

CT Projekt: Raycasting engine (Hundefels 2D)

Christian Korn

20.10.2021 - 11.01.2022

Inhaltsverzeichnis

1	Ziele	2
1.1	Muss-Ziele	2
1.2	Soll-Ziele	2
1.3	Kann-Ziele	2
2	Verwendete Technologien	3
2.1	Python	3
2.2	Dokumentation	3
2.3	Versionskontrollsystem	3
3	Mathematische Funktionsweise	4
3.1	Bewegung	4
3.2	Darstellung	4
4	Programmaufbau	5
5	Steuerung	6
5.1	Command-Line Argumente	6
5.2	Levelerstellung	6
5.3	Bewegung	6
5.4	UI	7

1 Ziele

1.1 Muss-Ziele

Wenn diese Ziele nicht erreicht werden, wird das Projekt als Fehlschlag angesehen.

- Anzeigen eines 2D Levels in 2,5D (Raycasting Methode)
- Bewegungsfreiheit im Level (Translation und Rotation)

1.2 Soll-Ziele

Diese Ziele müssen nicht unbedingt erreicht werden, sind aber für einen vollen Erfolg nötig.

- Laden von Leveln aus Dateien
- Anzeigen von anderen Objekten im Level (z.B. Gegner, Items)
- Kollisionserkennung

1.3 Kann-Ziele

Diese Ziele sind nicht nötig, können aber nach Vollendung der Höheren Ziele in Angriff genommen werden.

- Gegner KI
- Schießen
- Sprites
- Texturen für Wände
- visuelle Effekte (view bobbing, Blutspritzer)

2 Verwendete Technologien

2.1 Python

Das Projekt wurde mit Python 3.9.7 erstellt, müsste aber auch in späteren Versionen funktionieren.

Externe Libraries

- Pygame: Installation mit “`pip install pygame`” Verwendet für Darstellung.
- Numba: Installation mit “`pip install numba`” Für ‘magische’ Leistungsverbesserungen von besonders aufwändigen Funktionen durch JIT-Compilierung.

IDE

Es wurde die PyCharm Community Edition verwendet.

2.2 Dokumentation

Die Projektdokumentation wurde mit \LaTeX erstellt, UML Klassendiagramme wurden mit YUML erstellt.

2.3 Versionskontrollsystem

Ein GIT Repository wurde angelegt. Es kann unter <https://github.com/MacAphon/hundefels2d> gefunden werden.

3 Mathematische Funktionsweise

Die Berechnungen werden 1 mal pro Frame ausgeführt. Idealerweise heißt das, dass sie 60 mal pro Sekunde erfolgen. wenn die Rechenleistung nicht ausreicht wird eine Warnung angezeigt.

3.1 Bewegung

Der aktuelle Bewegungszustand und die Position werden in den Variablen `_state` und `_position` gespeichert.

