiben.
Die gewählte Richtung soll zurück gegeben werden.
Durch Eingabe einer 0 soll das Programm beendet werden.

Hinweis: Die Zahlen sind so gewählt, dass sie zu Richtungen auf dem Ziffernblock einer „normalen“ Tastatur passen. Wenn Sie hauptsächlich an einer Tastatur ohne Ziffernblock arbeiten, oder wenn Sie das einfach intuitiver finden, können Sie natürlich auch andere Tasten für Richtungen verwenden, beispielsweise q,w,e,a,s,d,y,x,c und irgendeine Taste zum Beenden. Die `move()` Methode liefert in dem Fall dann logischerweise einen `char` zurück.

4 Klasse `AbstractUI`

In der aktuellen Ausbaustufe verfügt die Klasse sonst nur über eine **rein virtuelle** Methode `void draw(Level*)`, die das übergebene Level auf dem Terminal ausgibt.

4.1 Klasse `TerminalUI`

Kapseln Sie alles, was mit der Ausgabe zu tun hat, in dieser Klasse (die Eingabe der Bewegungsrichtung können Sie aktuell noch in `Character` implementieren)!

5 Klasse `DungeonCrawler`

Dies ist die „Hauptklasse“ der Anwendung und verwaltet folgende Objekte:

- Einen Pointer auf eine `AbstractUI`
- Einen Pointer auf das aktuelle `Level`. In späteren Ausbaustufen werden wir hier mehrere Level Objekte verwalten und zwischen Leveln hin- und her wechseln. Aktuell genügt ein einziges Level, welches Sie mit dem `Level` Standardkonstruktor erstellen.

Initialisieren Sie im Konstruktor die Attribute entsprechend und implementieren Sie in einer Methode `play()` das eigentliche Spiel. Hierzu rufen Sie für jede Spielfigur des aktuellen Levels die `move` Methode auf und führen die unter Teilaufgabe 1.4 auf Seite 3 beschriebenen Schritte aus. Wiederholen Sie so lange, bis der Benutzer das Programm beendet.