Essay of Mini Project for the course Intro AI

Katharina Stengg, 11904546

March 7, 2023

Abstract

This essay is split up into two parts. The first part is about the implementation and testing of various agents for solving the computer game Minesweeper.
The second part is a discussion about ethic aspects of the scenario when the agent is developed and used in the real world in military context.

1 Implementation of various AI's to solve the game

1.1 Initial Setting

First the basic environment for the game needs to be created. I decided to do that from scratch and then my agent should play automatically and I watch the process via command line tool. Two methods are inspired by an online implementation as I struggled a bit at first. This methods are clearly marked as "not completely by me" in the code and have the url to the implementation in the header to not vioalte the code of conduct. Following libraries need to be installed to run:

- Python3
- Z3 (for SAT)
  pip3 install –user z3-solver
- Numpy
  pip3 install numpy

In this project for beginners level a grid size 8x8 with an amount of 8 mines and for intermediate level a grid size 16x16 with 40 mines was used. I tested the solvers in this settings but the values can be adapted in the class main.py. This is the class where the game and solver are initialized (you can also change the solver there, all are there as code comment, uncomment the solver you want to test and this one is used then, same with mines and grid sizes, everything can be changed in the main class) and where the game loop is run and stopped.

To run the program first go into the folder called **final_submission** and then type **python3 main.py**. Depending on the values set in the main.py file a game is now started.
Another important file of the initial setting is the boardClass.py. This class is responsible for the game field and the methods regarding this. Initially a public and a private board are created. The public board is always updated and at first filled with unknown values ('.'). The private board is initially filled with 0's and afterwards is filled with the mines randomly placed and their numbers which indicate adjacent mines. This approach is inspired by https://medium.com/swlh/this-is-how-to-create-a-simple-minesweeper-
The other classes are solver classes where each solver is in one class and has the same "main" method which is the method getMove() which gives the coordinates of a valid move to the gameloop of the main class.

1.2 Random AI

The first AI is a random AI which is done in the file **RandomAI.py**. Out of all unknown fields randomly a field is chosen. This is established via a recursive method, which is called as long as no free field is found via random coordinates.

1.2.1 Evaluation of random AI

The agent was the first approach to test the whole programs functionality as the implementation of the mine reveals were quite a challenge for me at first. The random AI fails almost every time on a big grid because there are so many options. On the smaller grid due to the 2 initial moves the random AI still has a winrate of about 28% but this is just because of the small field and therefore little possibilities. Also if a human tried to play the game without thinking any further about the numbers on the tiles the human will fail with a high chance.
Figure 1 and 2 show the result and winrate of 50 game runs with the random AI on the two tested grid sizes.

