UNIVERSITÄT BAYREUTH

Lehrstuhl für Angewandte Informatik III
(Robotik und Eingebettete Systeme)
Prof. Dr. Dominik Henrich

Bachelorpraktikum AI3

Aufgabe 1: Einlesen einer Textdatei

Gegeben ist eine Textdatei mit biologischen Daten verschiedener Kreaturtypen. Jeder Eintrag umfasst eine Zeile. Die einzelnen Daten (Felder) eines Kreaturtyps sind durch einen Komma getrennt. Vor dem ersten Feld und nach dem letzten Feld steht kein Komma. Es kann leere Felder geben (erkennbar durch zwei aufeinanderfolgende Komma oder einen Komma am Anfang bzw. Ende der Zeile).

Jede Zeile enthält die folgenden Felder in der angegebenen Reihenfolge:

```
Kreaturname,Stärke,Geschwindigkeit,Lebensdauer,Eigenschaften,Bild
```

Die Felder genügen den folgenden Beschränkungen:

- "Kreaturname" darf nur normale Buchstaben sowie Leerzeichen enthalten und muss angegeben werden. (bsp: Kreaturname = Hund)
- "Stärke", "Geschwindigkeit", und "Lebensdauer" dürfen nur positive Ganzzahlen sein.
- "Eigenschaften" beinhaltet beliebig viele, leerzeichen-getrennter Wörter, aus den Buchstaben a-zA-Z, Unterstrich, und Zahlen. (bsp: Eigenschaften = pflanze wasserbewohner)
- "Bild" muss ein relativer Dateipfad nach POSIX-Standard sein. (bsp: Bild = kreaturen/tannenbaum.jpg)

Schreiben sie ein Programm zum Einlesen einer Datei diesen Formats. Sind Fehler in einer Zeile, soll diese Zeile ignoriert werden, eine entsprechende Meldung über den Fehlerort und -typ auf der Konsole ausgegeben werden und die Datei weiter eingelesen werden bis zu ihrem Ende. Treten Fehler beim Öffnen der Datei auf, so soll auch dies gemeldet werden.

Den Dateinamen zum Einlesen erhält dieses Programm als ersten und einzigen Parameter auf der Kommandozeile in der main-Methode (siehe argc, argv).

Geben Sie nach dem Einlesen statistische Daten an: Anzahl der insgesamt eingelesenen Zeilen, Anzahl der korrekten und der fehlerhaften Zeilen, etc.

Ebenso sollen die Daten nach dem Einlesen in einer geeigneten Datenstruktur im Hauptspeicher für die Weiterverarbeitung abgelegt werden. Diese Datenstruktur sollte insbesondere ihre Elemente (also einzelne Kreaturtypen) selbständig verwalten, und automatisch löschen.

Der Code ist zweiteilig zu dokumentieren: Eine Header-Dokumentation für Klassen und Funktionseinstiegspunkte, sowie eine Source-Dokumentation für Fallstricke in der Implementierung. Triviale Dokumentation ist nicht erwünscht, die Dokumentation sollte auf Codeblöcken statt auf Codezeilen gründen.

Nach Abschluss der Programmierarbeiten ist der Code aufzuräumen. Insbesondere ist eine konsistente Konvention für Variablen-, Funktions- und Klassennamen, für Einrückung, Klammernsetzung, sowie Whitespace unerlässlich. Unlesbarer oder unübersichtlicher Code wird nicht akzeptiert.

Hinweise:

- Bereits für diese Aufgabe ein strukturiertes Projekt aufzusetzen ist nicht nötig, spart später aber viel Arbeit.
- Verwenden Sie zum Umgang mit Daten und Dateien nach Möglichkeit die Klassen und Methoden der Standardbibliothek (vector, list, fstream, getline, string, stringstream).
- Achten Sie beim Anlegen Ihrer Klassen und Datenstrukturen auf eine saubere Architektur. Insbesondere soll das Hinzufügen neuer Spalten bzw. das Ändern von Spaltenbeschränkungen ohne viel Aufwand realisierbar sein.
- Achten Sie auf korrektes C++. Dazu gehören korrekter Umgang mit Speicher, Ausnahmen, und Stackvariablen. Speicherlecks sind zu vermeiden.
- Achten Sie auf sinnvolles Klassendesign: Member-Methoden, private/public-Spezifikationen, statische Lademethoden, Ausnahmen für Fehler.
- Sinnvolle Google-Stichwörter: STL, vector, list, destructor, constructor, const reference, pass-by-value, pass-by-reference, exception safety

UNIVERSITÄT BAYREUTH

Lehrstuhl für Angewandte Informatik III
(Robotik und Eingebettete Systeme)
Prof. Dr. Dominik Henrich

Bachelorpraktikum AI3

Aufgabe 2: Einlesen einer Bilddatei in den Arbeitsspeicher

Ziel dieser Aufgabe ist das Einlesen einer Bilddatei in den Arbeitsspeicher.

