AI310/CS361 ARTIFICIAL INTELLIGENCE FALL 2024

COURSE PROJECT INSTRUCTIONS

Instructions to Students:

- This is a group work project. Each group consists of **Five to Six students** "Mainstream Programme" / **Four to Six** "both Soft. Eng. & Med. Info. Programmes" *(the Teaching Assistant must approve group members through registration)*. Each group must develop the idea assigned to them using Python.
    - **Project Objectives:** The objectives of this project can be summarised as applying the main ideas, fundamental concepts, and basic algorithms in the fields of artificial intelligence and machine learning.
- **Submission:** Submission is done according to the following schedule:
    - **Week 12: Submission and Discussion of the** (1) **Project** and (2) **Documentation**. *The report should include the following: (1) Project idea in detail, (2) Main functionalities, (3) Similar applications in the market, (4) A literature review of Academic publications (papers) relevant to the idea (at least 4 to 6 papers, as per the number of team members), (5) the Dataset employed (preferably a publicly available dataset), (6) Details of the algorithm(s)/approach(es) used and the results of the experiments, and (7) Development platform.*
- **Assessment:** Assessment will be on the reports, code submitted, and discussions with team members. All the team members must contribute to all the phases, and the role of each member must be clearly stated in each report.
    - The Project will be assessed based on the following criteria:
        - The complexity of the problem, & the correctness of the algorithms employed.
        - The quality/comprehensiveness of your experiments & documentation.
        - The correctness of your analysis and design diagrams.
        - Implementation correctness.
- **Feedback:** If requested, further details and feedback could be provided for each group through discussions with the teaching assistant(s) during the weekly-labs/office-hours.
- You can only submit your work. Any student suspected of plagiarism will be subject to the procedures set out by the Faculty/University (including failing the course entirely).
    - **Academic Integrity:** The University's policies on academic integrity will be enforced on students who violate University standards of academic integrity. Examples of behaviour that is not allowed are:
        - Copying all or part of someone else's work and submitting it as your own;
        - Giving another student in the class a copy of your work and
        - Copying parts from the internet, textbooks, etc.
        - If you have any questions concerning what is allowed, please don't hesitate to discuss them with me.
    - **We understand that you might positively refer to AI tools to support your research. Please note that, in case there are any concerns about AI-generated submission content, you will be invited for an opportunity to verify and defend your work.**

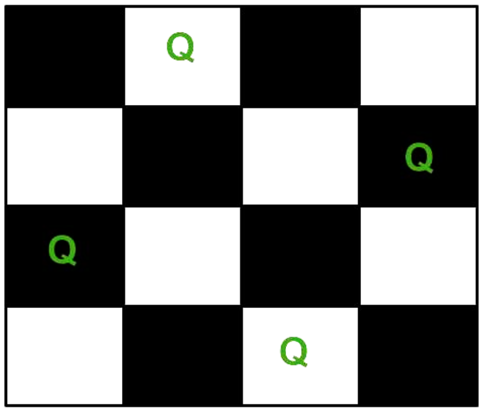
AI310/CS361 Artificial Intelligence Fall 2024

Course Project Descriptions (20 Ideas)

Table of Contents

1) N-Queens Problem Solver (for different sizes – n should be selected by the user) using the Backtracking Search Algorithm, a Best-First Search, a Hill-Climbing Search, AND a Genetic Algorithm.

N-Queens Problem Solver Project Overview: The N-Queens Problem Solver project aims to develop an intelligent system capable of solving the N-Queens problem for various board sizes. The N-Queens problem is a classic chessboard puzzle where the objective is to place N queens on an N×N chessboard in such a way that no two queens threaten each other. Threatening means no two queens share the same row, column, or diagonal. The value of N, representing the board size and the number of queens, can be chosen by the user. For example, following is a solution for 4 Queen problem:



Problem Description: In the N-Queens problem, the challenge is to arrange N queens on an N×N chessboard, ensuring that no two queens attack each other. The difficulty arises from the restrictive nature of queen movements; they can traverse horizontally, vertically, and diagonally. The project involves exploring multiple algorithms to efficiently find solutions to this puzzle.

Backtracking Search Algorithm: The Backtracking Search Algorithm is a systematic method for exploring potential solutions to a problem. In the context of the N-Queens problem, it involves placing queens on the board one by one and backtracking if a conflict is detected. This algorithm guarantees finding all possible solutions.

Best-First Search Algorithm: Best-First Search is an algorithm that intelligently selects the most promising path based on a heuristic evaluation. In the N-Queens context, this algorithm evaluates board configurations using a heuristic to guide the placement of queens, prioritizing paths that seem most likely to lead to a solution.

Hill-Climbing Search Algorithm: Hill-Climbing is a local search algorithm that continually moves towards higher elevations in the solution space. Applied to the N-Queens problem, it involves iteratively adjusting queen placements to ascend towards a configuration with fewer conflicts. It may get stuck in local optima.

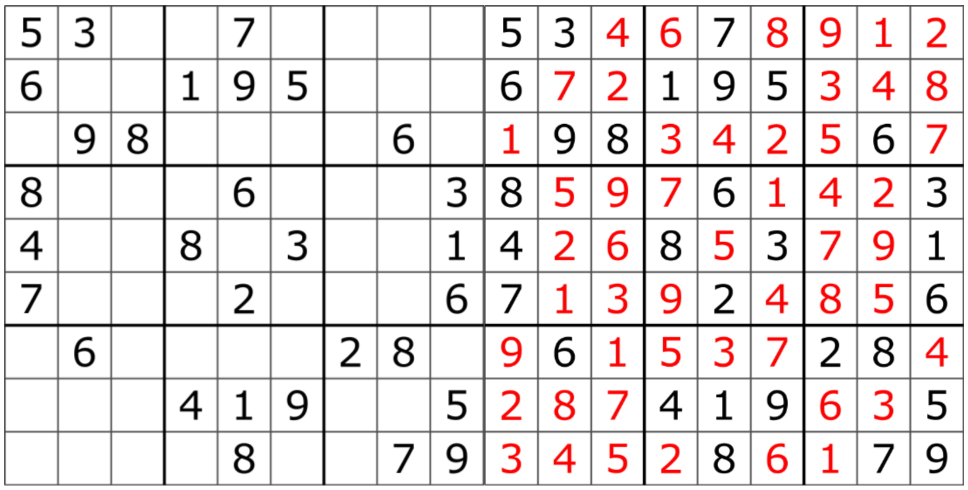
Genetic Algorithm: The Genetic Algorithm mimics the process of natural selection to evolve solutions. In the N-Queens context, a population of potential solutions undergoes genetic operations like crossover and mutation. This approach explores a diverse solution space, often finding effective solutions.

User Interaction: The user has the flexibility to choose the size of the chessboard (N), making the project adaptable to different scenarios. The system will present solutions generated by each algorithm, allowing the user to compare their effectiveness and efficiency. Additionally, the project provides insights into the strengths and limitations of diverse search and optimization techniques.