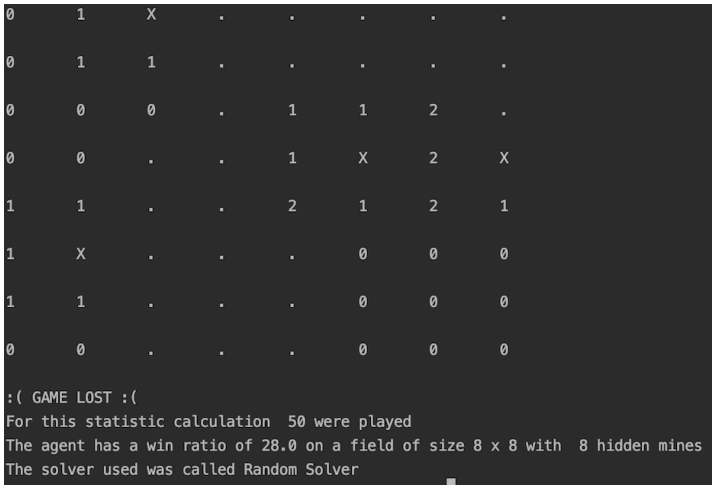

Figure 1: Winrate of random solver on 50 game runs on beginner level

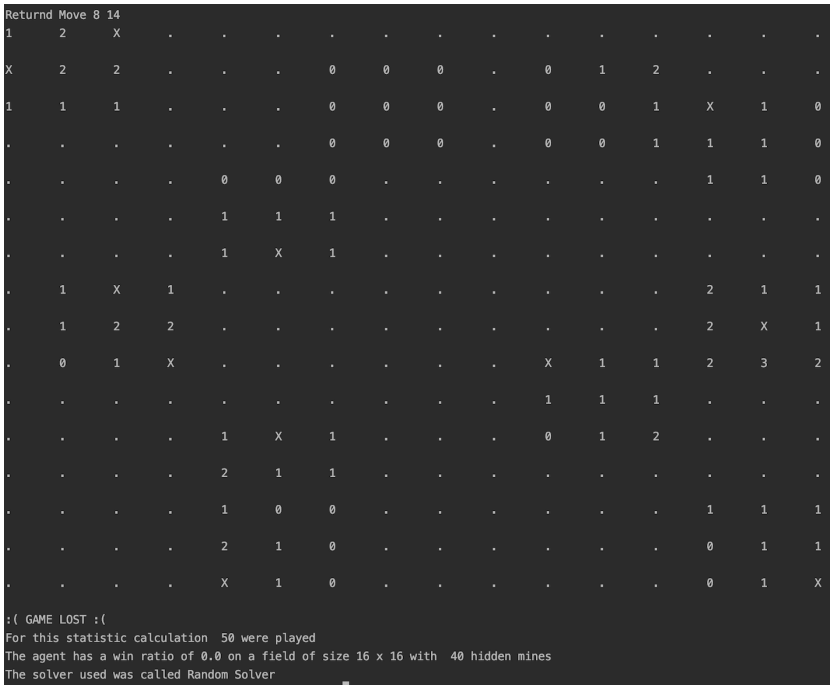

Figure 2: Winrate of random solver on 50 game runs on intermediate level

For all runs the winrate is calculated which basically is the percentage of games won out of the total games played.

1.3 Probability AI

The main idea of the AI is to use the field of least risk to contain a mine. This risk is calculated using the information given on the board. The solver loops over all fields on the public board and a dummy board variable which is manipulated to make things easier called board_r is introduced. For each mine ('X') found revealed on the public board the adjacent fields of this mine in the array board_e are reduced by 1 (if they are known and numbers). If a field is known (contains no '.' but a number or X) the solver skips this field as it can't be chosen anyway. For each unknown field the solver calculates the coordinates of the fields adjacent to this field via the method getCoords(). The solver then loops through the adjacent fields and counts the occuring values:

- If the field has the value 0 the original field can be safely chosen as when a adjacent field has value 0 the field can't contain a bomb.
- If the field is a bomb (X) the **counter_bomb** is increased by 1.
- If the field is unknown (.) the **counter_unknown** is increased by 1.
- If the field is a number (0 to 8) the **sum** is increased by this number

If the first case of above is true the coordinates are added to an array called safe_cells. If safe_cells is not empty, coordinates of one of those cells are returned as they all can be safely chosen. Otherwise if no safe cells are found the other three values of the list above are used to find another field of least risk. The probabilty of the field and it's adjacent fields is then calculated via following formula:

$prob = \frac{sum*100+counter_unknown*50}{len(coords)}$
This formula was invented by me by thinking about how I would choose a field. The probabilty and the fields coordinates are added to an array.
After the loop which loops through the fields is finished the array coords_and_probabilities contains all probabilities and coordinates of fields which could be chosen as move. This array is now sorted according to probability ascending. The solver now chooses to return the row and column of the array entry with the lowest probability value as this is the field with the least probability to contain a mine. If the array is empty a random move is made.

1.3.1 Evaluation of Probability AI

Although this approach is rather simple it works quite well especially on smaller grid sizes. Figure 3 and 4 show the winrate of the solver on 50 games on a 8x8 grid with 8 mines and on 16x16 grid with 40 mines. The AI achieved a result of 84% winrate on the small field and 60% winrate on the big field after playing for 50 games each.

```
0      1      X      2      1      1      0      0

0      1      1      2      X      1      0      0

0      0      0      1      1      1      0      0

1      1      0      0      1      1      1      0

X      1      0      0      1      X      1      0

1      1      0      0      1      1      1      0

0      0      0      0      0      0      0      0

0      0      0      0      0      0      0      0

YEY GAME WON YEY
For this statistic calculation  50 were played
The agent has a win ratio of 84.0 on a field of size 8 x 8 with  8 hidden mines
The solver used was called Probability Solver
```

