American University of Armenia

College of Science and Engineering

Artificial Intelligence Project

# Minesweeper Game

Team: Meri Asatryan, Vram Papyan, Lili Kostanyan,
Jeni Balaban

Fall, 2023

**Abstract**

This paper explores strategies to play the Minesweeper game effectively by leveraging various Artificial Intelligence algorithms. The project involves the identification and implementation of suitable algorithms to enhance Minesweeper gameplay. The primary algorithm discussed in the paper is designed to intelligently uncover mine locations. Additionally, the project considers potential future improvements based on the outcomes of this research and insights from other related studies. This work was conducted during an Artificial Intelligence course as a group project in the academic year 2023/2024 at the American University of Armenia.

**Keywords:** AI, Minesweeper, Game Strategies, algorithms, problem solving, heuristic function, decision-making, efficiency, complexity.

# 1 Introduction

## 1.1 The Structure of the Project Paper

This paper is structured first to introduce the problem and its relevance to our AI course material. Following that is a comprehensive literature review to examine previous works and publications in solving the Minesweeper game, concentrating on their methodologies. The method section will detail our approach to solving the game, which will be followed by the evaluation of our results. Next we conclude by reflecting on our achievements concerning the project objectives and suggesting routes for future research. Lastly, there is the Bibliography, used to acknowledge the sources we consulted.

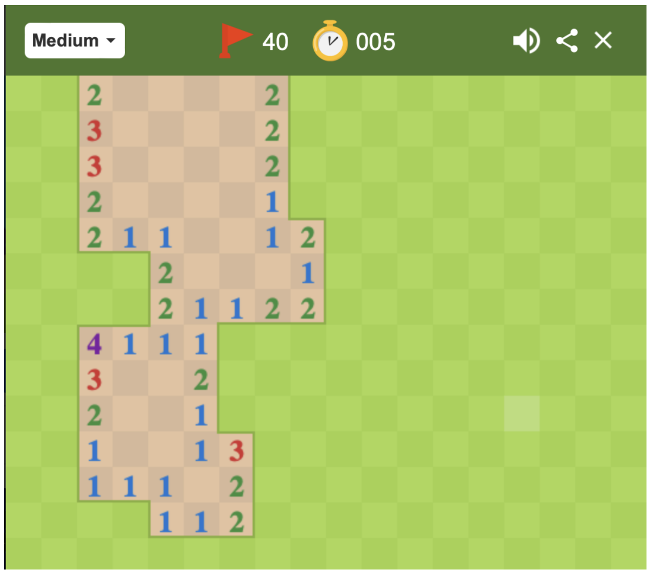


Figure 1: Board during the game: numbers on squares indicating the numbers of mines in neighborhood

## 1.2 Problem Setting and Description

The field of Artificial Intelligence (AI) contains a wide range of computational problems, and game solving is quite a fascinating aspect of it. Minesweeper is a classic computer game that provides a perfect platform for exploring AI because of its intrinsic complexity and decision making skills required for the gameplay. Minesweeper is a single agent puzzle solving game. The original game called Mine was created between 1989 and 1990 by Robert Donner, and he used Curt Johnson's code. The game was inspired by another game called Mined-Out, developed by Ian Andrew and published in 1983 (Minesweeper Game, n.d.). So, at the beginning of the game, the player sees the board filled with squares. The player can interact with the board by clicking on the squares, which is called probing. Each square hides a number inside that the player can only see after doing the probing action. And that number describes the number of bombs in the adjacent squares ranging from 0 to 8 mines. Besides, some squares also hide a mine before probing. In Figure 1, we present a snapshot of the game board during play. The game is lost when the player probes a square with a mine. The goal of the game is to open all the squares in such a way that none of them contain a bomb. Minesweeper, commonly included in Windows operating systems, has been proven to be NP-complete, meaning that solving this game is as challenging as some of the most complex problems known in computational theory. Our Artificial intelligence course covers topics such as intelligent agents, different search algorithms, game theory, and logic, laying the base for understanding such complex problems. The principles learned in these areas can be applicable to develop an Artificial intelligence agent capable of efficiently solving Minesweeper. Our project aligns with the course material, especially in the parts related to search algorithms and game strategies, giving us practical application of our theoretical concepts. Our interest in this project derives from the challenge it raises and its applicability to broader Artificial Intelligence research. Minesweeper game encapsulates critical factors of AI, for example, pattern recognition, decision making under uncertainty, and optimization - all of which are essential in developing intelligent systems. By choosing Minesweeper, we engage with a familiar game from childhood and go into profound computational challenges that reflect real world problem solving scenarios. The primary goal of our project is to create an AI agent that will optimally solve the Minesweeper game. By calling it "optimal," we mean the agent's ability to make the most efficient decisions based on the available information, reducing the guesswork and increasing successful outcomes. During this project, we will analyze existing strategies and algorithms for Minesweeper, use many different approaches, and combine all these solutions into our AI agent.

