Decision Tree Toolkit

A simple toolkit for creating complex AI gameplay and NPC decisions

MorbidCamel

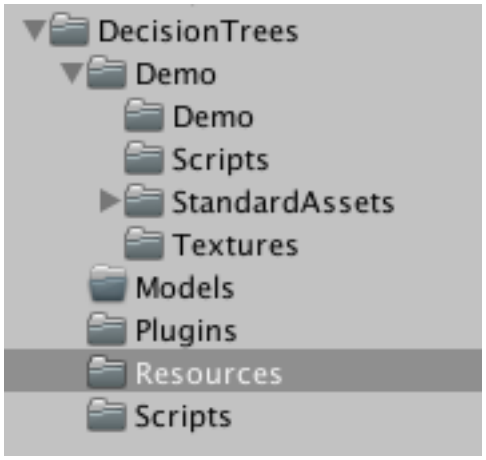
1st Edition

Introduction

In the field of machine learning in games one of the most popular mechanisms is using decision trees. Decision tree algorithms have less overhead than more complex solutions like Neural Nets and Logistic Regression because virtually no floating point operations is required to make a prediction. The decision tree represents a “baked” static representation of the underlying dataset and therefore resolving a decision is a simple matter of following a path down the branches of the tree until a conclusion is reached.

Getting Started

Simply import the package from the Unity asset store. Once imported you should see a structure looking like the screenshot below.



Unboxing

A brief overview of the package

Once imported it is important to understand where everything is located. The package uses a SQLite database so in order to create your own decision trees you will need a SQLite editor.

You do not necessarily need to know SQL to use the toolkit unless you want to edit the decision tree data at runtime. The component already takes care of loading and building the decision trees for you.

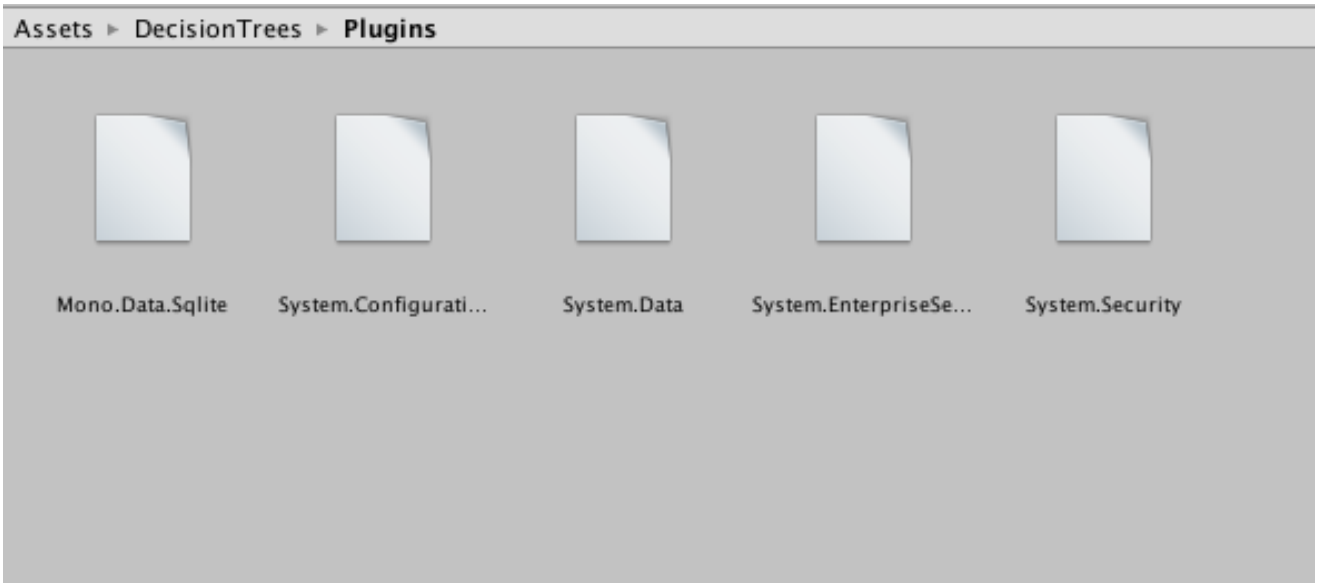
Decision Tree Database

The out-of-the-box decision tree database is located in the "**Resources**" folder and is called "**DecisionTreeData.db.bytes**". You can use this database and simply modify it for your needs or create a new database with the ".bytes" extension. You can also use an existing database if you are already using a SQLite database and configure it in the Unity editor.

