CT Projekt: Raycasting Game Engine II (Huskyrock)

Christian Korn

14.01.2022 - 04.06.2022

Inhaltsverzeichnis

1 Ziele

1.1 Muss-Ziele

Hundefels

Das neue Projekt muss die Kernfunktionalität des Vorgängerprojektes enthalten:

- Anzeigen eines 2D Levels in 2,5D per Raycasting ✓
- Bewegungsfreiheit im Level ✓
- Anzeige von Entities im Level ✓
- Kollisionserkennung ✓

Neues

Natürlich soll das neue Projekt auch eigene Features und Technologien enthalten, die es vom Alten unterscheiden:

- Wand-Texturen ✓
- Sprites für Entities ✓
- KI für Entities ✓

Das neue Projekt wird in der Programmiersprache Rust geschrieben, anstatt eines objektorientierten Designs wird ein Entity component system benutzt.

1.2 Kann-Ziele

Hundefels

Einige Features des Vorgängerprojektes werden als Kann-Ziele klassifiziert, da sie nicht für die Funktionalität des Projektes absolut notwendig sind.

- Laden von Leveln aus externen Dateien (durch neue features komplizierter)
- Command-Line Argumente (Funktionalität größtenteils überflüssig oder besser als Konfigurationsdatei zu lösen)

Andere

- Visuelle Effekte (view-bobbing, etc.)
- Schießen
- Kartenansicht
- WebAssembly
- unterschiedliche Sprites für verschiedene Drehwinkel

2 Verwendete Technologien

2.1 Rust

Rust ist eine Multiparadigmen-Systemprogrammiersprache mit einem Fokus auf Performance und Sicherheit. Dies wird erreicht, indem anstatt eines Garbage Collectors oder händischem Memory managements ein sogenannter Borrow Checker verwendet wird, der Lifetimes und Scope von Variablen überprüft und Regeln für Referenzen durchsetzt.

2.2 Entity component system

Entity component system (kurz ECS) ist ein Softwarearchitekturmuster, welches vor allem in der Spieleentwicklung genutzt wird.
Im Gegensatz zu objektorientierten Programmen werden im ECS Daten und Funktionalität nicht gruppiert, sondern getrennt in sogenannte "Components" (Daten) und "Systems" (Funktionalität). Zentral im ECS sind die "Entities": Objekte, die Components enthalten, auf welche Systems angewandt werden.
Systems können spezifizieren, welche Components für sie notwendig sind und werden nur auf die Entities angewandt, die alle benötigten Components enthalten.
Zum Beispiel wird in diesem Projekt mit einem Component definiert, ob eine Entity der Spieler ist, und deshalb eine Kamera enthalten sollte, mit einem anderen ob die Entity mit einer KI ausgestattet ist. Entities ohne diese beiden Components sind stationär oder bewegen sich gleichförmig ohne steuerung (abhängig davon, ob sie die Bewegungs-Component enthält).

2.3 A*

Der A*-Algorithmus ist ein Suchalgorithmus, der einen Optimalen Pfad zwischen zwei Punkten findet. Durch eine Kostenfunktion und eine Heuristik wird dabei die Laufzeit des Algorithmus optimiert.

2.4 Raycasting

Raycasting ist eine Methode mit welcher die Entfernung zu Geometrie effizient berechnet werden kann. Es wurde in frühen 3D Spielen genutzt um einen 3D-Effekt in einer 2D Welt vorzutäuschen.

2.5 Tools

- IDE: Intellij IDEA mit Rust Extension
- Grafiksystem: SDL2
- Dokumentation: Overleaf online LaTeXEditor, `https://www.overleaf.com`
- Pixelart: Pixilart, `https://www.pixilart.com/`

3 Mathematische Funktionsweise

Alle Berechnungen außer dem A* Algorithmus werden 60 mal pro Sekunde ausgeführt, A* wird pro Entity ein mal pro Sekunde berechnet.