## 2 Literature Review: Algorithmic Approaches

Jan Cicvárek's work titled "Algorithms for Minesweeper Game Grid Generation" delivers innovative algorithms to solve the game and generate solvable girds of the game. The idea searches through the complexities of this NP-complete (non-deterministic) problem, showing insightful solutions (Cicvárek, 2017). His research concentrates on algorithms that can solve the game in either polynomial or exponential time with different success rates. The goal is to make a solver with high success rates and an algorithm that consistently renders solvable minefields. Jan's thesis presents a new difficulty rating system that classifies the complexity of Minesweeper puzzles and acts as an input parameter for the minefield generator (Cicvárek, 2017). The paper provides us with an overview of the fundamental concepts of the Minesweeper game. It includes the layout of the board, game rules, and the meaning of the numbers displayed on the board. Also, it helps to determine between solvable and unsolvable boards, which can help to make algorithmic advancements (Cicvárek, 2017). Jan used existing methods such as the CSPStrategy, Equation Strategy, Single Point Algorithm, and Simon Tatham's Portable Puzzle Collection (Cicvárek, 2017). His thesis furthermore acknowledges Richard Kaye's seminal proof of Minesweeper's NP-completeness (Cicvárek, 2017). The explanation of the technical background of constraint satisfaction problems is significant because CSPs are fundamental to Cicvárek's proposed solutions. He goes deeper into strategies such as the Minimum Remaining Values Heuristic, Degree Heuristic, Most Constraining Value Strategy, Forward Checking, and Arc Consistency (Cicvárek, 2017). The focus is theoretical, and all these are integrated into the solver and board generator algorithms.

### 2.1 Single Point Strategy

The algorithm to solve Minesweeper uses a combination of advanced techniques. For example, human solver methodologies, build constraints (the term "constraints" refers to the rules that must be followed throughout the game), form clusters (the term "clusters" refers to the group of adjacent squares in the game that are analyzed together to deduce mine locations), identify open spaces and relevant squares, and apply the Single Point Strategy (SPS). Jan's solver examines mine combinations in clusters and uses mine count to solve the board. This algorithm transforms unsolvable boards into solvable ones by adding or reducing mines in clusters while maintaining solvability (Cicvárek, 2017). For SP, there are two expected outcomes to occur while taking a single cell: understanding the accessible cell location or finding neighboring mines. The first outcome is conducted when the influential label is zero (the term "influential label" refers to the number on the Minesweeper square indicating the count of adjacent mines). The agent concludes that all neighbor cells are free if this is true. The second outcome defines the mine location. When the adequate label (the term "adequate label" refers to the the number on the Minesweeper square which equals the count of unmarked adjacent cells, indicating they are all mines) number equals the unmarked adjacent cells, all unmarked cells should be marked as mine, as there is no other place for the missing mines. So the author names those assumptions all free neighbors (AFN) and all mine/mark neighbors (AMN) moves, respectfully. Jan's implementation of algorithms goes under a strict evaluation. The identification of solvable squares shows the success rates of the problem solver, its algorithmic categorization compared to existing solutions, and the required computation time. Likewise, the Minesweeper board generator is evaluated to add empirical weight to the theoretical constructs (Cicvárek, 2017).

## 2.2 Cellular Automaton

The CA approach for solving Minesweeper was based on Andrew Adamatzky's model (Becerra, 2015). In this model, each state that has its representation by a set of tuples from set Q(cell states) represents the cell's current state and transition function, defining the changes from one state to another over time.
The tuple has the form $(x^t, v(x))$, where $x \in Q$, $x^t = \#$ means $x$ contains a mine at time $t$, $x^t = \cdot$ means $x$ is a covered square at time $t$, and $x^t = \circ$ means $x$ is an uncovered cell at time $t$. The second item in the tuple is the label of cell $x$. $x^t \Rightarrow \#, \cdot, \circ$ and $v(x) = Label(x) \in \{0, 1, \ldots, 8\}$ (Becerra, 2015). The advantages of it are that the CA model gives a clear, rule-based structure for learning and succeeding in the game. Also by including AFN and AMN in its transition function, the CA technique directly addresses Minesweeper's essential deductive components.On the other hand CA technique requires a partially exposed configuration, which means it cannot start a game from a covered board, inability to handle non-deterministic scenarios (i.e., scenarios needing guesses), widespread in Minesweeper, is a severe shortcoming. Because of this constraint, CA cannot complete most Minesweeper games independently. Potential for stuckness as it can achieve a stationary configuration without successfully finishing the game, demonstrating a propensity to become "stuck" in circumstances where further advancement involves stochastic decision-making. Summing up cellular automaton approach for Minesweeper is structured and rule-based but struggles with non-deterministic scenarios and stationary states, requiring additional strategies or stochastic elements (Becerra, 2015).

## 2.3 Naive Single Point

For Naive SP, the author takes an approach from Kasper Pederson and the author of Programmer's Minesweeper (PGMS), which focuses the solution on randomness (Becerra, 2015). So, the algorithm maintains a set S of safe cells to probe. If S is empty, it chooses the next cell randomly and adds that to S. Then, the cell is either AFN or AMN based on the influential label of the cell. If the influential label (x) is zero or is equal to the number of unknown neighbor cells, then x is an instance of AFN or AMN, respectively. The solver will then probe or mark all adjacent unknown squares depending on which case was found. The advantages of this approach is that it introduces unpredictability to overcome limitations of deterministic logic, which allows the game to continue. The main drawback is the "ordering problem", which happens when the sequence of evaluating cells leads to missed opportunities for correct deductions. Another aspect that should be considered is that random selection can result in inefficiency, especially at complicated designs where random probabilistic predictions do not always match the actual distribution of mines. To address these challenges, two solutions are considered: the PGMS approach, which adapts the cell evaluation order based on neighboring cell states, and Pederson's approach:, which prioritizes cells using a priority queue, focusing on those most likely to indicate mines. In conclusion, Becerra comprehensively reviews Naive Single Point approaches in Minesweeper, stressing their simplicity and versatility while resolving the ordering problem and reliance on random selection. Although the methodologies proposed by PGMS and Kasper Pederson are fascinating, the tactics may still need to develop to provide the strategic depth required for more complex game scenarios.

