Institiúid Teicneolaíochta Cheatharlach
INSTITUTE of TECHNOLOGY CARLOW
At the Heart of South Leinster

Viking Chess Using MCTS

Final Report

Declan Murphy – C00106936
Supervisor: Joseph Kehoe
2016

Contents

1. Introduction

This is the final project report for the Viking Chess Using MCTS project developed as part of the 4th year BSc.(Hons) in Software Development at IT Carlow.

This document details the development of the Viking Chess game, the game itself, the challenges faced throughout development, the learning outcomes from the project and a review concerning the development and game.

2. Project Description

2.1. Application Description

The Viking Chess game is an open source Windows 10 game that recreates the ancient Tafl games played by the Vikings and the people they conquered. The game implements the Monte Carlo Tree Search algorithm to create a competent AI player. Development of the game began in October 2015 and concluded in April 2016.

The game contains five different variations of Tafl. These are:

- **Hnefatafl** – *The Norse Variant*. This is one of the most popular variants and it is believed that it is this variant that the other variants are derived from. Hnefatafl is played on an 11x11 board with 24 Attackers vs 12 Defenders and a King.
- Brandubh – The Irish Variant. This variant is played with the smallest amount of pieces with 8 Attackers vs 4 Defenders and a King on an 7x7 board.
- **Ard Rí** – *The Scottish Variant*. Like Brandubh, Ard Rí is played on a 7x7 board but uses twice as many pieces.
- **Tablut** – The Sami Tribe Variant. This variant of the game was still being actively played by the Sami People of Scandinavia in the 19th Century. It is played on a 9x9 board with 16 Attackers vs 8 Defenders.
- **TawlBwrdd** – *The Welsh Variant*. This variant is very similar to Hnefatafl in that it has the same number of pieces and is played on an 11x11 board with just a slightly different layout of pieces.

In addition to a choice between what variant to play, players have the choice to play against another human locally on the same machine or against the CPU which uses the MCTS algorithm.

As the game is not widely known about, the decision was taken to add a tutorial detailing how to play the game. To keep code to a minimum, the decision was taken to create a single page for this tutorial using buttons to activate different animations and text showing and detailing the rule specified.

2.2. UI/UX Description

2.2.1. Introduction

As with any game, the User Interface (UI) and User Experience (UX) had to be given careful consideration. It was important for me to create a UI that players would associate with Vikings. Also, since Tafl is a board game, the UI and UX would have to be simple to understand and straight-forward to use.

When designing the UI theme, I tried out a number of different concepts before settling on a final design. Once I had settled on the design, I detailed the elements of the design into a single image to use as a reference when creating the UI elements, as can be seen below.

All images used in the application were created by myself.