This project not only addresses the fundamental challenge of the N-Queens problem but also serves as an educational tool for understanding and comparing various search algorithms in artificial intelligence. It combines classic problem-solving techniques with contemporary optimization strategies.

2) A Sudoku Puzzle Solver using the Backtracking Algorithm AND a Genetic Algorithm.

Description: The project aims to create an intelligent Sudoku puzzle solver utilizing two distinct algorithms: the Backtracking Algorithm and the Genetic Algorithm. Sudoku is a popular number-placement puzzle where a 9x9 grid must be filled with digits from 1 to 9, ensuring each row, column, and 3x3 subgrid contains all digits without repetition. Solving Sudoku involves exploring possible configurations and finding a combination that satisfies all constraints. For example, the (left) figure demonstrates a typical Sudoku puzzle, and its solution (right).



5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Backtracking Algorithm: The Backtracking Algorithm is a systematic approach to problem-solving, particularly effective for constraint satisfaction problems like Sudoku. It explores possible solutions incrementally, placing digits in empty cells and backtracking when conflicts arise. In the context of Sudoku, the Backtracking Algorithm will intelligently fill cells, ensuring adherence to the rules of the puzzle.

Genetic Algorithm: The Genetic Algorithm is an evolutionary approach inspired by natural selection. It operates on a population of potential solutions, evolving towards optimal configurations over successive generations. In the Sudoku context, the Genetic Algorithm will create a population of Sudoku grids, applying genetic operations like crossover and mutation to generate new solutions. The fitness of each solution is determined by its adherence to Sudoku rules.

Project Requirements:

User Interaction: The project will provide a user-friendly interface where individuals can input a Sudoku puzzle or select from pre-existing ones.

Algorithm Integration: Both the Backtracking Algorithm and the Genetic Algorithm will be implemented and integrated into the solver. Users can choose which algorithm to employ.

Solution Visualization: The solver will visually display the step-by-step process of solving the Sudoku puzzle, allowing users to understand the algorithms' decision-making.

Customization: Users can customize the size of the Sudoku grid (e.g., 4x4, 6x6, 9x9) to cater to various puzzle complexities.

Performance Metrics: The solver will provide performance metrics, such as the number of iterations, to help users evaluate the efficiency of each algorithm.

This project not only enhances understanding of fundamental algorithms but also provides a practical tool for Sudoku enthusiasts to effortlessly solve puzzles of varying complexities. Users can experiment with different algorithms, gaining insights into their strengths and limitations in tackling Sudoku challenges.

3) An Intelligent Connect-Four Player using the Minimax Algorithm, Alpha-Beta Pruning, and Heuristic Functions.

Build an artificial Connect-Four player that can play games against a human opponent. Connect Four *(also known as Four Up, Plot Four, Find Four, Captain's Mistress, Four in a Row, Drop Four, and Gravitrips in the Soviet Union)* is a two-player connection board game, in which the players choose a color and then take turns dropping colored discs into a seven-column, six-row vertically suspended grid. The pieces fall straight down, occupying the lowest available space within the column. The objective of the game is to be the first to form a horizontal, vertical, or diagonal line of four of one's own discs. Connect Four is a solved game. The first player can always win by playing the right moves.

Project Description: The goal of this group project is to design and implement an intelligent Connect-Four player, leveraging AI techniques such as the Minimax Algorithm, Alpha-Beta Pruning, and Heuristic Functions. Connect Four is a popular two-player game where the objective is to connect four of one's own colored discs in a horizontal, vertical, or diagonal row before the opponent does.

Minimax Algorithm: Minimax is a decision-making algorithm used in two-player games with perfect information, like Connect Four. It evaluates possible moves by simulating the entire game tree and selects the move that minimizes the maximum possible loss. The Minimax Algorithm can be implemented to create an intelligent Connect Four player capable of making optimal moves to maximize its chances of winning or achieving a draw.

Alpha-Beta Pruning: Alpha-Beta Pruning is an optimization technique applied to the Minimax Algorithm. It reduces the number of nodes explored in the game tree by eliminating branches that are guaranteed not to impact the final decision. Integrating Alpha-Beta Pruning enhances the efficiency of the Connect-Four player, making it more responsive and capable of handling larger search spaces.

Heuristic Functions: Heuristic functions provide a shortcut for evaluating game states without exhaustively exploring all possible moves. In Connect Four, heuristic functions can assess the current board state based on factors like the number of connected discs and potential winning configurations. By incorporating heuristic functions, the AI player can make informed decisions without exhaustively searching the entire game tree.

Project Requirements:

User Interface: Develop a user-friendly interface allowing players to interact with the Connect-Four AI seamlessly.

Algorithm Integration: Implement the Minimax Algorithm with Alpha-Beta Pruning to create an efficient decision-making process for the Connect-Four player.

Heuristic Design: Devise heuristic functions that capture essential aspects of the game, guiding the AI player towards optimal moves.

Difficulty Levels: Provide adjustable difficulty levels to accommodate players of varying skill levels. Higher difficulty levels may involve deeper search depths or more sophisticated heuristics.

Visual Feedback: Display the Connect-Four AI player's thought process, moves, and decision-making to enhance the player's understanding and engagement.

Testing and Evaluation: Conduct rigorous testing to ensure the AI player performs effectively across different scenarios. Evaluate its performance using metrics such as win rate and average decision time.

This project offers an opportunity to delve into the realm of artificial intelligence and game-playing algorithms while creating an intelligent Connect-Four player that challenges and entertains players. The combination of Minimax, Alpha-Beta Pruning, and heuristic functions will contribute to the development of a strategic and adaptive game-playing agent.

4) Knight's Tour Problem Solver (for different sizes – n should be selected by the user) using the Backtracking Search Algorithm, AND a Genetic Algorithm.

The Knight's Tour is a captivating chess problem where the challenge is to find a sequence of moves for a knight to visit every square on the chessboard exactly once. This project, "Knight's Tour Problem Solver," aims to develop a software solution that efficiently tackles this intriguing problem using two distinct approaches: the Backtracking Search Algorithm and a Genetic Algorithm.

Problem Overview: In the Knight's Tour, the knight, a chess piece capable of unique L-shaped moves, must traverse the chessboard, covering each square precisely once. The problem is both combinatorial and algorithmic, requiring strategic exploration to determine a viable sequence of knight moves.

Approaches:

(a) Backtracking Search Algorithm: Backtracking involves systematically exploring potential moves while intelligently abandoning paths that lead to dead-ends. This algorithm efficiently navigates the knight through the chessboard. The Backtracking Search Algorithm is a systematic and resource-efficient method for solving the Knight's Tour. It optimizes exploration by discarding unfruitful paths, ensuring a feasible solution.