## 2.4 The Double Set Single Point(DSSP)

The ordering problem in board games can be solved by the double set single point (DSSP) method. It keeps two sets, S and Q, of safe squares that the solver explores. The supplementary set Q marks doubtful points as such and puts them into Q. When the solver has enough information to utilize, it returns to these points. Q is the current board state's frontier, including all cells constituting the boundary between covered and uncovered squares. The element q in Q is an unsolved square, and the solver must consider q in the future. The algorithm cycles between probing and marking steps. The first phase is a probing step in which the solver iterates across each square x in S. If S is empty, the algorithm chooses a covered square at random to put into S. If x is an AFN occurrence, the solver adds x's covered neighbors into S. The squares that are not AFN instances are removed from S and added into Q (Becerra, 2015). The second stage, known as the marking phase, looks for AMN cases in Q. If cell q in Q is AMN, the neighbors of Q are marked and eliminated. The final stage is another probing phase, iterating through Q rather than S in search of AFN instances. Q will have problematic squares on the frontier at the end of the three stages that cannot be resolved with AFN and AMN alone (Becerra, 2015). The algorithm is based on a player's gaming strategy, the first click reveals all nearby zeros until a mining border is reached. This propagation is reflected in the initial probing phase. Repeating the first step, the player then searches for deductions, marks mines, and uses flags to investigate other squares (Becerra, 2015). Now let's discuss the advantages of this method. Advanced SP techniques involve more sophisticated decision making mechanisms, allowing for better management of complicated game conditions. It can better deal with circumstances when the board does not give enough or straightforward information for prediction, by decreasing the need for random guessing. The disadvantage of this approach is the increased complexity and higher computational demands as sophistication grows, so does complexity, making these tactics challenging to apply, comprehend and require more significant computational resources (Becerra, 2015).

## 2.5 Basic Limited Search Algorithm

One of the main algorithms described in the article "The Complexity of Minesweeper and Strategies for Game Playing", is the basic limited search algorithm. The basic limited search algorithm provides a well organized way of Minesweeper playing (Pedersen et al., 2004). The main idea of this approach is to iteratively explore squares with concentrating on computational efficiency, based on the predetermined limitations. The information about adjacent or neighboring mines and labels on each tile is being updated all the time to direct the newt moves. But in situations where a more thorough investigation of the Minesweeper grid is required, this method is lacking. The main benefit of this algorithm is that it is efficient in the worst case (Pedersen et al., 2004). The Basic Limited Search algorithm strikes a compromise between processing resources and accuracy by limiting the search depth. Nevertheless, in the case of this algorithm the difficulties arise in the situations where a better comprehension of the Minesweeper grid is necessary for making the best decisions. There are advantages and disadvantages to Minesweepers Basic Limited Search Algorithm. One of the advantages is that it operates well in the game's most challenging situations, handling complex and unpredictable grids with reasonable efficiency. This makes it a dependable option for navigating in the more complex grids. Apart from this, the algorithm has its disadvantages. One of which is its shallow search depth. Although it is very handy when it comes to the complex cases, it fails when a deeper analysis of the Minesweeper grid is needed.

