Alexey Khokhlov Michel Witt Ayaz Khudhur

Zyklus 4: Konzept

Inhalt

Monster und Held: Nahkampf (5p)

Beschreibung der Aufgabe

- Held/Monster sollen im Nahkampf Schaden verursachen
- Konzept Vorgaben erweitern
- Rückschlag bei Treffer

Beschreibung der Lösung

Realisierung eines Nahkampfsystems, der Held soll automatisch bei Kollision mit einem Monster angreifen, das gleiche gilt für das Monster, wenn es mit dem Helden kollidiert.

Wenn das Monster stirbt, soll der Held XP erhalten und ein zufälliges Item wird vom Monster gedroppt.

Methoden und Techniken

Der Code wird mit JavaDoc ggf. mit normalen Kommentaren (private) dokumentiert. Logger wird an sinnvollen Stellen gesetzt (Kollision, Schaden, Sterben, XP).

Es werden keine weiteren Klassen erzeugt, sondern nur Methoden implementiert.

Ansatz und Modellierung

Es wird ein Melee System eingebaut, das auf Kollisionen reagiert und automatische angriffe auslöst, es werden bei dem Helden und bei den Monstern nur neue Methoden hinzugefügt, die auf den Helden/Monster Werten von Zyklus 1 aufbauen.

In der update() Methode, die bei jedem Frame aufgerufen wird findet das eigentliche „Kampfsystem“ statt das mit hitEnemy() aufgerufen wird, wenn die jeweilige Hitbox (Hero oder Monster) eine Kollision wahrnimmt mit gegnerischem Part Held => Monster und Monster => Held wird ein Boolean Wert auf true gesetzt und der/das Held/Monster erhält schaden über Zeit, solange die Kollision besteht, wird die Kollision verlassen wird der Boolean Wert wieder auf false gesetzt.

Wenn ein Monster stirbt dropt es ein zufälliges Item (Zufälliges Item in Zyklus 1 schon implementiert) mit der onDeath() Methode, zusätzlich soll hier auch die XP für den Helden angerechnet werden.

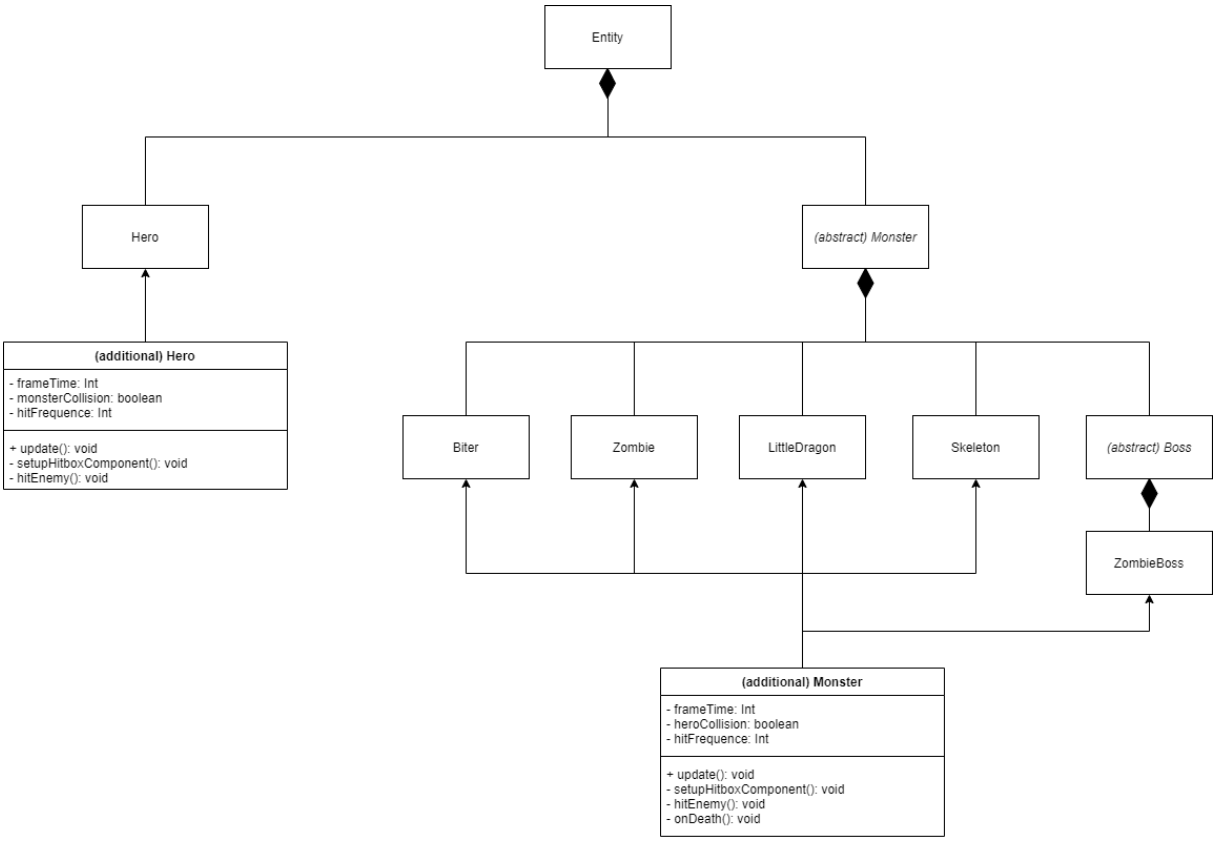
Damit der/das Held/Monster nicht in jedem Frame schaden erhält gibt es Hit Cooldown und Frame Zähler Variablen.