(b) Genetic Algorithm: Genetic Algorithms emulate evolutionary processes, employing genetic operations like crossover and mutation to evolve a population of potential solutions. The algorithm adapts over successive generations to find an optimal knight's tour. Genetic Algorithms bring a stochastic and adaptive approach to solving the Knight's Tour. By exploring a diverse set of move sequences and evolving over generations, the algorithm aims to discover effective paths.

Requirements:

User-Selected Board Size: An additional feature of this project is user customization. Users can select the size of the chessboard (n), allowing them to explore the Knight's Tour for different board dimensions. This flexibility enhances the project's applicability and provides users with a tailored experience.

Interactive Interface: Develop a user-friendly interface allowing users to input the chessboard size (n) and visualize the knight's tour solutions.

Backtracking Implementation: Implement the Backtracking Search Algorithm to systematically explore and find a solution for the Knight's Tour based on user-defined board size.

Genetic Algorithm Implementation: Design and integrate a Genetic Algorithm to provide an alternative solution strategy, fostering diversity and adaptability.

Visualization: Include a graphical representation of the knight's movements on the chessboard, aiding users in understanding and appreciating the solutions generated.

Optimization: Strive for optimized algorithms to handle larger chessboard sizes efficiently.

This project aims to not only solve the Knight's Tour problem but also to offer users an engaging and interactive experience, allowing them to explore the fascinating world of chessboard traversal using advanced algorithms.

5) An Intelligent Tic-Tac-Toe Player using the Minimax Algorithm, Alpha-Beta Pruning, and Heuristic Functions.

Problem Description: The project aims to develop an intelligent Tic-Tac-Toe player that employs advanced algorithms and heuristic functions to make strategic decisions during gameplay. Tic-Tac-Toe is a classic two-player game where opponents take turns marking X or O in a 3x3 grid, aiming to form a line of three of their symbols horizontally, vertically, or diagonally. While Tic-Tac-Toe is a simple game, creating an intelligent player involves implementing algorithms that can explore the game tree efficiently, make optimal moves, and potentially predict the opponent's actions.

Requirements of the Project:

Minimax Algorithm Implementation: The team will implement the Minimax algorithm, a decision-making strategy suitable for adversarial games like Tic-Tac-Toe. Minimax evaluates all possible moves, considering the best and worst outcomes, to make optimal decisions.

Alpha-Beta Pruning Integration: The project requires the integration of Alpha-Beta Pruning with the Minimax algorithm. Alpha-Beta Pruning is a technique to enhance the efficiency of Minimax by eliminating unnecessary branches in the game tree, reducing computational overhead.

Heuristic Functions Design: The team will design heuristic functions to provide the AI player with a quick evaluation of the game state. Heuristic functions guide the AI in making strategic decisions by assigning values to different board configurations, helping prioritize moves that lead to favorable outcomes.

User Interaction: The Tic-Tac-Toe player should have a user-friendly interface for interaction. This includes displaying the game board, allowing users to make moves, and presenting the AI's decisions in a clear and understandable manner.

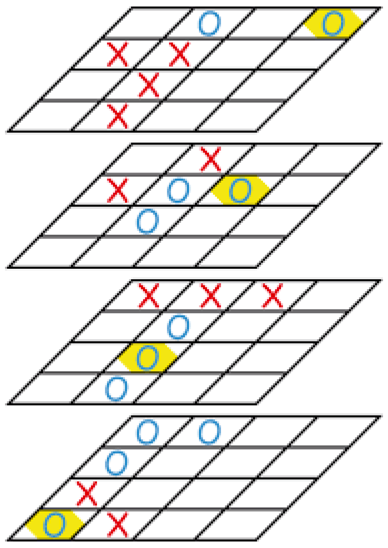
Optimization and Performance: The team should focus on optimizing the algorithms and ensuring the player responds swiftly to user inputs. Performance enhancements, such as parallel processing or algorithmic improvements (such as Symmetry Reduction), can be explored to achieve a seamless gaming experience.

By combining the Minimax algorithm, Alpha-Beta Pruning, and heuristic functions, the project aims to deliver an intelligent Tic-Tac-Toe player capable of challenging human opponents with strategic and optimal gameplay. The successful completion of this project will showcase the team's proficiency in implementing algorithms for game playing scenarios.

6) An Intelligent Cubic Player using the Minimax Algorithm, Alpha-Beta Pruning, and Heuristic Functions.

Introduction: The project aims to create an intelligent player for the Cubic game, a variant of Tic-Tac-Toe played on a 4x4x4 three-dimensional grid. The player will be designed to make strategic moves using artificial intelligence techniques, specifically implementing the Minimax algorithm, Alpha-Beta Pruning, and heuristic functions.

Problem Description: Cubic is an engaging board game that introduces an additional layer of complexity compared to traditional Tic-Tac-Toe. Played on a 4x4x4 grid, each player takes turns placing their markers in an attempt to create a line of four in any direction within the three dimensions. The challenge lies in the spatial nature of the game, requiring players to think strategically in three dimensions to secure a winning position.



Project Requirements:

Minimax Algorithm Implementation: Develop the Minimax algorithm tailored for the Cubic game. The algorithm should traverse the three-dimensional game space, evaluating possible moves and selecting the optimal move that maximizes the chances of winning or minimizes the chances of losing.

Alpha-Beta Pruning Integration: Implement Alpha-Beta Pruning alongside the Minimax algorithm to enhance computational efficiency. Alpha-Beta Pruning eliminates unnecessary evaluations in the game tree, reducing the time needed to find the best move.

Heuristic Functions Design: Design heuristic functions specific to Cubic to provide a quick assessment of the current board state. Heuristics should capture spatial patterns and configurations that lead to winning positions, aiding the AI player in decision-making.

