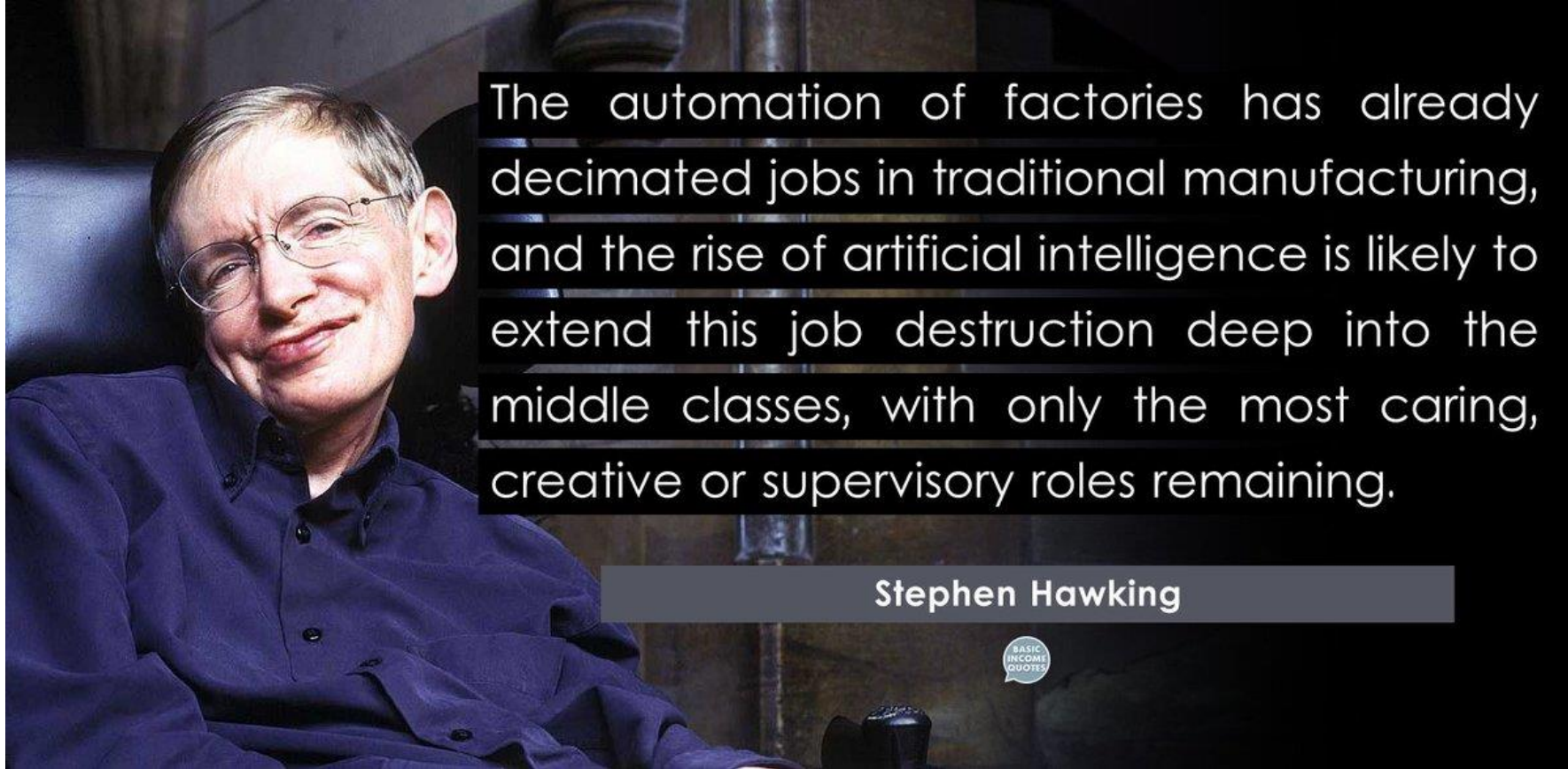


INTRODUCTION TO AI – CHESS + MACHINE LEARNING

Philip Mortimer



The automation of factories has already decimated jobs in traditional manufacturing, and the rise of artificial intelligence is likely to extend this job destruction deep into the middle classes, with only the most caring, creative or supervisory roles remaining.

Stephen Hawking

BASIC
INCOME
QUOTES



It [AI] would take off on its own and redesign itself at an ever increasing rate. Humans, who are limited by slow biological evolution, couldn't compete and would be superseded.

— *Stephen Hawking* —

AZ QUOTES

RECAP

- The minimax algorithm – an optimal adversarial search
- Alpha-beta pruning applied to minimax
- Design strategies to make effective AI
- Start by formulating problem in terms of input(s) some black box programmatic procedure and output(s)
- Break large problem down into solvable sub problems – known as decomposition
- We applied this to Tic Tac Toe to create AI that plays game perfectly
- Alpha-beta pruning drastically reduces search space – but even so, search space is exponentially large
- Depth limited minimax and why it's needed in most real world problems
- Iterative Deepening and why it's surprisingly more efficient than just depth limited search

OVERVIEW

- Build Chess AI with various approaches and evaluate their effectiveness
- Introduction to Machine Learning and Supervised learning

CHESS

- I imagine most people know what chess is
- But it's similar to tic tac toe in that two players (White and Black) take it in turn to make moves until one wins.





CHESS

- In order to put what we've learnt so far to the test, we'll discuss various strategies that we can use to produce an AI that plays chess.
- Chess is a classic AI problem which has decades of research dedicated to it.
- I have coded a chess application which can be downloaded here:
<https://github.com/philipmortimer/AI-Course/tree/main/Programs/Part%203/Chess%20User>
- You will need to install Java, which can be done here:
<https://github.com/philipmortimer/AI-Course/blob/main/Installation%20Help/AI%20Intro%20Course%20Install%20Guide%20-%20Java.pdf>
- If there are any issues, we can resolve them now

HOW TO USE APP

- Go to: <https://github.com/philipmortimer/AI-Course>

The screenshot shows the GitHub repository page for `philipmortimer / AI-Course`. The repository is public and has 1 branch and 0 tags. The main branch is selected. The repository contains a README.md file and a directory named `AI-Course`. The README.md file is the first commit, made 27 minutes ago. The `AI-Course` directory was updated 4 minutes ago. The repository description states: "Contains teaching materials used to teach my course which proves an introduction to AI." The repository has 0 stars, 1 watching, and 0 forks. There are no releases published.

Search or jump to... Pull requests Issues Marketplace Explore

philipmortimer / AI-Course Public

Pin Unwatch 1 Fork 0 Star 0

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

main 1 branch 0 tags

Go to file Add file Code

philipmortimer Updated some links 9c16948 4 minutes ago 3 commits

Installation Help	Added data	26 minutes ago
Programs	Updated some links	4 minutes ago
Slides	Updated some links	4 minutes ago
README.md	first commit	27 minutes ago

README.md

AI-Course

About

Contains teaching materials used to teach my course which proves an introduction to AI.

Readme

0 stars

1 watching

0 forks

Releases

No releases published

[Create a new release](#)

HOW TO USE APP

- Click on green “Code” button

The screenshot shows the GitHub interface for the repository 'philipmortimer / AI-Course'. The repository is public and has 1 branch and 0 tags. The 'Code' button is highlighted with a yellow arrow. The repository description is 'Contains teaching materials used to teach my course which proves an introduction to AI.' The repository has 0 stars, 1 watching, and 0 forks. The README.md file is visible at the bottom.

philipmortimer / AI-Course Public

Pin Unwatch 1 Fork 0 Star 0

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

main 1 branch 0 tags

Go to file Add file Code

philipmortimer Updated some links 9c16948 4 minutes ago 3 commits

Installation Help	Added data	26 minutes ago
Programs	Updated some links	4 minutes ago
Slides	Updated some links	4 minutes ago
README.md	first commit	27 minutes ago

README.md

AI-Course

About

Contains teaching materials used to teach my course which proves an introduction to AI.

Readme

0 stars

1 watching

0 forks

Releases

No releases published

[Create a new release](#)

HOW TO USE APP

- Press Download zip to download zip folder
- This will download a zip file to your computer

The screenshot shows a web browser displaying the GitHub repository page for 'philipmortimer / AI-Course'. The URL in the address bar is 'https://github.com/philipmortimer/AI-Course'. The repository is public and has 1 branch and 0 tags. The 'Code' button is highlighted, and its dropdown menu is open, showing options: 'Clone' (with sub-options: HTTPS, SSH, GitHub CLI), 'Open with GitHub Desktop', and 'Download ZIP'. The 'Download ZIP' option is highlighted with a yellow box and a yellow arrow pointing to it. The repository description states: 'Contains teaching materials used to teach my course which proves an introduction to AI.' The repository also has 0 stars, 1 watching, and 0 forks. The 'Releases' section shows 'No releases published' and a link to 'Create a new release'.