Rahmenbedingungen

a) Die Bilddatei muss im TGA-Format gelesen werden können.

b) Das Resultat des Einlesevorgangs soll eine Instanz einer ebenfalls zu entwickelnden Bild-Klasse sein. In dieser Klasse sollen RGB(A)-Daten mit einem Byte pro Farbkanal und Pixel vorgehalten werden. Der Ladevorgang soll durch statische Methoden innerhalb der Bild-Klasse umgesetzt werden.

c) Beim Schreiben der Bild-Klasse ist ein C++-übliches Vorgehen zu nutzen, insbesondere das RAII-Pattern (Resource acquisition is initialization) zum Vermeiden ungültiger Klassenzustände. In anderen Worten, eine Konstruktion eines Bildes soll bereits vollständige Parameter erfordern. Fehlerzustände müssen im Konstruktor erkannt und ungültige Klasseninstanzen durch Ausnahmebehandlung vermieden werden. Die Bildklasse soll Speicher intern automatisch verwalten und bei Zerstörung eines Bildobjekts automatisch freigeben.

d) Korrupte oder nicht unterstützte Bilddaten sind zu erkennen und über geeignete Mechanismen (Exceptions) zu propagieren. Speicherlecks sind zu vermeiden, ebenfalls sind offene Dateien korrekt zu schliessen.

e) Redundanzen im Code sind durch geschickte Funktionsaufteilung zu vermeiden. Alle Variablen, Methoden, und Klassen sind entsprechend dem Anwendungs-Kontext sinnvoll zu benennen. Dokumentation und Formatierung analog zur ersten Aufgabe.

Einlesen einer TGA-Datei

Das Einlesen einer TGA-Datei ist von Hand zu implementieren, es darf keine Bibliothek genutzt werden. Falls Tutorials aus dem Netz verwendet werden, ist der Code dennoch selbst neu zu schreiben. Copy und Paste ist nicht zulässig, und fälllt spätestens bei Detailfragen im Testat auf.

Eine Beschreibung des TGA-Formates findet sich unter:

http://de.wikipedia.org/wiki/Targa_Image_File

Es müssen nicht alle möglichen TGA-Dateien und TGA-Eigenschaften unterstützt werden, sondern nur die folgenden Wertebelegungen:

Länge Bild-ID = 0
Farbpalettentyp, Palettenbeginn, Palettenlänge = 0
Bildtyp = 2
X, Y-Nullpunkt = 0
Bits pro Bildpunkt = 24 oder 32

Nach dem Einlesen des Bilddaten-Blocks mit Pixelfarben kann der Einlesevorgang abgebrochen werden. Weitere Metainformationen sind nicht zu lesen.

Google-Stichwörter

ifstream binary, auto_pointer, vector, tga tutorial, RAII

UNIVERSITÄT BAYREUTH

Lehrstuhl für Angewandte Informatik III
(Robotik und Eingebettete Systeme)
Prof. Dr. Dominik Henrich

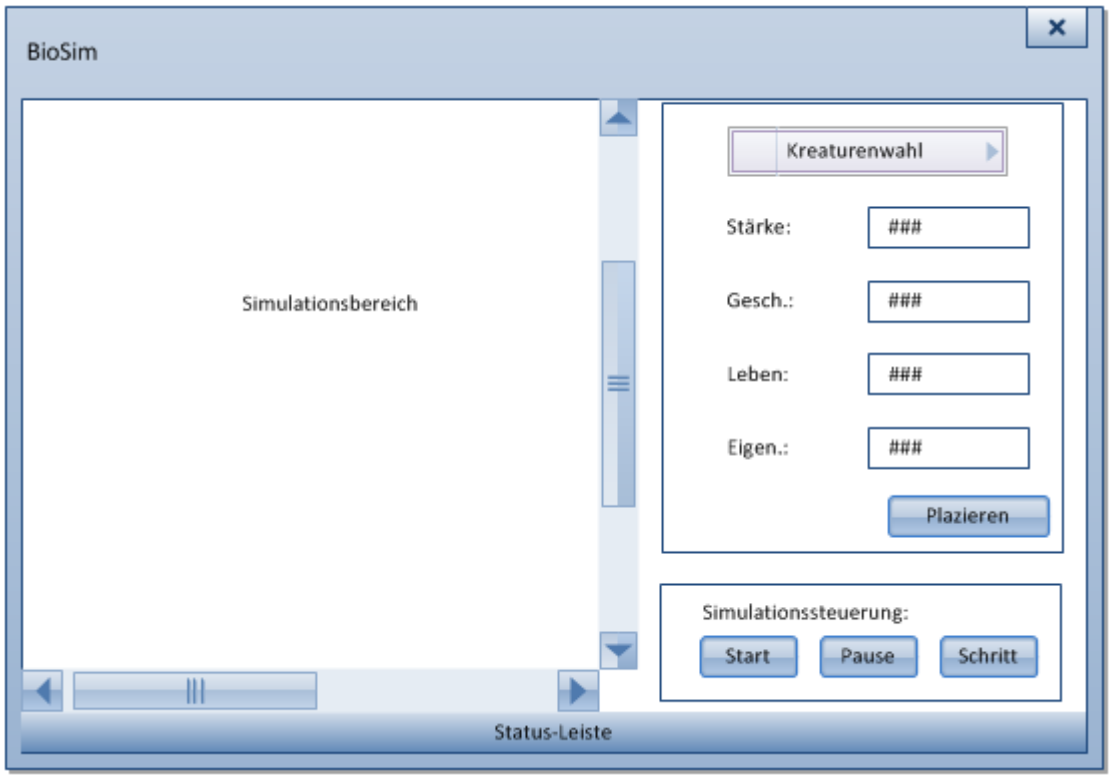
Bachelorpraktikum AI3

Aufgabe 3: Entwicklung einer GUI für die Simulationsumgebung

Ziel dieser Aufgabe ist das Erstellen einer GUI für die spätere Simulationsumgebung.

