Board	
• Display the grid of the board on the screen • Display the numbers and letters for each intersection on the screen • Store each of the pieces on the screen • Check if each move is legal before accepting a move by using the rules class • Keep track of who the current player is so that the AI can query the board • Return a game state of the board after a certain move is applied • Store all of the legal possible moves by using the rules class • Check if a player has won the game in a given board state	• Piece • Go Rules • Monte Carlo Class • Game class

Piece	
• Store the position of the piece on the board • Store the colour of the piece • Display the representation of the piece on the screen	• Colour • Board

Colour	
• Store the colour of a piece on the screen • Store as an empy colour if the piece has not been placed yet	• Piece • Go Rules

player_turn	
• Store who's turn it is supposed to be	• Game • Rules • Board

Go Rules	
• Return if a move is legal • Go through each rule and check that a move complies by it • Be able to find all of the places on the board where a legal move can be played so that the ai is able to make a move • Give an evaluation for a position • Remove all captured pieces from the state when a legal move is played	• Piece • Board • Monte Carlo • Alpha Beta

Monte Carlo Tree	
• Calculate the best move from the current position • Play out random games from the current position to try and find the best move • Store all of the previous game states to be able to remember what the best move was • Store how many moves they calculated • Abide by the time limit of the game	• Board • Game • Go Rules

Game

- Run the game loop
- Check for inputs from the user
- Display the UI
- Make sure that the correct screen is being displayed, eg. Main menu, game and game over
- Render the Game board
- Alter who's turn it is after each move
- Query the Alpha Beta for a move if it is their turn and give correct time limit and board state
- Query the Monte Carlo Tree Search for a move when it is their turn with correct time limit and board state
- Send move data from algorithms to the database
- Send game data to the database

- Board
- Piece
- Main function
- Alpha Beta
- Monte Carlo
- DatabaseCRUD
- DatabaseMove
- DatabaseGame
- PlayerTurn
- GoRules

DatabaseGame

- Store the results of the game
- Store the game id
- Store which algorithm/player each player was
- Store the time allowed

- database CRUD operations
- game

DatabaseMove

- Store the colour of the player
- Store the player related to the move (Alpha beta, Monte Carlo etc)
- Store the number of calculated moves (not applicable for actual player)
- Store the board size
- Store the time allowed
- Store the move number
- Store the game id

- Game
- Database CRUD

DatabaseZobrist

- Store the id for the specific move
- Store the score related to the state

- Alpha Beta
- Game

DatabaseCRUD

- Insert moves into the move table
- Get all of the moves by a specific player
- Update a specific move
- Remove a move from the database
- Insert a game into the games table
- Get games by a specific player for both colours
- Remove games from the database

- Game
- Database Move
- Database Game

Generate Database

- Create go_data database
- Create moves table
- Create games table

- Command line usage only

generate zobrist database

- Create Zobrist Database
- Create states table

- command line only

Zobrist	
• Create hash values for a specific state	• Game • Go Rules • Board

PlayerType	
• Store if a human or algorithm is playing	• Game

ProximalPolicyOptimisation	
• Load environment for model • Load model with correct setup • Train model • Save model to file	

PPO_Load_Model	
• Load environment correctly • Load PPO model from file • Play the environment using the model and a real player	

DQN	
• Create model layout for the deep q network • Implement ability to remember states, along with their reward and move taken • Implement the ability to act on a specific state • Implement an ability to replay as to not forget old information	• Train DQN

Train DQN	
• Create DQN agent • Train dqn model for a certain number of episodes and steps per game • Change action to random legal action if invalid move is played by the model • Reshape the environment for the model to be able to play correctly • Save the model to a file after all episodes or after max score is reached • Call functions on the dqn to get it to remember states and replay • Progress the state based on the actino taken • Reset State properly after each episode	• DQN