Grundeigenschaften

Held:
- Der in Zyklus 1 gesetzte Schaden wird für die Auto Kampf Funktion genutzt.
- Die in Zyklus 1 gesetzte int HP wird auf das HealthComponent geändert.

Monster:
- Der in Zyklus 1 gesetzte Schaden wird für die Auto Kampf Funktion genutzt.
- Die in Zyklus 1 gesetzte int HP wird auf das HealthComponent geändert.
- Das in Zyklus 1 gesetzte Item wird im Dungeon Level gedroppt.
- Die in Zyklus 1 gesetzte XP wird dem Helden angerechnet, wenn das Monster stirbt.

UML



Beschreibung

Held:
- frameTime // Anzahl der Bilder pro Sekunde
- monsterCollision // Variable für den auto hit zustand => Das auslösen von CollisionEnter setzt diesen Wert auf true (nur bei Monstern), CollisionLeave setzt den Wert auf False
- hitFrequence // Anzahl der Schläge pro Sekunde

Monster:
- frameTime // Anzahl der Bilder pro Sekunde
- heroCollision // Variable für den auto hit zustand => Das auslösen von CollisionEnter setzt diesen Wert auf true (nur beim Helden), CollisionLeave setzt den Wert auf False
- hitFrequence // Anzahl der Schläge pro Sekunde

Game-Over (5p)

Beschreibung der Aufgabe

Wenn der Spieler stirbt, soll „Game Over“ auf dem UI angezeigt werden.

Beschreibung der Lösung

Durch Verwendung des ScreenController und ScreenImage können Bilder auf der Benutzeroberfläche dargestellt werden. In der addListener() Methode wird ein Listener erwartet, der die TextButtonListener Klasse verwendet und die clicked() Methode überschreibt.

In der Hero Klasse wird geprüft, ob die HP (Lebenspunkte) kleiner oder gleich 0 sind (onDeath() Methode). Wenn dies der Fall ist, wird das GameOver Menü angezeigt.

Der Spieler wählt zwischen „restart“ und „exit“.

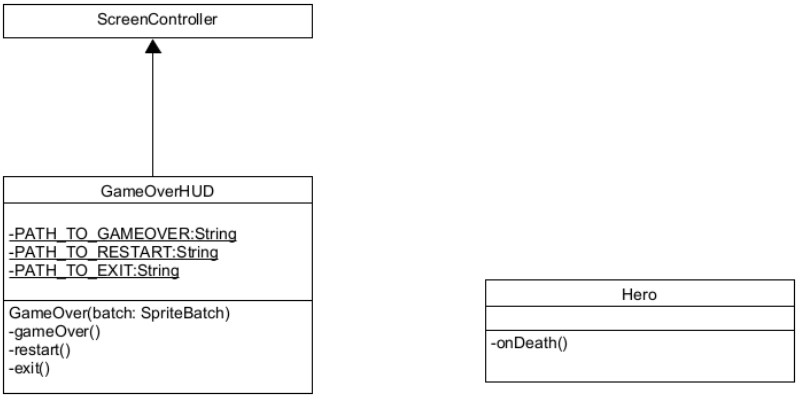
Restart
- Ein neues Level wird gestartet und die Spieler Eigenschaften werden auf 0 gesetzt.

Exit
- Das Spiel wird beendet

Eigene Assets werden verwendet.

GameOver Screen

UML



Dialog System (5p)

Beschreibung der Aufgabe

Das zu lösende Problem besteht darin, ein dynamisches Dialogsystem zu implementieren, das Benutzereingaben anhand von regulären Ausdrücken erkennt und entsprechend darauf reagiert. Zusätzlich soll ein Rätsel im Spiel implementiert werden, das eine Eingabe erfordert, die zu einem bestimmten regulären Ausdruck passt.