GUI-Muster

Die GUI soll zu Beginn die folgenden Eingabemöglichkeiten bieten:



Initialfunktionalität

In der ersten Implementierung soll die GUI folgende Funktionen bereitstellen:

- Befüllen der Kreaturenwahl-Liste mit Kreaturentypen aus der Eingabedatei, Vorbelegung mit dem ersten Kreaturentyp aus der Liste
- Eintragen der Attribute einer Kreatur in die zugehörigen Felder nach Auswahl in der Kreaturenliste
- Reagieren auf Anklicken bei den verbleibenden Knöpfen (z.B. durch Darstellen eines Nachrichtendialogs mit dem Namen des gewählten Knopfes)
- Programmende über den Exit-Knopf

- Fehlermeldung und Programmende, falls beim Start ein Fehler auftritt

Der GUI-Code selbst soll durch Wahl geeigneter Software-Entwurfsmuster modular gestaltet werden. Es bieten sich an:

- Model-View-Presenter als Klassen-Modell
- Signal/Slot-System oder Event-System für die GUI-Logik

Noch nicht zu implementieren ist die weitergehende Simulationsfunktionalität, insbesondere:

- Darstellung der virtuellen Welt im Simulationsbereich
- Reagieren auf die Knöpfe „Start“, „Pause“, „Schritt“, „Bildschirmausschnitt verschieben“, „Kreatur plazieren“, ...

Analog zu vorangehenden Aufgaben ist bei der Implementierung ist auf korrektes C++, vollständige und sinnvolle Dokumentation, sowie saubere Formatierung und Strukturierung zu achten.

Wahl der GUI-Bibliothek

Im Rahmen dieser Aufgabe ist keine eigene GUI-Bibliothek zu erstellen, sondern eine bestehende GUI-Bibliothek zu nutzen. Je nach Betriebssystem bieten sich verschiedene Lösungen an.

- **QT:** systemunabhängig, IDE-Integration schwierig, zusätzlicher Build-Schritt
- **wxWidgets:** systemunabhängig, externes Lib, kein zusätzlicher Build-Schritt
- **GTK:** systemunabhängig, Einbindung in Windows umständlich
- **FLTK:** systemunabhängig, leicht veraltet
- **Win32-API:** nur Windows, kann direkt verwendet werden, systemnah
- **X-Window System:** nur Linux, kann direkt verwendet werden, systemnah

Vor der Wahl einer Bibliothek empfiehlt es sich, die verfügbaren Tutorials für alle Kandidaten zu überfliegen, und dann nach Bauchgefühl zu entscheiden.

Hinweis: Für viele der genannten Bibliotheken gibt es bereits bestehende Editoren, in denen die spätere GUI über eine einfache Benutzerschnittstelle zusammengestellt werden kann.

Google-Stichwörter

<Name der GUI-Bibliothek>, Model-View-Presenter, Signal-Slot-Mechanismen, Event Handling

UNIVERSITÄT BAYREUTH

Lehrstuhl für Angewandte Informatik III
(Robotik und Eingebettete Systeme)
Prof. Dr. Dominik Henrich

Bachelorpraktikum AI3

Aufgabe 4: Virtuelle Umgebung

Ziel dieser Aufgabe ist das Erstellen, Verwalten, und Zeichnen der virtuellen Umgebung für die Biologiesimulation.