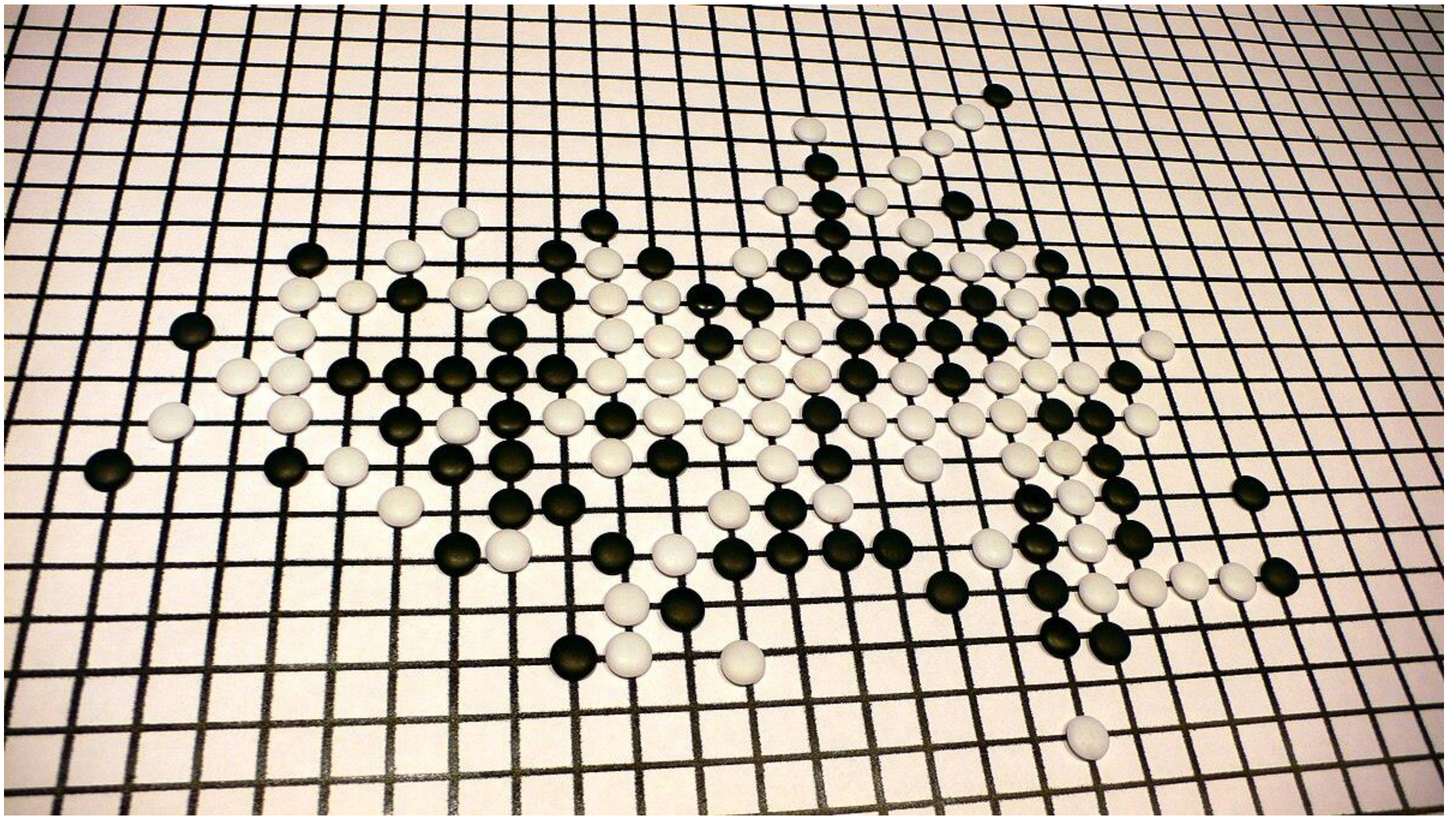
User Interface Development: Create a user-friendly interface for interacting with the Cubic player. The interface should allow users to make moves on the 4x4x4 grid and display the AI's decisions in an understandable manner, enhancing the overall user experience.

Optimization: Optimize algorithms for responsiveness and efficiency, considering the computational demands of the three-dimensional game space.

Expected Outcome: The project's successful outcome will be an intelligent Cubic player capable of making strategic moves on the 4x4x4 grid using the implemented Minimax algorithm, Alpha-Beta Pruning, and heuristic functions. The player should provide a challenging and enjoyable gaming experience, showcasing the capabilities of artificial intelligence in mastering three-dimensional spatial reasoning.

7) An Intelligent Connect-6 Player using the Minimax Algorithm, Alpha-Beta Pruning, and Heuristic Functions.

Description: The project aims to develop an intelligent player for the Connect-6 game, an extended version of Connect Four, where players strive to connect six markers in a row horizontally, vertically, or diagonally on a 19x19 grid. This board game introduces increased complexity compared to Connect Four, making it a challenging problem for artificial intelligence (AI) systems to navigate strategically.



Project Requirements:

Minimax Algorithm Implementation: The project requires the implementation of the Minimax algorithm, a decision-making approach commonly used in two-player games. Minimax explores potential moves by recursively evaluating game states to determine the optimal strategy for maximizing wins or minimizing losses.

Alpha-Beta Pruning Integration: Integrating Alpha-Beta Pruning with the Minimax algorithm is essential for optimizing the AI player's decision-making process. Alpha-Beta Pruning helps reduce the number of evaluated game states, making the algorithm more efficient without compromising strategic depth.

Heuristic Function Design: Develop heuristic functions that provide a quick evaluation of a given board state. Heuristics should capture patterns and configurations conducive to winning, guiding the AI player's decisions in situations where an exhaustive search is impractical.

User-Selected Board Size (n): Allow users to select the size of the Connect-6 board (n), introducing flexibility and adaptability to different game scenarios. The AI player should be capable of adjusting its strategy based on the chosen board size, accommodating variations in complexity.

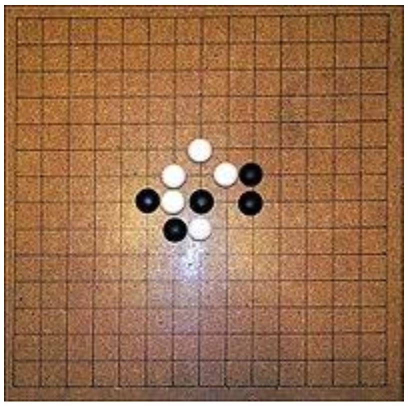
User-Friendly Interface: Design a user-friendly interface that allows players to interact seamlessly with the intelligent Connect-6 player. The interface should facilitate input from users regarding the desired board size and enable them to engage in a strategic game against the AI.

Optimization and Performance: Optimize the AI algorithms for performance, considering computational efficiency and responsiveness. Strive to create an intelligent player that balances strategic depth with real-time responsiveness during gameplay.

Conclusion: This project aims to deliver an intelligent Connect-6 player that combines the strategic depth of Minimax, the efficiency of Alpha-Beta Pruning, and the adaptability of heuristic functions. The implementation should provide users with a challenging and engaging gaming experience, showcasing the capabilities of AI in strategic decision-making within the context of board games.

8) An Intelligent Gomoku Player using the Minimax Algorithm, Alpha-Beta Pruning, and Heuristic Functions.

Description: The goal of this group project is to create a smart player for the Gomoku game using advanced AI techniques. Gomoku, also known as Five in a Row, is a classic board game where two players take turns placing their markers on a (usually 15x15) grid. The objective is to be the first to achieve a continuous sequence of five markers either horizontally, vertically, or diagonally.



Game Rules:

__Board__: Gomoku is typically played on a 15×15 grid, though variations with different board sizes exist.

***Players**: The game is played between two players, Black plays first.*

***Turns**: Players take turns placing their symbols on vacant intersections. Players take turns placing their markers (usually black and white stones or "X" and "O") on the empty grid.*

***Objective**: The first player to achieve an unbroken row of five symbols wins. The row can be horizontal, vertical, or diagonal.*

***Blocked Board**: If the board is filled without a winner, the game is a draw.*

In some rules, this line must be exactly five stones long; six or more stones in a row does not count as a win and is called an overline.

