CAB402 Tic Tac Toe Assignment

John Santias n9983244

Queensland University of Technology

Introduction

The focus of this assignment is to learn programming in a "*pure"* functional style and differentiate it with traditional styles impure and object-oriented paradigms. Minimax with & without alpha beta pruning game theory concepts, heuristic scores, and model was implemented in the Tic-Tac-Toe game.

Tic-tac-toe is a game for two players where each take turns placing a Nought (X) or a Cross (O) in a 3x3 grid. To win the game a player needs to place their marks in a vertical, horizontal or diagonal row. Otherwise the game is declared a draw when board is full without any straight rows.

Experience

My experience to functional programming in F# allowed me to explore a different style of programming that emphasizes the use of immutable data and functions. I noticed that functional programming can shorten your code with the use of pipes, sequences, and functions that work like data types and more. Thus, making It easier to debug, faster to execute and easier to comprehend.

Functions allowed me to assign values that can be any data type (int, array, string et.) which can eliminate mistakes from setting its type. An example for this is in C# is if we specified a private int variable called *'nextValue'* in a new class, it can be hard to keep track of its value during debugging. Functions can do more than just assigning variables. It can also pass a parameter, return a function as the result of another function, calculate on a given value or even create another function.

F# operators like pipes, arithmetic operators and the use of sequences calculate values in a much more efficient way. This eliminated the use of for loops, switches, creating new lists which can take longer to execute. Extracting the head or tail from a sequence, list or array is easy also if prepending to a list.

The use of mutable types in F# and C# helps keep track of values. However, in C# if an object has more than 10 properties you may have to implement more code to update it potentially leading to more bugs. Creating immutable types made code cleaner, easier to understand and it created less bugs for me which was easy to track down. It was found that multiple bugs were created when implementing the solution in C#. It took longer and was harder to debug.

It was found that F# impure was the most efficient of all programming styles when it comes to finding the best move for the given player. Majority of the test results were very similar to the test named 'TestFindBestNodeCount251'.

C# Impure Alpha Beta	F# Impure Alpha Beta	F# Pure Alpha Beta	F# Pure Minimax
0:00:00.000505	0:00:00.000006	0:00:00.0000097	0:00:00.0015564

It is also evident that using basic minimax would take longer because it searches through each node unlike the alpha beta pruning method. When implementing this method in games, it would be best to search less nodes which can make the program use less resources and be more efficient. C# clearly takes more time because of the frequent property updates between each class and the loops generating new values.

In conclusion, making everything immutable makes programs easier to understand and fix bugs. It can be much more efficient to run program that uses less resources during execution. For future, I would try to program with immutable states especially if it's a big project to eliminate the use of updating multiple properties and receiving unexpected change in variables.