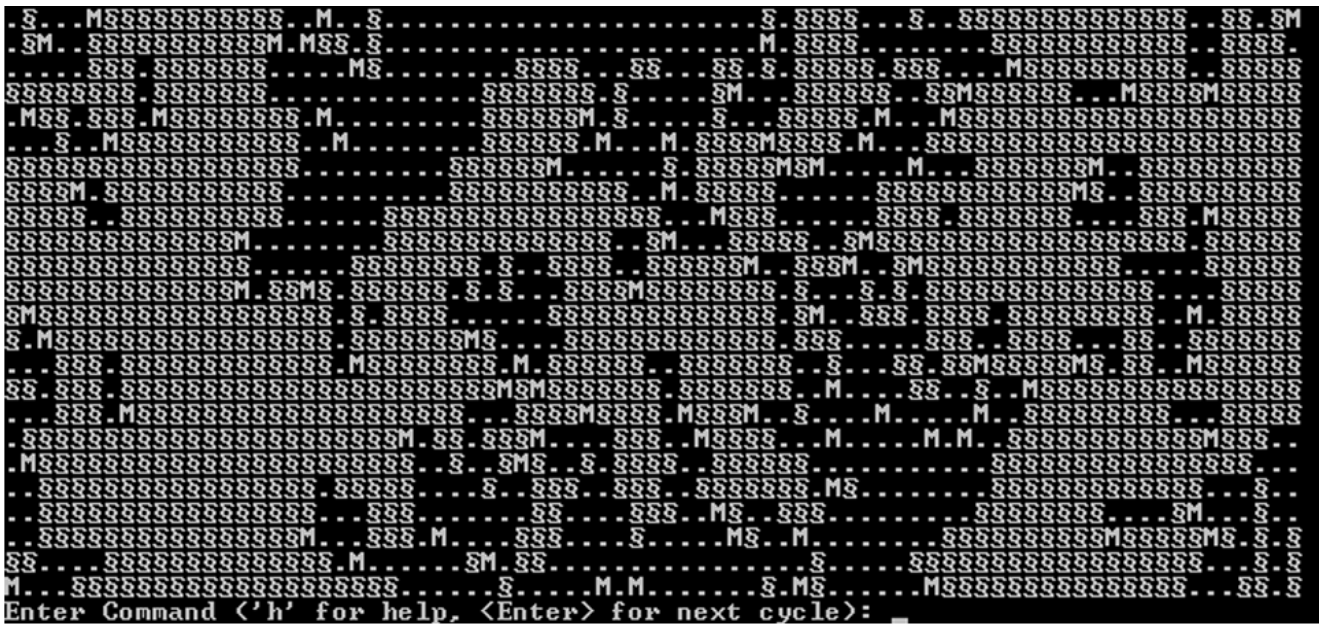
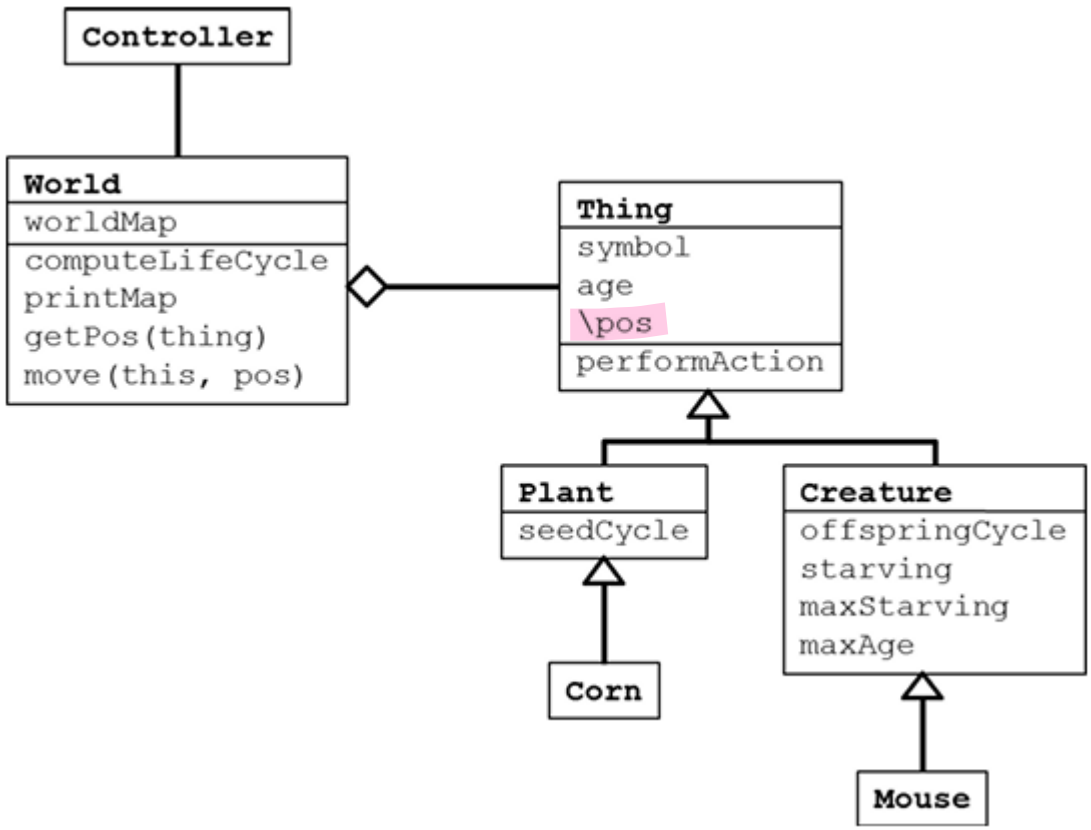
Simulation PyCreatures