> *In order to edit the contents of the database you will need a SQLite database editor available for free. In our case we are using "Base" available from the Mac AppStore.*

Plugins

In order for the toolkit to work it requires the "*Mono.Data.SqlLite.dll*" library. This is available as part of the Mono framework that Unity uses and is platform independent. However, it is not included by default and therefore these dlls needs to be imported using the "Plugins" folder. As a Unity rule of thumb if you include "dlls" in a folder called "Plugins" they will be referenced in you solution and copied to the output. Without these dll's the code will not compile. Below is a screenshot of all the dependencies used in the package.



Scripts

The “Scripts” sub-folder contains the C# scripts and Unity behavior components of the toolkit.

Script	Description
AttributeInfo.cs	Describes the meta-data for the decision tree algorithm
AttributeValue.cs	Structure describing the value of an attribute
DecisionAgent.cs	A proxy component that can be used on NPC characters or game objects that needs to use decision trees
DecisionExtensions.cs	A utility class containing some extensions used by the toolkit.
DecisionQuery.cs	An instance of this class is used to resolve a query with the decision tree.
DecisionTreeController.cs	The main controller class which contains the decision tree definitions and transforms the data in the database.
DecisionTreeInfo.cs	Describes the decision tree and maps the information to the decision tree data.
DecisionTreeOutcome.cs	Used to split a single dataset into multiple outcomes. In the example provided this would describe the “Perform” column.

Script	Description
DecisionTreeOutcome.cs	Used to split a single dataset into multiple outcomes. In the example provided this would describe the “Perform” column.
Hawk.cs	A reusable utility class that makes debugging easier and also adds the ability to strip out the debug code from the final build.
ID3.cs	An implementation of the ID3 Decision Tree algorithm
QueryCache.cs	Caching decision queries are contained in an instance of this class.
QueueSize.cs	Enum used in the logic.

Demo

The “Demo” sub-folder contains the assets used by the demo scene. You can use this to see how to implement it in you code. The demo uses a number of Unity standard assets to illustrate the decision tree usage.

Building a Decision Tree

A guide to build a decision tree in Unity

Building your own decision trees is easy using the toolkit. To illustrate how to do this we are going to use the Demo scene as an example. In the demo example we have built a decision tree based on a dataset describing instinctive responses to needs. When building your own decision tree there is a number of considerations to take into account. The accuracy and effectiveness of you decision tree will be largely dependent on how accurate the data you provide is.

Defining your Data Table

The out-of-the-box decision tree database already has some data preloaded. We will explain how the data looks and how to configure a decision tree using this data in Unity. The basic structure should look like this

Column	Purpose
<Input1>	The first input attribute
<Input2>	The second input attribute
<Input3>…<Inputx>	The third input and beyond attribute. You can have as many inputs. The amount of inputs and the range of values of those inputs will determine the complexity of the resulting decision tree.
<Predict>	The resulting prediction or output based on the other inputs. We recommend that this is an integer representing a “state” but this is not required.

Instinct Demo

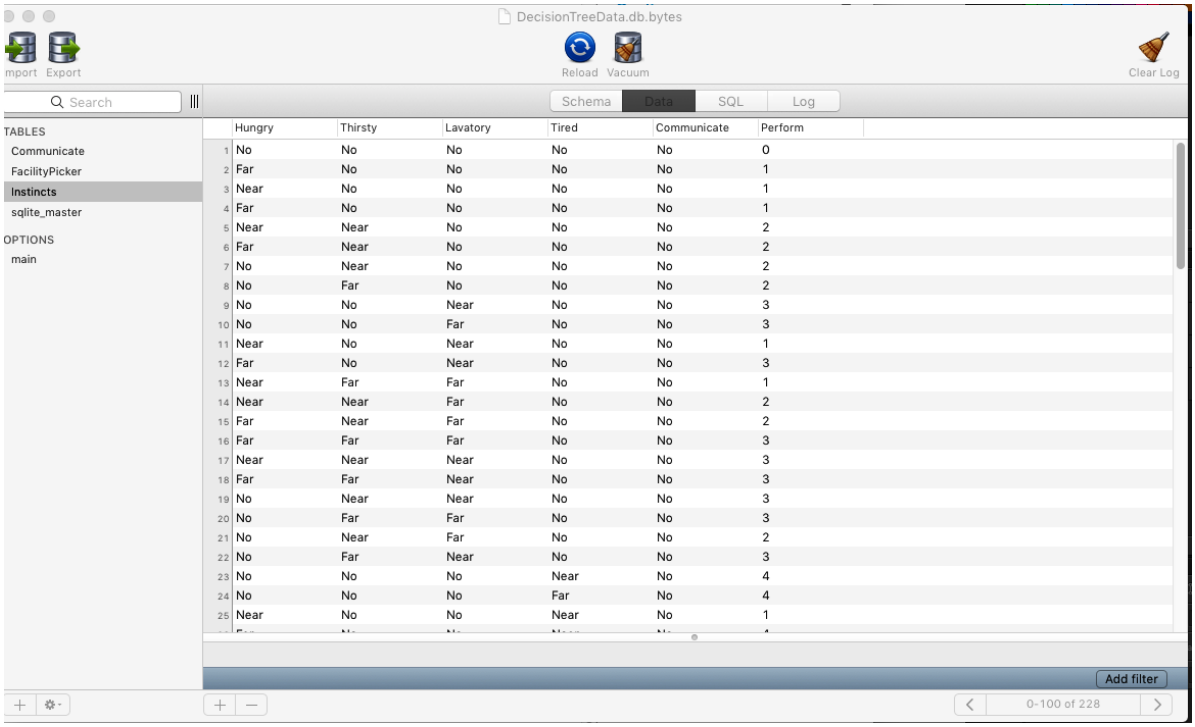
The table “Instincts” contains the data for the demo scene. The data describes a pre-captured decision making process based on the needs of a character. In the example it is obvious that if you need to drink water and you’re tired as well you are going to drink water first and then go to sleep. This reasoning is based on emotional and instinct responses in your neural pathways. By using these rules in a static dataset we can teach the NPC to ‘reason’ like a human.