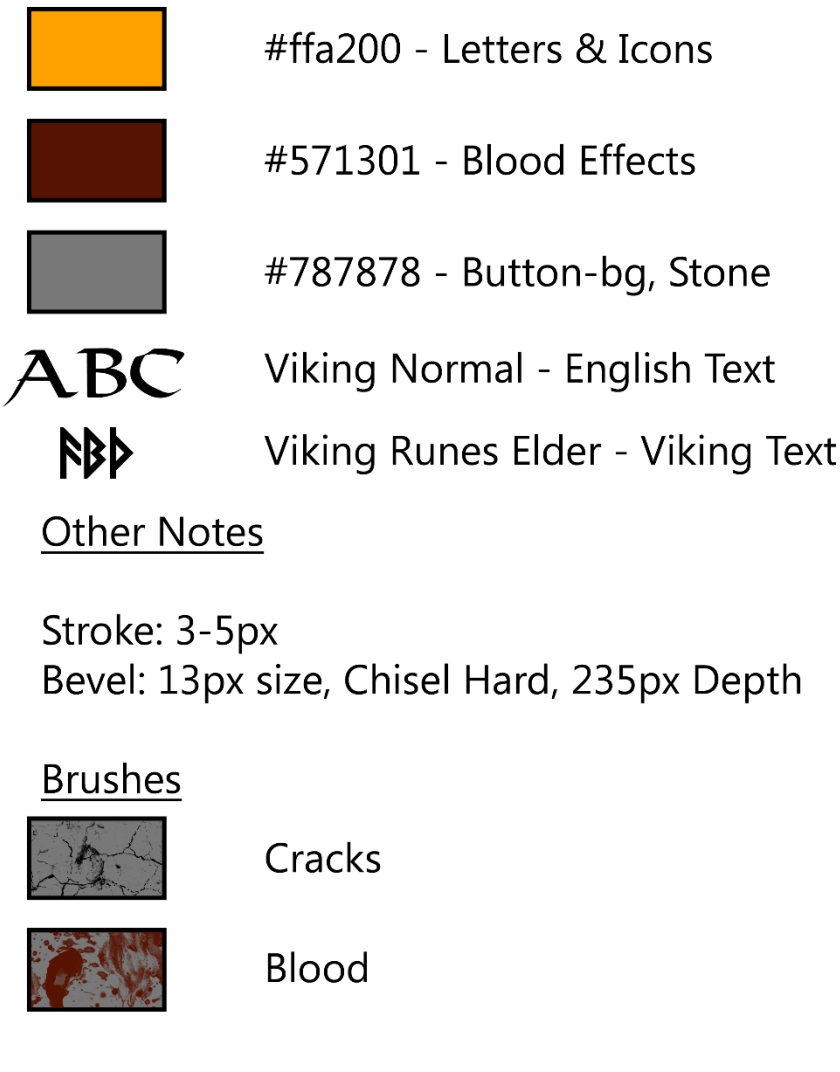

Fig 1 - Theme Details

2.3.2. Screens

The following is a description of all screens and their purpose present in the game.

2.3.2.1. Main Menu

The main menu is the entry point of the game.

Fig 2 - Main Menu Page

2.3.2.2. Settings Page

The Settings Page contains the configurable options for a new game of Tafl and forwards the player onto their chosen variant.