← → ↻ 🏠 🔒 https://github.com/philipmortimer/AI-Course 🔊 ⭐️ ✅ ⚙️ ⌵ 🔖 📁 👤 ..

🐱 Search or jump to... / Pull requests Issues Marketplace Explore 🔔 + 👤

📁 philipmortimer / AI-Course Public 📌 Pin 👁 Unwatch 1 🍴 Fork 0 ⭐ Star 0

<> Code ⓘ Issues 🔗 Pull requests ⏮ Actions 📁 Projects 📖 Wiki 🛡 Security 📈 Insights ⚙ Settings

🔗 main 🌿 1 branch 🏷 0 tags Go to file Add file ▾ Code ▾ ⚙ About

📁 philipmortimer Updated some links

📁 Installation Help	Added data
📁 Programs	Updated some links
📁 Slides	Updated some links
📄 README.md	first commit

📄 README.md

🔖 # AI-Course

📄 Clone ⓘ

HTTPS SSH GitHub CLI

https://github.com/philipmortimer/AI-Cour: 📄

Use Git or checkout with SVN using the web URL.

📄 Open with GitHub Desktop

📄 Download ZIP

📖 Readme

⭐ 0 stars

👁 1 watching

🍴 0 forks

Releases



No releases published

[Create a new release](#)

HOW TO USE APP

The zip file will have been downloaded to where your browser stores it downloads.

On Windows this section is called “Downloads”

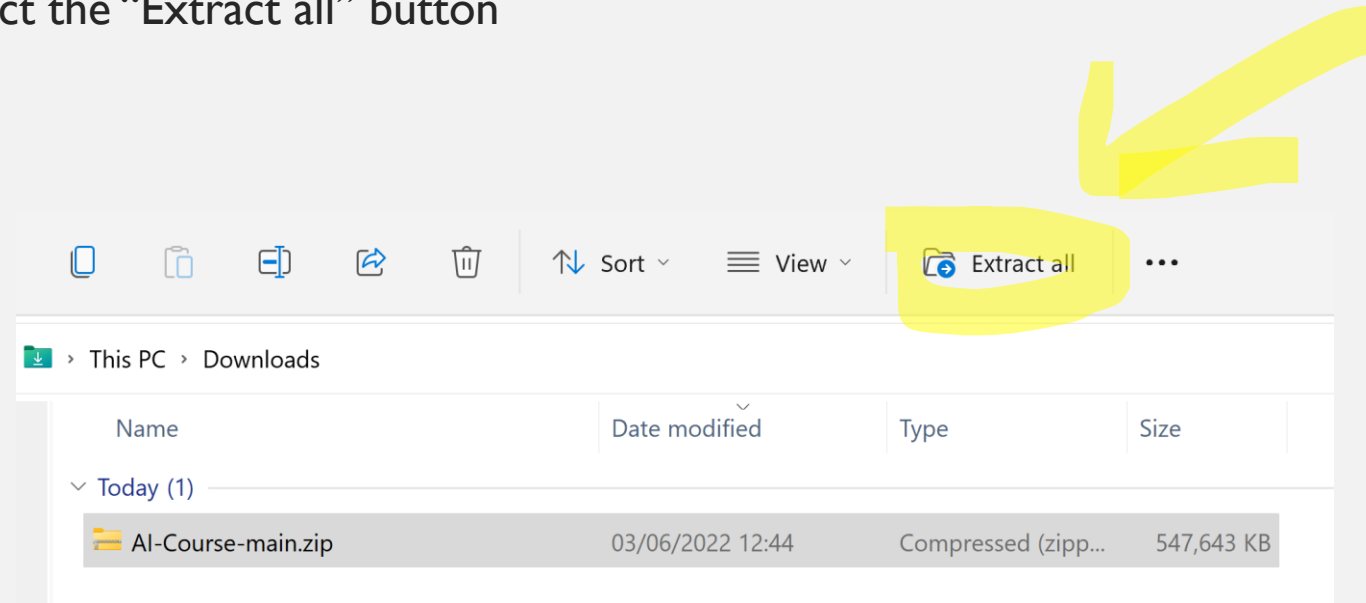
 > This PC > Downloads				
Name	Date modified	Type	Size	
▼ Today (1)				
 AI-Course-main.zip	03/06/2022 12:44	Compressed (zipp...	547,643 KB	

HOW TO USE APP

Extract zip file (this may already been done) to a folder.

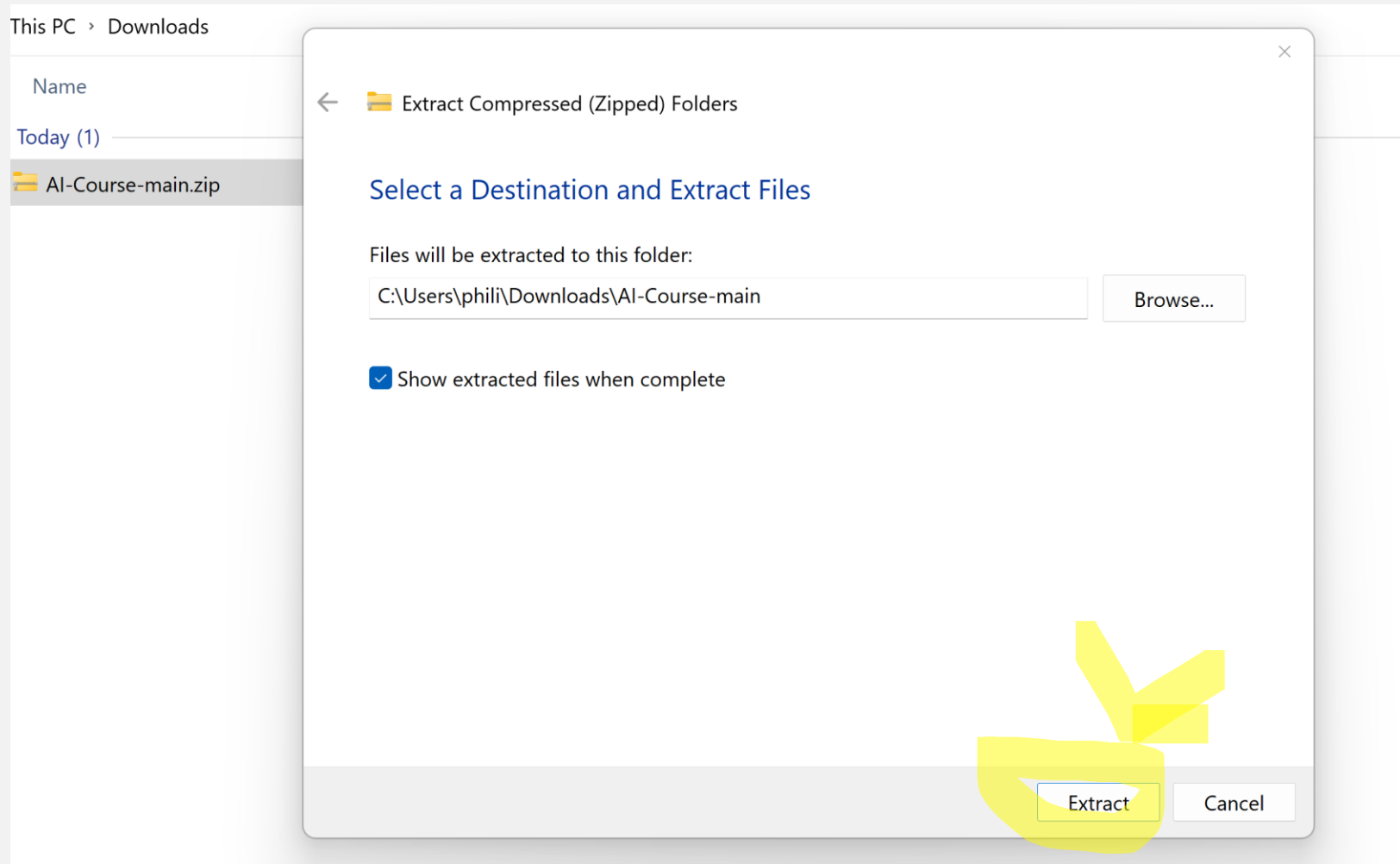
To do this on windows, first left click once on the ZIP file.