Project Requirements:

Minimax Algorithm Implementation: Develop the Minimax algorithm, a strategic decision-making tool that examines potential moves and predicts outcomes. It ensures the AI player makes moves maximizing its chances of winning and minimizing the chances of losing.

Alpha-Beta Pruning Integration: Integrate Alpha-Beta Pruning with Minimax for efficient decision-making. Alpha-Beta Pruning helps the AI player disregard unpromising branches in the decision tree, optimizing the search process.

Heuristic Function Design: Create heuristic functions that quickly evaluate the current state of the game. These functions should capture strategic elements like potential threats, defensive moves, and advantageous positions to guide the AI's decision-making.

User Interface: Design a user-friendly interface that allows players to interact with the intelligent Gomoku player. The interface should display the game board, accept player moves, and provide a seamless experience.

Adaptive Difficulty Levels: Implement varying difficulty levels that adapt to different player skill levels. The AI player should dynamically adjust its strategy, offering a challenging experience for both beginners and experienced players.

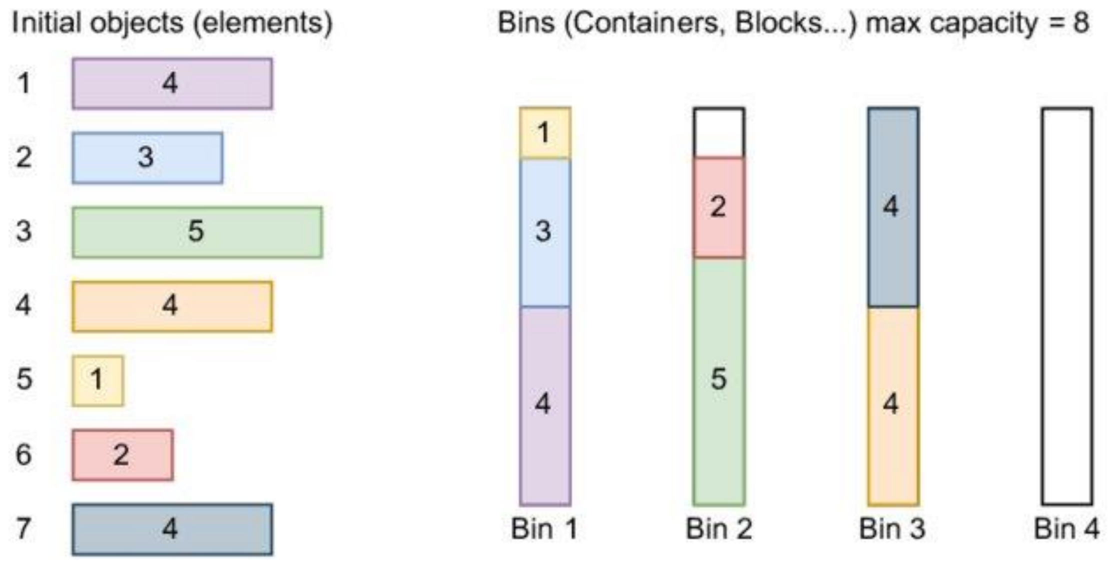
User-Selected Board Size (n): Allow users to choose the size of the Gomoku board (n), providing flexibility for different gaming scenarios. The AI player should be capable of adapting to various board sizes.

Optimization: Optimize the AI algorithms to balance strategic sophistication and responsive performance. The intelligent Gomoku player should provide real-time responses during gameplay.

Conclusion: The Intelligent Gomoku Player project aims to deliver a sophisticated AI opponent for the Gomoku game. By implementing Minimax, Alpha-Beta Pruning, and heuristic functions, the project seeks to provide users with an enjoyable and challenging gaming experience while showcasing the capabilities of AI in strategic board games.

9) Bin Packing Problem Solver using the Backtracking Search Algorithm, AND a Genetic Algorithm.

Description: The Bin Packing Problem is a classic optimization challenge where the objective is to efficiently pack a set of items into a minimum number of bins or containers. Each item has a specific size, and the bins have a limited capacity. The goal is to find an arrangement that minimizes the number of bins used. This problem has real-world applications in logistics, resource allocation, and space optimization.


Initial objects (elements)
Bins (Containers, Blocks...) max capacity = 8
1
4
2
3
3
5
4
4
5
1
6
2
7
4
Bin 1
Bin 2
Bin 3
Bin 4

Problem Rules:

***Items**: There is a set of items, each with a defined size or volume.*

***Bins**: A limited number of bins or containers are available, each with a maximum capacity.*

***Packing Objective**: The objective is to pack all items into the bins in a way that minimizes the number of bins used.*

***Item Placement**: Items cannot be split or partially placed in different bins; they must be placed whole.*

***Bin Capacity**: The total size of items in a bin cannot exceed its maximum capacity.*

Project Requirements: The project aims to develop a Bin Packing Problem Solver using two different approaches – the Backtracking Search Algorithm and a Genetic Algorithm. The solver will address the following key components:

Backtracking Search Algorithm: Develop the Backtracking Algorithm to explore possible combinations of item placements in bins. Enhance the algorithm to efficiently prune branches and explore feasible solutions.

Genetic Algorithm: Design a genetic representation of bin configurations and explore genetic operators like mutation and crossover. Implement a fitness function to evaluate the quality of a given bin configuration. Allow the genetic algorithm to evolve populations toward better solutions.

User Interface: Enable users to input item sizes, bin capacities, and other relevant parameters.

Algorithm Selection: Provide an option for users to choose between the Backtracking Algorithm and the Genetic Algorithm.

Visual Representation: Implement a visual representation of the bin-packing solutions generated by both algorithms.

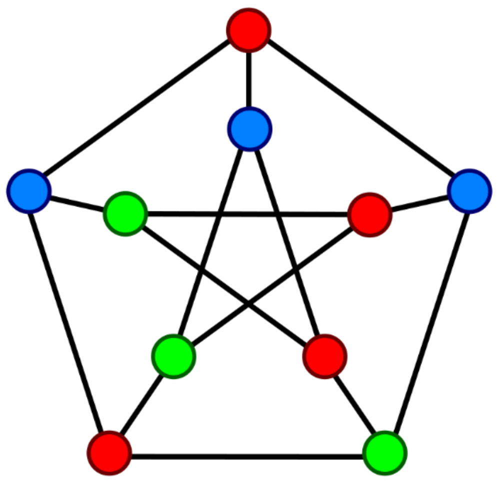
Comparison Display: Allow users to compare the solutions produced by the two algorithms.

Performance Metrics: Implement metrics to assess the efficiency and effectiveness of both algorithms. Analyze and display comparative results, such as solution quality and computation time.