Instincts

A brief description of the content of the demo table. It is important to note that the values has to be “meaningful” - the values reflect the state in the game. In our case we using the values “No”, “Far” and “Near”. So instead of simply “Yes” we use both “Far” to indicate that yes it’s a need and a facility is far from the character and “Near” if yes and a facility is close to a character. This technique makes it easy to cater for in game state. For example, if you are hungry and thirsty you might choose to first go buy a sandwich if it’s closer to you than the nearest water.

Column	Definition	Values
Hungry	The state of hunger of a character	No Near Far
Thirsty	The state of thirst of a character	No Near Far
Lavatory	The need to go visit a lavatory	No Near Far
Tired	The state of sleepiness	No Near Far
Communicate	The need to communicate	No Near Far
Perform	The resulting decision to perform one of the above	0 - Don’t do anything 1 - Eat 2 - Drink 3 - Lavatory 4 - Tired 5 - Communicate

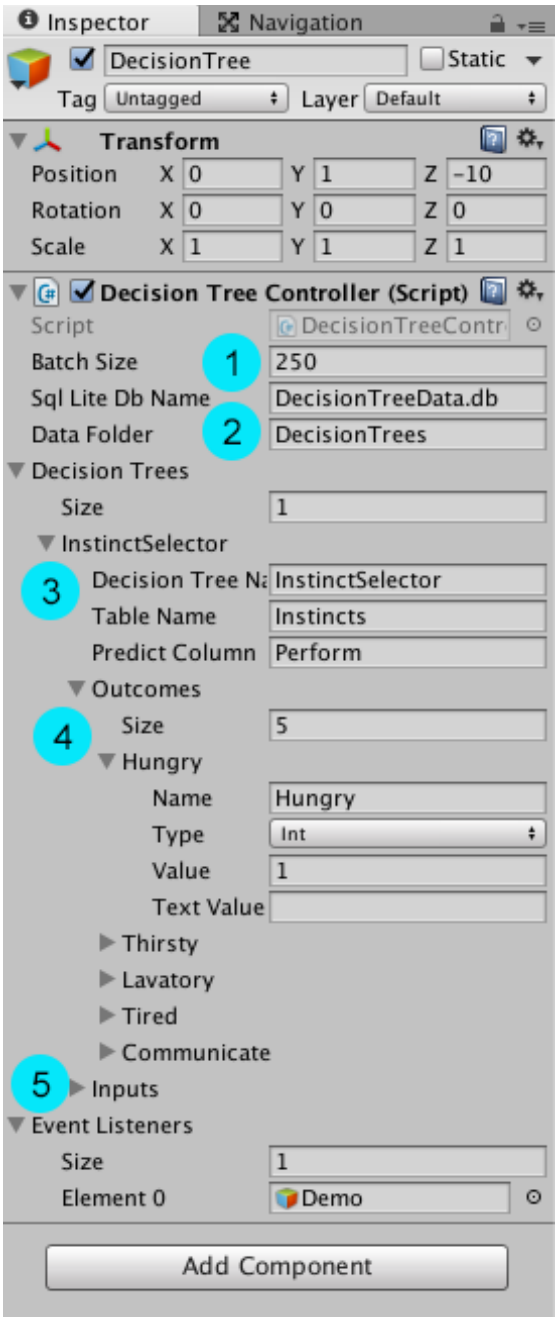
Table describes the content of the demo decision tree