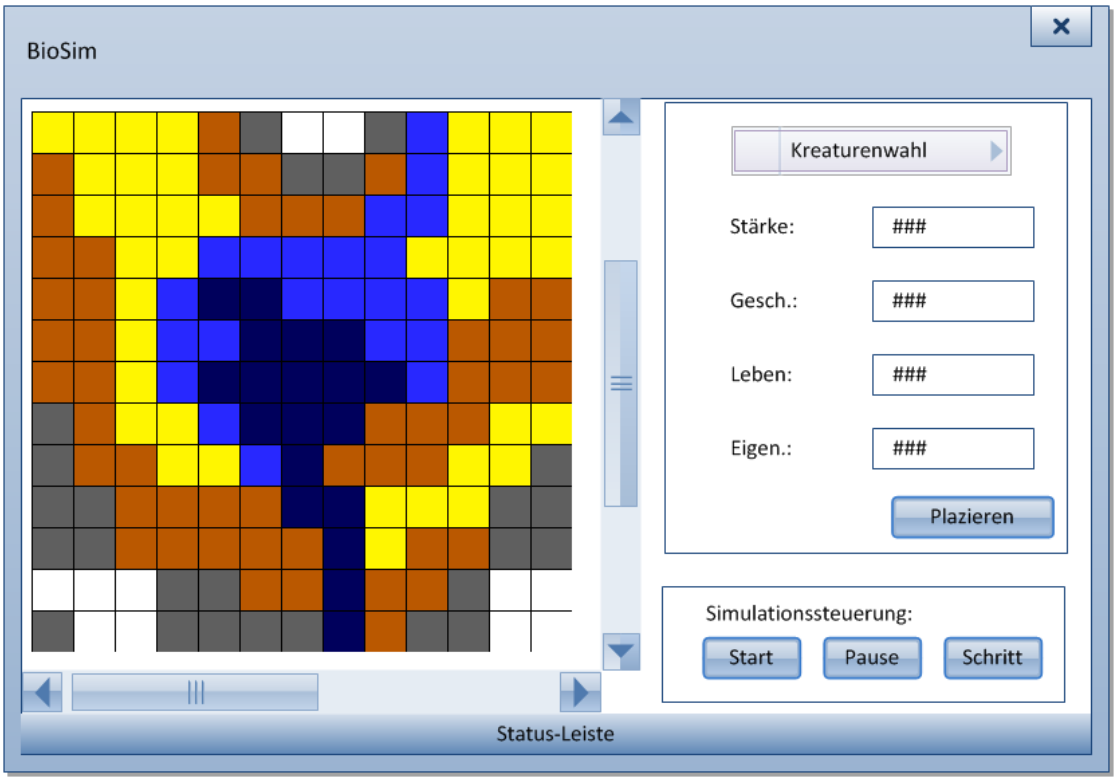
Grundlagen

Die Landschaft ist als ein zweidimensionales Gitter zu realisieren. Jede Gitterzelle entspricht dabei einem Feld („Kachel“), auf dem sich eine oder mehrere Kreaturen aufhalten können. Jede Kachel hat ferner einen zugeordneten Klima-Typ.

Die folgenden sechs Klimata sollen unterstützt werden: Tiefes Wasser, seichtes Wasser, Sand, Erde, Steine, Schnee

Die Größe der Landschaft ist nicht fest ins Programm zu integrieren, sondern soll über eine geeignete Variable zur Compilezeit gewählt werden können.

Die folgende Skizze zeigt exemplarisch eine kleine Kachel-Landschaft im Simulationsfenster:

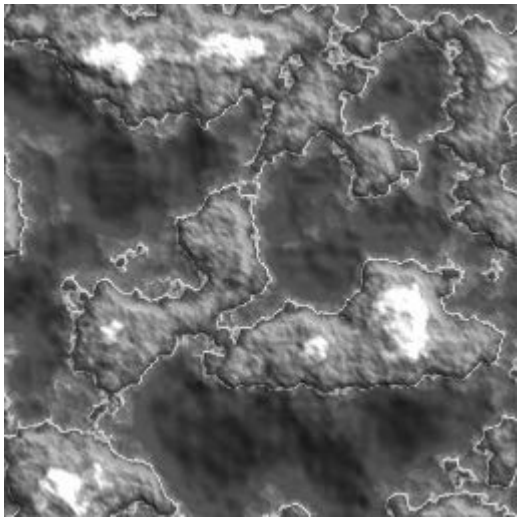


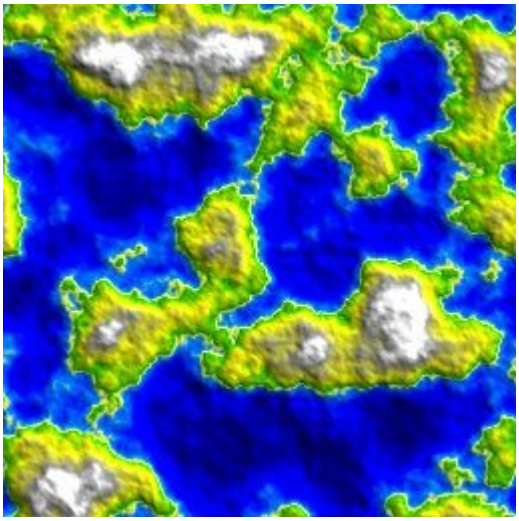
Erstellen der Landschaft

Um bei jedem Start der Simulation einen anderen Simulationsablauf mit jeweils neuen Herausforderungen zu generieren, ist die Landschaft zufällig zu erstellen.

Um ein realistisches Landschaftsbild zu erhalten, ist es jedoch nicht ausreichend, über einen Zufallsgenerator für jedes Gitterfeld einen beliebigen Landschaftstypen zu wählen. Stattdessen muss eine andere, geeignete Zufallsfunktion herangezogen werden. Hier bietet sich die Zufallsfunktion Perlin Noise an.

Die Anwendung von Perlin Noise ergibt im Regelfall für jede Kachel einen einzelnen Ganzzahlwert. Durch passende Assoziation von Wertebereichen mit Klimazonen lassen sich die Ganzzahlwerte schliesslich auf Klimatypen abbilden. Die folgenden Bilder zeigen als Beispiel links ein Ergebnis von Perlin Noise, und rechts eine über Wertebereiche zugeordnete Klima-Einfärbung.





Bildquelle:
http://libnoise.sourceforge.net/tutorials/images/newheightmap.jpg

Verwalten der Landschaft

Für das Verwalten der Landschaft im Arbeitsspeicher ist eine geeignete Datenstruktur zu nutzen.

Dabei ist zu bedenken, dass später auf jedem Feld der virtuellen Landschaft eine oder mehrere Kreaturen stehen können. Ferner wird in späteren Aufgaben (z.B. zur Bahnplanung) ein wahlfreier Zugriff auf einzelne Felder in Abhängigkeit von (x, y)-Positionen erforderlich sein. Achten Sie schliesslich darauf, eine möglichst speicherkompakte Repräsentation zu wählen, um auch große und sehr große Karten zu unterstützen.