Im Mittelpunkt dieser Übung steht eine Simulation mit dem Titel „PyCreatures“. In dieser virtuellen Welt können verschiedene Dinge wie z.B. Tiere oder Pflanzen existieren. Beide können sich vermehren. Tiere können Pflanzen oder andere Tiere fressen, sich fortbewegen und an Alter oder an Hunger sterben. Die Welt ist in einer Raster von m∗n Felder aufgeteilt. Die Größe der Welt soll beim Aufruf des Programms konfigurierbar sein. Jedes Feld ist entweder gar nicht belegt oder von genau einem Ding. Abbildung 1 zeigt den Schnappschuß eines Zustands der Simulation, so wie er von Ihnen entwickelt werden soll. Alle Dinge, die in der virtuellen Welt existieren haben bestimmte Attribute und Ver haltensweisen. Als Starthilfe und zur Orientierung ist die Grundstruktur der Klassen sowie deren Assoziationen vorgegeben (Abbildung 2). Sie zeigt alle Klassen sowie einige denkbare Attribute und Operationen. In der konkreten Ausgestaltung haben Sie viele Freiheiten. Da die Objektorientierung und Generalisierung im Fokus dieser Übung stehen ist jedoch die Vererbungsstruktur bindend. Überlegen Sie für jede Klasse: Welche Attribute und Verhaltensweisen sind für alle Objekte dieser Klasse und deren Unterklassen gleich? Alle Dinge auf der Welt altern zum Beispiel bei jeder Simulationsrunde um eins. Dies muss also nicht für jede Klasse neu festgelegt werden. Jedes Objekt einer Klasse welche in der virtuellen Welt existieren kann darf jede Spielrunde eine Aktion ausführen. Es macht also Sinn einmal eine entsprechende Operation zu definieren und diese dann von Fall zu Fall zu überschreiben und ggf. anzupassen. Beachten Sie, dass es in Python möglich ist, die Implementierung der Super-Klasse aufzurufen. So kann zuerst das Verhalten der Elternklasse ausgeführt werden, bevor das Verhalten der Kind-Klasse ausgeführt wird.



Die Welt

Die Welt wird durch die Klasse World repräsentiert. Sie enthält ein Verzeichnis (Dictio nary bzw. Map) aller Dinge auf der Welt und deren Position in x/y Koordination. Die Zählung der x/y Positionen beginnt immer bei 0. D.h. die auf einer Karte möglichen x Positionen liegen im Intervall [0,width−1]. Analog gilt für die y-Positionen das Intervall [0, height − 1]. Zum anderen enthält die Klasse ein zweidimensionales Array bzw. einer Matrix in welcher die Objekte der Welt repräsentiert werden. Diese Modellierung der Po sitionen ist redundant, aber es erleichtert die Programmierung. Die Klasse World stellt darüber hinaus alle Operationen bereit um Informationen über sie auszulesen oder ihren Zustand zu verändern. Die Welt in PyCreatures ist rund: Jedes Feld hat vier Nachbar felder. Diagonale Felder sind keine Nachbarfelder. Befindet sich ein Feld z.B. am oberen Rand so befindet sich sein oberer bzw. nördlicher Nachbar am unteren Rand der Karte. Analog befindet sich der linke Nachbar eines Feldes am linken Rand ganz rechts in der gleichen Zeile usw. Diese runde-Welt-Eigenschaft gilt für jegliche Aktionen von Dingen auf der Welt (Fortpflanzung, Bewegung etc.). Die Operation computeLifeCycle iteriert über alle Dinge die in der Welt aktuell existieren und lässt sie eine Aktion ausführen (performAction). Ein Ding führt dabei zunächst einmal nichts anderes aus als sein eigenes Alter um eins zu erhöhen. Alle weiteren Dinge erben von dieser Klasse und fügen Attribute und Verhaltensweisen hinzu.



