Python-Dokumentation

Kurze Beschreibung:

In dem Projekt kann ein kleines Ökosystem simuliert werden. In dem Ökosystem werden dabei Hasen und Pflanzen dargestellt. Während der Simulation kann beobachtet werden, wie die Hasen die Pflanzen suchen und dabei um das Überleben kämpfen.

Zielplattform: Windows

Klassen / Funktionen:

Class Game(ShowBase)

Die Game-Klasse ist der Startpunkt der Anwendung und erbt von ShowBase. Die Vererbung von ShowBase wird benötigt, damit wichtige Funktionen, wie das Rendern von Objekten, möglich sind. Außerdem werden in der Game-Klasse alle wichtigen Parameter für den Start der Simulation gesetzt, wie zum Beispiel die Fenstergröße oder die Anzahl der Hasen.

__init__(self)	In der Init-Funktion werden alle Eigenschaften, wie oben erwähnt gesetzt. Des Weiteren wird die Update-Funktion registriert, damit diese nachher auch erkannt wird und ausgeführt werden kann.
update(self, task)	Die Update-Funktion ist die Funktion, die bei jedem Frame aufgerufen wird und alle Daten halt updatet. Durch das return-Statement wird die Funktion nicht nur einmal ausgeführt, sondern gelooped
+ spawn_objects(self, count_rabbits, count_grass)	Diese Funktion ist für das Spawnen der Hasen und der Gräser verantwortlich (Hätte man auch in der __init__-Funktionen machen können, aber durch das Auslagern in eine separate Funktion wird der Code übersichtlicher)

Class Object()

Die Object-Klasse ist die Grundlage für die Grass- und Rabbit-Klasse. In der Objekt-Klasse werden Funktionen und Eigenschaften ausgelagert, welche sowohl vom Rabbit als auch vom Grass gebraucht werden, wie eine delete-Funktion, welche das Objekt aufräumt.

__init__(self, pos, model)	In der Init-Funktion werden die Position und das Modell für das Objekt gesetzt
delete(self)	Die Delete-Funktion löscht alle Referenzen auf das Objekt und räumt es sicher auf

Class Grass(Object)

Die Grass-Klasse erbt von der Object-Klasse und übernimmt somit alle Funktionen. Des Weiteren hat es eine Funktion, welche die aktuelle Position des Grass-Objektes zurückgibt (wird gebraucht, da in der Rabbit-Klasse damit gerechnet wird). Die Grass-Klasse wird außerdem noch gebraucht, um das Grass anzuzeigen und damit Aktionen, wie „wurde gegessen", auszuführen.

__init__(self, pos, model)	In der Init-Funktion werden dieselben Eigenschaften wie aus der Object-Klasse gesetzt. Außerdem wird noch die Eigenschaft „was_eaten" gesetzt.
Return_pos(self)	Gibt die Position des Grass-Objektes zurück

Class Rabbit(Object)

__init__(self, pos, speed, health)	Initialisiert die Klasse mit Eigenschaften
update(self, food_sources)	Update-Funktion wird in der Update-Funktion von der Game-Klasse aufgerufen
move(self)	Wenn gerade kein Essen in Reichweite ist, wir zufällig entschieden in welche Richtung sich bewegt wird
check_if_map_border(self, vector, vector_pos)	Bei der Move-Methode muss überprüft werden, ob aus der Welt herausgelaufen wird
random_direction(self, choice)	Gibt per Zufall ausgehend von der jetzigen Richtung (keine 180 Grad Drehung möglich) eine Richtung zurück

Klassenlose Funktionen

setup_point_light(render, pos)	Setzt eine Lichtquelle an einer bestimmten Koordinate

Module:

direct.showbase.ShowBase import ShowBase	Grundklasse von der geerbt werden muss für eine Panda3D Anwendung
random	Zufallszahlen oder Zufallsauswahl
direct.actor.Actor import Actor	Actor beinhaltet Funktionen zum Arbeiten mit Objekten (Rabbit und Grass)
math	Wird verwendet, um die Wurzel zu ziehen
panda3d.core import WindowProperties	Wird gebraucht, um das Window anzupassen
panda3d.core import Vec3	Rechnen mit 3 dimensionalen Vektoren
from panda3d.core import PointLight	Licht für die Anwendung

Umgesetzte Feature:

- Environment in Panda3D darstellen und mit Blender modellieren
- Dynamisches Spawnen von Nahrung (Grass) in dem Environment
- Ein Entity (Rabbit) mit Blender modellieren und in dem Environment simulieren
- Entity Rabbit:
    - Hungertrieb über ein Health System dargestellt (kontinuierliches senken der Health & beim Essen wird Health wieder auf das maximum erhöht)
    - Reproduktion (Immer wenn etwas gegessen wird, spawnt ein neuer Rabbit)
- Selbstgeschriebener Pathfinding- Algorithmus

Verworfene Features:

- Reproduktion mit 2 Partnern und Vererbung von Eigenschaften
- Übersicht am Ende oder während der Simulation
- Weitere Entitys

Begründung für verworfene Features:

Primär wurden die Features wegen stark steigender Komplexität verworfen. Die Reproduktion mit 2 Partnern und weitere Entitys (wie ein Fuchs) wären für das Projekt zu aufwendig. Die Übersicht am Ende oder während der Simulation wurde verworfen, da der Aufwand für so ein kleines Feature zu groß geworden wäre bzw. nach längeren rumprobieren verworfen wurde.

Zusammenfassung des erzielten Resultats:

Im Grunde wurden die Anforderungen des Projektes erfüllt. Es wurde kein fertiger Algorithmus für Pathfinding verwendet, sondern ein eigener geschrieben (GameObjects.py / Rabbit / update-Funktion), welcher vorher Out-of-Scope war. Die meisten Nice-To-Haves wurden verworfen, wegen der steigenden Komplexität. Ein bekannter Fehler ist, dass die Rabbits sich auf dem Weg zum Grass nicht in die richtige Richtung drehen und ein Fehler in der DIE angezeigt wird, der keiner ist.

Anleitung zum Ausführen des Programmes befindet sich in der README.md