Figure 3: Winrate of probability solver on 50 game runs on beginner level

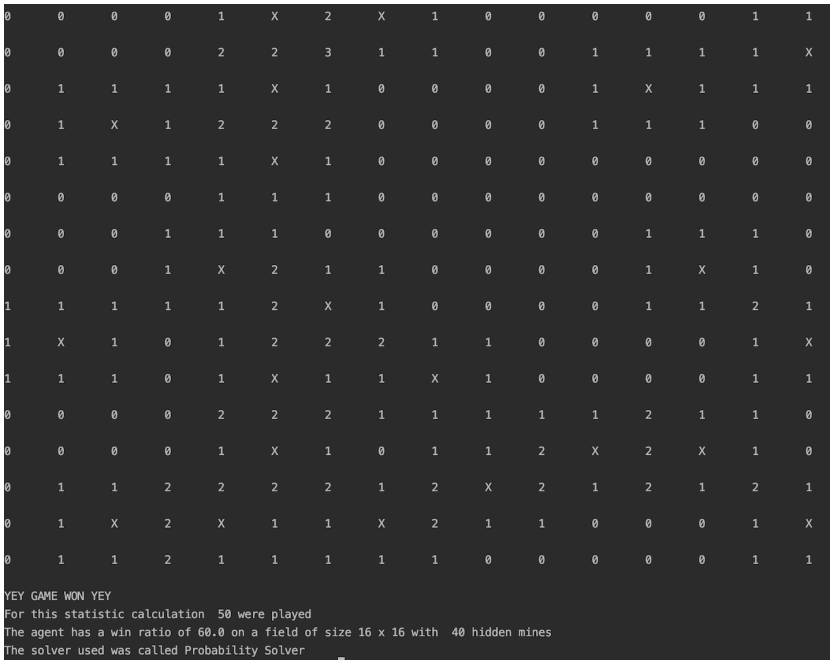

Figure 4: Winrate of probability solver on 50 game runs on intermediate level

1.4 Backtracking AI

This AI uses the principle of backtracking to find a valid placement for all mines missing on the grid. We systematically explore all possibilities until we find locations for all mines. Afterwards a field where no mine was placed is chosen. Due to the fact that especially with little information more than one placement for the mines is possible this AI sometimes fails.

The solver has an array visited_fields, an array mines and an array board_r. As an initial step the mines array is set to True at positions where the given public board contains mines. Simultaneously the board_r array's values around the coordinates of the field where the mine is are reduced by 1. This array is now helpful to decide on whether a cell is safe (a bomb can't be placed there) which is the case when the value of board_r is 0 as no bomb can be in any adjacent field of this field. Initally the board value could have been higher than 0 but due to the fact that there is an adjacent bomb already it got reduced and this fact got taken into account. If a field was visited the value of visited_fields at the corresponding row and column value is set to True. This helps to not visit any already visited field again as this could lead to circles.

Following code shows the backtracking process. First it will be checked whether all mines are found which returns true as this is the goal. If all fields were visited but not all mines found the function returns False as no suitable mine grid result was found via backtracking.
Afterwards a loop over all fields of the grid is done. If a field is not visited and can be set it is checked whether a bomb can be set on this field. For this the function **cellCanBeBomb** is used. This function basically checks the fields neighbors (board_r values). If a value is 0 no adjacent cell and therefore also the current cell could contain a bomb. In this case the function returns False otherwise it returns True. If the return value is True a bomb is set at current row and column coordinates in the mine grid and the values of board_r get adapted. A recursive call of the function is started to place additional mines. This works as due to the visited_fields array the coordinates of already placed mines are skipped. If the recursive call doesn't give any sufficient result the bomb placement is reverted (as well as the board_r calculation) and another recursive method call is done to find alternative bomb placements. Again this works because the current field is skipped due to being visited already. If this call also fails the field is set to unvisited and we return False.

```
def return_Solution(self, board, visited_fields, mine_grid, iterations, board_r):
```

```
if self.AllBombsFound(mine_grid):
    return True, mine_grid, visited_fields
# if all cells are visited but not all bombs found -> no solution
if self.AllVisited(visited_fields) and not self.AllBombsFound(mine_grid):
    return False, mine_grid, visited_fields

# try placing mines until solution for all placements is found
for row in range(self.board.height):
    for column in range(self.board.width):
        # mine can't be place when already taken
        if visited_fields[row][column]:
            continue # to not go into none result bringing fields again
        if board[row][column] != '.':
            visited_fields[row][column] = True
            continue
        visited_fields[row][column] = True
        if self.cellCanBeBomb(row, column, board_r):
            # set mine to current row/col
            mine_grid[row][column] = True
            board_r[row][column] = 'X'
            # after mine is set reduce board_r value of adjacent neighbors
            board_r = self.reduceBombCountField(board_r, row, column)
            # if bomb is placed -> reduce count of surrounding by 1 as risk is reduced

            # Recursive Call
            if self.return_Solution(board, visited_fields, mine_grid, iterations, board_r):
                return True, mine_grid, visited_fields

            # if recursive call with set mine failed -> reset to state before
            # call to backtrack and try another step
            mine_grid[row][column] = False
            board_r[row][column] = '.'
            board_r = self.improveBombCountField(board_r, row, column)
        # recursive call without mine being set
        if self.return_Solution(board, visited_fields, mine_grid, iterations, board_r):
            return True, mine_grid, visited_fields
        # visited field doesn't lead to any result and therefore we delete the try
        visited_fields[row][column] = False
return False, mine_grid, visited_fields
```