Select the “Extract all” button



HOW TO USE APP

Extract folder by pressing “Extract” button



HOW TO USE APP

You have extracted all the data needed for the course. The folder may be stored at a location along the lines of: “C:\Users\phili\Downloads\AI-Course-main” or “C:\Users\phili\Downloads\AI-Course-main\AI-Course-main” (phili is a username on my personal computer)

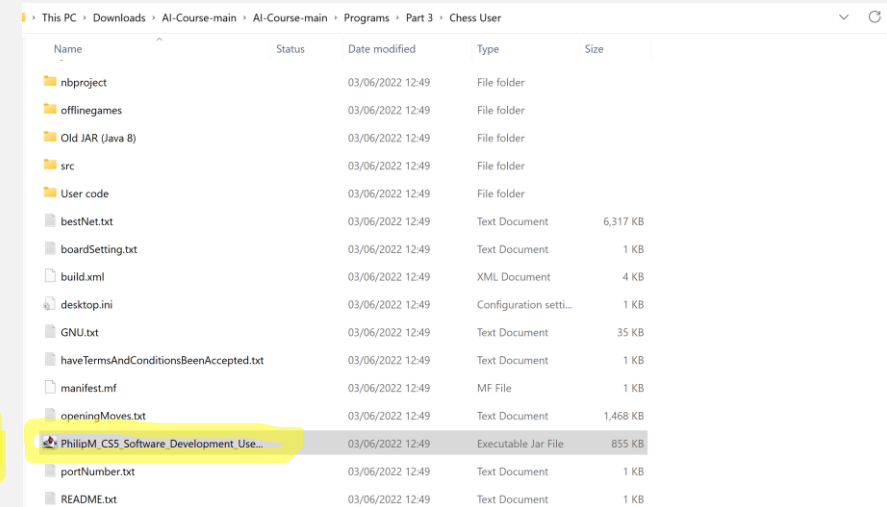
The folder needed for today should be in this folder. It can be found at somewhere along the lines of: “C:\Users\phili\Downloads\AI-Course-main\AI-Course-main\Programs\Part 3\Chess User”

There is a lot of code / resources in this folder. You can open this folder in NetBeans or IntelliJ to view the source code and also to run it.

You can also simply double click on “PhilipM_CS5_Software_Development_User_System.jar” to run it.

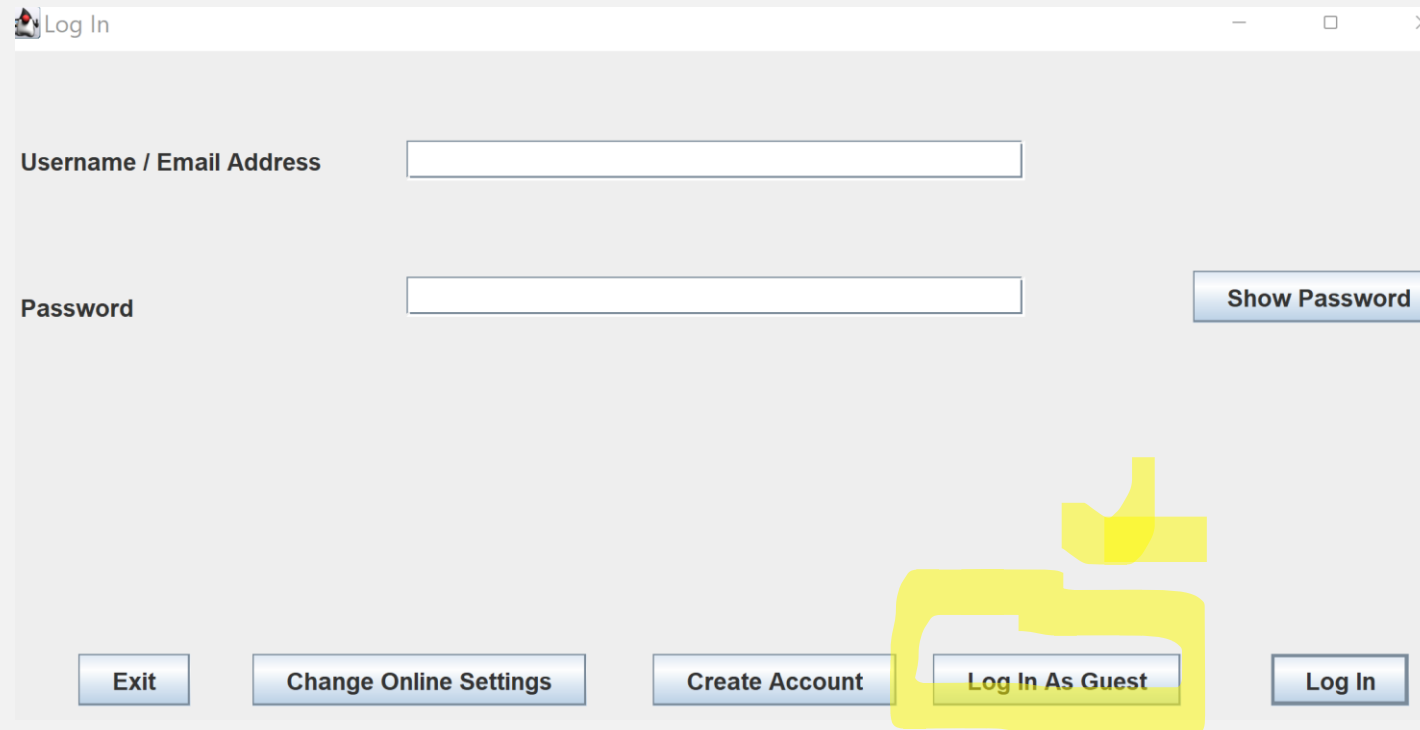
If you encounter issues with this, open the directory in the command line and type “java -jar .\PhilipM_CS5_Software_Development_User_System.jar”

This should run the program



HOW TO USE APP

Select “Log In As Guest”



The screenshot shows a 'Log In' window with a title bar containing a small icon and the text 'Log In'. The window has standard minimize, maximize, and close buttons. The main area contains two input fields: 'Username / Email Address' and 'Password'. To the right of the password field is a 'Show Password' button. At the bottom, there is a row of five buttons: 'Exit', 'Change Online Settings', 'Create Account', 'Log In As Guest', and 'Log In'. The 'Log In As Guest' button is highlighted with a yellow rectangular box, and a yellow arrow points to it from the right.