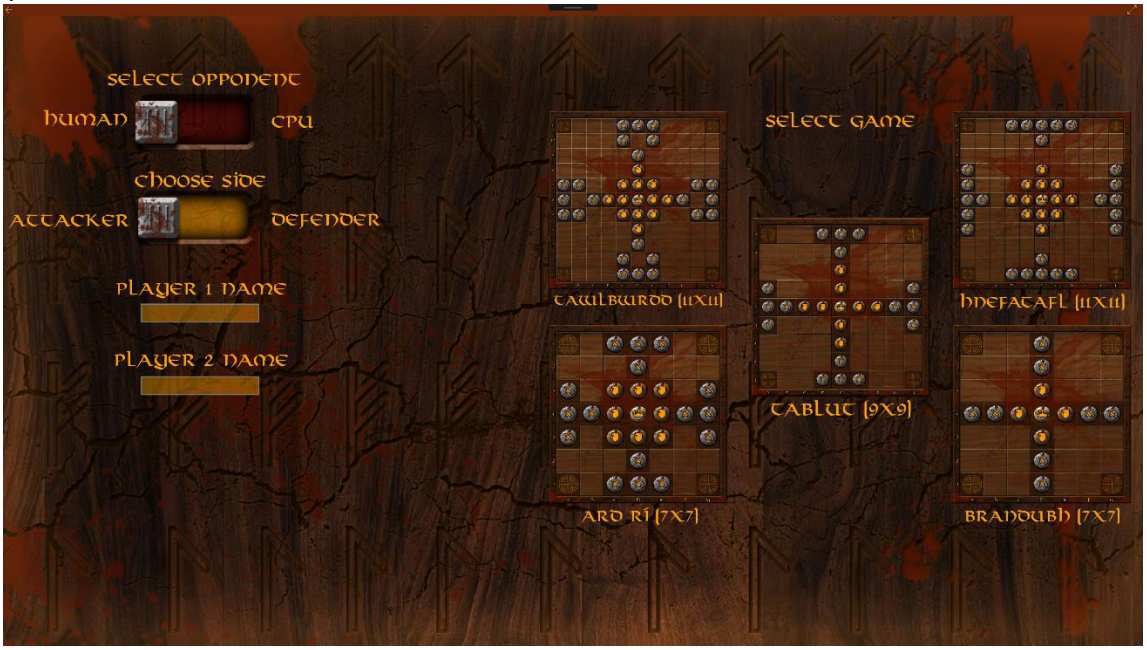

Fig 3 - Settings Page

2.3.2.3. Game Page

The Game Page is where a game of Tafl is played out. In the centre of the screen, the board and pieces of the chosen variant are displayed and interacted with. At either side of the board is a list of the previous moves made by each player and a turn timer for each player. Below the board lies the total board timer and above the board, relevant notifications about the game inform the user of various statuses such as the CPU making a move or the King piece being in a Check position.

Fig 4 - Game Page

2.3.2.4. Rules Page

The rules page contains information about how to play the game. Each button on the left-hand side of the screen activates an animation that plays in the upper-right corner and a text description of the rule which itself may be animated if more information is required to be delivered.

Fig 5 - Rules Page

2.3.2.5. About Page

The About Page displays information on the game and its development.

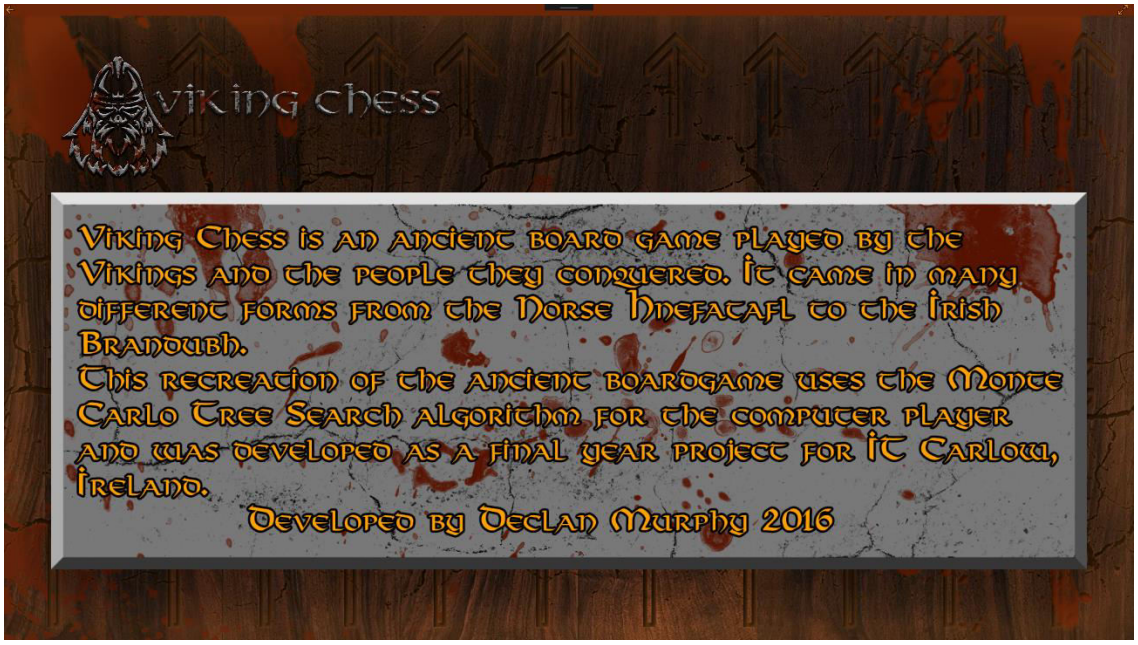

Fig 6 - About Page

3. Specification Conformance

"Good design adds value faster than it adds cost."

-Thomas C. Gale

It is difficult to gauge how well this project stuck to the initial specification. In many ways, the project stayed on track and the initial design and development synchronised well. However, with the relatively short development lifecycle, I sometimes felt it was necessary to forge ahead with coding a solution before any clear specification was drawn up for the problem of the time. This compounded the conformance drift since once I had coded a solution to a problem, I found myself resistant to changing it and instead opted for updating the specification to match.

