**HO CHI MINH UNIVERSITY OF TECHNOLOGY AND EDUCATION**

**FACULTY FOR HIGH QUALITY TRAINING**

**-------------------------------------------------------**

Logo

Description automatically generated

**FINAL TERM PROJECT**

**Course name: Artificial Intelligence**

**PROJECT IDEAS**

**GROUP 4:**

|  |  |  |
| --- | --- | --- |
| Nguyễn Hoàng Anh Khoa | 19110514 | 25% |
| Chu Nguyễn Hoàng Sơn | 19119128 | 25% |
| Nguyễn Quốc Hoàng | 19110128 | 25% |
| Biện Quang Huy | 19110005 | 25% |

**LECTURER: Hoàng Văn Dũng**

Ho Chi Minh city, November 1, 2023

COMMENTS FROM THE INSTRUCTOR

*Ho Chi Minh City, 11/2023*

INSTRUCTOR

**Table of Contents**

[Gratitude 1](#_Toc152031886)

[Work Assignment 2](#_Toc152031887)

[Chapter 1: Introduction 3](#_Toc152031888)

[**1.1.** **Background** 3](#_Toc152031889)

[**1.2.** **Theory** 3](#_Toc152031890)

[**1.3.** **Aims** 4](#_Toc152031891)

[Chapter 2: Overview 5](#_Toc152031892)

[**2.1.** **Preparations** 5](#_Toc152031893)

[**2.2.** **Algorithms** 6](#_Toc152031894)

[**2.2.1.** **Breadth First Search (BFS)** 6](#_Toc152031895)

[**2.2.2.** **Depth-first Search Algorithm (DFS)** 7](#_Toc152031896)

[**2.2.3.** **Ant Colony** 8](#_Toc152031897)

[**2.2.4.** **Uniform-cost search (UCS)** 8](#_Toc152031898)

[**2.2.5.** **A-star** 9](#_Toc152031899)

[Chapter 3: Design 10](#_Toc152031900)

[**3.1. Overall** 10](#_Toc152031901)

[3.2. Class design 11](#_Toc152031902)

[Chapter 4: Test results and discussions 14](#_Toc152031903)

[**4.1.** **Algorithm’s parameter** 14](#_Toc152031904)

[**4.2.** **Test results comparison** 15](#_Toc152031905)

[**4.3.** **Game interfaces and tutorials** 18](#_Toc152031906)

[**4.3.1.** **Main menu** 18](#_Toc152031907)

[**4.3.2.** **Life chances selection menu** 19](#_Toc152031908)

[**4.3.3.** **Game difficulties selection** 20](#_Toc152031909)

[**4.3.4.** **Gameplay** 21](#_Toc152031910)

[Chapter 5: Conclusion 22](#_Toc152031911)

[**5.1.** **Achievement obtained** 22](#_Toc152031912)

[**5.2.** **Features extensions** 22](#_Toc152031913)

[REFFERNCES 24](#_Toc152031914)

# Gratitude

Firstly, I would like to express my sincere gratitude to the University of Technical Education for incorporating the Artificial Intelligence course into the curriculum. Special thanks go to the course instructor, Mr. Hoang Van Dung, for guiding and imparting valuable knowledge throughout my study. During the time in Mr. Dung's class, I have gained profound insights and a comprehensive understanding of this field of study. Undoubtedly, these insights will serve as precious knowledge and a foundation for my future endeavors.

Due to the limitations in knowledge and understanding, there may be mistakes and constraints in the report. I sincerely hope for your guidance and contributions, Mr. Dung, to refine and enhance our report. We eagerly anticipate your valuable insights and comments.

I extend my heartfelt thanks!