Log In

Username / Email Address

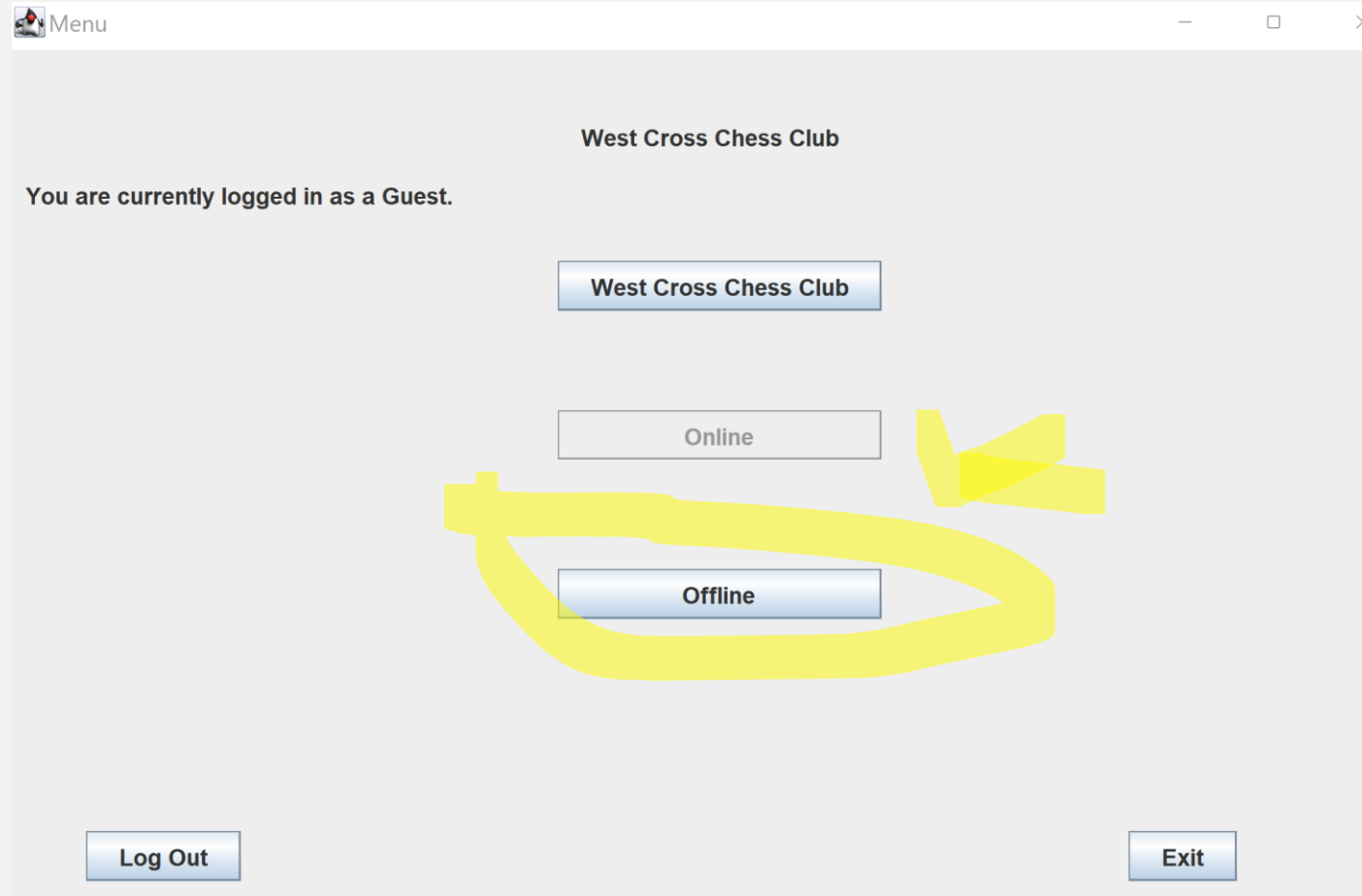
Password

Show Password

Exit Change Online Settings Create Account Log In As Guest Log In


HOW TO USE APP

Select “Offline”



HOW TO USE APP

- From here, the menu is fairly self-explanatory. Select new game against computer to play against AI.



The screenshot shows a window titled "Create New Game" with standard window controls (minimize, maximize, close) in the top right corner. The window has a light gray background and contains the following elements:

- Game Title:** A text input field containing "Game1".
- Computer Difficulty:** A row of five buttons: "1 (Easiest)", "2", "3", "4", and "5 (Hardest)".
- You play as:** A row of three buttons: "White", "Random", and "Black".
- Time Control:** A row of three buttons: "5 Minutes", "10 Minutes", and "Unlimited".
- Back:** A button located at the bottom left of the window.
- Create Game:** A button located at the bottom right of the window.

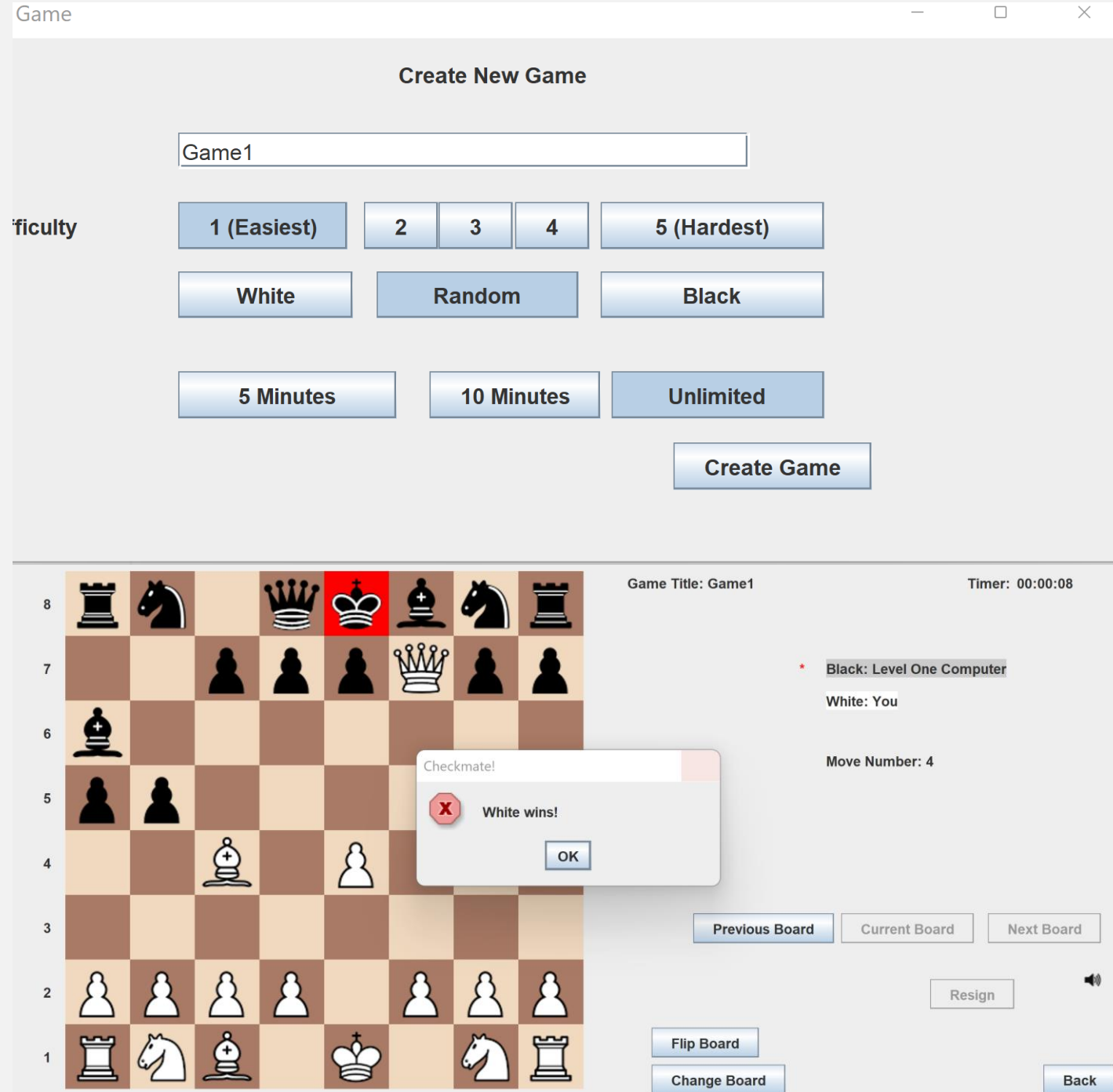
APPROACH #1 RANDOM

- The first approach, and by far the simplest, is to simply return a random move.
- This obviously plays very badly, but is sometimes our only option when we have little / no knowledge about our domain.
- Random moves can be combined with advanced techniques like Monte Carlo tree search, reinforcement learning, unsupervised learning and genetic algorithms.
- In fact, cutting edge AI is often applied to fields with little specific knowledge. Sometimes, blind AI that uses no pre-assumed knowledge actually performs far more effectively than domain specialised systems
- At any rate, picking just random moves is about as poor as an AI will ever play. We can see this when we play it ourselves on the app (difficulty setting 1)

```
Move randomMove(Board b){  
    List<Move> allMoves = b.getAvailableMoves();  
    Random randNumGenerator = new Random();  
    return allMoves.get(randNumGenerator.nextInt(allMoves.size()));  
}
```


APPROACH #1 RANDOM

- Even a novice like myself can checkmate the random AI very quickly!
- We obviously need a better approach



APPROACH #2 DEPTH-LIMITED MINIMAX

- Following on from last lesson, we saw how using the Minimax algorithm is an effective technique for solving turn based games like chess
- If we apply the same idea to chess, we would never compute an answer as the game space is too large.
- Hence, we would introduce a depth limit after which no position will be evaluated. This means that we will only ever search a maximum of **X** moves into the future.

APPROACH #2 DEPTH-LIMITED MINIMAX

- Like before, assign some values for a white win, draw and black win (say $+\infty$, 0, $-\infty$). These values will be assigned at terminal board states (i.e. checkmate and stalemate).
- If we reach the search depth but don't reach a terminal state, we use some heuristic to estimate the board value at the current position.
- We then propagate these values through the minimax tree. We use alpha-beta pruning to speed up the search.

APPROACH #2 DEPTH-LIMITED MINIMAX

- For our heuristic, we will take the weighted sum of the number of each piece a player has.

