Implementation of Basic games using Various Algorithms

Course Name: Data Structures and Algorithms
Semester:Winter Semester 2021-22
Slot: E2
Team Members:

1. Moola Sri Hemanth Reddy – 19BEE0021
2. Shreeya Santhanam – 19BEE0182
3. Pavitra Mathur – 19MIY0035
4. Satrajit Das – 20BEE0248
5. Anamay Krishna Tiwari – 20BEE0273

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

ABSTRACT :

Classic games have always been a good pass time for many people. Most of these games include Sudoku, Chess, Tic tac toe etc. There are various methods to solve and play these games. In this project, algorithms for various classic games are going to be implemented and hence corresponding codes will be written. The purpose of this project is to come up with a novice method in order to implement algorithms for each game and design an user interface in order to access the games and play the game of choice. Along with that the use of Artificial Intelligence will be depicted in order to find the optimal solution of each game.

INTRODUCTION :

In this project algorithms for 5 classic games have been designed and implemented. These games are usually played on paper and there are several permutations and combinations to play those games. Our aim was to find out the most optimal solution for each step and come up with an algorithm for each of the games in order to solve them efficiently, precisely without consuming much time. For some games AI based algorithms have been implemented and the corresponding solutions were found. In these games, are many modern libraries present in this.

Some of the known games such as Chess, Pacman, Minesweeper and Snake games have been implemented along with Sudoku solver which can find answers for a given sudoku game by scanning the image of the game. All the implemented games require use of some important data structures such as arrays, maps, linked lists, etc. This data structures help in organising the data properly and help the computer access the required data(moves) for effective play against the user. Chess has been implemented using Minimax algorithm which is a type of backtracking algorithm. Sudoku solver has also been implemented using Backtracking algorithm. Pacman has been implemented using Breadth first search algorithm which is used to traverse through trees and help find solutions. The snake game has been implemented using LinkedList. Minesweeper has been implemented using mapping.

LITERATURE SURVEY

Zhou et a.l[1] has proposed a comparison between the use of breadth first search strategy and the best first strategy. It has been concluded that the breadth first strategy is more efficient. It prevents the nodes from regenerating. This makes it easier to implement. It can be used to various problems including many conventional problems which otherwise take a long amount of time when solved using traditional algorithms. Various theorems, proofs and corollaries have been proposed in this paper using which the use of Breadth First search can be proved.

Kang et al.[2] has proposed how to implement connect 4 game using Minimax algorithm. It has been demonstrated how Minimax algorithm can decrease the time taken for a game and how efficiently it can map solutions for various games. Three experiments have been conducted in order to find relation between functions, searching depth and to find the number of features. Alpha beta pruning has been used to delete the unused nodes which otherwise are of no use and increase the time of execution of the game. Hence this approach is not time consuming along with efficient and precise.

Nasa et al.[3] has proposed the use Minimax algorithm to solve the connect 4 game. It is a backtracking algorithm and is used to find the most appropriate and optimised solution. Various

applications of this algorithm have been specified which include chess, tic tac toe, etc. Moreover in order to increase the time efficiency and precision Alpha Beta Pruning has been suggested. It helps in decreasing the unused nodes which otherwise lead to increase in time taken for the execution of the game and storage of unwanted data. A comparison between mini max and alpha beta has been performed in order to specify the differences between the two in terms of computation time, operation time and the total number of iterations required to perform the execution.

Buffer et al. [4] has present Upper Confidence Tree and domain specific solvers. The use of UCT has been proposed. The various methods of applying the UCT to constraint problems have been proposed. The sequential decision making research has been proposed. The various requirements and implementation techniques have been proposed. The application of UCT to minesweeper has also been presented.

Mossel et al.[5] has proposed a model to develop minesweeper game solutions. The model is initiated through movement of mines on a latice. It then tries to search for an infinite path if sites which are mine free. When a free site is discovered, the player is provided with information related to other sites which are adjacent to the present site. Then the parameter values have been compared. This can be done using a strategy in which it needs a critical parameter which is used to locate an ordinary mine. Bounds for the values are derived when there are difficult constraints when finding out the next move for the player. Further some issues and drawbacks related to the model have been discussed.

Tingting et al. [6] has performed a study on how the various traditional algorithms when applied to minesweeper lead to excessive complexity along with issues with efficiency and time complexity. Hence game refinement theory has been proposed through which the model designed proves to be effective along with the success rate of minesweeper being high.

Vomlelov´a et al[7] has proposed the use of Bayesian Networks for implementation of Minesweeper game. It was observed that compared to traditional techniques Bayesian networks provided better and efficient results. Moreover Ransk One decomposition has been applied and conditional probability tables have been used. This implementation helps in decreasing the time complexity for solving the problem using Bayesian Networks. Various experiments have been performed in order to test the working of Bayesian networks and the outputs have been analysed and compared with traditional approaches.

Robles et at. [8] has proposed tree search method for Ms. Pac Man. The tree search method has proved to be useful and efficient when applied to other conventional games. The work proposed describes the implementation and application of tree search method to Ms pac man. It has been proved to be very efficient and time consumed is lesser.