In retrospect, greater attention should have been paid early on to properly documenting the project before proceeding with coding solution. Doing so would have alleviated many problems I encountered while coding and saved more time in the long run so that more features could have been added.

Nevertheless, many specification details were transferred successfully. First and foremost, the UI design changed very little from what was laid out in the functional specification. This allowed any UI development to move along at a quick and constant pace. Additionally, the use cases I designed were sufficient for guiding me through coding. Finally, the development of the MCTS algorithm was, for the most part, true to design with changes made to the design arising as a result of unforeseen complexities in tailoring the algorithm towards the game's domain space.

4. Learning Outcomes

4.1. Personal

To say that this project was eye-opening would be an understatement. Having never before dedicated so much time and effort into a single software project, I found the experience to be one of constant learning. By that I mean the project was challenging, frustrating, rewarding, and educational; in equal measure and often simultaneously.

The project challenged me in many ways. It forced me into adopting better time management skills than I had when I started. It humbled me when I believed myself to have found a solution only to discover my gung-ho attitude to coding led me to overlook an obvious oversight.

I found myself frustrated on many occasions as I fought through a lack of a proper testing framework and debugged the program at a snail's pace and reinforced in me the benefits of test-driven development and the importance of suffice documentation.

Throughout the development of the game, I was rewarded for taking the time to learn about the MCTS algorithm and the domain space. It led me to considering ways to play the game that I would have to consider in my evaluation function and allowed me to create a relatively code-light algorithm from spec.

In all these ways and more, the project was educational. Developing the game as a one-man team over the course of six months revealed to me the challenges that small developers face can seem overwhelming but not insurmountable. In essence, it thought me that with the right tools (good documentation and testing), the necessary skills (coding), and a bit of hard work, a project that at first appears difficult can soon be overcome.

4.2. Technical

In my 4 years doing this course, I have been introduced to a number of different languages. Many of these were cursory delves into the little differences and intricacies of one language versus another. I have never felt that I could say I could competently develop an application in a language other than Java. Until now. Though C# shares much with Java, there was also a lot of learning to do along the way and at the end of this project, I now feel confident in being able to say I can take advantage of the C# language.

Similarly, I have never before developed a Windows application to any real extent yet with this project, I not only got to learn how the .Net framework operates under the hood but as I was developing for the relatively new Universal Windows Platform (UWP), I gained valuable knowledge in developing applications for the Windows 10 family of devices and discovered the ease-of-use that XAML provides in creating a good and responsive UI.

Designing and coding Viking Chess and the MCTS algorithm also have me great insight into creating board games with AI players. Though the rules of a game like Tafl are a bit simpler than creating a complex AI for a complex video game, researching the MCTS algorithm and related

algorithms such as Alpha-Beta Search, etc has allowed me to better understand how one would go about creating a more advanced AI.

5. Project Review

5.1. Introduction

When I chose to select this project as my top choice, there were a number of motivators that led to the decision. First and foremost, I want to direct the skills I have learned over the last four years towards game development. Viking Chess provided me the perfect opportunity to get experience and start building my portfolio for the future.

Secondly, the importance of artificial intelligence in today's games cannot be understated. I felt that it was important for me to gain first-hand knowledge and experience about AI in games and finish this course with the ability to implement decent AI in future projects.

Finally, before choosing Viking Chess as my project, I had never heard of the board game. This project drew parallels with my personal interest in the ancient history of Northern Europe and delving into the history of Viking Chess as a game was a personally rewarding experience.

5.2. Achievements

Overall, I am happy with the final outcome of the project. Implementing the MCTS algorithm for Viking Chess was difficult but in the end, I am relatively happy with how the algorithm turned out. Creating a board game from beginning to end was a first and although I faced many hurdles throughout development, I'm glad that I was able to implement a working version of the game.