Group 4

# Work Assignment

A black grid with white lines

Description automatically generated

**Chapter 1: Introduction**

1. **Background**

Sudoku is a popular number puzzle game that involves placing the numbers 1 through 9 in a 9x9 grid made up of 3x3 subgrids, with each number only appearing once in each row, column, and subgrid. The game starts with some numbers already filled in, and the goal is to fill in the remaining squares without breaking the placement rules. Solving Sudoku puzzles relies on logic and reasoning rather than mathematical computations. Players analyze the initial number placements to deduce where other numbers must go based on the constraints. There are a few common techniques used when solving Sudoku such as scanning rows, columns and subgrids to identify missing numbers, noting eliminated number candidates, and identifying naked and hidden subsets. Given the defined ruleset, each Sudoku puzzle has just one solution. For difficult puzzles, Sudoku solvers can be helpful tools. These solvers use computer algorithms like backtracking, pattern matching or constraint programming to logically determine the solution to any valid Sudoku grid. Solvers allow players to check their work or provide hints when they get stuck.

1. **Theory**

Solving Sudoku relies heavily on logical reasoning rather than mathematical computations. Players use theory and strategies to deduce the placement of numbers based on the constraints of the 9x9 grid. Techniques like noting eliminated candidates and identifying naked and hidden subsets utilize the theory of process of elimination. Pattern recognition theorizes that by identifying recurring patterns in the initial placements, you can determine where similar patterns will occur. Another theory applied is symmetry analysis, looking at how numbers are symmetrically placed within each region to deduce missing numbers. Analyzing grid subsets forms theories about interconnections between regions. Puzzle solvers theorize that by gathering information from regions revealing single candidates, they can logically eliminate possibilities in connected regions, progressively determining the solution.

1. **Aims**

Through exploration of this project, we want expand our knowledge of key algorithms including depth-first search, breadth-first search, backtracking, hill climbing, and more. Analyzing the mechanisms and applications of these algorithms enable us to better comprehend their strengths, weaknesses, and optimal use cases. Our learnings equip us with enhanced technical acumen that can be leveraged across myriad domains. This undertaking promises to bolster our algorithmic aptitude and readiness to wield these tools in addressing complex computational challenges.

**Chapter 2: Overview**

1. **Preparations**
2. Programming tool



Visual Studio Code, also commonly referred to as VS Code, is a source-code editor developed by Microsoft for Windows, Linux and macOS. Features include support for debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git. Users can change the theme, keyboard shortcuts, preferences, and install extensions that add functionality.

1. Programming language & libraries

Python is a high-level, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation. Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including structured, object-oriented and functional programming.

* **Pygame** is a free and open-source cross-platform library for the development of multimedia applications like video games using Python. It uses the Simple DirectMedia Layer library and several other popular libraries to abstract the most common functions, making writing these programs a more intuitive task
* **Math** a set of built-in math functions, including an extensive math module, that allows you to perform mathematical tasks on numbers.
* **Tkinter** is a standard library in Python which is used for GUI application. Tkinter has various controls which are used to build a GUI-based application.
* **Time:** This repository contains the code used to combine macroscopic tractography information with microscopic multi-fixel model estimates in order to improve the accuracy in the estimation of the microstructural properties of neural fibers in a specified tract
* **Random:** a built-in module that you can use to make random numbers.

1. **Algorithms**
2. **Breadth First Search (BFS)**

A computer code with text

Description automatically generated with medium confidenceBFS is an algorithm that explores all the vertices of a graph level by level. In the context of Sudoku, each state represents a partial assignment of numbers to the grid. BFS systematically explores all possible assignments, ensuring that the solution found is the shortest path to a valid solution.

Implementation:

The Sudoku grid can be represented as a state space, where each node represents a possible assignment. BFS involves maintaining a queue of states, starting with the initial state (unsolved puzzle), and iteratively exploring neighbors until a solution is found.

1. **Depth-first Search Algorithm (DFS)**

DFS explores as far as possible along each branch before backtracking. In the context of Sudoku, DFS can be used to systematically search through possible assignments, potentially finding a solution faster than BFS in certain scenarios.

Implementation

A screenshot of a computer code

Description automatically generated

DFS can be implemented using recursion or a stack. The algorithm starts with the initial state and recursively explores each branch until a solution is found or the entire state space is explored.

1. **Ant Colony**

ACO is a metaheuristic inspired by the foraging behavior of ants. It uses artificial ants to explore the solution space and deposit pheromones on the paths they traverse. The pheromones guide subsequent ants towards promising solutions.