Outcome: The project's success will be measured by the ability of the developed solver to efficiently and accurately solve the Bin Packing Problem using both the Backtracking Search Algorithm and the Genetic Algorithm. The user interface should provide a clear understanding of the solutions generated by each algorithm, enabling users to compare their performance.

10) Graph Colouring Problem Solver using the Backtracking Search Algorithm, AND a Genetic Algorithm.

Introduction: The Graph Coloring Problem is a well-known combinatorial optimization challenge that involves assigning colors to the vertices of a graph in such a way that no two adjacent vertices share the same color. This project aims to develop an intelligent Graph Coloring Problem Solver utilizing both the Backtracking Search Algorithm and a Genetic Algorithm.



Problem Description: In the Graph Coloring Problem, we have a graph composed of vertices and edges. The objective is to color each vertex in a way that no two connected vertices have the same color. The minimum number of colors required to achieve this is known as the chromatic number of the graph. Our task is to find a valid coloring solution that minimizes the chromatic number.

Backtracking Search Algorithm: The Backtracking Search Algorithm is a systematic exploration approach that examines different color assignments for each vertex. It works by recursively trying out colors and backtracking when conflicts arise. This algorithm explores the solution space while intelligently pruning paths that lead to infeasible colorings.

Genetic Algorithm: The Genetic Algorithm is an optimization technique inspired by natural selection. In this context, we represent potential colorings as chromosomes. The algorithm evolves a population of colorings over generations using genetic operations such as crossover and mutation. The fittest colorings, those with fewer conflicts, are more likely to be passed to the next generation.

Project Requirements:

User-Defined Graphs: The system should allow users to input graphs with specified vertices and edges.

Backtracking Implementation: Develop a Backtracking Search Algorithm to find valid colorings for the given graph.

Genetic Algorithm Implementation: Implement a Genetic Algorithm that evolves colorings over generations.

User Interface: Create an intuitive user interface where users can input graphs, choose algorithms, and visualize the colorings.

Solution Visualization: Provide a visual representation of the graph with colored vertices to demonstrate the effectiveness of the algorithms.

Performance Metrics: Implement metrics to measure the quality of solutions, such as the chromatic number and computational time.

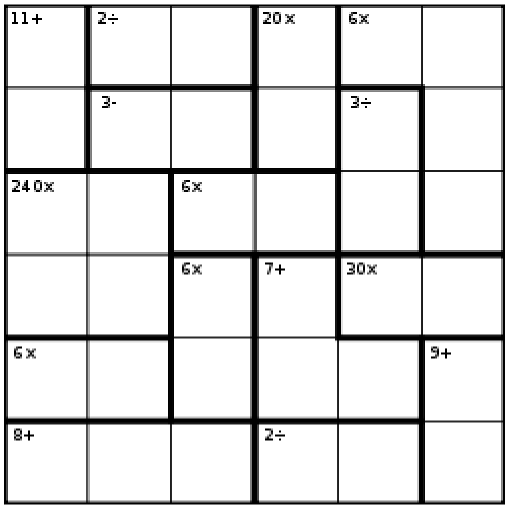
Parameter Tuning: Allow users to adjust parameters for the genetic algorithm, such as population size and mutation rates, to explore their impact on solution quality.

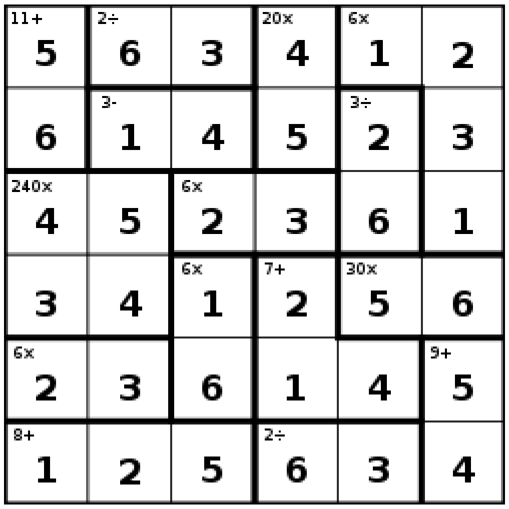
Outcome: The project aims to deliver a versatile and user-friendly Graph Coloring Problem Solver that demonstrates the strengths of both the Backtracking Search Algorithm and the Genetic Algorithm. Users will gain insights into how different algorithms approach and optimize the coloring of graphs, contributing to a deeper understanding of combinatorial optimization challenges.

11) KenKen (KenDoku) Puzzle Solver using the Backtracking Search Algorithm, AND a Genetic Algorithm.

The KenKen Puzzle is a mathematical and logical grid-based puzzle that combines aspects of Sudoku and arithmetic operations. The goal is to fill a grid with digits so that each row and column contains unique numbers, and certain outlined regions (cages) follow specified arithmetic rules.

Problem/Game Description: In a KenKen Puzzle, you are presented with an N×N grid, where N is usually a perfect square (e.g., 4, 6, 9). The grid is divided into regions or cages, each outlined with bold borders. Each cage contains a target number and an arithmetic operation (addition, subtraction, multiplication, or division).





Rules of the KenKen Puzzle:

Digit Placement: Fill the grid with digits from 1 to N in such a way that each row and each column contains unique digits.

Cage Operations: The digits within each cage must satisfy the specified arithmetic operation to achieve the target number. For example, a cage labeled "8÷" with a target of 2 means that the digits in the cage must divide to give 2. Thus, the numbers in the cells of each cage must produce a certain "target" number when combined using a specified mathematical operation (one of addition, subtraction, multiplication or division). Another example, a linear three-cell cage specifying addition and a target number of 6 in a 4×4 puzzle must be satisfied with the digits 1, 2, and 3. Digits may be repeated within a cage, as long as they are not in the same row or column. The target number and operation appear in the upper left-hand corner of the cage.

Requirements of the Project: The objective of the KenKen Puzzle Solver Project is to design an intelligent system capable of solving KenKen puzzles using both the Backtracking Search Algorithm and a Genetic Algorithm.

Project Components:

Backtracking Search Algorithm: Implement the Backtracking Algorithm to explore possible digit placements in the grid. The algorithm should backtrack when constraints are violated and efficiently navigate through the solution space.

Genetic Algorithm (GA): Develop a Genetic Algorithm to approach KenKen puzzle-solving as an evolutionary process. Represent potential solutions as individuals in a population, apply genetic operators, and evolve the population over generations to find optimal or near-optimal solutions.

Integration: Integrate the Backtracking Algorithm and the Genetic Algorithm into a cohesive system. Allow users to choose between the two methods for puzzle-solving.

User Interaction: Create a user-friendly interface that allows users to input KenKen puzzles, visualize solutions, and observe the solving process.