Die Datenstruktur ist gemäß dem Model-View-Presenter-Muster in das Modell der bestehenden Anwendung zu integrieren.

Darstellung der Landschaft

Schliesslich soll die Landschaft im Simulationsfenster der bestehenden GUI-Anwendung dargestellt werden.

Als vorbereitender Schritt müssen bei Programmstart zuerst die Grafiken für die Klimakacheln in den Arbeitsspeicher geladen werden. Dazu ist der bestehende Algorithmus zum Einlesen von Bildern zu nutzen. Wichtig ist hier, jedes Eingabebild nur einmal einzulesen und nur einmal im Speicher zu halten. Passende Klimata-Bilder finden sich im eLearning-Kurs.

Beim Zeichnen der Landschaft wird nun die Datenstruktur aus dem Modell traversiert und über die Benutzeroberfläche dargestellt. Beim Darstellen der Kacheln kann für jedes Vorkommen eines Klimatyps die entsprechende, bereits vorgehaltene Grafik gewählt werden. Zum Zeichnen selbst bietet es sich an, bestehende Grafik-Funktionalität der ausgewählten GUI-Bibliothek zu nutzen. Je nach Bibliothek kann das entweder den Einsatz von OpenGL oder die Arbeit mit bibliotheksspezifischen Funktionen bedeuten.

Bei großen Landschaften darf im Simulationsfenster nur ein Ausschnitt der gesamten Karte dargestellt werden. Um exzessivem Rechenaufwand vorzubeugen, dürfen in diesem Fall nur Kacheln berücksichtigt werden, die sich zumindest teilweise innerhalb des Kartenausschnitts befinden. Damit fallen alle Lösungen weg, die einmalig bei Programmstart eine Grafik des gesamten Terrains erzeugen. Stattdessen muss dynamisch nach jeder Scrolloperation der Bildausschnitt nur mit den aktuell sichtbaren Kacheln gefüllt werden.

Eine Verschiebung des aktuellen Karten-Ausschnitts muss über die Scrollbalken im Simulationsfenster ermöglicht werden. Dabei soll das Verhalten den üblichen GUI-Standards entsprechen. Beispielsweise soll die Größe der Scrollbalken einen Aufschluss über die Grösse der aktuellen Ansicht relativ zur gesamten Umgebung geben. Das Scrolling soll weich auf Pixelbasis erfolgen, nicht mit Kachelsprüngen.

Rahmenbedingungen

Analog zu vorangehenden Aufgaben ist bei der Implementierung ist auf korrektes C++, vollständige und sinnvolle Dokumentation, sowie saubere Formatierung und Strukturierung zu achten.

Google-Stichwörter

enum, perlin noise, <Name der GUI-Bibliothek> graphics, OpenGL, blitting, tile sets

UNIVERSITÄT BAYREUTH

Lehrstuhl für Angewandte Informatik III
(Robotik und Eingebettete Systeme)
Prof. Dr. Dominik Henrich

Bachelorpraktikum AI3

Aufgabe 5: Integration von Kreaturen

Ziel dieser Aufgabe ist das Laden von Kreaturgrafiken, sowie das Verwalten, Plazieren, und Zeichnen von Kreaturinstanzen.

Laden von Kreaturgrafiken

Beim Einlesen der Kreaturenliste soll die zugehörige Grafik für jeden Kreaturentyp eingelesen und sinnvoll im Arbeitsspeicher abgelegt werden.

Die einzulesenden Kreaturgrafiken liegen im TGA-Format vor (siehe Archiv im eLearning-Kurs). Um mehrere Kreaturen übereinander auf der selben Kachel darstellen zu können, ist es unerlässlich, zusätzlich zu den RGB-Farbwerten auch einen Alpha-Kanal zu unterstützen. Der Alpha-Kanal definiert die Transparenz eines jeden Bildpixels, und wird als zusätzlicher Byte-Wert neben den RGB-Bytes in den Eingabebildern gespeichert.

Dementsprechend ist der bestehende Code zum Laden einer TGA-Datei um eine alternative Lademöglichkeit für RGBA-TGA-Dateien zu erweitern. Beachten Sie, dass der TGA-Header ausgewertet werden muss, um RGB- von RGBA-Bildern zu unterscheiden. Insbesondere muss ein Laden von TGA-Dateien im RGB-Format weiterhin möglich bleiben.

Verwalten von Kreaturinstanzen

Zum Verwalten von Kreaturinstanzen sind insgesamt drei Änderungen an dem Programm erforderlich:

- Erzeugen einer Klasse, die eine Instanz eines gegebenen Kreaturtyps repräsentiert. Insbesondere sollen hier die aktuellen Werte einer besonderen Kreatur vorgehalten werden, also etwa die aktuelle Stärke oder aktuelle Eigenschaften. Auf sinnvolle Datentypen für eine spätere Nutzung durch die Simulationslogik ist zu achten (z.B. keine Strings für Zahlenwerte, und eine passende Datenstruktur für die Eigenschaften).
- Hinzufügen einer Liste aller Kreaturinstanzen zum Anwendungsmodel. Insbesondere ist es für die spätere KI wichtig, alle Kreaturen nacheinander durchgehen zu können, ohne dabei über die Kacheln der Simulationsumgebung suchen zu müssen.
- Hinzufügen einer Liste aller Kreaturinstanzen auf einer bestimmten Anwendungskachel. Diese Liste wird zur schnellen Darstellung sowie für manche KI-Operationen benötigt.