## 2.6 Limited Search with Probability Estimates

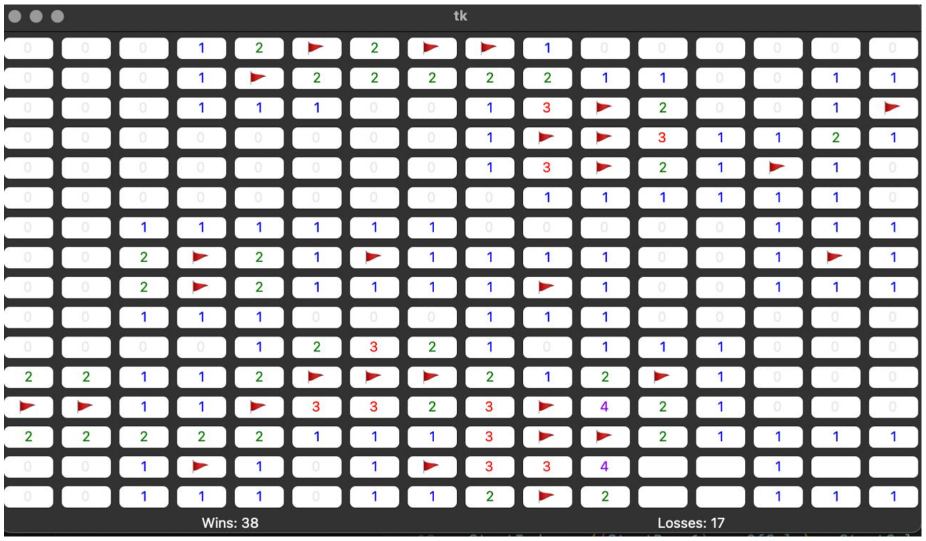
Limited Search with Probability Estimates In the same article discussed in 2.5, the author also introduces probability estimation for constrained search to overcome the shortcomings of simple constrained search. Minesweeper gameplay becomes more complex due to this advanced strategy of integrating stochastic calculations into the decision-making process (Pedersen et al., 2004). The Limited Search with Probability Estimates tracks pairs of squares and their related mine probabilities in a set which is called the safelist. The calculation of probabilities based on previous search results is one of the main components of this method. When there are multiple possible outcomes (squares with mine or no mine), squares are assigned probabilities to ease higher-level decision making. Thanks to the safe list, only the areas with the lowest probabilities of having mines are considered for further exploration. Even though it increases computational complexity, it improves Minesweeper game playing by allowing players to make more strategic decisions. The decision making process gains a new dimension with the addition of probability, providing a more challenging and informed way of playing Minesweeper. Unlike deterministic strategies, probabilistic or stochastic approaches take uncertainty into account and reflect the unpredictability of placement and location of the mines. The safelist mechanism gives more priority to the locations with the least number of mines and it speeds up the decision making process by that. The safelist works as a filter, which allows the algorithm to focus on fields where it is less likely to have mines (Pedersen et al., 2004). The safelist set is crucial and helpful for minimizing the space of the search and for allowing the algorithm to concentrate on the squares that are considered safer based on their probability estimated values. This feature increases the complexity and improves the efficiency of the game, as the Limited Search with Probability Estimates algorithm uses strategies based on probabilities to break through the minefield (Pedersen et al., 2004). This article describes a comprehensive complexity analysis used to assess the effectiveness of demining strategies at various levels of difficulty. The basic limited search algorithm is being effective in worst case situations because of its stable nature but can be limited in average situations. Including probability estimations adds a new layer of complexity and increases overall performance metrics, on the other hand it also increases the computational complexity. When developing mine detection strategies, it is important to find the right balance between exactness and complexity (Pedersen et al., 2004). The ability of each algorithm to adapt to different game scenarios and find the best compromise between computational efficiency and strategic decision making determines the algorithm's effectiveness. The complex interplay of these variables creates a situation for minesweeper strategies and highlights the need for unique and flexible methods (Pedersen et al., 2004).

## 2.7 Minesweeper Consistency Problem

Minesweeper Consistency Problem Meredith Kadlac, under the guidance of Paul Cull, performed research on Minesweeper at Humboldt State University. And the result is the "Explorations of the Minesweeper Consistency Problem" paper. It contributes to the Minesweeper game being considered an NP-complete problem (Kadlac, 2003). Meredith thoroughly studies the complexities of solving Minesweeper and shows innovative approaches to handle the challenges. This research also focuses on the Minesweeper Consistency Problem (MCP), which evaluates whether a Minesweeper grid is consistent. That means, if there is at least one arrangement of mines and open squares that aligns with the grid's specified patterns (Kadlac, 2003). The ways to address the issue are two pivotal components, such as developing a C++ program for searching all consistent configurations for a given Minesweeper grid and examining the problem in a simplified one-dimensional context (Kadlac, 2003). The C++ program, inspired by Neville Mehta's work, uses a Breadth-First Search (BFS) algorithm to explore all possible consistent configurations for a given Minesweeper grid. The program categorizes each new grid based on consistency and goal status, using a systematic method to classify and prune grids throughout the search process. Despite its exponential time complexity, when the grid has numbered squares then the program presents practical tractability for various inputs (Kadlac, 2003). Next, in order to simplify the problem, Kadlac redefines the consistency problem in a one-dimensional context. This version limits the puzzle to a single row of blanks and numbers indicating adjacent bombs. Kadlac demonstrates that this simplified version of MCP is solvable in polynomial time by developing a deterministic finite state machine (FSM), showing that local inconsistencies in a Minesweeper string can be detected effectively and ensuring that every possible inconsistency is accounted for (Kadlac, 2003). The FSM's design maps consistent strings to corresponding consistent bomb configurations by allowing it to process a Minesweeper string of any length. This approach demonstrates that every consistent Minesweeper string translates into a possible bomb layout, reinforcing that the one-dimensional MCP is not only solvable but efficient (Kadlac, 2003).