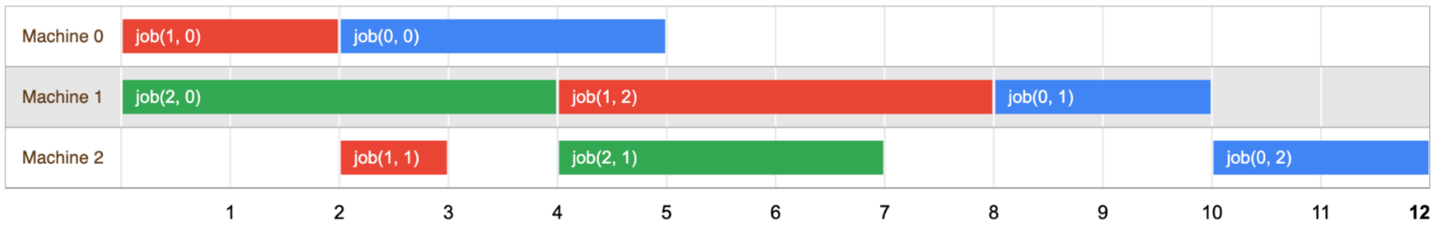
Performance Metrics: Implement metrics to evaluate the efficiency and accuracy of the solver, considering factors such as solution time and correctness.

Challenges: Consider the challenges associated with implementing Genetic Algorithms for KenKen puzzles, including suitable representations, genetic operators, and the handling of constraints.

Outcome: Upon completion, the KenKen Puzzle Solver Project will serve as a valuable tool for enthusiasts and puzzle-solvers, showcasing the application of both Backtracking and Genetic Algorithms in tackling mathematical and logical challenges. Users can gain insights into the strengths and weaknesses of each algorithmic approach and witness their performance in real puzzle-solving scenarios.

12) Job Scheduling Problem Solver using the Backtracking Search Algorithm, AND a Genetic Algorithm.

Description: The Job Scheduling Problem (JSP) is a classic optimization challenge encountered in various industries, where the goal is to efficiently assign a set of jobs to available resources while respecting constraints and optimizing a specific objective. The project aims to develop intelligent algorithms, specifically employing the Backtracking Search Algorithm and a Genetic Algorithm, to solve instances of the Job Scheduling Problem.



Problem Overview: In the Job Scheduling Problem, a set of jobs needs to be scheduled on available resources, considering constraints such as time, resource capacity, and dependencies between jobs. Each job has specific processing times and may require certain resources, and the objective is to find an optimal schedule that minimizes the completion time or maximizes resource utilization.

Backtracking Search Algorithm: The Backtracking Search Algorithm is a systematic approach that explores different job assignments recursively. It starts with an initial assignment, explores possibilities, and backtracks when constraints are violated. The algorithm incrementally builds a schedule, making decisions at each step, and aims to find a valid and optimal solution.

Genetic Algorithm: The Genetic Algorithm is an evolutionary approach inspired by natural selection. It maintains a population of potential solutions (schedules), applies genetic operators (crossover, mutation), and evaluates the fitness of each schedule based on the optimization objective. Over successive generations, the algorithm evolves schedules that exhibit desirable traits, leading to improved solutions.

Project Requirements:

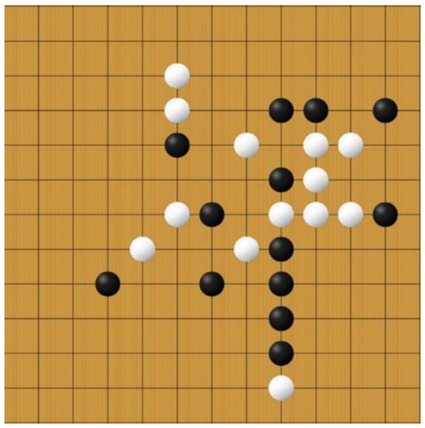
> **Problem Representation**: Define a suitable representation for the Job Scheduling Problem, considering job details, resource constraints, and objective function.
>
> **Algorithm Implementation**: Implement the Backtracking Search Algorithm to explore the decision space systematically. Additionally, implement a Genetic Algorithm to evolve schedules over multiple generations.
>
> **Constraints Handling**: Develop mechanisms to handle constraints, including resource capacity, temporal constraints, and any specific rules governing job assignments.
>
> **Optimization Objective**: Clearly define the optimization objective, whether it's minimizing the makespan, maximizing resource utilization, or another relevant metric.
>
> **Performance Evaluation**: Evaluate the performance of both algorithms on various instances of the Job Scheduling Problem, comparing their effectiveness in finding optimal schedules and their computational efficiency.
>
> **User Interface**: If feasible, consider implementing a user-friendly interface allowing users to input problem instances, visualize schedules, and understand algorithmic decisions.

Expected Outcomes: The project aims to deliver effective solutions to the Job Scheduling Problem using both Backtracking and Genetic Algorithms. The team is expected to provide insights into the strengths and limitations of each algorithm, helping users make informed decisions when selecting approaches for specific instances of the problem. The project will contribute to understanding and solving real-world scheduling challenges efficiently.

13) An Intelligent Pente Player using the Minimax Algorithm, Alpha-Beta Pruning, and Heuristic Functions

Introduction: The project aims to develop an intelligent Pente player, a classic two-player strategy board game. Pente, derived from the Japanese word for "five," involves placing stones on a gridded board with the goal of getting five stones in a row. The game is known for its strategic depth, requiring players to plan ahead and block opponents' moves. The intelligent player will be implemented using the Minimax algorithm, enhanced with Alpha-Beta Pruning for efficiency, and heuristic functions for quicker decision-making.

Game Description: Pente is played on a gridded Go board, typically 19x19 intersections. Players take turns placing stones of their color on the intersections (White always assume the opening move), aiming to form an unbroken row of five stones horizontally, vertically, or diagonally. Players can also capture their opponent's stones by surrounding them with their own stones. The first player to achieve five in a row or capture ten opponent stones wins.



Project Requirements:

Game Representation: Define a data structure to represent the Pente board and track the positions of stones for both players.

Move Generation: Implement algorithms to generate legal moves for both players, considering the rules of stone placement and capture.

Minimax Algorithm: Integrate the Minimax algorithm, a decision-making approach that explores possible moves and evaluates outcomes to make optimal decisions.

Alpha-Beta Pruning: Enhance the Minimax algorithm with Alpha-Beta Pruning to optimize the search and reduce the number of explored nodes.

Heuristic Evaluation Functions: Design heuristic functions to evaluate the desirability of a game state, capturing strategic elements and patterns that influence decision-making.

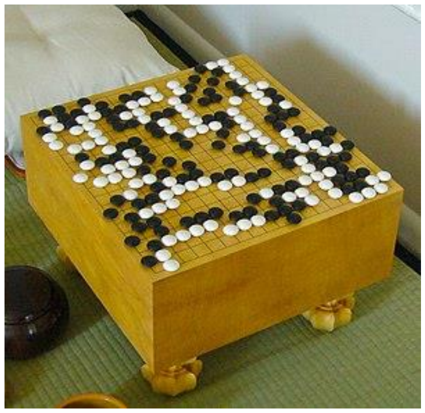
User Interface: Create a user interface to visualize the Pente board, allowing users to play against the intelligent Pente player and observe its moves.