Beachten Sie, daß Kreaturinstanzen von verschiedenen Stellen aus referenziert werden können. Um spätere Probleme beim Löschen von Kreaturen zu vermeiden, sollten Sie eine geeignete Strategie zur Verwaltung der Lebenszeit von Kreaturinstanzen finden.

Plazieren von Kreaturinstanzen

Zum Plazieren von Kreaturinstanzen muss zuerst eine geeignete Möglichkeit zur Wahl einer Zielkachel implementiert werden. Konkret soll der Anwender durch Klick in die Simulationsumgebung eine Kachel auswählen können. Diese Kachel wird als aktive Kachel markiert, und ein entsprechendes Cursor-Symbol (siehe eLearning) wird über der Kachel dargestellt.

Beim Klick auf den Plazieren-Knopf wird schliesslich eine Instanz der aktuell gewählten Kreatur erzeugt und auf der aktiven Kachel abgesetzt.

Zeichnen von Kreaturinstanzen

Nach dem Zeichnen jeder sichtbaren Umgebungskachel sollen alle Kreaturen dieser Kachel ebenfalls gezeichnet werden. Dabei sollen aus Performancegründen nur Kreaturinstanzen berücksichtigt werden, die zur entsprechenden Kachel gehören. Insbesondere ist zu keiner Zeit die Liste aller Kreaturen aus dem Anwendungsmodell zu nutzen.

Sind mehrere Kreaturinstanzen auf einer Kachel, so ist die Zeichenreihenfolge beliebig. Der Cursor für die aktive Kachel sollte jedoch immer über allen Kreaturen dargestellt werden.

Rahmenbedingungen

Analog zu vorangehenden Aufgaben ist bei der Implementierung ist auf korrektes C++, vollständige und sinnvolle Dokumentation, sowie saubere Formatierung und Strukturierung zu achten.

UNIVERSITÄT BAYREUTH

Lehrstuhl für Angewandte Informatik III
(Robotik und Eingebettete Systeme)
Prof. Dr. Dominik Henrich

Bachelorpraktikum AI3

Aufgabe 6: Wegfindung

In dieser Aufgabe ist die Bewegungsplanung für Kreaturen vorzubereiten.

Die Wegfindung von Kreaturen soll auf Basis des a*-Verfahrens durchgeführt werden. Eine grundlegende Beschreibung dieses Algorithmus findet sich beispielsweise in der Wikipedia unter http://de.wikipedia.org/wiki/A*-Algorithmus.

Der Grundalgorithmus ist für die Anwendung in der Bio-Simulation anzupassen:

1. Eingabe des Algorithmus ist eine Kreatur, sowie die Zielkachel der Kreatur.
2. Ausgabe des Algorithmus ist eine Folge von benachbarten Kacheln, die die Kreatur auf dem kürzesten Weg bis zum Zielort durchschreiten muss.
3. Als Knoten werden in der Bio-Simulation die Kacheln der aktuellen Umgebungskarte genutzt.
4. Die Gewichte der Kanten im A*-Algorithmus werden über auswechselbare Tabellen berechnet. Dabei gibt jeder Eintrag einer Tabelle die Schwierigkeit an, ein bestimmtes Gebiet zu betreten. Ein Eintrag von -1 bedeutet, dass das entsprechende Gelände nicht betreten werden kann. Die zu verwendenden Tabellen hängen von der Art der Kreatur ab, zu finden bei den Kreatureneigenschaften. Beispieltabellen stehen untenan.
5. Eine geeignete obere Schranke für zu betrachtende Kacheln ist z.B. über eine statische Konstante festzulegen. Kann innerhalb dieser Schranke kein Weg zwischen Kreaturenposition und Ziel gefunden werden, gibt der Algorithmus eine Fehlermeldung „unerreichbar“ an den Aufrufer zurück.

Zu Testzwecken ist es erforderlich, den Algorithmus an das Auswählen einer aktiven Kachel anzuhängen. Dabei wird jeweils ein Weg zwischen der vorherigen und der aktuellen aktiven Kachel geplant und grafisch dargestellt.

Tabelle für Landbewohner:

T. Wasser	S. Wasser	Sand	Erde	Steine	Schnee
-1	-1	1	1	4	2

Tabelle für Wasserbewohner:

T. Wasser	S. Wasser	Sand	Erde	Steine	Schnee
3	1	-1	-1	-1	-1

Rahmenbedingungen

Analog zu vorangehenden Aufgaben ist bei der Implementierung ist auf korrektes C++, vollständige und sinnvolle Dokumentation, sowie saubere Formatierung und Strukturierung zu achten.

UNIVERSITÄT BAYREUTH

Lehrstuhl für Angewandte Informatik III
(Robotik und Eingebettete Systeme)
Prof. Dr. Dominik Henrich

Bachelorpraktikum AI3

Aufgabe 7: Künstliche Intelligenz

Ziel dieser Aufgabe ist die Implementierung der Simulationslogik.

