# CAB402 Programming Paradigms

## Assignment 1 Report

N9747575

Jerald Jericho Limpin

## Overall Experience

It was a tough challenge to adapt to a functional paradigm due to the majority of my experience being in imperative languages. The more I reasoned about and developed my functions the more I began to appreciate the advantages of using pure functional programming. Being able to use higher order functions such as Map and Filter allowed me to make more concise code that I could understand. Furthermore I was able to more easily to explain the concept behind my functions. Though I saw the pitfalls of functional programming in terms of computational efficiency and general lack of convenience when using variables. Being able to use mutability in F# impure and C# made my code more efficient computationally but at the cost of readability.

## 3 Approaches

### F# Pure:

### Pros:

- Higher order functions results in more concise code
- Don't have to deal with any pitfalls to do with mutability
- If written at a higher level the code is easiest to understand and read
- Type inference makes the code more readable and less clouded
- Rewards more concise, higher level code as opposed to a brute forced solution (e.g. use of higher order functions and Lambda functions such as in GameOutcome)
- More easily threadable and parallelised
- Least easy to write due to lack of experience, possibly easiest to write due to how concise the code becomes
- Easier to debug because a function is self contained and only input parameters matter

### Cons:

- Certain scenarios (i.e. Alpha Beta Pruning) require recursion which is less efficient computationally and more memory intensive
- Lack of mutability leads to having to create a whole new object when updating (applyMove) which is less efficient computationally

## F# Impure:

### Pros:

- More efficient when updating the board’s state since only need to change variables not create whole new object
- More convenient to store mutable variables so that recursion isn’t required
- Able to reuse most of the code from F# pure with a few tweaks for computational efficiency
- More readable only where originally in F# pure a recursive function was used (i.e. Alpha beta pruning) since a while loop is easier to reason about then the use of head and tail notation
- Leads to more computationally efficient code than F# Pure

### Cons:

- An undo move function is required since the board is mutable otherwise an applied move would be permanent
- Becomes less readable and harder to reason about than F# pure

## C#:

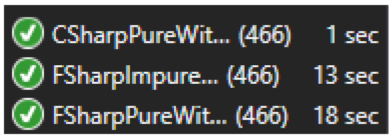
### Pros:

- Easiest way to conceptualise the functions and the steps required
- Able to reuse variables across functions such as in Game class
- Helps break down the code into many more easily maintainable chunks (i.e. breaking down the game into multiple classes)
- Strong typing forces the input and output to result into as exactly as expected typing
- Clearer structure for modelling the game in terms of classes and functions
- Most efficient in terms of execution time
- Easiest to write code due to how easily a solution can be brute forced (i.e. gameOutcome)

### Cons:

- An undo move function is required since the board is mutable otherwise an applied move would be permanent
- Makes certain functions such as GameOutcome more brute forced and less readable even if the function was split into multiple smaller functions
- Inheritance isn’t useful for reusability when looking at only C#. It is only useful for easily defining the game board across all 3 implementations
- Least readable implementation

## Tests for Computational Efficiency



```
✓ CSharpPureWit... (466)   1 sec
✓ FSharpImpure... (466)   13 sec
✓ FSharpPureWit... (466)  18 sec
```

Figure 1 Execution Time for tests

| Programming Style | Average Time to Compute (Seconds) |
|---|---|
| C# | 0.02 |
| F# Impure | 0.241 |
| F# Pure | 0.609 |

Figure 2 Execution Time for 3 x 3 for a move that results in the same 2080 nodes

In figures 1 and 2 show that C# is definitely the most computationally efficient while F# pure is least efficient.

## Conclusion

Overall it is too hard to say which style leads to the most efficient code. Since functional programming isn't the easiest to model a game like Tic Tac Toe which much more easier to conceptualise using an OOP based paradigm. Though functional programming does results in the most concise and easily readable code it is definitely not the most computationally efficient in this scenario. Therefore there is no objective way to say which code is most efficient.