Symbol					
Piece	pawn	knight	bishop	rook	queen
Value	1	3	3	5	9

- Note this essentially means we say that a bishop is worth roughly three pawns
- Our heuristic is expressed mathematically in the following way:

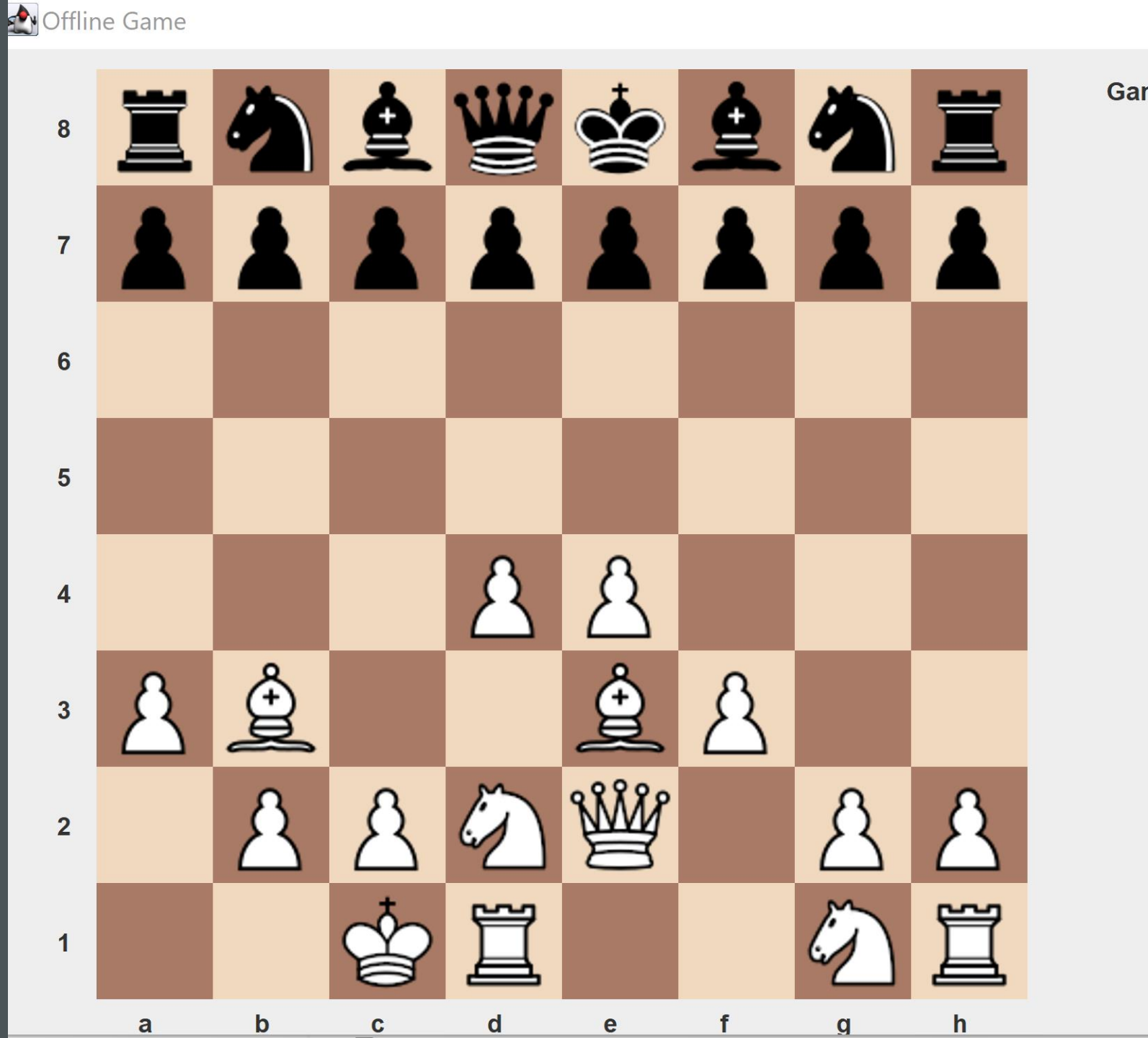
$$\text{boardValue} = (\text{whitePawns} + 3\text{whiteKnights} + 3\text{whiteBishops} + 5\text{whiteRooks} + 9\text{whiteQueens}) - (\text{blackPawns} + 3\text{blackKnights} + 3\text{blackBishops} + 5\text{blackRooks} + 9\text{blackQueens})$$

APPROACH #2 DEPTH-LIMITED MINIMAX

- This heuristic is what's commonly taught to new chess players.
- And in combination with a computer that can search many moves, it actually performs pretty well – it beats me more often than not
- If you select difficulty level 3, you can see this implemented in the program.
- The AI does very well in complex middle and end game positions where both players can make many captures.
- However, it plays awfully at the beginning. Simply shuffling random pieces back and forward. Why is this?
- The system can't search far enough forward to determine which initial moves lead to losing bad board states. In short, the AI has no sense of positional awareness – only materialistic equality.

APPROACH #2 DEPTH-LIMITED MINIMAX

- Currently, our AI says that this board game is completely equal. This is because both players have the same number of pieces. Clearly however, white is massively better (and will win this game with perfect play)



APPROACH #3 POSITIONAL DEPTH-LIMITED MINIMAX

- Let's adopt the same framework but choose a more advanced heuristic that encourages positional play as well as material play
- Firstly, we will use very similar weights for each of the pieces
- For each piece, we will have a matrix which stores some value for roughly how good each square is for that piece.

- Here is the pawn table:

```
// pawn
0, 0, 0, 0, 0, 0, 0, 0,
50, 50, 50, 50, 50, 50, 50, 50,
10, 10, 20, 30, 30, 20, 10, 10,
5, 5, 10, 25, 25, 10, 5, 5,
0, 0, 0, 20, 20, 0, 0, 0,
5, -5, -10, 0, 0, -10, -5, 5,
5, 10, 10, -20, -20, 10, 10, 5,
0, 0, 0, 0, 0, 0, 0, 0
```

- Notice how the table encourages pawns to push forward so that they may be promoted to a queen. Pawns in the centre are encouraged to delve into enemy territory whilst pawns in the corners may wish to hold back to protect the king.

APPROACH #3 POSITIONAL DEPTH-LIMITED MINIMAX

- Similar tables are available for all pieces (and for various phases of the game). Such an example may be found at:
https://www.chessprogramming.org/Simplified_Evaluation_Function
- These tables lead to good positional awareness, but don't do enough to ensure strong opening chess moves.
- This is because the opening is so far from the end of a chess game that we are unable to search deep enough down the game to really see the consequence of our moves.
- To remedy this, I downloaded a small database of ~12,000 chess grandmaster games. I extracted the opening five moves from this database. If our AI encounters a move from this database, it simply plays the next move in the database (i.e. a move played by a Grandmaster).
- This leads to much better opening play for database positions.
- Of course, advanced chess players could deliberately stray from chess theory in order to force the computer to play a poor move in the opening phase.

APPROACH #3 POSITIONAL DEPTH-LIMITED MINIMAX

- Let's try this new approach and play against the computer at difficult setting level 4!
- This AI is actually pretty good and rather difficult to beat.
- For future enhancements, we could implement iterative deepening (which is especially effective given that chess uses time controls).
- We could store search results to a table and, use move ordering techniques. For example, any capture move made by a pawn is likely to be a good move.

FLAWS

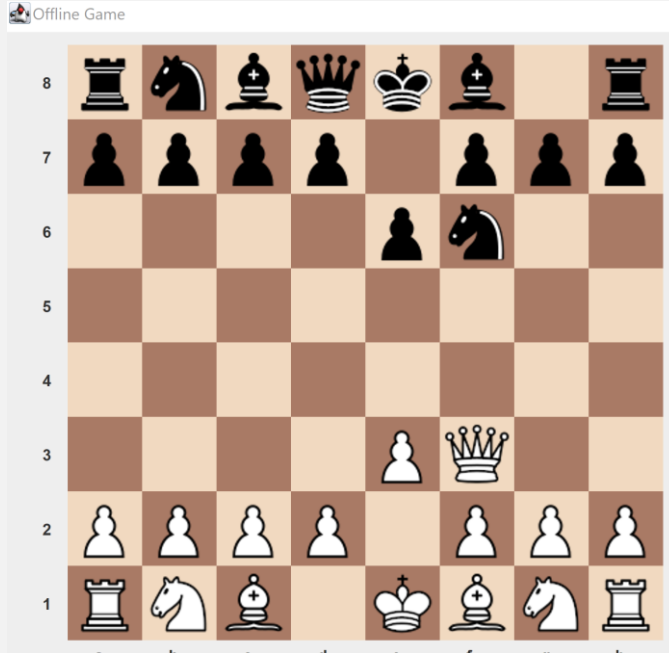
- There are many flaws – the AI is still fairly materialistic. It also can't search deep enough into the future to see some things that are still obvious to humans.
- Perhaps the biggest problem is that it stops searching at a given depth.
- Assume this current position is at a depth one below the maximum depth.