Von Dingen zu Pflanzen

Thing ist die Basisklasse aller Dinge. Es enthält Informationen über das ASCII-Symbol mit welchem Instanzen dieser Klasse auf der Weltkarte angezeigt werden sowie das Alter. Das Alter wird in Spielrunden gerechnet- dazu später mehr. Die Klasse Plant (Pflanze) erbt von der Klasse Thing. Pflanzen können sich alle n Runden fortpflanzen indem auf einem freien Nachbarfeld ein neues Objekt dieser Klasse entsteht. Der feste Zyklus mit dem eine Pflanze dies tut ist ein Merkmal der Klasse. Klassen die davon erben können diesen Wert anpassen, aber das Verhalten der Fortpflanzung bleibt gleich. Im Sinne der Objektorientierung sind Thing und Plant abstrakte Klassen die zwar Attribute und Operationen bereitstellen aber nicht selbst instanziiert werden sollen. Daher soll nun die Klasse Corn von Plant erben um damit eine Maispflanze zu repräsentieren

Kreaturen

In diesem Stand gibt es nun also eine Welt (ein Objekt der Klasse World) auf der Instan zen der Klasse Corn existieren können. Die Instanzen der Klasse Corn sind zum einen in einem Dictionary abgelegt, welches vom Corn-Objekt auf dessen x/y Koordination zeigt. Zum anderen sind die Instanzen in einem 2-dimensionalen $m * n$ Array abgelegt. Bei jedem Aufruf von computeLifeCycle auf dem World-Objekt wird die performAction Operation jedes Thing-Objekts (bzw. den Erben) aufgerufen. Da der Mais in dieser Welt nicht aus Altersgründen stirbt oder gefressen wird füllt sich die Welt langsam aber si cher mit Mais. Damit das nicht zu langweilig wird, fügen wir den Welt nun noch Tiere hinzu. Die Klasse Creature erbt von Thing. Kreaturen bzw. Tiere haben analog zu Pflanzen einen festen Zyklus nachdem sie sich fortpflanzen. Da es manchmal schon recht voll auf der Spielwelt sein kann gibt es jedoch eine Besonderheit: Ein Nachkomme kann auf einem freien Nachbarfeld entstehen oder aber auf einem Nachbarfeld auf dem ein fressbares Ding steht. Dazu gleich mehr. Ist dennoch kein Platz frei kann wie bei Pflanzen in diesem Zyklus kein Nachkomme erzeugt werden. Darüber hinaus haben Tiere ein maximales Lebensalter. Sobald dieses erreicht ist, stirbt es und wird von der Welt entfernt. Damit ist wieder ein Feld auf der Karte frei. Aber nicht nur das Alter begrenzt das Leben, sondern auch der Hunger. Für jede Kreatur wird gespeichert wieviele Runden es nichts gefressen hat. Wird der, für Objekte einer Klasse festgelegte Maximalwert erreicht, stirbt die Kreature ebenfalls. Eine Kreatur muss also etwas essen. Dazu erhält die Klasse Creature noch ein Attribut welches alle Typen von Dingen aufzählt welche die Kreatur fressen kann. Wie Plant ist Creature im Grunde eine abstrakte Klasse die nicht selbst instanziiert wird. Eine Kreature kann sich fortpflanzen sowie an Alter oder Hunger sterben. Es braucht also einen Erben welcher der Sache Leben einhaucht. Wir lassen daher die Klasse Mouse von Creature erben. Sobald eine Maus in den vier Nachbarfeldern ein Ding findet was sie fressen kann, soll sie auf dieses Feld gehen. Das zu fressende Ding wird gelöscht und der Zähler der den Hunger der

Kreature misst wird wieder auf Null gesetzt. Gibt es nichts zu fressen zieht die Maus in ein zufällig gewähltes, freies Nachbarfeld. Ist kein Feld frei, bleibt sie einfach stehen

Programmablauf

Mit diesen Zutaten lässt sich bereits eine interessante Simulation erstellen. Der Pro grammablauf soll wie folgt gestaltet sein

