**Individual writing exercise – Demian Till, 1003594**

This report attempts to introduce the mind-game team project that me and my team are

working on, explaining the nature of the project, the direction in which we have decided to

go and some of the things we have learnt.

The reader of this report is expected to be familiar with terms and concepts related to

general computer science but not necessarily artificial intelligence. The reader is also

expected to be familiar with the game of chess.

Mind games are games in which there is no element of luck. The outcome of every action

can be precisely calculated allowing, in theory, a player with enough time on their hands to

play perfectly. In other words, such a game can be ‘solved’ in theory.

Of the many popular mind games, we thought that chess would best fit our purposes.

Playing a game of chess is complicated enough a problem to be well beyond being solved

by any foreseeable computer. This means that subtler techniques need to be employed in

order to create an AI that can play effectively such as estimating the value of different

positions that could occur in the future and using these estimates to narrow down which of

the available moves should be considered further when taking a turn. This provides

interesting programming and intellectual challenges.

Another point in favour of chess is that the rules are complicated enough to provide further

programming challenges. There are many different pieces, each with different ways of

moving and capturing. Additionally, some moves rely on what has happened previously in

the game such as castling which relies on the king and the rook never having moved, and

en passant which is available only for a single turn after a particular event has occurred.

Also, checking for stalemate requires comparing previous positions for repetition.

Finally, chess was the only mind game fulfilling the above criteria with which every

member of the team was familiar. It is also one of the most popular games in the world,

which will make it all the more satisfying to build an AI for.

Unlike some of the other projects, a huge amount of work has already been done on the

particular problem of writing chess AIs. With our very limited resources, we have

absolutely no chance of bettering existing projects in any way, with the possible exception

of adding some innovative UI features. This means that we have rather modest aims

relative to the current state of the art.

Following requirements gathering and discussion with the project supervisor, we are

aiming, at the minimum, to build a chess AI that can consistently beat a novice player,

obeying and accounting for all rules of the game with the possible exceptions of en

passant and stalemate, along with a functional UI displaying the state of the game and

allowing the player to make moves. The AI would need to guarantee making moves within

a short time frame, say, 0 – 20 seconds on a typical home computer.

If time allows, we aim to make an AI capable of beating a fairly skilled player,

acknowledging the full rule set, providing many more UI features such as displaying legal

moves, displaying previously played moves, allowing the user to undo moves and saving

and loading games. We would also like to offer multiple difficulty levels/playing styles and

enable the AI to play effectively to a clock, aborting the search before time runs out and

playing the best move according to the most accurate estimates made so far.

The extensive past work on the subject offers a rich pool of knowledge to guide our efforts.

The standard basis of a chess AI is a min-max search algorithm. It estimates the value of

each available move by observing the tree of following positions rooted at each. That is, it

considers the moves that the opponent could make in response to each of ours, and the

moves that we could make in response to each of the opponent’s moves and so on until

some fixed depth. For the resulting positions at this maximum depth, it estimates their

values using an ‘evaluation function’ which takes into account qualities of the positions

such as the difference in material value between the two sides. Now, assuming a faultless

opponent, it finds the estimated values of each of the AI’s available moves from the

current position by taking the worst (from the AI’s perspective) of the moves that the

opponent could make in response. The values of each of the opponent’s moves are found

by taking the best (again, from the AI’s perspective) of the moves that we could make in

response. It continues like this until it gets to the maximum depth, at which point it simply

uses the estimates made by the evaluation function.

A standard modification to this search pattern is called alpha-beta pruning, which reduces

the number of positions that must be searched by using the fact that some branches of the

search tree cannot influence the decision. For example, if we are trying to find the best of

two moves and one of the moves was estimated to have value -8 based on the 2 moves

that the opponent could make in response which had values -4 and -8, and we find that

one of the opponents responses to the other move that we could make had value -10, then

we know that we will not play the latter move since -10 is already worse that the worst

outcome of playing the former move. We don’t need to find the value of the other move

that the opponent could make in response because it could only make it worse.

Alpha beta pruning yields the most benefit if a cheap algorithm is used to order the moves

from a given position based on the expected utility for the player making the move. This

will increase the chances of better moves being found before lesser ones, allowing the

searches rooted at the lesser moves to be abandoned as soon as they are discovered to be

as such.

The above search strategy applied to chess assumes the search of the tree being cut off at

a certain point, using an evaluation function to estimate the value of the reached positions

simply by studying the state of the board as it is. However, using a naive cut-off point such

as having reached a certain depth can lead to problems when a move is chosen based on

a future position looking relatively good when actually, if the search had gone one level

deeper, it would have revealed a major change in direction such as the loosing of a queen.

This can be avoided by using a more sophisticated algorithm for deciding when to cut off

the search based on the level of activity of the position. For example, the search is cut off

if the depth is greater than some set depth AND no major pieces can be taken on the next

turn. This strategy is called quiescence.

Another trick we learnt was a way of representing the board to make legal moves very

easy and quick to generate. Using a 128 cell array rather than a 64 cell one and only using

every other 8 cells to refer to squares on the board (0 – 7, 16 – 23, and so on) allows

positions which are not on the board to be easily checked for. All indices in the other set of

cells (8 – 15, 24 – 31, and so on) have a high value for their 4th bit and thus will yield a nonzero

number when combined in a bit-wise AND operation with the number 0x88

(10001000). Similarly, all indices higher than 127 (or lower than zero if 8-bit two’s

compliment indices are used) will contain a high value for their 8th bit and so too will yield

a non-zero value when combined with 0x88.

So far, we have produced a program capable of arbitrating legal play (with a couple of

exceptions) between two humans playing at a command line interface. It can also replace

one or both of the human players with an AI using a simple min-max search and an

evaluation function which only takes into account the value of material on the board.

The next step is to add alpha-beta pruning and quiescence search, iterative deepening

(continuously re-doing the search, each time to a greater depth, possibly discarding moves

which were estimated to be poor in previous iterations). All the while we will be

researching and experimenting with new evaluation functions as well as developing a more

sophisticated UI.

The rest of the dissertation will detail the process and results of requirements gathering,

and then go on to explain the final design of our program and the process that lead to it.