## 3 Methodology

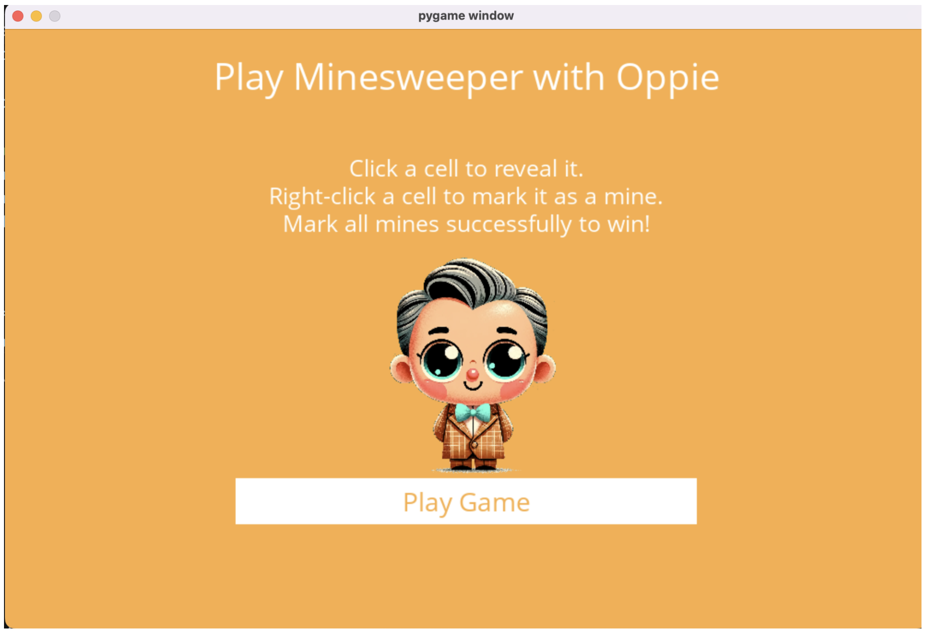
After analyzing the existing methods we decided and chose 3 methods of solving Minesweeper Agent: CSP Agent, Probability-based Agent, and Knowledge-based Agent. As our group consisted of data scientists and faculty students we implemented everything in Python. For each method we tried our best to make an interface, so now let's talk about the board creation with its features and implementations. All implementations share the same logic of creating the board by creating a window that contains the grid of tiles. The tiles are created as buttons, these are constructed in a grid layout with the provided number of rows and columns. The AI has the option of clicking or flagging each button. Mines are placed randomly with a state attached to it to indicate the mine present. For each none mined tile, it is calculated neighboring mine and displays the number. Starting with the Knowledge-Based (KB) agent, the graphical interface was developed using the Tkinter model, which is a wrapper for Tcl/Tk functions. This module provides classes allowing you to display, position, and control the widgets. The script begins by importing necessary modules, such as random, math, and time, and initializing variables for game settings, including the number of bombs, rows, columns, etc. The Tkinter root window (`root = Tk()`) is then created to house the game's graphical interface. The game board is constructed with a grid of tiles (buttons) using a nested loop with specified rows and columns. Each button is added to the `btn` list. Subsequently, mines are added by randomizing their positions with the help of `random.sample`, and the mine locations are stored in the `isBombList`. Functions such as `flagClick`, `my_fun`, and `ChordClick` are responsible for managing flagging and clicking. They handle tasks such as flagging tiles, revealing tile contents, and enforcing the game's rules when a tile is clicked. Additionally, functions like `spiral`, `spiralGuess`, and `spiralMaybes` are designed to traverse the board in a spiral pattern, revealing known tiles from the knowledge base (KB). The `restartSweep` function restarts the game in case of a win or loss, and a counter keeps track of wins and losses. The script enters the Tkinter main loop (`root.mainloop()`) to initiate the game and manage AI interactions. Upon completion, the terminal prints information such as the time, win count, and loss count.



Wins: 38 Losses: 17

Figure 2: Board of Knowledge Based Agent during the game: numbers on squares indicating the numbers of mines in neighborhood

The CSP and PB board creation logic is the same. In both cases, Pygame is used, which is a set of Python modules designed for writing games. It is written on top of the excellent SDL library. This allows you to create fully featured games and multimedia programs in the Python language. It's again in the first place called necessary modules and libraries. Pygame includes GUI library init which creates a window to host the game. This window will contain the grid of tiles. The tiles represent a matrix with given column and row numbers and are randomized in placement. The tiles with mine have specific attachments that let the program know that they are mine. Also in those versions, we added options, as from the beginning the user starts the play the game and then if needed let the AI make a move, show what moves the AI prefers by clicking on show inference, or let the AI play by clicking on the AI name. The AI players as an autoplay until the user does not stop it.



Play Minesweeper with Oppie

Click a cell to reveal it.
Right-click a cell to mark it as a mine.
Mark all mines successfully to win!

