Table of contents

[Table of contents 1](#_Toc473033877)

[Analysis 2](#_Toc473033878)

[Description 2](#_Toc473033879)

[Target Audience/Stakeholders 2](#_Toc473033880)

[Why a computer is suitable for the task 2](#_Toc473033881)

[Research 2](#_Toc473033882)

[Features and limitations 2](#_Toc473033883)

[Requirements 2](#_Toc473033884)

[Success Criteria 2](#_Toc473033885)

[Design 2](#_Toc473033886)

[Decomposition of the problem 2](#_Toc473033887)

[Proposed structure of the program 2](#_Toc473033888)

[Algorithms 2](#_Toc473033889)

[Usability 2](#_Toc473033890)

[Key variables and data structures 2](#_Toc473033891)

[Test Data for development 2](#_Toc473033892)

[Test Data for beta testing 3](#_Toc473033893)

[Development 3](#_Toc473033894)

[Iterations of development 3](#_Toc473033895)

[Prototypes 3](#_Toc473033896)

[Evidence of modular code 3](#_Toc473033897)

[Evidence of validation 3](#_Toc473033898)

[Review 3](#_Toc473033899)

[Evaluation 3](#_Toc473033900)

[Testing 3](#_Toc473033901)

[Testing of usability features 3](#_Toc473033902)

[Overall evaluation 3](#_Toc473033903)

[Future Maintenance 3](#_Toc473033904)

Analysis

# Description

[TODO]

# Target Audience/Stakeholders

There is no target audience due to the simple/casual nature of the game, and the fact that most people seem to know what tic-tac-toe is. This means that the stakeholders will most likely be anyone who is 6 years or older.

# Why a computer is suitable for the task

Computers are very fast at performing calculations, and the only errors they make are generally due to human errors (coding mistakes, for example). For a game as simple as tic-tac-toe, a computer is more than capable of calculating what it should do in a reasonable amount of time, and can possibly be almost impossible to beat.

As an example, when two human players are playing against each other, Player 1 may be one move away from winning the match with Player 2 not noticing they can stop them; however, when a human is against a computer, the computer can be made so it will always block the human from winning, if there’s a chance to. This is due to the point made earlier, the only errors a computer can make are usually due to human errors, so if a human tells the computer to always block the other player, then it will do so without fail (assuming there’s no errors in the code).

As another example, for a simple game like tic-tac-toe, the computer may be able to plan ahead of time and think of the most optimal route to take, similar to a human. The difference is that a computer can analyse the paths it can take significantly faster than a human, and a computer will be able to ‘remember’ them all perfectly, whereas a human might forget something or make mistakes in their logic.

# Research

While researching on what algorithms I might use when writing the AI for the game, I came upon the Minimax [1] algorithm.

After some more research, I came upon a website [2] that describes how they used the Minimax algorithm with tic-tac-toe. To summarise, they calculated every possible route the AI could take, and used points to weigh between which route will have the least chance for the AI to lose, and which ones would make the AI win/tie (with some other tweaks to make the algorithm work well).

The issue with this algorithm is, it creates an unbeatable AI, which is not fun for the human to fight against (nor does it seem terribly interesting to code). The upside is, this algorithm is a perfect example of how a computer is suitable for playing tic-tac-toe, and can be better at it than humans.

The idea of weighing which path is most likely to win/lose was interesting to me, and my tutor during an earlier session was discussing about possibly using machine learning, where the computer stores data of past games and then uses that data to determine which moves have led to a win in the past.

The advantage of the AI using past data, instead of calculating the best moves to make on the spot, is that it can attempt to ‘learn’ the best way to win which I see as an acceptable compromise between ‘impossible to beat’ and ‘impossible to lose against’. At the start, when the AI lacks data, it should be pretty easy to beat; but as time goes on the AI will gradually get more data and will be able to perform better than when it started.

Similar to how the minimax algorithm would create a tree of moves to analyse, my AI can store the data of its past games in a tree. For example, it may be formatted like:

“empty grid” -> “X is placed in the top-middle slot” -> “O is placed in the bottom-right slot”  
 -> “X is placed in the top-left slot” -> “O is placed in the bottom-middle slot” etc.

[1] <https://en.wikipedia.org/wiki/Minimax>

[2] <http://neverstopbuilding.com/minimax>

# Features and limitations

[TODO]

# Requirements

[TODO]

# Success Criteria

[TODO]

Design

# Decomposition of the problem

[TODO]

# Proposed structure of the program

[TODO]

# Algorithms

[TODO]

# Usability

[TODO]

# Key variables and data structures

[TODO]

# Test Data for development

[TODO]

# Test Data for beta testing

[TODO]

Development

# Iterations of development

[TODO] (Self note, remember to use Git to ‘go back in time’)

# Prototypes

[TODO]

# Evidence of modular code

[TODO]

# Evidence of validation

[TODO]

# Review

[TODO]

Evaluation

# Testing

[TODO]

# Testing of usability features

[TODO]

# Overall evaluation

[TODO]

# Future Maintenance

[TODO]