I am also very proud of how the user interface turned out. Initially, I threw together a basic UI that served the purpose of visualising the game during initial playtests. Very early into playtesting however, I realised that the interface was, for lack of a better term, ugly. Many elements blended together due the monotone nature of the colours along with low resolution images. The result was a UI that was both difficult to read and unappealing. I believe spending the time to think about the UI design and create high resolution images paid off in creating a clear and attractive UI that evokes the spirit of Viking life.

In addition to spending time on the user interface, I also created a handful of sounds that play when a player presses a button, selects a piece, moves a piece and captures a piece. With more time, I would have liked to have created a full suite of sounds and music. As it stands however, the experience of playing the folly artist for my own game proved to be an enjoyable experience in terms of finding the right objects to use (Luckily, I had a nice bamboo chessboard and pieces lying around) and editing the recorded audio afterwards to reduce noise and improve the sound quality.

5.3. Challenges

The project presented many challenges over the course of development. Thankfully, I was able to overcome most of them and when the result was less than I had expected, I did my best to mitigate the problems.

5.3.1. Performance

When starting the project, the greatest challenge was deciding how to design the board game. There are many established methods for creating games such as Chess but I felt some of these would be ill-suited to the multiple boards and layouts of Tafl. In the end, I decided on a purely object-oriented approach. However, once I settled on this design and coding began and progressed, it became harder and harder to change the design. Initially, I was happy with the object-oriented approach. Then as I began developing and testing the MCTS algorithm, I became more concerned with the performance overhead from creating the multiple objects required by the algorithm.

These concerns showed themselves to be well justified when I began playtesting the MCTS algorithm. The time complexity of the MCTS algorithm can grow exponentially. The reason for this is simple: The greater the number of playouts performed by the MCTS algorithm, the greater the number of nodes created. With each node created, performing the four steps of the algorithm takes longer to complete. With an object-oriented approach, the performance impact of processing the playouts is high as each object takes a relatively high amount of time and memory to create when compared to basic data types. To control this performance impact, the number of playouts performed was limited to 1000. This keeps the time it takes for the CPU player to make a move relatively low, although the time it takes is also dependent on the state of the current board and the variant being played. The following table shows a quick run-down of the average first-move turn time for the CPU player in each variant:

Variant	Time
Ard Rí (7x7)	19s
Brandubh (7x7)	18s
Hnefatafl (11x11)	2m41s
Tablut (9x9)	1m8s
Tawlbwrdd (11x11)	2m35s

Fig 7 – MCTS Time to Completion Table

As you can see, the larger boards with more pieces and thus more possible moves take quite a bit longer to process through the playouts than the smaller boards. One solution to this problem might have been to decrease the number of playouts depending on the variant being played. However, as the number of playouts is strongly tied to the strength of the move the MCTS algorithm makes, it was decided to keep the number of playouts the same across the board with the number chosen to attempt to balance the time it takes against the quality of the move.

5.3.2. The Evaluation Function

Aside from the performance challenges, the greatest difficulty was in implementing a good evaluation function for the MCTS algorithm. The purpose of the evaluation function is to score the state of a board at the end of the simulation phase. If the board is in a terminal state (i.e. The game has been won by a simulated player) then scoring is straight-forward and the score returned is predetermined by a constant which is positive or negative depending on if the win is in the CPU player's favour.

Scoring boards that conclude in the mid-game is more difficult however. At first, I had considered just counting the number of pieces each side had remaining and scoring the board accordingly. However, this proved insufficient at providing an accurate reflection of the state of the board. That being said, it did prove to be a good starting point. I then reflected on the problem some more and realised that controlling the ranks and files of the board are important. In other words, pieces on the outside ranks and files are worth more than pieces on the inner ranks and files. This is because the closer a piece is to the edge of the board, the easier it would be for the piece to block the King from moving to the edge or prevent a block from occurring in the first place.

So all together, my evaluation function scores based on Win/Loss, Piece Count and Board Control. I believe this is sufficient enough given the time I had to complete the project. However, with more time, I would have liked to put more effort into considering more ways of scoring the mid-game and improve my evaluation function. As it stands, it makes relatively smart moves but falls apart against a player familiar with the game as it cannot evaluate the board to the degree that the player can.