The AI was inspired by various backtracking board game solving implementations I found online. An example of a code snippet I read through before trying out my approach is available under https://www.geeksforgeeks.org/minesweeper-solver/ which basically uses the same approach of backtracking to find a suitable mine allocation for the grid. I didn't copy any of the code from there but it inspired my approach so I wanted to mention it just to be on the safe side in the context of the code of conduct. It is also linked in the header of the solver class to be on the safe side.

1.4.1 Evaluation of Backtracking AI

The Solver doesn't perform as good as expected but still achieves sufficient results. Figure 5 shows the results on a 8x8 grid with 8 mines (70% winrate) and Figure 6 the results on a 16x16 grid with 40 mines (40% winrate).

```
1       2       X       1       0       0       0       0

X       3       2       2       0       1       1       1

1       2       X       1       0       1       X       1

0       1       1       2       1       2       2       2

0       0       0       1       X       1       1       X

1       1       0       1       1       1       1       1

X       1       0       0       0       0       0       0

1       1       0       0       0       0       0       0

:( GAME LOST :(
For this statistic calculation  50 were played
The agent has a win ratio of 70.0 on a field of size 8 x 8 with  8 hidden mines
The solver used was called Backtracking Solver
```

Figure 5: Winrate of backtracking solver on 50 game runs on beginner level

```
X       2       1       1       0       0       0       1       1       1       0       0       1       X       1       0

1       2       X       1       0       0       0       1       X       1       0       1       2       3       2       1

0       1       1       1       0       0       0       1       1       2       1       2       X       2       X       1

0       0       0       0       0       0       1       1       1       1       X       2       1       2       1       1

0       0       1       1       1       0       1       X       1       1       2       2       1       0       0       0

0       0       1       X       1       0       1       1       1       0       1       X       1       0       0       0

0       0       2       2       2       1       1       2       1       1       2       2       2       1       1       1

0       0       1       X       1       1       X       2       X       1       1       X       1       1       X       1

0       0       1       1       2       2       2       2       1       1       1       2       2       2       1       1

0       0       0       0       1       X       1       0       0       0       0       1       X       1       0       0

1       1       1       1       2       1       1       0       0       0       0       1       1       1       0       0

X       1       1       X       1       0       0       0       1       1       1       0       0       0       1       1

1       1       1       2       2       2       1       1       1       X       1       0       0       0       1       X

1       1       1       1       X       2       X       1       1       1       1       0       0       0       1       1

1       X       1       .       .       .       .       .       .       .       .       .       .       0       0       0

1       1       1       .       .       .       .       .       .       .       .       .       .       0       0       0

:( GAME LOST :(
For this statistic calculation  50 were played
The agent has a win ratio of 40.0 on a field of size 16 x 16 with  40 hidden mines
The solver used was called Backtracking Solver
```

Figure 6: Winrate of backtracking solver on 50 game runs on intermediate level

1.5 SAT Solver using Z3 library

The Python library Z3 has various useful functions. A Solver can be created. Constraints can be asserted. A check method is available to solve the constraints. If a solution is found the result is SAT otherwise UNSAT.
This principle can be used to determine whether any move for the AI is possible using the provided constraints. The possible moves should be safe and therefore in no way contain mines. For this we place a bomb at each possible coordinate and look at whether we then achieve status UNSAT. If so,

constraints are not met anymore and therefore we can deduce that we can place a bomb there safely. The AI then checks whether the returned move is actually safe (for example if a adjacent cell contains the number 0 as this indicates that there can't be any bomb) and if so returns the coordinates of the chosen cell. If this approach returns no safe cell a random move is made. This solver was inspired by a document available under following link: https://sat-smt.codes/SAT_SMT_by_example.pdf (pages 50-51) where all available moves for a given field are gathered via a similar approach. This document really helped me to get a feeling for using solver as I really struggled with this assigment at first.