3.1 Bewegung

Die aktuelle Position, Rotation, maximale Geschwindigkeit, relative Geschwindigkeit und Rotationsgeschwindigkeit werden, soweit anwendbar, in den Components `Position`, `Rotation`, `VelocityMultiplier` und `VelocityRelative` gespeichert, das Input des Spielers in `PlayerInput` und die Kontrollarten werden durch `UserControlled` und `HasAI` markiert.

Die neue Position und Rotation jeder Entity werden berechnet mit:

$$x_{t+1} = x_t + (-vr_x * \cos r - vr_y * \sin r)$$
$$y_{t+1} = y_t + (-vr_y * \cos r + vr_x * \sin x)$$

Zur Kollisionserkennung werden 4 Punkte um die Entity überprüft, ob sie ein Block sind, wenn ja wird die Position der Entity angepasst.

3.2 Rendering

Raycasting

Von der Entity ausgehend werden Strahlen in Paaren ausgesendet:

Der horizontale Strahl überprüft bei vertikalen Linien, der vertikale Strahl bei horizontalen Linien. Mit dem Unterschied der Richtung funktionieren beide insgesamt gleich, die Erklärung konzentriert sich daher auf den horizontalen Strahl.

Zuerst wird der x-Wert des Strahls x_{ray} auf die nächste Linie gesetzt, danach der y-Wert y_{ray} mit folgender Formel bestimmt:

$$y_{ray} = x_{entity} - x_{ray} * \frac{-1}{\tan(\alpha_{ray})} + y_{entity}$$

Nun wird der Strahl wiederholt um 1 in x-Richtung und um

$$-block\_size * \frac{-1}{\tan(\alpha_{ray})}$$

in y-Richtung verschoben. Nach jedem Verschieben wird überprüft, um es sich beim neuen Block um eine Wand handelt, wenn ja wird abgebrochen und die zurückgelegte Entfernung, sowie die x- und y-Koordinaten des Strahls zurückgegeben.

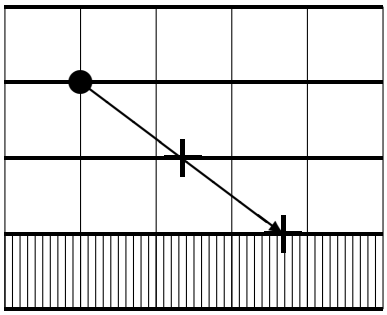
Von den zwei so berechneten Werten wird nun der kürzere Strahl ausgewählt, auf der Karte gezeichnet und als vertikale Linie der Länge

$$v\_offset = \frac{1}{dist + 0,001} * v\_multiplier$$

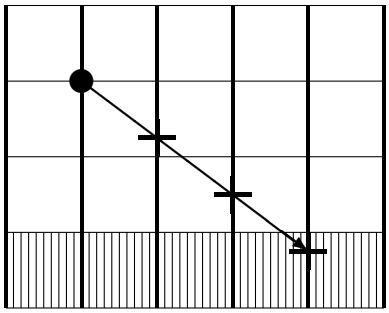
im Viewport angezeigt (*dist* ist die Länge des Strahls, die zwei festen Werte verhindern, dass durch 0 geteilt wird und erhöhen den Wert auf eine sichtbare Länge).

Beispiel

Horizontaler Check, vertikaler Strahl:



Vertikaler Check, horizontaler Strahl:



Entities

Die x-Position einer Entity im Viewport wird mit folgender Formel berechnet:

$$\frac{r_{player} + atan2(dy, dx)}{fov} * fensterbreite$$

Einige Korrekturen werden dabei noch vorgenommen, um die Position der Entity auf dem Bildschirm nicht umherspringen zu lassen.

Zuerst werden die Strahlen hinter der Entity angezeigt, dann die Entity selbst und zuletzt die Strahlen vor der Entity.

3.3 KI

Steuerung

Jede Entity mit den notwendigen Components überprüft jeden Frame, ob der Spieler hinter einer Wand ist. Ist dass der Fall, wird der nächste Punkt auf dem per A* berechneten Pfad als Ziel gewählt, ansonsten der Spieler. Zeigt die Entity auf das gewählte Ziel, geht sie mit voller Geschwindigkeit vorwärts, ansonsten dreht sie sich in Richtung Ziel und geht mit verringerter Geschwindigkeit vorwärts.