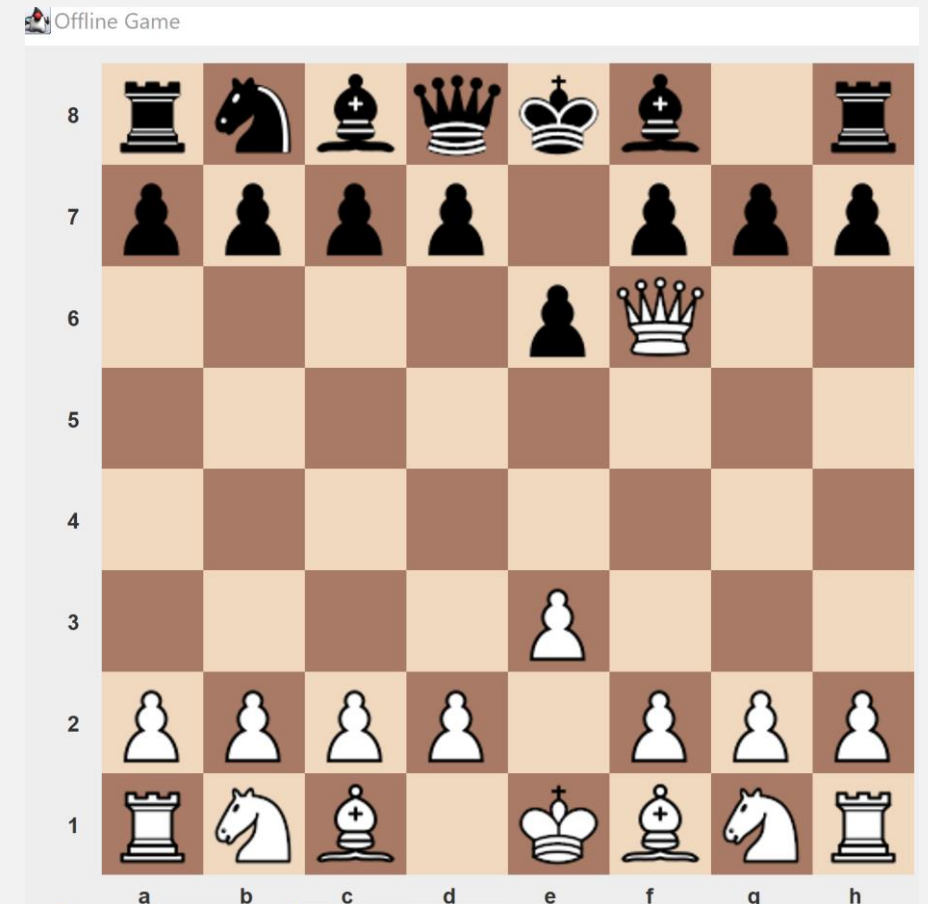
Offline Game



FLAWS

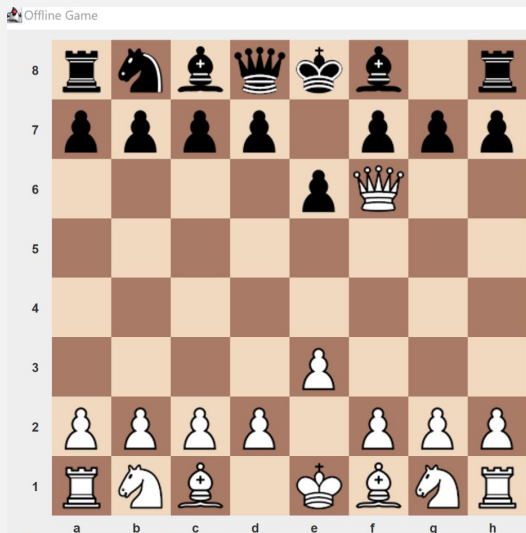
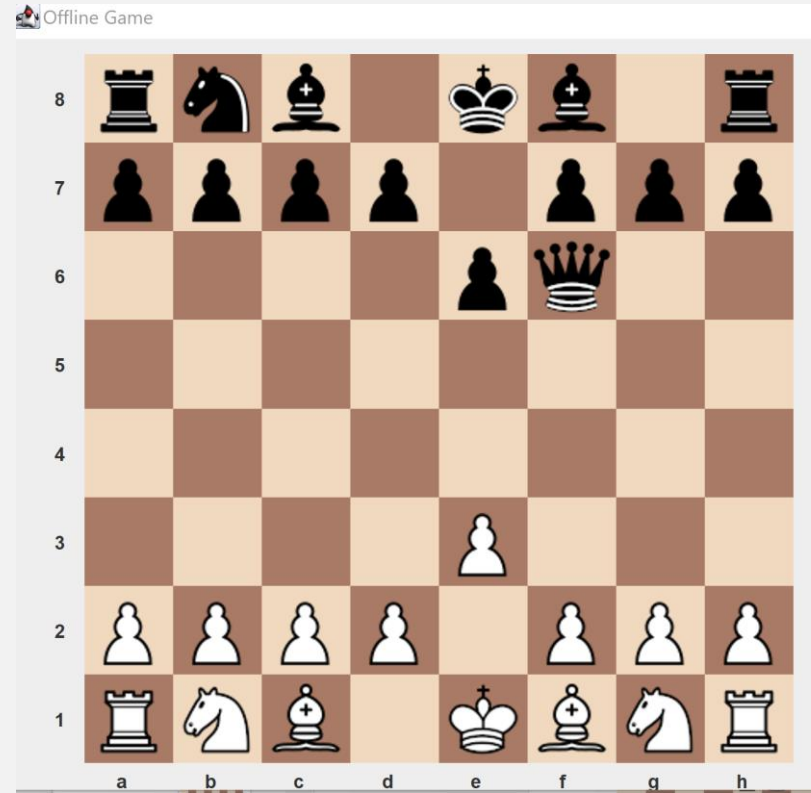
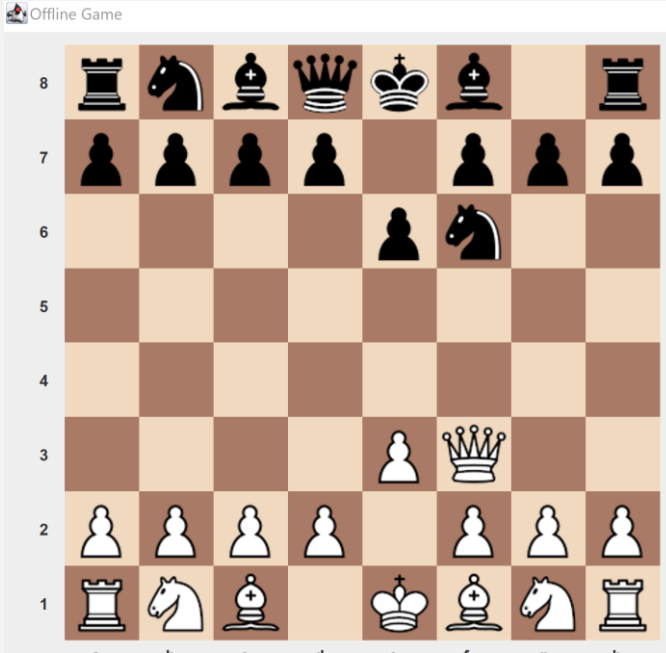


- The AI would search the position where the Queen captures the knight, but it wouldn't search any deeper than that.
- According to our evaluation function, the position is +3 (we are a knight ahead of our opponent)



FLAWS

- The AI would search the position where the Queen captures the knight, but it wouldn't search any deeper than that.
- According to our evaluation function, the position is +3 (we are a knight ahead of our opponent)
- However, on the very next turn, our opponent captures our Queen with no consequences. Thus the actual position is -6.



FLAWS

- In order to address this flaw of strict depth limited searches, we use a Quiescence Search at our search depth.
- This search performs a much smaller minimax style search, only searching capture moves. This forces our system to only apply our heuristic to stable states (states where no capture moves are possible on the very next turn).
- There are a whole range of advanced optimisations that can be made, but the truth is that an alpha-beta pruned system using a positional heuristic can achieve very high level play.