Figure 3: Starting page of CSP Agent Oppie

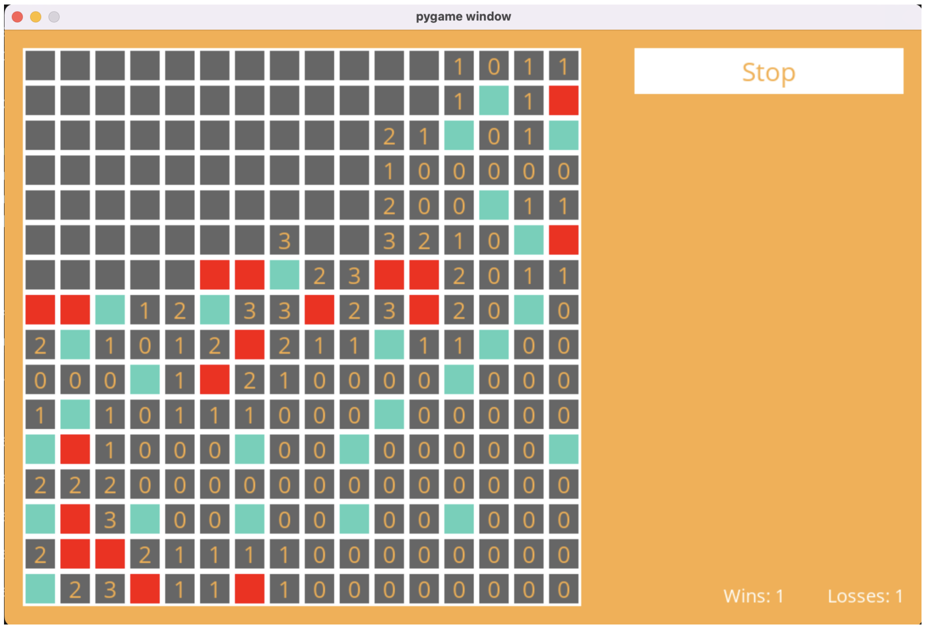
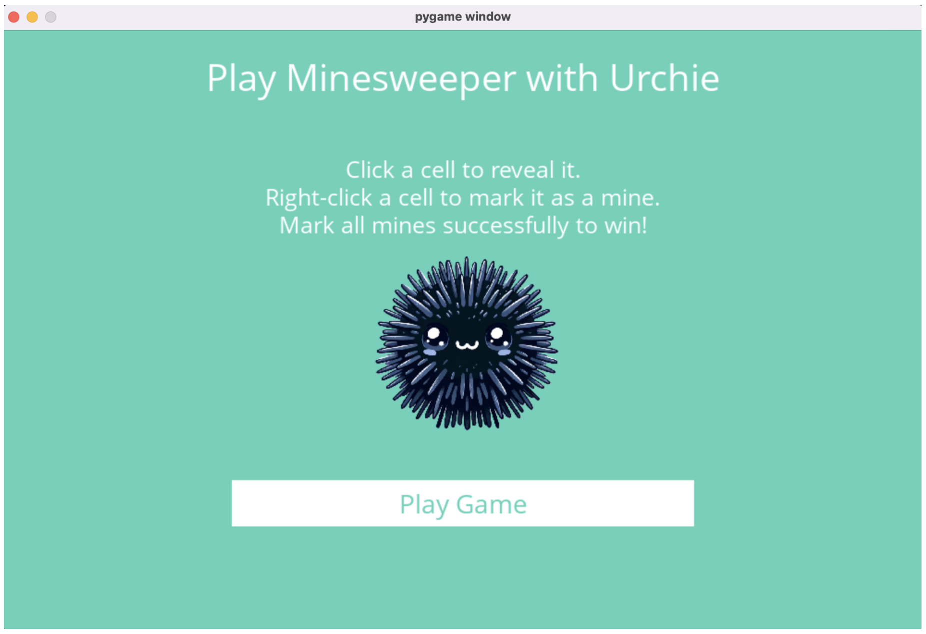


Figure 4: Board of CSP Agent during the game: numbers on squares indicating the numbers of mines in neighborhood



Play Minesweeper with Urchie

Click a cell to reveal it.
Right-click a cell to mark it as a mine.
Mark all mines successfully to win!

Figure 5: Starting page of Probability Based Agent Urchie

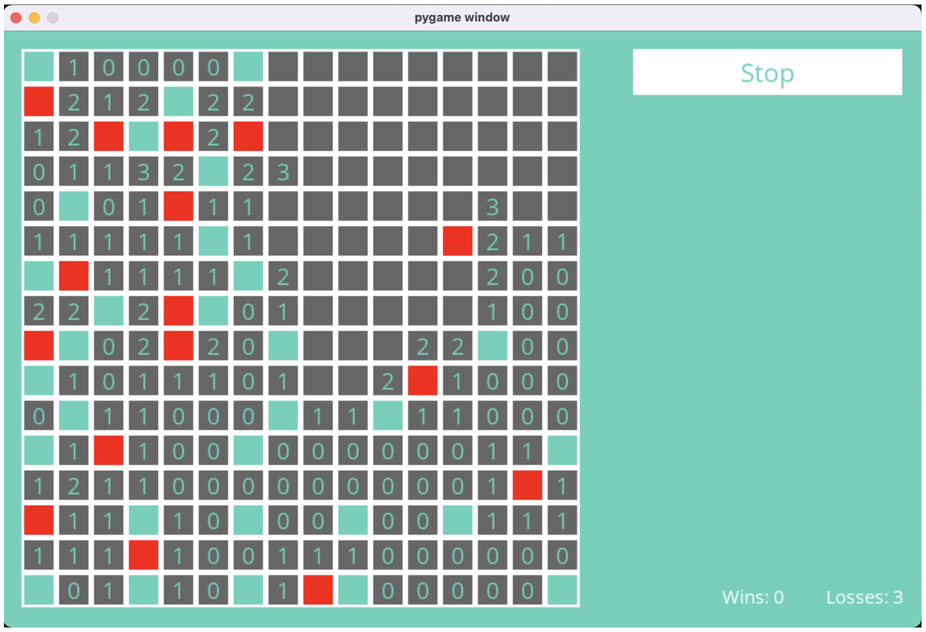


Figure 6: Board of Probability Based Agent during the game: numbers on squares indicating the numbers of mines in neighborhood

## 3.1 Minesweeper as CSP

Our initial MinesweeperAI, named Oppie (derived from Oppenheimer's name), can be characterized as an agent that utilizes an algorithm based on the principles of the Constraint Satisfaction Problem (CSP) to solve the Minesweeper game. A set of variables, domains for each variable, and constraints limiting the possible values for variable combinations define the problem in CSP. The way our algorithm solves the game board and fits into the CSP framework is as follows:

**Problem Definition:**

- Variables: Each Minesweeper board cell individually.
- Domains: Every variable represents a domain of possible states, i.e., safe, revealed, or mine.
- Constraints: The Minesweeper rules implicitly define the constraints. The number of nearby mines for every revealed cell serves as the constraint.