Grundlegend läuft die Simulation in einzelnen Simulationsschritten ab. Ein einzelner Simulationsschritt entspricht einer Runde eines Brettspiels: Jede Kreatur muss eine bestimmte Aktion durchführen. Mögliche Aktionen sind beispielsweise Bewegung, Fortpflanzung, Nahrungsaufnahme, oder Ausruhen. Manche Aktionen haben Folgen für den Simulationszustand - etwa die Verschiebung der Kreatur auf der Karte oder den Verbrauch von Lebenszeit.

Die Aktionswahl einer Kreatur wird von verschiedenen Parametern bestimmt, und lässt sich als endlicher Automat darstellen. Passende Automaten für die Simulation finden sich am Ende dieser Aufgabenstellung.

Die programmatische Realisierung der KI besteht aus drei Teilaufgaben:

1. **Aktionsdurchführung für eine einzelne Kreatureninstanz**
   Bestimme auf Grundlage des aktuellen Kreatur- und Simulationszustands die nächste Aktion, und führe diese durch. Hierbei sind die untenstehenden Automaten anzuwenden.
2. **Einzelschritt-Logik**
   Bei Klick auf den Schritt-Knopf in der GUI soll für jede Kreatur ein einzelner Aktionsschritt durchgeführt werden. Die Aktionsreihenfolge innerhalb der Kreaturen ist dabei beliebig, muss aber über alle Simulationsschritte konstant bleiben. Nach Abarbeitung des Schrittes für alle Kreaturen muss die Darstellung im GUI-Fenster aktualisiert werden.
3. **Animationslogik**
   Nach Klick auf den Start-Knopf soll in festem Zeitabstand (beispielsweise 1s, statische Konstante) jeweils ein Einzelschritt durchgeführt werden. Über den Pause-Knopf lässt sich dieses Verhalten jederzeit stoppen.

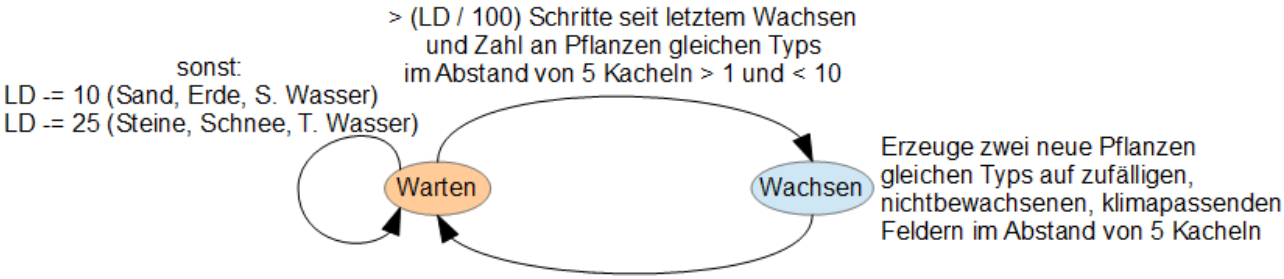
   Während die Simulation abgespielt wird, soll kein weiterer Einzelschritt durch den Schritt-Knopf erlaubt werden.

   Hinweis: Vermeiden Sie eine thread-basierte Lösung! In der Regel bietet Ihre GUI-Bibliothek bessere Alternativen, z.B. Timer-Objekte. Falls Sie dennoch mit Threading arbeiten wollen, müssen Sie geeignete Synchronisierungs-mechanismen nutzen.