A*

Einmal Pro Sekunde wird der A* Algorithmus von jeder Entity mit den nötigen Components ausgeführt.

Jedem Knoten werden im Laufe des Algorithmus 4 Werte zugeordnet: f, g, h und der Vorgängerknoten. g ist die bisherige Kostenfunktion auf dem weg zum Knoten, h ist die Heuristik, mit der die Kosten zum Ziel geschätzt werden und f ist die Summe aus g und h.
Es werden zuerst zwei Listen angelegt: die OpenList und die ClosedList. Der OpenList wird nun der Startknoten mit einem f von 0 hinzugefügt.
solange die OpenList nicht leer ist passiert folgendes:
Es wird der Knoten q mit dem niedrigsten f von der OpenList entfernt, seine Nachfolgeknoten generiert und für jeden dieser Knoten nun überprüft, ob er der Zielknoten ist, in welchem Fall die Suche beendet wird.
Ansonsten f für den Knoten generiert. Wenn der Knoten mit niedrigerem f in der OpenList oder ClosedList ist wird er übersprungen. Ansonsten wird der Knoten der OpenList hinzugefügt. Zuletzt wird q der ClosedList hinzugefügt.

Nachdem der A* Algorithmus selbst beendet ist wird vom Zielknoten ausgehend die Route generiert, indem sich entlang der Vorgängerknoten durch die ClosedList gehangelt wird und die entstandene liste umgedreht wird.

4 Programmaufbau

- `main.rs` ist der Programmeinstiegspunkt und enthält den Großteil des Setups, sowie die Gameloop.
- `init.rs` enthält code zum Erstellen von Entities für den Spieler, Gegner und Gegenstände.
- `components.rs` enthält die Components für das ECS
- `ai.rs` enthält das KI-System und die A* Implementation
- `input.rs` ist für die Verarbeitung von Playerinput zu Bewegungsvektoren verantwortlich
- `physics.rs` führt die Bewegungsbefehle des Spielers oder der KI aus
- `render.rs` stellt entities und rays dar
- `rays.rs` berechnet das Raycasting

5 Steuerung

5.1 Bewegung

Translation (Laufen)

- Vorwärts: ‘W’
- Links: ‘A’
- Rückwärts: ‘S’
- Rechts: ‘D’

Rotation

- Links: linke Pfeiltaste (←)
- Rechts: rechte Pfeiltaste (→)

5.2 UI

Zu sehen sind:

- Verschiedene Wandtexturen (Lehmziegel, Steinziegel), bei der Kartenerstellung frei wählbar
- Verschiedene Texturen für verschiedene Seiten, automatisch (abhängig von der Seite)

- Zwei Entities mit verschiedenen Texturen, bei der Kartenerstellung frei wählbar

6 Quellen

Das GIT-repository findet sich unter
`https://github.com/MacAphon/husky_rock`

- Viel Code wurde direkt vom Vorgängerprojekt zu Rust übersetzt und wiederverwendet.
  `https://github.com/MacAphon/hundefels2d`
- Steve Klabnik & Carol Nichols: "The Rust Programming Language"
  `https://doc.rust-lang.org/stable/book/`
- "Rust by Example"
  `https://doc.rust-lang.org/rust-by-example/`
- "Rust Coockbook"
  `https://rust-lang-nursery.github.io/rust-cookbook/`
- "Learn Game Development in Rust"
  `https://sunjay.dev/learn-game-dev`
  `https://github.com/sunjay/rust-simple-game-dev-tutorial/`
- 3DSage: "Make Your Own Raycaster Part 1"
  `https://youtu.be/gYRrGTC7GtA`
  Quellcode verfügbar unter
  `https://github.com/3DSage/OpenGL-Raycaster\_v1`
- A* Search Algorithm - GeeksforGeeks
  `https://www.geeksforgeeks.org/a-search-algorithm/`