Studienleistung 3

Zuletzt bearbeitet von Alexander Bazo

Studienleistung 3

Wichtige Informationen zur Bearbeitung der Aufgabe

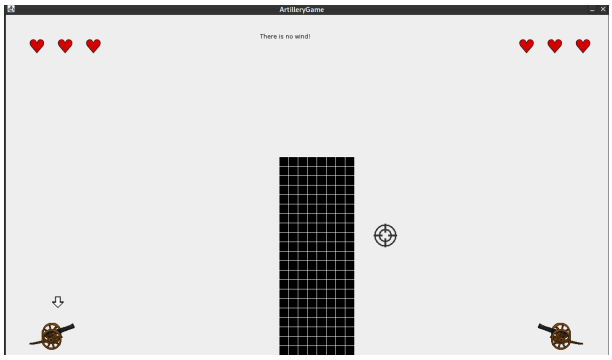
- Informationen zur Entwicklungsumgebung *IntelliJ IDEA*
- Informationen zum Im- und Export von Projekten
- GraphicsApp

Starterpaket

Ein vorbereitetes Starterpaket zur selbständigen Implementierung der Aufgabe finden Sie hier.

ArtilleryGame

Im Rahmen dieser Aufgabe entwickeln wir ein Spiel aus dem Genre der Artillery Games. Tank Wars ist ein Beispiel für ein solches Spiel. Unser ArtilleryGame bleibt dieser Formel treu und ist demnach auch ein 2-Spieler, rundenbasiertes Strategiespiel.



Auf GRIPS finden sie zusätzlich ein kleines Demo-Video, das den allgmeinen Spielablauf zeigt.

Beschreibung

Auf der jeweils linken und rechten Seite des Spielfeldes befinden sich zwei Kanonen, stellvertretend für Spieler:in 1 (linke Kanone) und Spieler:in 2 (rechte Kanone). Wessen Zug es gerade ist, wird durch den weißen Pfeil über der entprechenden Kanonene angezeigt. Die beiden Kanonen sind anhand von Terrain von einander getrennt. Diese Terrain ist zerstörbar. Spieler:innen benutzen Tastatur und Maus, um ihre Kanone zu bewegen (Tastatur), mit einem Fadenkreuz zu zielen (Maus) und einen Schuss abzugeben (Maus, linke Maustaste). Spieler:innen haben genau einen

Schuss pro Runde. Dieser Schuss beendet den Zug und die andere Person ist dran mit ihrem Zug. Die abgefeuerte Kanonenkugel wird von Gravitation und Wind beeinflusst. Wind kommt in unterschiedlichen Facetten vor - kein Wind, Windstärke 1, 2, 3 und 4. Wird eine Kanone dreimal getroffen, gewinnt die Person, die den Schuss abgegeben hat, das Spiel. Dies wird den Spieler:innen anhand eines Game-Over-Sreens angezeigt. Das Spiel startet neu, wenn man in diesem Game-Over-Screen die linke Maustaste drückt. Das Spiel beinhaltet eine sich wiederholende Hintergrundmusik und Sounds, die abgespielt werden sollen, wenn eine Kanone abgefeuert wird und eine Kanonenkugel etwas trifft (z.B. Kartenrand, Kanone, Terrain). Alle benötigten assets liegen im Ordner `data/assets`.

Anforderungen

- Die Klasse `ArtilleryGame` muss als Einstiegspunkt für Ihr ArtilleryGame verwendet werden
- Teilen Sie Ihre Anwendung in sinnvolle Komponenten ein und legen Sie entsprechende Klassen dafür an
- Trennen Sie die Daten von Objekten (z.B. Kanonen) von deren Darstellung
- Teilen Sie das ArtilleryGame in konkrete Teilaspekte ein, z.B.:
  - Szenen (Spiel, Game Over)
  - Spielobjekte (Actors, z.B. Kanonen, Terrain)
- Zerlegen Sie das Spiel in konkrete Phasen und Phasenübergänge.
- Verwenden Sie sinnvolle Datenstrukturen (z.B. `ArrayList`)
- Praktizieren Sie `Decomposition`

Hinweise

Berechnung der Schussrichtung

- Die Schussrichtung muss als zweidimensionaler Richtungsvektor berechnet werden, um die korrekten x- und y-Koordinaten für die Kanonenkugel pro Frame berechnen zu können
- Die Schussrichtung wird durch zwei Punkte definiert: die Mündung des Kanonenrohrs und dem von den Spieler:innen bewegbaren Fadenkreuz
- Bei Mausklick soll anhand dieser Richtung ein Richtungsvekor berechnet werden
- Verwenden Sie folgenden Code für die Berechnung (tauschen Sie Platzhalter entsprechend aus):

```
private Point calculateShotDirection() {
    Point barrelPosition
        = new Point(<x-Koordinate der Kanonenmündung>, <y-Koordinate der Kanonenmündung>);

    Point crosshairPosition = new Point(<x-Koordinate des Mittelpunkts des Fadenkreuzes>,
        <y-Koordinate des Mittelpunkts des Fadenkreuzes>);

    float directionX = crosshairPosition.getXPos() - barrelPosition.getXPos();
    float directionY = crosshairPosition.getYPos() - barrelPosition.getYPos();

    Point directionVector = new Point(directionX, directionY);

    float directionXSquared = directionVector.getXPos() * directionVector.getXPos();
    float directionYSquared = directionVector.getYPos() * directionVector.getYPos();
```

```
    double directionVectorLength = sqrt(directionXSquared + directionYSquared);

    float normalizedDirectionX = v.getXPos() / directionVectorLength;
    float normalizedDirectionY = v.getYPos() / directionVectorLength;

    Point normalizedDirectionVector
        = new Point(normalizedDirectionX, normalizedDirectionY);

    return normalizedDirectionVector;
}
```

Mögliche Erweiterungen