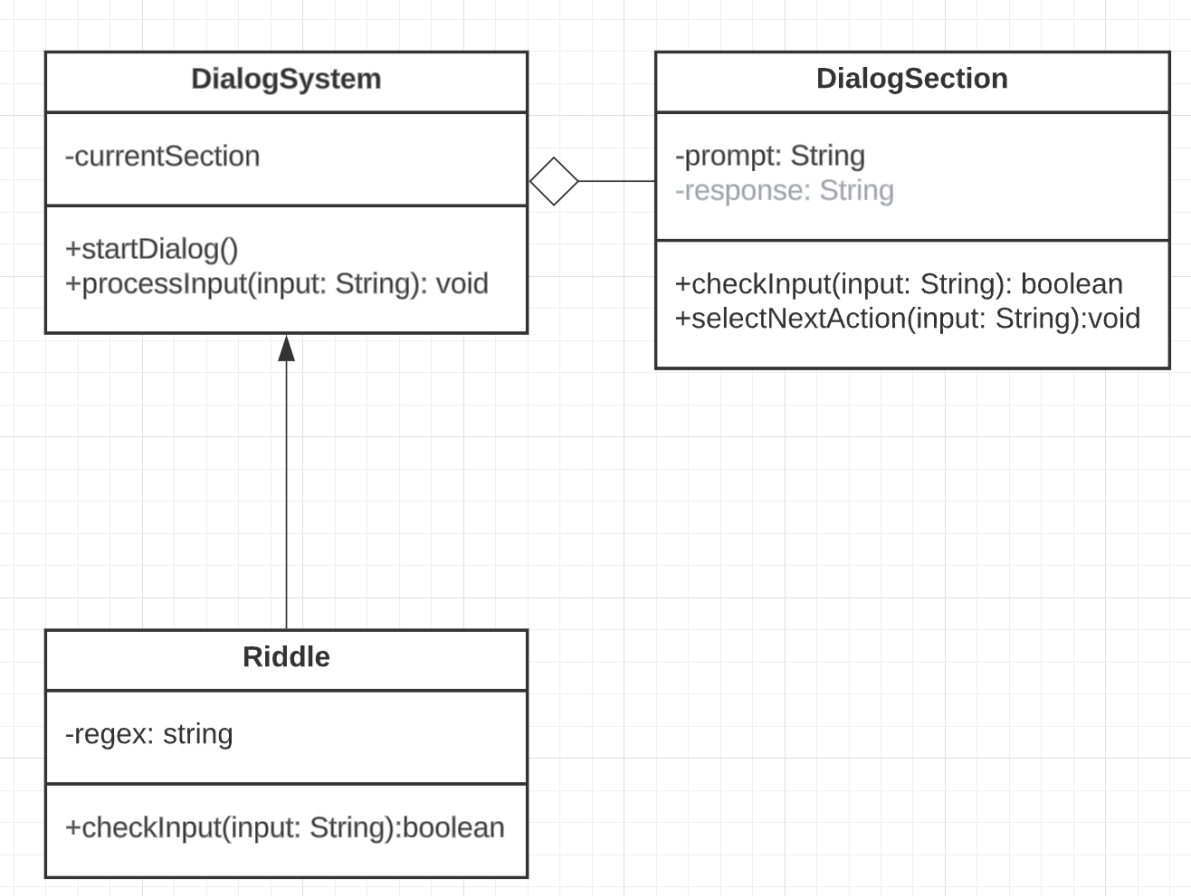
Beschreibung der Lösung

Die Lösung besteht darin, ein Dialogsystem zu entwickeln, das die Benutzereingaben analysiert und basierend auf vordefinierten Mustern die passende Aktion oder Antwort ausführt. Das System sollte in der Lage sein, Dialogabschnitte zu verwalten, Rätsel mit regulären Ausdrücken zu implementieren und den Spieler interaktiv am Spiel teilhaben zu lassen.

Methoden und Techniken

Um die Lösung umzusetzen, werden wir reguläre Ausdrücke (RegExp) verwenden, um die Benutzereingaben zu analysieren und Muster zu erkennen. Dadurch können wir gezielt auf bestimmte Eingaben reagieren.

Ansatz der Modellierung und UML



- Die Klasse "DialogSystem" verwaltet den Dialogfluss und enthält eine Referenz auf den aktuellen Dialogabschnitt ("currentSection"). Die Methode "startDialog()" startet den Dialog, und "processInput(input: string)" verarbeitet die Benutzereingabe.

- Die Klasse "DialogSection" repräsentiert einen Abschnitt des Dialogs und enthält eine Eingabeaufforderung ("prompt") und mögliche Antworten ("responses"). Die Methode "checkInput(input: string)" überprüft, ob die Eingabe mit einer der möglichen Antworten übereinstimmt, und "selectNextAction(input: string)" wählt die nächste Aktion basierend auf der Eingabe aus.

- Die Klasse "Riddle" repräsentiert ein Rätsel im Spiel und enthält einen regulären Ausdruck ("regex") für die erwartete Eingabe. Die Methode "checkInput(input: string)" überprüft, ob die Eingabe mit dem regulären Ausdruck übereinstimmt.