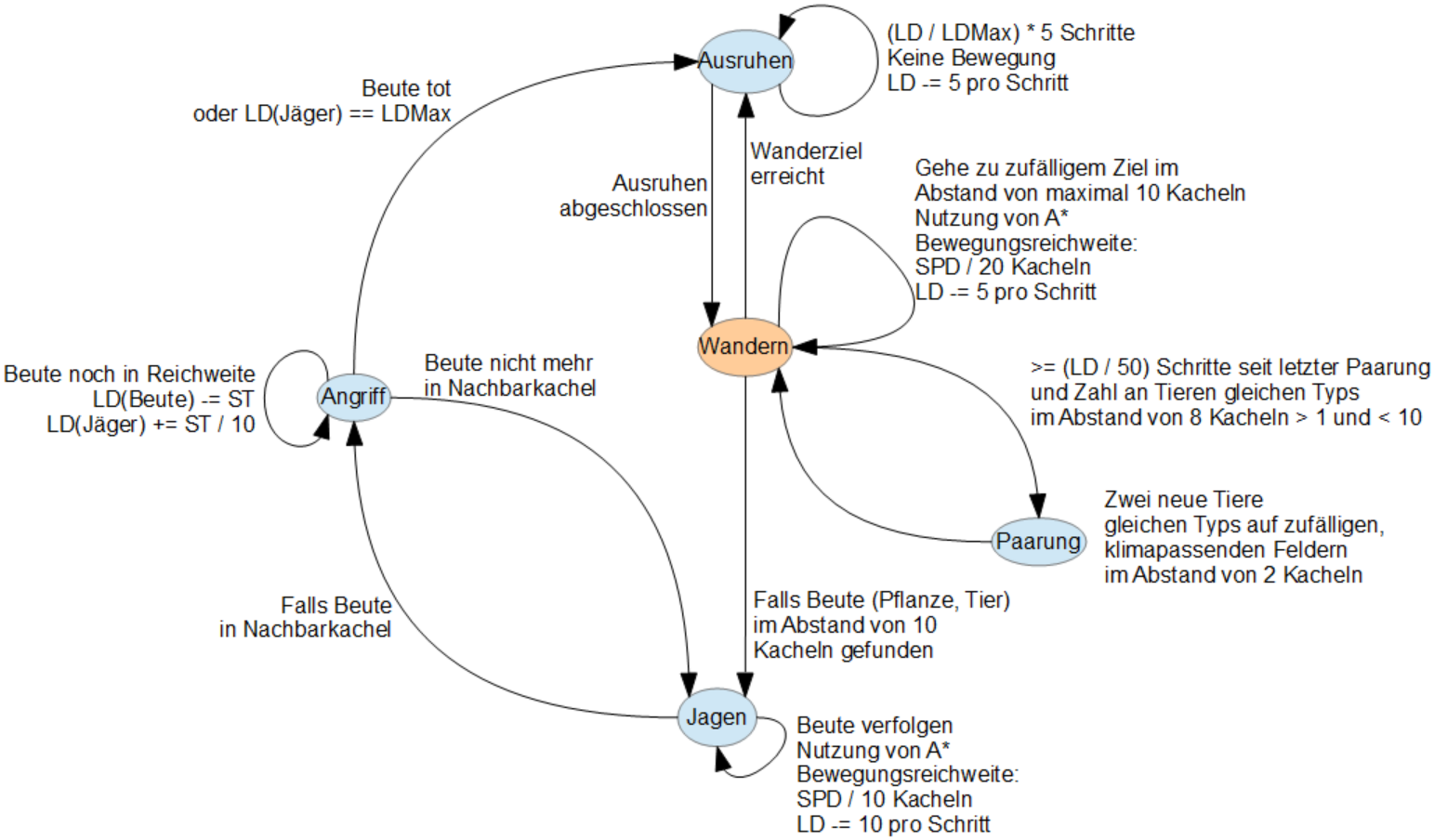
KI-Automaten

Im Folgenden finden Sie die zu implementierenden Automaten.

Pflanzen verhalten sich wie folgt:

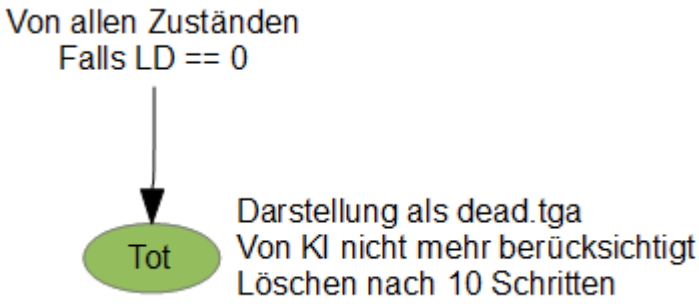


Alle Tiere nutzen einen anderen Automaten:



Dabei unterscheiden sich die beiden Tiertypen lediglich durch die Wahl von Beutetieren: Bei Pflanzenfressern werden nur Pflanzen als Beute berücksichtigt, Fleischfresser berücksichtigen nur Tiere als Beute.

Schliesslich modelliert ein letzter Teilautomat das Ausscheiden und Löschen gestorbener Kreaturen:



Bedingungen für die Wahl einer Aktion (Eingaben der Automaten) sowie Folgen einer Aktion (Ausgaben des Automaten) stehen jeweils informell bei den Aktionspfeilen oder Zuständen. Die Startzustände sind in allen Automaten gelb, Endzustände grün markiert. Ferner werden die Abkürzungen LD (=verbleibende Lebensdauer), LDMax (=Maximale Lebensdauer aus der Kreaturenliste), SPD (=Geschwindigkeit), und ST (=Stärke) genutzt.

Rahmenbedingungen

Analog zu vorangehenden Aufgaben ist bei der Implementierung ist auf korrektes C++, vollständige und sinnvolle Dokumentation, sowie saubere Formatierung und Strukturierung zu achten.

UNIVERSITÄT BAYREUTH

Lehrstuhl für Angewandte Informatik III
(Robotik und Eingebettete Systeme)
Prof. Dr. Dominik Henrich

Bachelorpraktikum AI3

Aufgabe 8: Test

In dieser Aufgabe ist die abgeschlossene Simulationsumgebung ausgiebig zu testen.

Insbesondere sind die folgenden Punkte zu testen:

1. **Fehlerarmut der Anwendung** (z.B. kein Absturz bei Tod einer Kreatur)
2. **Verhalten der Anwendung unter Last** (z.B. viele Kreaturen und große Karten)
3. **Semantik der Anwendung** (z.B. Balance der Kreaturen)

Überlegen Sie sich selbst, wie Ihre Anwendung möglichst sinnvoll in diesen Punkten zu testen ist. Zeichnen Sie Ihre Überlegungen in Form eines strukturierten, tabellarischen, und umfangreichen Testplans schriftlich auf. Führen Sie die entsprechenden Tests durch, und halten Sie die Ergebnisse im Testplan fest.

Eventuell auftretende Programmfehler vom Typ 1 sind zu korrigieren. Für Fehler vom Typ 2 ist keine Korrektur nötig, jedoch muss ein entsprechender Vorschlag zur Optimierung im Testplan festgehalten werden. Probleme vom Typ 3 sind lediglich festzuhalten, eine weitergehende Analyse ist nicht nötig.