Implementation

A screenshot of a cell test

Description automatically generated

ACO involves representing the Sudoku grid as a graph and simulating the movement of ants on this graph. Pheromones are updated based on the quality of solutions found. Over time, the algorithm converges towards an optimal solution.

1. **Uniform-cost search (UCS)**

UCS is a variant of Dijkstra's algorithm that takes into account the cost of reaching each state. In Sudoku, the cost is associated with the number of steps taken to reach a particular assignment.

Implementation

UCS maintains a priority queue based on the cost of reaching each state. It explores the states with the lowest cost first, ensuring an optimal solution is found.

1. **A-star**

The A\* algorithm is adapted to solve Sudoku by treating the puzzle as a search problem. Nodes in the search space represent different puzzle configurations, and the goal is to find a configuration that satisfies the rules of Sudoku. The algorithm uses a heuristic to estimate the remaining cost to the goal and guides the search accordingly.

Implementation:

The Python implementation involves defining a Sudoku class that encapsulates the puzzle and the A\* algorithm. The A\* algorithm includes the following steps:

Initialization: Create an initial node representing the given puzzle configuration.

Priority Queue: Maintain a priority queue to select nodes with the lowest combined cost and heuristic.

Expand Nodes: Generate successor nodes by applying valid moves to the current configuration.

Heuristic Function: Develop a heuristic function to estimate the remaining cost to the goal.

Goal Check: Determine when a solution is found by checking if the puzzle is complete and follows the rules of Sudoku.

**Chapter 3: Design**

**3.1. Overall**

The code is divided into packages with clear names. These packages are connected to represent the whole program.

A diagram of a software system

Description automatically generated

They are described as followed:

1) Sudoku package deals with:

a. Cell – having the cell limit of only one number from 1 to 9.

b. Grid – having 81 cells organized as 9x9 grids.

c. Game – the main puzzle game. Grid which filled with random numbers in random Cells.

d. Create Solver – empty Grid. Users can fill some of the Cells, and the program shows solution(s).

e. Tests – to make sure it works fine.

2) Features package explained in the next sub-section.

3) Solver and Generator package:

a. Solver – 5 algorithms and different logics to solve puzzles.

b. Generator – randomly generate a real Sudoku puzzle. This check for reality and validity for a puzzle by sending it to the Solver. Four levels provided (easy – medium – hard - expert).

c. Tests – to make sure it works fine.

## 3.2. Class design

|  |  |  |
| --- | --- | --- |
| No | Class name | Purpose |
| 1 | Ant() | represent an ant navigating a grid, making decisions on values for each cell based on pheromone information and local constraints |
| 2 | AntSolver() | Simulating the foraging behavior of ants and updating pheromones on the grid |
| 3 | BFSProblem() | Generate possible actions, apply actions to the current state, and check the legality of a state |
| 4 | BFSNode() | Encapsulate the concept of a node in the BFS search tree. Each node represents a specific state in the search space |
| 5 | Cell() | Represent individual cells in the Sudoku grid |
| 6 | DFSProblem() | Define a problem for solving Sudoku using the DFS algorithm. It provides methods for legality checking, filtering possible values for specific contexts, and generating possible actions for the DFS search |
| 7 | DFSNode() | Encapsulate the concept of a node in the DFS search tree. Each node represents a specific state in the search space |
| 8 | Grid() | Provide a comprehensive set of methods for interacting with a Sudoku grid, including setting and retrieving cell values, propagating constraints, deducing values based on constraints, checking grid validity, and converting the grid to a 2D array |
| 9 | SudokuHeuristic() | provides a heuristic measure for Sudoku puzzles based on the count of filled cells |
| 10 | AProblem() | represent a Sudoku problem, check the legality of a given state, filter possible values for empty spots, generate possible actions, and calculate a heuristic value for a state |
| 11 | Anode() | this class is likely used in conjunction with a search algorithm to represent nodes in a search space |
| 12 | UCSProblem() | representation for solving Sudoku puzzles using the UCS algorithm. It defines methods for checking the legality of a state, filtering possible values for empty spots, finding the first empty spot, and generating possible actions (successor states) with a cost of 1 |
| 13 | UCSNode() | conjunction with the UCS algorithm for searching through a state space |
| 14 | Board() | GUI representation of a Sudoku board, allowing users to interact with the puzzle, insert values, and visualize the state of the game |
| 15 | Game() | provides a foundation for a Sudoku game with a graphical user interface (GUI) and additional features such as algorithmic puzzle solving |
| 16 | Square() | designed to represent and visually render an individual square in a Sudoku puzzle on a Pygame GUI |