1.6 Evaluation of SAT Solver

The Solver doesn't perform as expected but still performs better than the random solver. On the small grid results are not that bad (winrate 64%) but on the bigger grid it fails a lot. Figure 7 shows the results on a 8x8 grid with 8 mines.

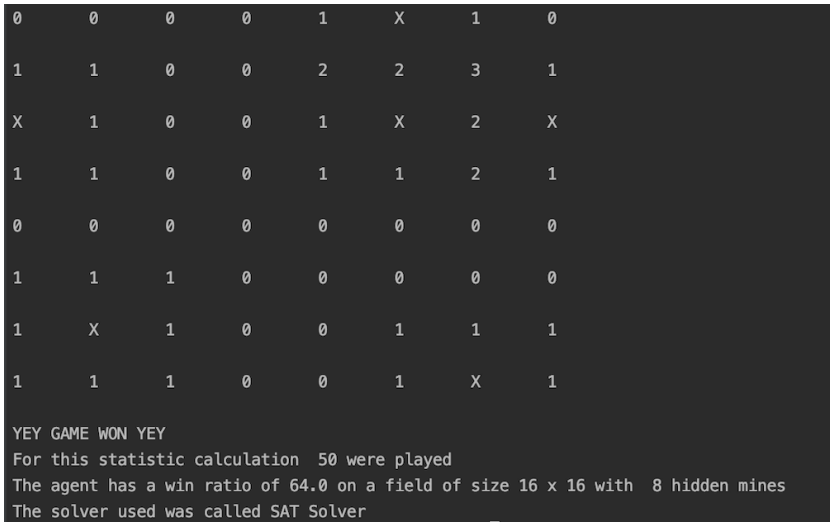

Figure 7: Winrate of backtracking solver on 50 game runs on beginner level

I can't provide an image of the results of the SAT Solver on a bigger grid here as it took way too long on my machine and I need to submit this assignment now :D I additionally tested on a 8x4 grid to show that the program also can handle different heights and widths and with this setting 3 out of 5 games were won.

1.7 Conclusion on project

Overall the solvers achieve somewhat okay results. They definitely can be improved but at least they show good results compared to the random solver which just chooses fields without any further logic. On other grid sizes some solvers (for example Probability and Backtrack Solver on 8x4 grid with 8 mines) achieve even better results close to 100%.
For the backtracking and SAT solver I did a lot of research and read through various papers to get an idea of how to implement such things. The documents which inspired my approaches were linked in the corresponding sections to avoid any violation of the code of conduct (I didn't copy anything on purpose but I am always scared of getting accused of plagiarism or anything so just to be on the safe side :D).The exercise definitely improved my skills.
I really enjoyed working on the project although it was really hard for me to achieve results. I am good in theory but still lack in programming skills so this exercise was really good to improve my research on approaches and invent my own way of implementing things. I am still far away from perfect with these solvers but I hope that they are at least enough for a positive grade as I invested a lot of time and work into this project and really wanted to present good results.

2 Discussion on ethics of deploying such methods in real life

Sending a drone into the minefield in general is an interesting idea as the drone is a machine and can be replaced when destroyed. Therefore if we send a drone into a dangerous area no human life will have to be put into this dangerous zone as especially the task of mine detection can get very danergous. At first this sounds like the risk for soldiers is drastically reduced. In the ideal scenario they could enter the area safely after the drone searched it through but is this really the case?
The first question which came to my mind was, what happens if the machine fails? Of course also humans make mistakes and the task of mine detection is really hard but a small bug in the code of the drone could make a huge impact especially if the software is on various drones worldwide. What happens if the signals the drones work with are distorted?
Another topic which is present especially in modern times is the Cybersecurity aspect. Can the drone be hacked of people who want to harm the soldiers? The drone could then be manipulated to tell the soldiers that there are no minefields but actually all of them will detonate when the soldiers enter the area and more harm would happen than at the case where a human was searching for the mines.
Also the magentic measurements which are done beforehand could be falsified by enemies on purpose.

If we consider the ethical aspect of automated mine searching in the military context there are good and bad perspectives. First of all war always has ethical problems as war is about gaining victory via eliminating people and causing destruction. Therefore using such machines in the context of war is also ethically problematic in general. On the other hand using such machines if they are tested enough could safe soldiers which is a good thing as lifes are saved but still those saved soldiers could harm their enemies so the machine again helps to harm people.
I can conclude that especially in the context of war everything is ethically difficult and I am not sure whether I would have a good feeling if my machine would be used in times of war.