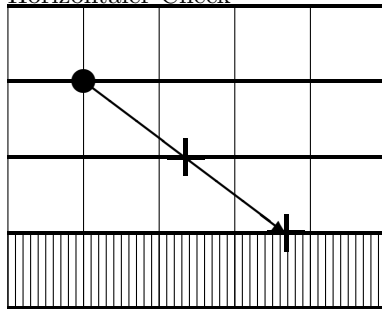
Drehung

Solange \leftarrow oder \rightarrow gedrückt werden wird der

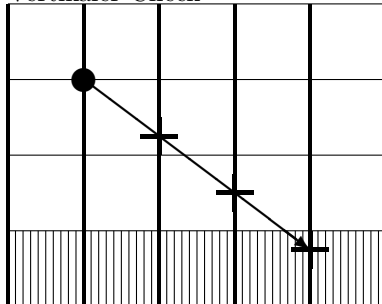
Laufen

3.2 Darstellung

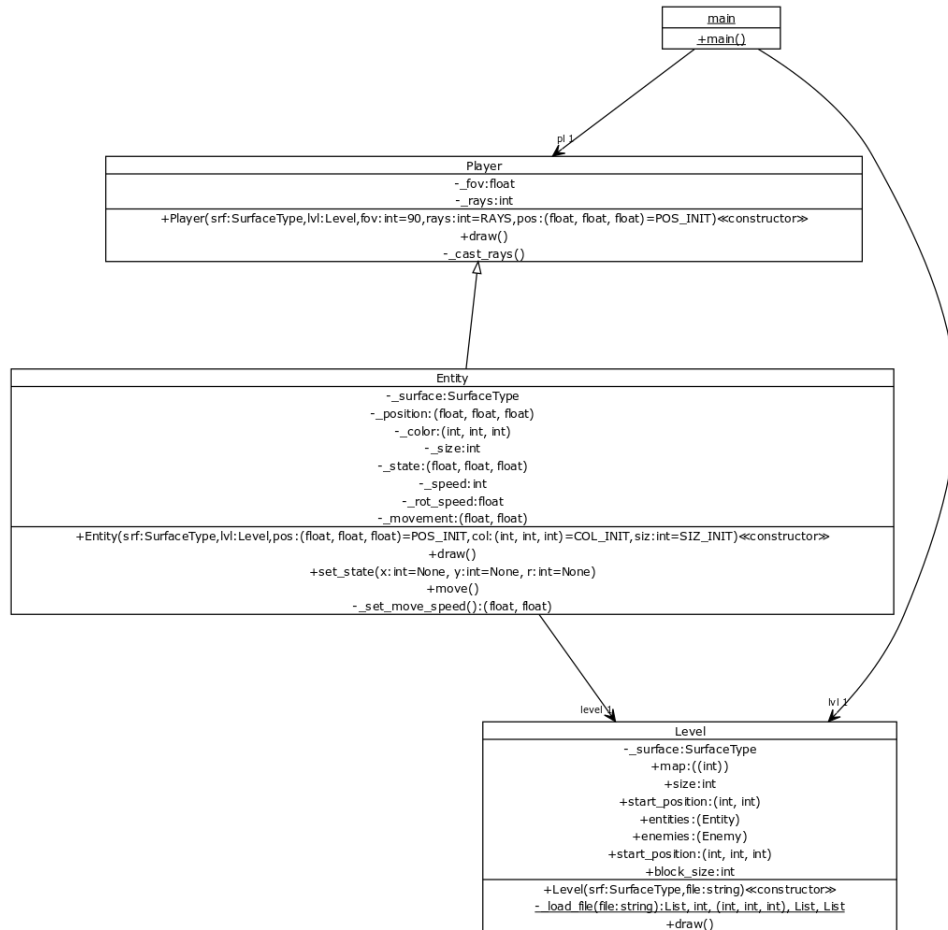
Horizontaler Check



Vertikaler Check



4 Programmaufbau



CREATED WITH YUML

5 Steuerung

5.1 Command-Line Argumente

- `-h --help` zeigt die CLI Argumente und beendet das Programm.
- `-l --level` lädt das angegebene Level oder die angegebene Level Datei.
- `--fov` ändert den Blickwinkel (angegeben in Grad) Standardwert ist 90°.
- `--rays` ändert die horizontale Auflösung (Anzahl der gesendeten Strahlen) Standardwert ist 90. Höhere Werte können die Leistung beeinträchtigen.

5.2 Levelerstellung

Level werden im JSON-Format gespeichert.

- `"map": [[int]]` Die Map: 1 entspricht einer Wand, 0 Leerraum. Die Map muss quadratisch sein (ansonsten crasht das Programm)
- `"size": int` Die Größe der Map. Muss dem tatsächlichen Wert entsprechen.
- `"start_pos": [x: int, y: int, r: int]` Die Startposition des Spielers. `x` und `y` sind Werte zwischen 0 und 512, sie geben die Position in Pixeln an. `r` ist zwischen 0 und 360 und ist die Drehung in Grad.
- `"entities": []`, `"enemies": []` Enthalten aktuell keine Werte und werden für zukünftigen Gebrauch freigehalten.

5.3 Bewegung

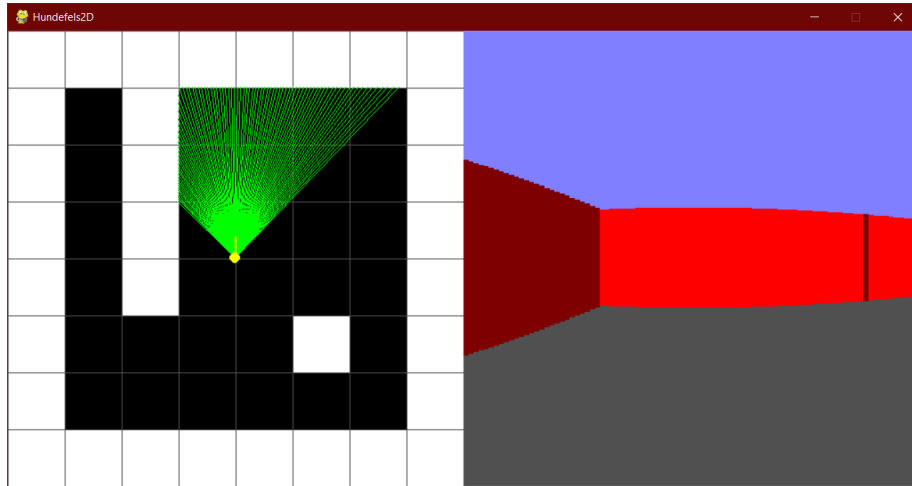
Translation (Laufen)

- Vorwärts: 'W'
- Links: 'A'
- Rückwärts: 'S'
- Rechts: 'D'

Rotation

- Links: linke Pfeiltaste (\leftarrow)
- Rechts: rechte Pfeiltaste (\rightarrow)

5.4 UI



Das Anzeigefenster ist 1024 auf 512 Pixel groß.

Die linke Hälfte enthält die Kartenansicht. Der Spieler wird mit einem gelben Punkt dargestellt, Entities mit (standartmässig) blauen Punkten.

Die rechte Hälfte des Fensters ist der First-Person Viewport.

Literatur

- [1] 3DSage: “Make Your Own Raycaster Part 1”
<https://youtu.be/gYRrGTC7GtA>
Quellcode verfügbar unter
https://github.com/3DSage/OpenGL-Raycaster_v1
- [2] Pygame tutorial: <https://www.pygame.org/docs/tut/MakeGames.html>