DEEP BLUE

- Throughout the years, there have been many attempts to develop AI opponents capable of beating top chess players.
- Most famously, IBM's deep blue computer beat reigning chess world champion Gary Kasparov 3.5 – 2.5 in 1997. This was the first time a computer had beaten a chess world champion.
- This famous win came after three previous losses against Kasparov. The system undeniably exhibited mastery of chess. There is a valid school of thought that argues that Kasparov was still a superior chess player on average.
- For example a glitch in the 44th move of the second game led to the computer selecting a random move as a fail safe to exit an infinite loop. This move spooked Kasparov, who misattributed the pointless move to “superior intelligence”.
- However, it certainly did beat Kasparov fair and square. And deep blue was essentially just using a heuristic optimised by several grandmasters through repeated trial and error. It still used an alpha-beta pruned minimax search (albeit on a very powerful computer),
- Deep blue typically searched 6-8 moves ahead (although it searched up to 20 moves ahead in some positions).



LIMITATIONS OF HARD-CODED SYSTEM

- Deep blue and our style of deterministic programming have just about reached the peak that such hard-coded systems can reach
- A hard coded system is a system that relies on human knowledge that can be easily explained and programmatically expressed. We can express exactly how our heuristic function works (summing all the pieces). We can also express exactly how the minimax search works
- There are some systems where this just isn't possible. For example, if chess had been invented yesterday, we would have no idea what constitutes a good heuristic and no easy way of measuring it.
- In short, the skill of our AI is highly dependent on programmers having specialist knowledge on a field. This is a problem for complex topics
- Additionally, there are some fields which are simply too complex for humans alone to develop effective heuristics.

MACHINE LEARNING

- Machine learning is the solution to these systems
- Machine learning is a **subset** of AI
- Machine learning (often abbreviated as ML)
- ML covers a series of techniques used to allow machines to learn from data intelligently
- **Machine Learning** - the use and development of computer systems that are able to learn and adapt without following explicit instructions, by using algorithms and statistical models to analyse and draw inferences from patterns in data
- This allows us to apply data to a problem without explicitly understanding the nature of the solution

TYPES OF LEARNING

- ML can broadly be broken down into three categories: Supervised learning, unsupervised learning and reinforcement learning.
- Supervised learning is the most common application of ML and extremely powerful. Supervised learning uses a large dataset of **labelled** data.
- Labelled data is a set of data that is human annotated. E.g. we may have 10,000 images with each image labelled as either “cat” or “dog”.
- Unsupervised learning also uses sets of data, however the data is not labelled. E.g. we may have 100,000 images but we have no idea whether they are “cat” or “dog”
- Reinforcement learning involves some form of model and program defined actions that are either good or bad. The model learns which actions to perform to maximise their score through trial and error. E.g., in a game of Mario, killing an enemy may lead to a reward of 10 points and reaching the end of a level may lead to 100 points. Dying may lead to -20 points. Through repeatedly playing the game, the AI will figure out how to beat a level.

TYPES OF LEARNING

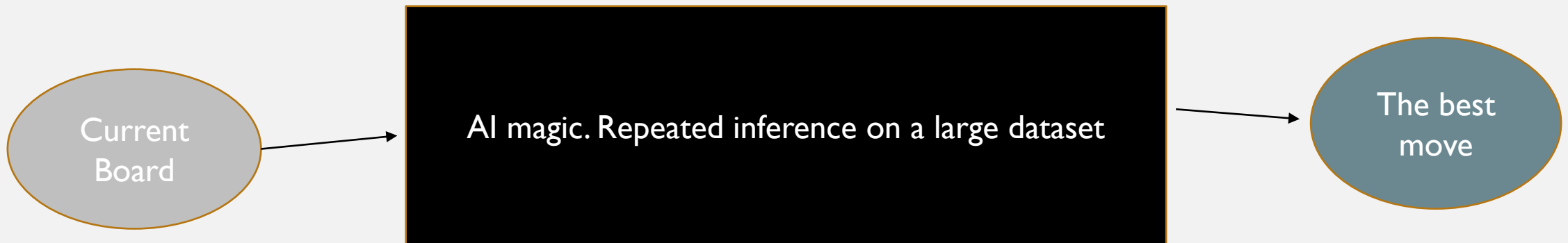
- Genetic algorithms are also a form of machine learning. They imitate biological process, like gene pool slicing, genetic mutation, natural selection and evolution in order to produce agents that can perform tasks.
- Genetic algorithms have also been applied to Mario. We may have 100 “Mario's” with certain “genes”. The 10 Mario's who make it furthest into the level will have their genetic pool spliced together to produce 100 new “Mario's”.
- Typically, supervised learning leads to the best results most quickly. However, gathering labelled data can be extremely expensive. AI systems are also limited by the quality of data labelling (and thus often, human intelligence).
- Unsupervised learning is slower but is not reliant on human annotation, sometimes leading to systems far superior to people. Also often more cost-effective.
- Genetic algorithms and reinforcement learning require no pre acquired data, only some valid model implementation. Many systems will simply not have data available to be gathered and hence these often represent great choices.

SUPERVISED LEARNING

- The most common (and currently the most important technique) in AI is surely supervised learning.
- Supervised involves a model repeatedly looking at a large amount of labelled data in order to better model a given problem space
- This process is iterative and can be performed for any number of iterations. The more time is given to training, the better the model will typically be.
- Supervised learning is often considered a black box process. We have access to the inputs and outputs of the AI but we don't really understand how it achieves what it does.
- We can empirically verify that it performs its task effectively. We know exactly how the model is trained.
- However, despite knowing the precise mathematical formula that converts our model input into an output, we often simply have no way of expressing what the model is doing in human terms.

SUPERVISED LEARNING

- Remembering our black box design approach from the Tic Tac Toe AI, we can actually use something rather similar for ML models:
- Unlike the Tic Tac Toe AI however, this time we let the machine decide how to implement the black box.



APPROACH #4 MACHINE LEARNING

- Supervised learning has been applied to chess for decades with mixed results. Up until the last ~10 years, supervised systems typically failed to beat advanced human players
- Current state of the art supervised systems however, can play better than any human player
- Let's attempt to design our own chess machine learning model using basic supervised learning

APPROACH #4 MACHINE LEARNING

- Lets define a model that takes in a chess board as input and returns the score of the board for the current game state.
- Some papers suggest that use of bitboards make an effective representation of a game of chess for ML systems.
- A bitboard is an $8 * 8$ grid of 0 and 1 's representing whether a piece is either there or not. So for white pawns we will have a bitboard to represent all of their locations. We will have an $8 * 8$ bitboard for each type of piece (e.g. white rook, black bishop etc.). We will have four number to store castling rights and one number for potential en passant rights.
- This gives us 773 input numbers.

APPROACH #4 MACHINE LEARNING

- We will use a neural network as our model (the thing that converts an input board to a game score).
- We will discuss neural networks in detail in a future lesson, but for now we can simply think of them as a magic black box. A neural network consists of a bunch of levers and dials that can be moved in the machine learning process to produce different values.
- The training process involved changing these dials in a mathematical way so that the system produces accurate predictions.

APPROACH #4 MACHINE LEARNING

- For our output we want to produce some score for the game state.
- Thus we will have two output nodes: the probability that the game will end in a white win or draw and the probability that the game ends in a black win or draw. Thus our score is simply $p(\text{whiteWin}) - p(\text{blackWin})$.
- We use two nodes instead of one, as this separation of output classes has been shown to often achieve strong results.
- We will experiment with different neural network hyperparameters and architectures (which will be discussed more in subsequent lessons).

APPROACH #4 MACHINE LEARNING

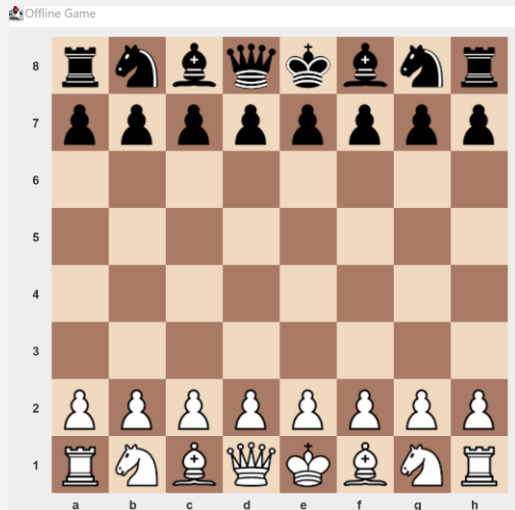
- To train our AI, I have downloaded a database of roughly 1.1 million games high level chess. This consists of both chess grandmaster games and other AI games
- From each game, I have extracted 10 random positions
- The goal of our model is simple: given a random position from a random position, predict the probability that the game will end in a win for white or a win for black.
- Games that end in a draw have been excluded as some papers advocate for this approach.