5.4. Dropped Features

There are a number of features I had hoped to include in the final design that I either never got around to implementing or dropped for a specific reason. If I wanted to, this list could be endless and given an infinite amount of time and passion, the feature-creep would be endless. However, here are a few of the features I most deeply considered that never appeared in the final version:

- **Rule Modifications**, including:
  - *Corner Escape*: King must reach a corner to escape
  - *King Captured by Two Pieces*: King is captured in the same way as all other pieces
  - *Dice Rolls determine moves made*: The roll of a die determines the number of moves a piece can make in a given turn. This is a rule speculated to have existed in Brandubh when it was first created and may have existed in other variants as well.
- **Difficulty Slider** – This was removed when I determined that increasing the strength of the MCTS algorithm by increasing the playout count would exponentially increase the time it takes for the CPU player to make a move. Instead, I chose a static number for the reasons discussed in the previous section (5.3. Challenges).
- **CPU vs CPU** – This would have been relatively straight-forward to implement given a little more time. However, as the final deadline loomed, I decided that it would be an aside to the main game and chose to drop it as a feature. That being said, I would be interested to see how the computer performs against itself using the MCTS algorithm, possibly with different playouts for each CPU.
- **Multiple Device and Resolution support** – As I had chosen to develop a UWP application, I had hoped to support multiple devices and resolutions. However, as this was my first foray in UWP development and XAML, I found it difficult to implement the UI scaling necessary to support them. As time dwindled, I decided to drop this feature in favour of focusing more on the underlying code.
- **Full Audio Suite** – I had planned to include additional sounds and music such as background music and noise, etc but ran out of time.

- **Online Play** – This was always a long-stretch goal but if time had allowed, I would have liked to explore the option of playing a peer-2-peer game over the internet.

5.5. Recommendations

As I reach the conclusion of this report – and as a consequence, this project – there are a number of recommendations I would like to make to anyone else attempting a project in the same vein as Viking Chess.

5.5.1. Document Early and Sufficiently

Although the bohemian soul in me fights against the idea of heavy planning, it is difficult to argue against its benefits, especially in the case of a large project such as this. My first piece of advice to anyone willing to listen is to spend the time to plan ahead and document that plan is as much detail as possible. As the project progresses, that initial documentation may become outdated and it may even turn out to be entirely wrong. Yet with that documentation, there's a very likely chance that spending an hour detailing a use case and creating a sequence diagram will save at least two hours later down the line coding.

5.5.2. Be Wary of Feature-Creep

One thing that I had to fight hard against over the course of development was my own active imagination. I could have included different board styles and different pieces. I could have tried out multiple variations of the MCTS algorithm. I could have included every Tafl variant ever mentioned. I could have continued on and on adding new features indefinitely.

Had I done so, the end result would have likely felt bloated; each disparate component less than what it should have been. Instead, I focused my energies on the main goal – implement the MCTS algorithm and the underlying game. Sure, there were times when I let my flights of fancy take me and looking back upon those occasions, I see that they were as much an escape from the current problem I was facing in development as it was about adding the feature I just thought of. In a way, I believe this to be healthy. Stepping back from a challenge and moving onto something new is a good way to find new vigour and return to the problem with a new approach head-on. So long as the escape doesn't become an extended departure.

5.5.3. Just Because They Look Alike Doesn't Mean They Are

Very early on in development I found myself running into errors and exceptions that I just couldn't track down. I did (or thought I did) everything I could to solve these problems but to no avail. Then the realization came to me – Copy Constructors.

Although present in Java, Copy Constructors are very important in C# when dealing with objects. Where with Java, you might get away with just a shallow copy of an object, there is very little chance that the same success will be found in C#.

This highlights an important lesson for anyone looking at a new language and thinking "That looks familiar to [insert language here] which I know very well." Do your research. Make sure early on that you have a good working knowledge of a language before you embark on a big project. Not doing so will only hold you back in the long run – and provide countless joyless hours of debugging.