Expected Outcomes: The project aims to deliver an intelligent Pente player capable of strategic decision-making, blocking opponents effectively, and showcasing the strengths of the Minimax algorithm with Alpha-Beta Pruning and heuristic functions. The team will evaluate the performance of the intelligent player through test games against human players and possibly other AI opponents. Success will be measured by the player's ability to make optimal moves, anticipate opponent strategies, and adapt to different game scenarios. The user interface, if developed, will provide an interactive platform for users to engage with the intelligent Pente player.

14) An Intelligent Chess-Player using an Alpha-Beta Depth-First algorithm *(designing & implementing at least 2 heuristic functions)*

Build an artificial Chess player, that can play games against a human opponent. Chess is a two-player strategy board game played on a chessboard, a checkered game-board with 64 squares arranged in an 8×8 grid. The game is played by millions of people worldwide. Chess is believed to be derived from the Indian game chaturanga sometime before the 7th century. Chess reached Europe by the 9th century, due to the Umayyad conquest of Hispania. The pieces assumed their current powers in Spain in the late 15th century with the introduction of "Mad Queen Chess"; the modern rules were standardized in the 19th century. Play does not involve hidden information. Each player begins with 16 pieces: one king, one queen, two rooks, two knights, two bishops, and eight pawns. Each of the six piece types moves differently, with the most powerful being the queen and the least powerful the pawn. The objective is to checkmate the opponent's king by placing it under an inescapable threat of capture. To this end, a player's pieces are used to attack and capture the opponent's pieces, while supporting each other. During the game, play typically involves making exchanges of one piece for an opponent's similar piece, but also finding and engineering opportunities to trade advantageously, or to get a better position. In addition to checkmate, a player wins the game if the opponent resigns, or (in a timed game) runs out of time. There are also several ways that a game can end in a draw.

15) An Intelligent Go-Player using an Alpha-Beta Depth-First algorithm *(designing & implementing at least 2 heuristic functions)*

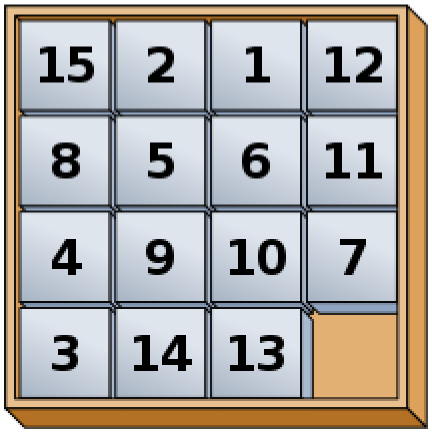


Build an artificial Go player, that can play games against a human opponent. Go is an abstract strategy board game for two players, in which the aim is to surround more territory than the opponent. The game was invented in China more than 2,500 years ago and is believed to be the oldest board game continuously played to the present day. The playing pieces are called stones. One player uses the white stones and the other, black. The players take turns placing the stones on the vacant intersections ("points") of a board. Once placed on the board, stones may not be moved, but stones are removed from the board if "captured". Capture happens when a stone or group of stones is surrounded by opposing stones on all orthogonally-adjacent points. The game proceeds until neither player wishes to make another move. When a game concludes, the winner is determined by counting each player's surrounded territory along with captured stones and komi *(points added to the score of the player with the white stones as compensation for playing second)*. The standard Go board has a 19×19 grid of lines, containing 361 points. Beginners often play on smaller 9×9 and 13×13 boards.

16) An Intelligent N-Puzzle Solver (for sizes: 8, 15, and 24) using a Best-First Search algorithm (designing & implementing at least 4 heuristic functions).

The 15-puzzle (also called Game of Fifteen, Mystic Square and many others) is a sliding puzzle that consists of a frame of numbered square tiles in random order with one tile missing. The puzzle also exists in other sizes, particularly the smaller 8-puzzle. If the size is 3×3 tiles, the puzzle is called the 8-puzzle or 9-puzzle, and if 4×4 tiles, the puzzle is called the 15-puzzle or 16-puzzle named, respectively, for the number of tiles and the number of spaces. The object of the puzzle is to place the tiles in order by making sliding moves that use the empty space.

The n-puzzle is a classical problem for modelling algorithms involving heuristics. Commonly used heuristics for this problem include counting the number of misplaced tiles and finding the sum of the taxicab-distances between each block and its position in the goal configuration.



17) Solving a Faculty's Timetable Scheduling Problem using Genetic Algorithms.

A very famous scenario where Genetic Algorithms can be used is the process of making timetables or timetable scheduling. Consider you are trying to come up with a weekly timetable for classes in a college for a batch/class. We must arrange classes and come up with a timetable so that there are no clashes between classes. Here, the task is to search for the optimum timetable schedule. A possible definition for the problem is: Given a set of lecturers, a set of courses on individual topics and a Course Requirements matrix with integer elements representing the number of hours a lecturer teaches a course during each week, the problem is to allocate times to these hours so that a student may take as many suitable combinations of courses as possible. Or, simply to create a practical timetable for a whole faculty in which courses offered by different departments may be combined in various ways to suit individual students.

18) Solving the VRP "Vehicle Routing Problem" using both Genetic Algorithms & Differential Evolution.

The vehicle routing problem (VRP) is a combinatorial optimisation and integer programming problem that asks: "What is the optimal set of routes for a fleet of vehicles to traverse to deliver to a given set of customers?". It generalises the well-known travelling salesman problem (TSP). The VRP concerns the service of a delivery company and how things are delivered from one or more depots that have a given set of home vehicles and are operated by a set of drivers who can move on a given road network to a set of customers. It asks for a determination of a set of routes, S (one route for each vehicle that must start and finish at its depot), such that all customers' requirements and operational constraints are satisfied and the global transportation cost is minimised. This cost may be monetary, distance or otherwise.

19) Solving the Knapsack Problem using Genetic Algorithms *(Solve both the 0-1 Knapsack Problem and the Unbounded Knapsack Problem)*.

The knapsack problem or rucksack problem is a problem in combinatorial optimization: Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. It derives its name from the problem faced by someone who is constrained by a fixed-size knapsack and must fill it with the most valuable items. The most common problem being solved is the 0-1 knapsack problem, which restricts the number of copies of each kind of item to zero or one. The unbounded knapsack problem (UKP) places no upper bound on the number of copies of each kind of item.

20) Genetic Algorithms for Function Optimisation.

Implement a Genetic Algorithm to find the global minimum of 5 benchmark optimisation functions.

- You may select any five benchmark optimisation functions from the following list: https://www.sfu.ca/~ssurjano/optimization.html