# Chapter 4: Test results and discussions

1. **Algorithm’s parameter**

Unlike other algorithms, only Ant Colony (ACO) needs set of parameters for producing different results. These parameters showed in the image have been optimizely configured.

A screen shot of a computer

Description automatically generated

**GRID\_SIZE:** Specifies the size of the grid in the ant colony optimization algorithm. The algorithm will operate on a grid with a size of 9.

**NUM\_OF\_ANTS:** Defines the number of ants in the colony. In this case, there is only one ant.

**LOCAL\_PHER\_UPDATE:** Represents the rate at which pheromone is locally updated on the grid. After an ant moves, it deposits pheromone, and this value determines the strength of that local update.

**GLOBAL\_PHER\_UPDATE:** Specifies the rate at which global pheromone update occurs. This happens after all ants have completed their moves, and this value defines the strength of the update.

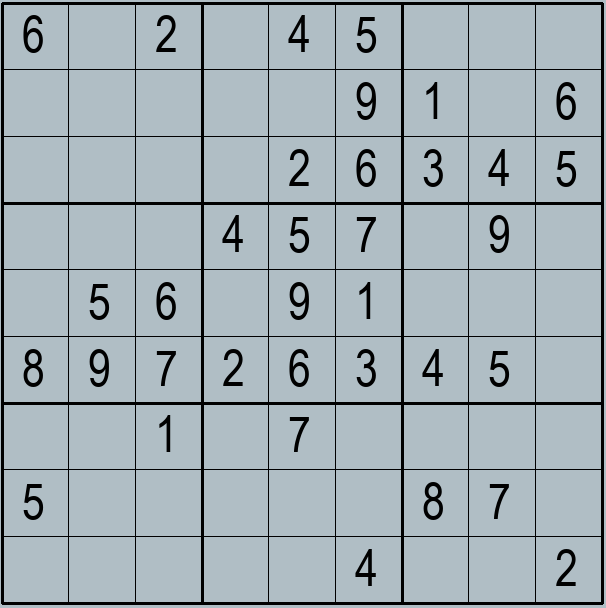
**BEST\_PHER\_EVAPORATION:** Defines the rate at which pheromone evaporates on the best path found. Pheromone evaporation helps prevent convergence to a suboptimal solution.

**GREEDINESS:** Represents the level of greediness in ant decision-making. A higher value makes ants more likely to choose the path with the most pheromone, promoting exploitation, while a lower value encourages exploration of alternative paths.

1. **Test results comparison**

We only record least node expanded results for each algorithm. Depend on the user’s computer/laptop specification and random generations in sudoku, the results may variety.

* Easy

 ACO: A grey background with black text

Description automatically generated

BFS: A grey background with black text

Description automatically generated DFS: A grey background with black text

Description automatically generated

UCS: A grey background with black text

Description automatically generated A\*: A grey background with black text

Description automatically generated

* Medium

A grey square with black numbers and a square grid

Description automatically generated ACO: A grey background with black text

Description automatically generated

BFS: A close-up of a grey background

Description automatically generated DFS: A grey background with black text

Description automatically generated

UCS: A grey background with black text

Description automatically generated A\*:A grey background with black text

Description automatically generated

* Hard

A square grid with numbers and symbols

Description automatically generated ACO: A close-up of a text

Description automatically generated

BFS: A close-up of a number

Description automatically generated DFS: A grey background with black text

Description automatically generated

UCS: A grey background with black text

Description automatically generated A\*:A close-up of a text

Description automatically generated

From above images, we can see that:

Ant Colony (ACO) gives the fastest solution and explored nodes nearly the same.