5.5.4. Time Management is Everything

In spite of a starting base of poor documentation and hasty coding, effective time management can turn a potential disaster into a potential exemplar.

Developing an application over the course of six months, all the while juggling other course responsibilities and home life is no easy task. It requires dedication, self-control and good time-management. At first, it seems like you have plenty of time to get the project done. Then the first iteration and the first demo is due and you have to get things up to scratch and put your best foot forward. Before you know it, the second iteration has passed you by and you're on the home stretch.

If you manage your time poorly, you are setting yourself up for failure. It's as simple as that. Six months is less time that you might imagine when you have a torrent of work and responsibilities getting in the way of making progress. This is where Recommendation 5.5.1 pays dividends. Plan correctly and you are already well on your way to managing your time more effectively.

5.5.5. Don't Stress the Little Things

One of the big things that surprised me while developing the Viking Chess game was just how easy it was to get something done when I got out of my way. There were so many times when I would get bogged down in the minutia of a problem and could no longer see the wood from the trees. I would then get frustrated and compound the problem when the solution didn't magically present itself after the well-practiced ritual of beard-stroking and head-scratching.

So my final piece of advice is this: Don't worry about the details *too much*. In my experience from working on this project, I have found time and time again that the problems that tend to niggle the most resolve themselves as you work your way through the bigger picture. In the end, it is when you put the stress and the worry aside when you get the most work done.

So relax, settle in and put the head down. It should all work out in the end.

6. Final Remarks

In closing I would like to briefly go over some outstanding discussion that I did not get to or which did not fit in the above sections.

6.1. What Would a Redo Look Like?

Had I the opportunity to start the project again from scratch, there are many things that I would do differently. Some of these are small, others are quite major.

First and foremost, I would use the knowledge I gained from this project to document a more complete model of the game. In doing so, I would consider from the outset the importance of performance to the MCTS algorithm and step away from, as much as I could, an object-oriented approach in favour of a more lightweight and stripped down representation. Zobrist Keys and Lookup Tables (as recommended by my supervisor Joseph Kehoe) would be my first port of call.

Next, I would create a concrete list of must-have features that is concise. I would plan each in detail and add them to the game as soon as possible and I would include no additional feature until I was sure everything else was to spec. I would also take greater care in developing a responsive UI from the start rather than trying to shoehorn it in after the leg-work has been done.

Finally, I would set aside as much time as possible to improving the evaluation function as I now believe this to be the critical component in making an effective MCTS algorithm for Viking Chess.

6.2. Did You Choose the Right Technologies?

The short answer: Yes, I believe so. The UI was always going to be a front and centre element of Viking Chess. Developing a UWP application using C# and XAML all but removed the grunt-work and difficulty that would have otherwise been involved in creating an attractive and engaging user interface.

In addition, the use of C# as a core language allowed me to take advantage of the similarities with Java to get a head start on development versus choosing a language I was less confident about.

6.3. What Were The Implications of Your Chosen Technologies?

Honestly, there were very little implications to the choices I made. Although XAML was completely new to me and there was a slight learning time before I felt confident developing with it, I gained a good grasp of it with relative ease.

If I was to highlight any implication, it would probably be that choosing C# immediately put me into an object-oriented mind-set which, as mentioned previously, became a source of consternation in regards to performance.

That being said, if I had the opportunity to choose new technologies for a redo, I would probably still go with the same, if only for the speed and ease at which code can be generated and UI's built.

7. Acknowledgements

I'd like to thank my supervisor Joseph Kehoe for making me focus on getting the game working before I worried too much about getting it working perfectly. I wish to thank him too for sitting through my attempts at trying to find the right questions to ask and then given me the answers I needed to hear.

I would also like to thank anyone who had to listen to me blather on about a board game they had little interest in and pretending that they had. I would also like to thank my friends and partner who play tested the game as much as I made them and given me honest, sometimes scathing critique.

All of you kept me in line with my feet on the ground and aided me in accomplishing what is, to date, my biggest software development project to date.