- nach dem Programmstart erscheint zunächst eine Darstellung der noch leeren Welt in ASCII-Zeichen
- nun können über die Tastatur Befehle eingegeben werden die mit Return bestätigt werden
- "h" ruft eine Hilfeseite auf
- "q" beendet das Programm
- Spawn <number> <type> erzeugt <number> zufällig positionierte Objekte der Klasse <type> auf der Welt. Beispiel:spawn 20 Mouse. Die Weltkarte wird danach neugezeichnet.
- <return> lässt genau einen Berechnungszyklus laufen und zeichnet danach die Weltkarte neu
- <number> lässt soviele Zyklen nacheinander durchlaufen wie <number> angibt. Man muss also nicht für jeden einzelnen Zyklus <Enter> drücken

Tabelle 1: Empfohlene Parameter

attribute	value
mapWidth	79
mapHeight	24
seedCycle	6
offspringCycle	12
starvingMax	7
maxAge	25

Experimentieren Sie mit den Attributen der Pflanzen und Tiere damit alle Arten längerfristig überleben können. Als Grundlage können Sie die Werte aus Tabelle1 nutzen. Denn leicht passiert es, dass alle Mäuse verhungern, weil sie sich zu schnell vermehrt haben und allen Mais aufgefressen haben. Gern können Sie auch weitere Pflanzen und Tiere erstellen.

Meilenstein I

Beginnen Sie mit der Implementierung von World und Thing. Schreiben Sie Operationen um Dinge in die Welt einfügen und wieder entfernen zu können. Schreiben Sie Operationen um die Welt auslesen zu können. Dazu zählt zumindest eine Operation um abzufragen welches Objekt (oder keines = None) sich an einer gewünschten Position x/y befindet. Auch eine Operation zur Abfrage des nördlichen/östlichen/südlichen/westlichen Nachbars einer Position sind hilfreich. Schreiben Sie eine Operation mithilfe welcher Sie die Welt als ASCII-Karte in die Konsole ausgeben können (wie auf Abbildung 1 zusehen). Schreiben Sie als Teil der Abgabe eine Abfolge von Befehlen, die folgende Punkte umfassen. Testen Sie diese und dokumentieren Sie die Rückgabewerte

- Erstellung einer Instanz der Klasse World mit einer Breite von 20 und einer Höhe von 10 Feldern
- Einfügung eines Objekts der Klasse Thing an die Position y=0, x=15
- Einfügung eines Objekts der Klasse Thing an die Position y=9, x=15
- Abfrage des Objekts an Position y=0, x=15 und Ausgabe des Zeichens auf die Konsole (’.’ falls None)
- Abfrage des Objekts an der nördlichen Nachbarposition von y=0, x=15
- Löschung des Objekts an Position y=9, x=15
- Abfrage des Objekts an der nördlichen Nachbarposition von y=0, x=15
- Weltkarte auf der Konsole ausgeben

Meilenstein II

Implementieren Sie nun die Klassen Plant und Corn entsprechend der Aufgabenstellung. Erweitern Sie die Klasse World um eine Operation computeLifeCycle welche der Reihe nach die performAction-Operation aller Objekte der Welt aufruft. Gehen Sie dabei nicht so vor, dass sie über die Karte iterieren sondern verwenden Sie als Grundlage das Dictionary welches jedem Objekt in der Welt seine Koordination zuordnet. Andernfalls sind die linken-oberen Dinge immer im Vorteil, weil sie zuerst dran sind. Idealerweise wird die Reihenfolge der performAction-Aufrufe der Objekte in jeder Runde zufällig neu bestimmt. Ergänzen Sie das Programm um eine einfache Menüfunktion wie in der Sektion 5.5 beschrieben. Testen Sie Ihr Programm anhand folgender Schritte:

- Start des Programms mit einer Karte von 79 Feldern Breite und 29 Feldern Höhe.
- Nutzen Sie Ihre Menüfunktion um zufällig 20 Objekte der Klasse Corn einzufügen.
- Lassen Sie die Simulation schrittweise einige dutzend Runden laufen. Die Karte sollte sich langsam mit Maus füllen

Meilenstein III

Implementieren Sie nun abschließend die Klassen Creature und Mouse. Testen Sie Ihr Programm anhand folgender Schritte:

- Start des Programms mit einer Karte von 79 Feldern Breite und 29 Feldern Höhe.
- Nutzen Sie Ihre Menüfunktion um zufällig 100 Objekte der Klasse Corn einzufügen.
- Nutzen Sie Ihre Menüfunktion um zufällig 100 Objekte der Klasse Mouse einzufügen
- Lassen Sie die Simulation schrittweise einige dutzend Runden laufen. Das Verhalten der Mäuse sollte der Aufgabenstellung entsprechen. Wenn die Simulation stabil läuft ändern sich die Populationsgrößen von Mais und Mäusen über die Zeit zyklisch. Es kann aber auch passieren, dass die Mäuse oder auch der Mais aussterben.