**Initialization:**

- The MinesweeperAI class initializes variables (`self.moves_made`, `self.mines`, `self.safes`) and a knowledge base (`self.knowledge`), setting up the initial state of the CSP.

**Deductive Reasoning (Constraint Propagation):**

- `add_knowledge` Method: The algorithm performs constraint propagation upon the revelation of a safe cell.
- It adds the cell to {`moves_made`} and designates it as safe.
- Based on the count and revealed cell, constructs a new sentence that represents the constraint.
- Updates the knowledge base by analyzing already-written sentences to draw new conclusions about safe cells and mines.

**Logical Inference (Constraint Propagation):**

- To update the knowledge base iteratively, the algorithm uses logical inference.
- It determines the relationships between already-formed sentences and makes inferences regarding safe cells and mines.
- The steps in the propagation of constraints are represented by the removal of duplicates (`remove_dups`) and the resolution of certain conclusions (`remove_sures`).

**Decision Making:**

- make_safe_move and make_random_move Methods: These methods represent the decision-making phase of the CSP algorithm.
- make_safe_move selects a safe cell based on the current knowledge.
- make_random_move makes a random move among unrevealed and unmarked cells, adhering to CSP constraints.

**Note:** If there is a situation that the AI agent cannot make a right and perfect decision, it performs a random move.

The Oppie agent's Minesweeper solver algorithm, which is based on the CSP paradigm, effectively distributes information, manages constraints, and makes choices to play the Minesweeper game wisely. It uses logical inference to quickly determine the locations of safe cells and mines, and it dynamically updates its knowledge base to reflect the changing state of the CSP.

## 3.2 The Probability-based agent

The Probability-based agent named Urchi works by combining a few strategies to make a move. The core of the agent is the MinesweeperAI class, which is responsible for playing the game.

**Problem Definition:** The problem definition is the same as Constraint Satisfaction Problems (CSP), except for constraints.

**Initialization:** The MinesweeperAI class has the following variables initialized:

- self.moves_made
- self.safes
- self.mines

It also maintains a knowledge base, self.knowledge, which is a list of Sentence objects.

**Inference:** Inference is made based on information in the knowledge base to understand the locations of safe and mined tiles after each decision and execution. The add_knowledge function is crucial for this process. The AI learns about the revealed or flagged cell and adds new information to the knowledge base. It deduces mines and safe cells from this new knowledge and marks them accordingly.

**Decision-making:** The main difference for the Probability-based strategy in decision-making is as follows:

- If no known_safe_cell is available, the make_advanced_move function is used to pick a cell before resorting to make_random_move.
- The advanced move calculates the probability of a mine in a tile using the knowledge of known and remaining mines. It chooses the tile with the lowest probability if the tile's probability is under a certain value, or if all tiles have the same probability, it chooses make_random_move.

## 3.3 Knowledge Based Minesweeper

This algorithm for solving Minesweeper employs probabilistic reasoning, rule-based logic, and strategic decision-making to flag possible mine locations and reveal tiles in a methodical manner. It dynamically adapts its strategy based on the changing game grid. The goal is to solve Minesweeper puzzles effectively and methodically by utilizing a combination of tactics. While it doesn't explicitly use a "knowledge base" in the traditional sense, the algorithm relies on a set of rules, heuristics, and strategies to explore and manipulate the Minesweeper grid based on gained information. Here's how it utilizes its knowledge effectively:

**Initialization:**
The Minesweeper tiles are represented by a grid of buttons at the beginning of the game, with a predetermined number of bombs, rows, and columns. A predetermined starting point is used to launch the algorithm.

**Deterministic Clicks:**
Using information about the environment, the algorithm initially makes deterministic clicks. The number on a tile, indicating the number of nearby bombs, appears if the tile has been clicked. Clicking a tile triggers the automatic clicking of its neighboring unexplored tiles if the number of flagged tiles surrounding it matches the number on the clicked tile. This deterministic approach helps find safe areas quickly with less random exploration.

**Predicting Clicks:**
The algorithm switches to a predicting strategy if deterministic clicks are not possible. It uses a spiral layout to traverse the grid, exposing tiles and identifying possible safe areas. The algorithm guesses whether bombs are present in nearby unexplored areas based on data from previously clicked tiles.

**Chord Clicks:**
When the number on a clicked tile and the number of flagged tiles surrounding it match, the algorithm clicks on the tile. To expedite the exploration process, a chord click simultaneously reveals all unflagged neighboring tiles.

**Recursive Checks:**
The algorithm remembers the tiles that have been clicked and avoids areas that have already been explored twice to ensure efficient exploration. Recursive checks are carried out to reveal connected safe areas.

**Flagging:**
Using data gathered from exploration, the algorithm systematically flags tiles it believes to be bomb sites. It employs a "maybe" flagging technique to identify tiles that could be bomb-filled. Educated guesses about possible bomb

locations are made based on the number on a tile and the nearby flagged tiles.

**Restart Mechanism:**

To prevent becoming stuck in a loop or making ineffective decisions, the algorithm includes a restart mechanism that kicks in after a predetermined number of iterations. It switches to a different probabilistic exploration strategy after several failed attempts.

**Game Completion:**

Until a bomb is clicked (loss) or all non-bomb tiles are revealed (win), the algorithm repeats the previous steps. The algorithm notifies the user of every bomb location in the event of a loss.

**User Interface Updates:**

The Tkinter interface is dynamically updated to show the locations of bombs, revealed numbers, and flags in relation to the states of the tiles.