DecisionTreeData.db.bytes
Import
Export
Reload
Vacuum
Clear Log
Search
Schema
Data
SQL
Log
TABLES
Communicate
FacilityPicker
Instincts
sqlite_master
OPTIONS
main
Hungry Thirsty Lavatory Tired Communicate Perform
1 No No No No No 0
2 Far No No No No 1
3 Near No No No No 1
4 Far No No No No 1
5 Near Near No No No 2
6 Far Near No No No 2
7 No Near No No No 2
8 No Far No No No 2
9 No No Near No No 3
10 No No Far No No 3
11 Near No Near No No 1
12 Far No Near No No 3
13 Near Far Far No No 1
14 Near Near Far No No 2
15 Far Near Far No No 2
16 Far Far Far No No 3
17 Near Near Near No No 3
18 Far Far Near No No 3
19 No Near Near No No 3
20 No Far Far No No 3
21 No Near Far No No 2
22 No Far Near No No 3
23 No No No Near No 4
24 No No No Far No 4
25 Near No No Near No 1
Add filter
0-100 of 228

Defining your Decision Tree in Unity

The "**DecisionTreeController**" component will need to be added to your scene. In the demo scene we have added it to the top of the hierarchy. The decision tree controller represents multiple decision trees and depending on your needs you might want to have multiple controllers in your scene. It is important to note that there's a "**DecisionAgent**" component that you can use as a proxy to the the decision tree controller.

Once added to your scene you use the properties of this component to define the decision tree(s) for your game. In the demo we have configured it to look at the "Instincts" table. Here's a guide to the component configuration.



Usage

1.The batch size controls how many samples are processed per frame. This is done to ensure that execution in the game does not slow down.

2.The SqlLite database name without the "**.bytes**" extension. The data folder is the relative path the database is going to be copied for use in runtime.

3.An array of decision trees. In this case only one element is specified and pointed to the "*Instincts*" table. The predict column is specified as "*Perform*" according to the table structure. Multiple decision trees pointing to different tables can be added here as long as they are in the same database.

4.Outcomes defines the predicted values. So an outcome of 1 == Hungry, 2 == Thirsty etc. Each outcome will have a separate decision tree.

5.The inputs of the decision tree as defined by the table definition. This is an array of values that lists all the input columns that needs to be used to create the decision tree.

Once the controller is configured have a look at the console to provide feedback when the scene starts up. If everything is successful it will show you the decision trees being calculated in the Unity Console window. Also use this information to troubleshoot

possible setup issues.

Using the Decision Tree

The demo scene included in the package shows you how to typically use the decision tree in the code. The "*DecisionQuery*" class has all the information needed to resolve queries against the decision tree. The "*DecisionTreeController*" class has a method called "**Decide**" which can be called to resolve the decision query,

This is an extract from the "**DemoUI.cs**" script for the instinct selector tree.

```
public void Decide()
{
    // Calculate the sqr to compare to the dot product
    float nearSqr = near * near;
    List<AttributeValue> values = new List<AttributeValue>();
    foreach(var need in needs)
    {
        var dist = need.target.transform.position - character.transform.position;

        //  If the need is selected we simple state whether its Near or Far
        values.Add(new AttributeValue(need.inputName,
            need.toggle.isOn
                ? ((dist.sqrMagnitude > nearSqr) ? "Far" : "Near")
                : "No"));
    }

    // Now we use the decision tree to see which of the needs we need to satisfy first
    // o In human reasoning thirst and hunger will always take precedence over other needs
    // o The distance to satisfy the need is also a factor
    // o Please note that in a game context these needs would have their own
    //   mecanisms such as using energy depletion as an indication of hunger.

    DecisionQuery query = new DecisionQuery();

    bool decided = false;
    StringBuilder builder = new StringBuilder();
    foreach(var need in needs)
    {
        if (!need.toggle.isOn)
            continue;

        query.Set(treeName, need.inputName, values.ToArray());

        tree.Decide(query);
        if (query.Yes) {
            character.target = need.target;
            decided = true;
        }
        builder.AppendLine(tree.PrintDecision(query));
    }
    // Print the last query
    text.text = builder.ToString();
    if (!decided)
        character.target = startPoint;
}
```