Breadth First Search (BFS) gives the slowest one with large number of nodes.

The rest algorithms’ nodes and time usage for solving increase decent amount depend on difficulties and game generations.

1. **Game interfaces and tutorials**
   1. **Main menu**

**A grey square with white text

Description automatically generated**

* 1. **Life chances selection menu**

**A screenshot of a computer

Description automatically generated**

* 1. **Game difficulties selection**

**A screenshot of a computer

Description automatically generated**

* 1. **Gameplay**

A screenshot of a game

Description automatically generated

# Chapter 5: Conclusion

1. **Achievement obtained**

We have created a Sudoku game application. In order to create the whole application, we needed to learn about the algorithms for generating and solving Sudoku. There are many ways of producing the solution of a puzzle. The idea was to integrate the solver logics with one of the algorithms in order to improve the efficiency time. The application uses two algorithms and human logics to solve all real Sudoku. Solver logics are implemented before the backtracking algorithm to reduce searching for a solution. Furthermore, the application is capable to generate 3 levels of difficulties. So users can play the level that is more enjoyable and satisfiable for them. In addition, the application enables users to create or copy any valid puzzle (i.e. newspapers) and then have the solution. The GUI created in the application was designed successfully with more options and features, so users should find it more attractive than other applications. There are no ambiguities in using the application. Moreover, we have added the instructions to explain how the game is played and how to use the whole application. But since users like to start playing rather than reading the instruction, the application has a clear sequence of use. Even the shortcuts can be easily figured out. On the other hand, this application has been transformed to another platform. we have used eclipse program in order to code the application into android software. The coding language was written in python, VSCode is an open platform and easy to find errors. This is the first phone application we have done. It was challenging, but due to less time we had in the end of the project, it was really simple. In addition, coding the solver logics increases my ability to become a better programmer thinker. Overall, the final application satisfied all the objectives outlined in the introduction.

1. **Features extensions**

The code implemented in the application has been well organised and also includes comments everywhere. Therefore, we believe that the application can be easily developed more, especially for the android version. However, there are many approaches that could be done in order to improve and extend the functionality of the application. For example, the generator can generate different size of Sudoku grids. Also, solver functionality can be improved to work faster with different grid sizes. Finally, the generating of levels could be more accurate than using loops.

# REFFERNCES

[1] Lee, W.-M. (2006). Programming Sudoku. New York: Apress.

[2] [Lewis, R. (2007). Metaheuristics can Solve Sudoku Puzzles. Retrieved Jan 16, 2011 from Publications and Presentations](http://www.rhydlewis.eu/)

[3] [Lweis, R. (2007, Apr). On the Combination of Constraint Programming and Stochastic Search: The Sudoku Case. Retrieved Jan 13, 2011 from Publications and Presentations:](http://www.rhydlewis.eu/talks/DortmundTalkRhydLewis.pdf)

[4] [Pramberg, J. (2010, Aug 3). Sudoku Solver and Generator. Retrieved Jan 13, 2011 from Code Project](http://www.codeproject.com/KB/game/sudoku.aspx)

[5] [Silas, G. (2009). The algorithm behind a sudoku solver. Retrieved Jan 15, 2011 from Yahoo! Answers:](http://answers.yahoo.com/question/index?qid=20090810061953AAR1mud)

[6] [Smallbone, N. (2005). Sudoku Solver - source code. Retrieved Jan 13, 2011 from Sudoku Solver](http://www.8325.org/sudoku/source.html)

[7] [Tellenbach, P. (2007, Mar 18). Backtracking to solve a sudoku puzzle. Retrieved Jan 13, 2011 from Heimetli](http://www.heimetli.ch/ffh/simplifiedsudoku.html)

[8] [What is Sudoku? (2007, Jan 9). Retrieved Jan 13, 2010 from Notelay](http://www.notelay.com/articles/did_you_know/what_is_sudoku/)