Overall, the algorithm intelligently combines deterministic and probabilistic strategies to efficiently solve Minesweeper games. It leverages information from clicked and flagged tiles to make informed decisions during exploration.

## 4 Evaluation

In this section, we compare the performance of three distinct algorithms for solving Minesweeper instances of varying complexity levels: an agent utilizing constraint propagation (Oppie), the agent with probabilistic solution algorithm (Urchie), and a knowledge based agent . The evaluation is conducted on two levels of Minesweeper games, one with an 8x8 grid and 8 mines (the easier one), and the other with a more challenging 16x16 grid and 30 mines (intermediate). All three agents have an option of changing the speed and waiting time between each step and each game. To assess the performance of each algorithm, we consider the following metrics:

**Winning Rate:** The percentage of games successfully solved within a given time limit.

**Average Time:** The average time taken by each algorithm to solve a Minesweeper instance.

| **Agent** | **Games Played** | **Time (Easy)** | **Win (Easy)** | **Time (Hard)** | **Win (Hard)** |
|---|---|---|---|---|---|
| Agent 1 | 1000 | 12.38 min | 665/1000 | 26.14 min | 733/1000 |
| Agent 2 | 1000 | 16.24 min | 654/1000 | 44.26 min | 671/1000 |
| Agent 3 | 1000 | 1.547 min | 678/1000 | 3.873 min | 699/1000 |

Table 1: Performance Summary of Minesweeper Solving Agents

For the first agent, we got that from 1000 games played, the agent was able to play the 1000 rounds of the easier greed in 12.38 minutes and won 665 out of 1000, and it was able to solve the 1000 rounds of the harder-intermediate greed in 26.14 minuter and won 733 out of 1000. It is important to mention that this agent only loses when it performs a random move.

For the second agent, we got that from 1000 games played, the agent was able to play the 1000 rounds of the easier greed in 16.24 minutes and won 654 out of 1000, and it was able to solve the 1000 rounds of the harder-intermediate greed in 44.26 minuter and won 671 out of 1000. It is important to mention that this agent only loses when it performs a random move.

For the third agent, we got that from 1000 games played, the agent was able to play the 1000 rounds of the easier greed in 1.547 minutes and won 678 out of 1000, and it was able to solve the 1000 rounds of the harder-intermediate greed in 3.873 minuter and won 699 out of 1000.

The three Minesweeper-solving agents—Urchie, Oppie, and the knowledge-based agent—perform admirably, each demonstrating a unique strategy for addressing the problems presented by Minesweeper instances. Of these agents, the knowledge-based algorithm stands out as being especially effective at solving Minesweeper puzzles quickly. Let's examine their performances in more detail using a comparative analysis. Oppies' strong performance is demonstrated by their winning rates of 66.5Urchie exhibits a competitive advantage with success rates of 65.4On the other hand, the knowledge-based agent presents a unique approach. This algorithm blends probabilistic and deterministic tactics, dynamically changing its strategy in response to the Minesweeper grid's evolution. With remarkably faster solving times of 1.547 minutes and 3.873 minutes, respectively, it outperforms both Oppie and Urchie in terms of efficiency, achieving winning rates of 67.8

The decision between these algorithms depends on a number of variables, such as the desired winning rate, the availability of computational resources, and the preference for probabilistic versus deterministic methods. The knowledge-based agent becomes an attractive option if computational efficiency and striking a balance between deterministic and probabilistic strategies are important factors. In conclusion, the knowledge-based agent stands out for its efficiency, achieving competitive winning rates with noticeably faster solving times, even though Oppie and Urchie give excellent performances in Minesweeper solving. This emphasises how crucial it is to take into account the subtleties of each algorithm's methodology when choosing the best course of action for solving Minesweeper puzzles.

Moreover, it is crucial to draw attention to a unique gameplay feature of the knowledge-based (KB) agent that differentiates it from Oppie and Urchie. After the first click, the KB agent uses a novel tactic by opening a sizable portion of the Minesweeper grid. In contrast to Oppie and Urchie, who usually concentrate on exposing one cell following the opening move

## 5 Conclusion

In conclusion, this study delves deeply into the use of artificial intelligence to win the logically challenging game Minesweeper. It contrasts three AI algorithms with different approaches and levels of efficiency: CSP, Probability-based, and Knowledge-based Agents. While the Probability-based Agent concentrates on chance, the CSP Agent concentrates on constraints, and the Knowledge-based Agent employs a combination of probabilistic and deterministic strategies. The outcomes show how well the Knowledge-based Agent performed, effectively overcoming the obstacles in the game. This study provides insightful information about how AI techniques can be applied to solve problems effectively, especially in games where a combination of probability and logic is needed.

## References

[Bec15] David Becerra. *Algorithmic Approaches to Playing Minesweeper*, 2015.

[Cic17] Jan Cicvárek. *Algorithms for Minesweeper Game Grid Generation*, 2017.

[Kad03] Meredith Kadlac. Explorations of the minesweeper consistency problem. *Oregon State University, Corvallis*, 2003.

[Ped04] Kasper Pedersen. The complexity of minesweeper and strategies for game playing. *Department of Computer Science University of Warwick*, 2004.

Minesweeper Game. (n.d.). Retrieved from https://minesweepergame.com
[Ped04]. [Bec15]. [Cic17]. [Kad03].