APPROACH #4 MACHINE LEARNING

- For a given game where white eventually wins, the result should 1,0 (i.e White is 100% certain to end up winning).
- We repeatedly adjust the dials within our neural network given our output and the correct output. This process produces an AI that becomes better over time.
- I have trained such an AI and it can be accessed using difficulty setting 2 on my chess program. Let's try it!

APPROACH #4 MACHINE LEARNING

- We simply substitute the machine learning heuristic into our minimax AI.



APPROACH #4 MACHINE LEARNING

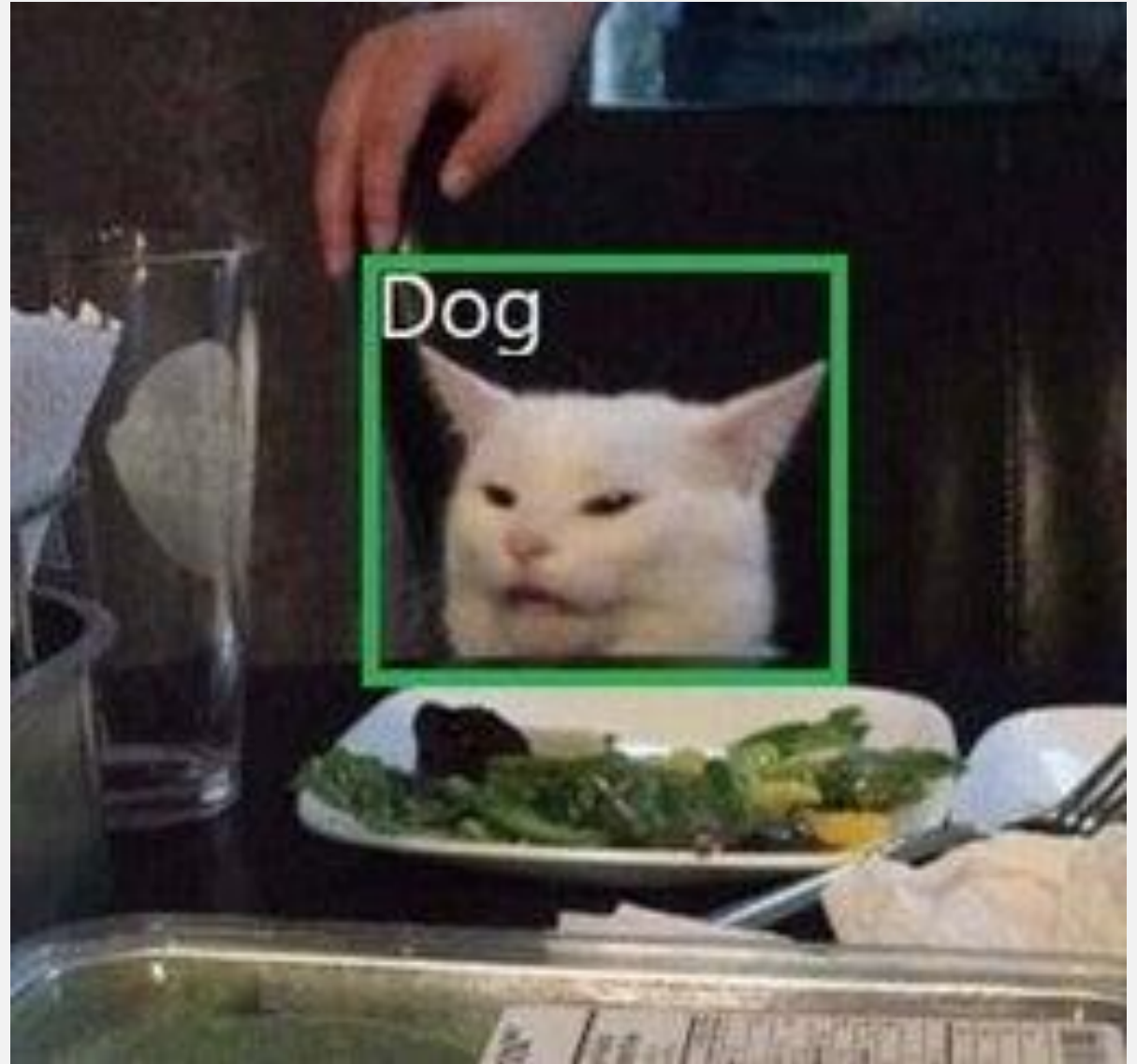
- Unfortunately, our AI system does not play chess very well. It shows flashes of brilliances. Occasionally, it will play really high level chess and positionally sound moves.
- Most of the time however, it will simply blunder away pieces
- There are a number of reasons for this poor play.
- The biggest is simply the lack of compute power. My model was trained on my personal laptop with poorly optimised code. Training machine learning model requires lots of energy, processor power and time.
- Given some tweaking and more compute time, similar models have proven their value when applied to chess.

APPROACH #4 MACHINE LEARNING

- Additionally, the dataset was poorly chosen. It turn out that computer chess is full of AI making massive mistakes, encouraging the computer to do the same.
- Some argue that perfect chess will probably end in a draw. Typically games where one player triumphs over another is because a mistake has been made by one player at some point in the game (often at the end)
- Therefore, many boards labelled as “eventual win for white” and “eventual win for black” are actually equal in value and simply just “draws” (which we failed to fully model).
- A better system may look at pairs of board states and determine which one is better. For example, if we have a board from a game that results in a win and one that results in a loss, we can say with decent confidence that $\text{score}(\text{win}) \geq \text{score}(\text{lose})$
- Given more compute time, I would have created a more complex neural network model. However, the more complex the model, the slower the minimax search. Thus bigger models mean searching fewer moves into the future (when often a simple heuristic and searching further into the future is more effective for weak hardware).

APPROACH #4 MACHINE LEARNING

- That being said, the model did undoubtedly learn and does sometimes play well. Especially towards the end of the game, the system will often produce move sequences far beyond the capability of the simplistic minimax algorithm.



APPROACH #5 STOCKFISH

- Stockfish is the world's best chess player and far exceeds the capability of any human player. The current best chess player has an ELO rating of 2700. Stockfish has a rating of 3400. The required amount to be a grandmaster is 2500 (i.e. Stockfish will basically never lose to a human player).
- Stockfish is actually an open source engine, meaning that anyone is able to view and distribute the source code and executable (even commercially).
- Stockfish use a minimax alpha beta search and iterative deepening. It uses parallel processing and many other optimisations. However, the core of the algorithm is nonetheless minimax.
- It uses a hand crafted heuristic (which is obviously very complex) but nonetheless explainable by a human.

APPROACH #5 STOCKFISH

- The latest releases of Stockfish also use a neural network evaluation function which far exceeds the hand crafted heuristic. This neural network uses supervised learning as part of the training process.
- Indeed, we can train a neural network to mimic stockfish. This is actually how the initial weights for the neural network were selected for the current Stockfish version.
- Stockfish proves that supervised learning (as well as other machine learning techniques), in combination with traditional AI algorithms can produce superhuman performance.

APPROACH #5 STOCKFISH

- To play against Stockfish select difficult level 5 on the AI. Note that this will not work on non-windows systems.
- If you don't have a windows system, you may play against Stockfish here <https://lichess.org/> or <https://www.chess.com/play/computer>
- If you manage to beat the AI, you are either a cheat or the next chess world champion!

SUMMARY

- We have looked at a number of ways to create AI systems that can play chess at various levels.
- These skill levels vary drastically
- We have looked at minimax implementations with various heuristic functions, random moves and machine learning as a way of producing superhuman heuristics
- We discussed the different form of machine learning, particularly supervised learning.
- We will revisit and discuss ML in further depth in future lessons.

QUESTIONS



FURTHER READING

- Stockfish: <https://github.com/official-stockfish/Stockfish>
- Heuristics: https://www.chessprogramming.org/Simplified_Evaluation_Function
- Simple Chess AI: <https://www.freecodecamp.org/news/simple-chess-ai-step-by-step-1d55a9266977/>
- Machine learning intro: <https://towardsdatascience.com/introduction-to-machine-learning-for-beginners-eed6024fdb08>
- Supervised ML model for chess: <https://arxiv.org/pdf/1711.09667.pdf>