Dokumentation zur Projektaufgabe für das Modul Künstliche Intelligenz für Spiele im Wintersemester 2022/23

Oliver Jakobs

16. Februar 2023

Thema und Motivation

Als Thema für meine Modulaufgabe habe ich mich für **Decision Trees** entschieden.

Einlesen eines Decision Trees aus einer XML-Datei und Frage-Antwort Spiel oder textbasiertes Adventure.

Das Konzept lässt sich neben künstlicher Intelligenz auch auf viele andere Bereiche anwenden. Ein Beispiel ist eine Alternative zu Zustandsautomaten um festzulegen welche Animation gespilet werden soll.

Allgemeine Designentscheidungen

Mein Programm lässt sich in zwei Abschnitte aufteilen. Der erste Teil ist die Implementiation des Decision Tress. Dieser wird aus einer .xml-Datei eingelesen und in eine passenden Datenstruktur geladen. Den zweiten Teil habe ich TreeWalker gennant. Dieser Teil beschäftigt sich mit Kommunikation zwischen dem Nutzer und dem Baum.

Das Programm habe ich bewusst im C-Style programmiert. Das heißt ich habe mich bei der Struktur des Programmes an C gehalten und habe lediglich einzelene Features (z.B. `std::vector` oder `std::variant`) aus C++ verwendet. Der Grund für diese Entscheidung war, dass ich privat hauptsächlich in C programmiere und so am komfortabelsten mit diesem Stil bin. Außerdem habe ich so weniger Aufwand, wenn ich die Implementation anpassen muss, wenn ich Decision Trees in meinen C-Programmen verwenden will.

Entscheidungen zur DecisionTree Implementation

Der DecisionTree wird aus mehreren TreeNodes aufgebaut. So eine TreeNode hat einen Type, einen Namen, einen Wert und eine Liste an Nachfolgern. Anhand der Types werden folgende Arten von TreeNodes unterschieden:

Decision
TreeNodes mit dem Type `decision` bilden den Grundbaustein meiner Implementation. An sich lässt sich alle Funktionalität von DecisionTrees nur mit TreeNodes dieser Art umsetzen. Der Wert einer `decision` ist eine Boolean Expression, in die zum Zeitpunkt der Entscheidung eine Variable eingesetzt werden kann. Operatoren in diesen Expressions sind die zu Erwartenden ($<, <=, >, >=$) und ein Intervall Operator (angegeben mit $v_1 : v_2$). Gibt die Expression `True` zurück, so ist die entsprechende TreeNode die richtige Antwort.

Option
Bei einer TreeNode mit dem Type `option` werden Zeichenkentten verglichen.

Final
Eine `final` TreeNode dient in der XML-Datei nur als Verständnishilfe. Im eigentlichen Programm ist jede TreeNode `final` die keine Nachfolger hat (also ein Blatt ist).

Invalid
Mit einer `invalid` TreeNode können bestimmte Werte blockiert werden. So können zum Beispiel alle Werte < 0 als ungültige Antworten makiert werden.

Wenn über einen Knoten im Baum entschieden wird, wird jeder Nachfolger nacheinander angeschaut. Dabei wird der erste Treffer wird akzeptiert. Die Antwort ist ungültig, wenn kein passender Knoten gefunden wurde oder wenn der (erste) gefundene Knoten `invalid` ist.

Entscheidungen zur XML-Datei

Ich bin eigentlich kein Fan vom XML Dateiformat, aber um Baumstrukturen darzustellen ist es die einfachste und verbreitetsten Form.

Der eigentliche Entscheidungsbaum besteht aus vier Tags (`decision`, `option`, `final`, `invalid`). Diese Tags entsprechen dem Type der TreeNodes, die aus diesem Element generiert werden.

Für den TreeWalker kann man noch einen weiteren Tag (`prompt`) angeben, der dem TreeWalker zusätzliche Informationen gibt. Mit deisem Tag lässt sich angeben welche Frage der TreeWalker für eine Entscheidung stellen soll.

Entscheidungen zur Darstellung

Bedienung

Um das Programm zu starten muss die Datei `MeshSimplifier.exe` ausgeführt werden. Dafür wird neben der `.exe` auch der gesammte Inhalt des Orderns `./res/` benötigt.

Die Ausführung des Programms findet in einem `read-eval-print loop (REPL)` statt. Dem Nutzer wird eine Frage gestellt, auf die er dann antworten muss. Ist die Antwort ungültig, das heißt die Antwort entspricht nicht der erwarteten Form oder ist keine der möglichen Antwortmöglichkeiten, so wird der Nutzer erneut aufgefordert zu antworten. Wenn ein Blatt des Entscheidungsbaumes erreicht wird, wird das Endergebnis angezeigt und das Programm beendet.

Bibliotheken

Da dieses Projekt textbasiert ist benötige ich nur eine Möglichkeit XML-Dateien einzulesen. Für diese Funktionalität habe ich mich für die Bibliothek **TinyXML-2** (`https://github.com/leethomason/tinyxml2`) entschieden.

Build

Zur Project Generation benutze ich Premake (`https://premake.github.io/`). Ich habe die entsprechenden Scripts meiner Abgabe beigefügt. Falls also Probleme mit der VisualStudio Solution autreten sollten, kann mit dem folgenden Befehl das Projekt neu generiert werden und so die Probleme hoffentlich gelöst werden:

```
.\premake\premake5.exe [action]
```

Für mein Projekt habe ich `vs2019` als `action` verwendet. Andere Möglichkeiten sind hier `https://premake.github.io/docs/Using-Premake` aufgelistet.