DeNero et al. [9] has proposed the application of AI concepts for implementation of Ms . Pac man. The various concepts that can be applied are state space search, multi agent search, probabilistic inference and reinforcement learning. Moreover the use of heuristics, evaluation and feature functions have been suggested in order to provided effective, efficient and optimal solution.

Grivokostopoulou et al. [10] has proposed the use of search algorithms in various algorithms. The use of the search algorithms have been proved to be very efficient and helpful for developing solutions for the Pacman game. The performance of various search algorithms have been proposed. Moreover visualisations have been provided in order to understand the working of these search algorithms and understand their application and implementation.

Wirth et al.[11] has proposed the implementation of Pac Man using influence Map model. The model suggested in simple in nature and takes into account most essential parameters. The

model has 3 parameters that are established on the basis of relationship with the agent's behavior. Model's performance has been enhanced using random and systematic global exploration and greedy algorithm.Further the parameters have been effectively optimised.

Duro et al. [12] has conducted a study on performance of various algorithms when designing the game of chess. Further a new approach has been proposed. The use of Particle Swarm optimisation has been proposed. It is used to find out the weights of heuristic function. Moreocer it has been compared with simulated annealing in order to find merits and demerits of the approach.

Wolfe et al. [13] has proposed various methods to get the snake model. Various algorithms and solutions to solve the snake game have been studied.

Job et al. [14] has proposed the use of recursive back tracking algorithm in order to solve a 9*9 sudoku game. The project has depicted the number of valid grids in a game of sudoku and a valid programming approach in order to find the right and optimised solutions in a chess game. The algorithm for solving sudoku has been proposed. The results and outputs have been achieved as expected.

Kamal et al. [15] has proposed an image processing technique in order to find the solutions for a game of sudoku by capturing and scanning the image of the game. Adaptive thresholding , Hough Transform, geometric transformation have been used to find the optimal solutions. Digital detection and decryption of sudoku haas been proposed. The algorithms implemented are Back Tracking, Simulated Annealing and Genetic algorithm which are a part of effective optimisation techniques.

Maji et at. [16] has proposed a minigrid based backtracking algorithm for implementing sudoku game. Traditional cell based sudoku games take long time to be solved. Hence minigrid approach has been implemented in order to find the solutions efficiently without wasting a lot of time. The above method finds application in Encryption, Authentication, watermarking etc. of images.

Berggren et al.[17] has proposed the study of three various algorithms to solve the Sudoku game. The parameters required are solving ability, difficulty level, generation and parallelization. These aspects are further compared with other algorithms. First the game is implemented using Backtracking. Further rule based and Boltzmann machines have been used to find an effective method to solve Sudoku puzzle effectively.

Hasanah et al.[18] has proposed the use of Ant Colony Optimisation method in order to find the optimal solution for a game of Sudoku. It has been observed that this method is faster and more efficient compared to the traditional backtracking algorithm. The proposed work helps in finding out the best possible solution for Sudoku game.

Wei et al. [19] has proposed the use of deep reinforcement learning to play the classic game. The outputs were observed to be sparse and delayed. By considering more parameters and by using refined deep reinforcement learning, this shortcomings can be satisfied.

Sridhar et al. [20] has proposed the use of artificial intelligence in snake game in order to find the best effective solution. First the game was built. Further propoer game data was collected in order to train the ML models. The data collected was cleaned and was made suitable for training. The training was performed and was integrated to an user interface. Further necessary features were added to make the interface interactive and user friendly.

Becerra et al. [21] has proposed an algorithm in order to solve Minesweeper game. The parameters considered are start of the game, the heuristics to perform guesses and ways available in to perform probable deductions. The work proposed has studied the use of single

point apporach. Moreover constraint satisfiability has been studied in order to conclude the best possible solution.

Golan et al. [22] has proposed several algorithms and has conducted a comparison between all the algorithms to find out optimal algorithm. Further a method has been proposed to calculate the winning probability at the end of each possibility.

Iyer et al.[23] has proposed a review on on sudoku solving using various patterns. Various patterns such as Hidden singles pattern, locked cadidates 'pattern, Naked and hidden pair patterns. The use of these patterns can help in reducing the net time required by an algorithm to find the best possible solution.

Holdsworth et al. [24] has proposed the use of Breadth first search. The study proposed traverses through each and every parameter of BFS. A brief analysis and comparison has been performed between various BFS algorithms.

3. SNAKE GAME

Snake is a video game where the player aims at a food block placed on a random location and with each block consumption, the size of the snake increases gradually leading to multiple failure cases such as collision within the snake body due to size increase or collision to the game screen. The game ends leading to any of such situations with the maximum score equivalent to no of times food bloc, eaten before collision.

The gaming user controls the movement of the snake's head (up, down, left, or right) in order to move the whole body upon the food block target. The snake moves continuously around the gaming screen and the player is not able to stop the snake from moving while the game is in progress, and cannot make the snake go in reverse.

The data structures and algorithm behind the game is equivalent to a singly-linked list concept where it is assumed that the snake head is the linked-list tail whereas the snake tail is the linked-list head. The concept diagram shown below shows how the snake segments, by being associated with a particular row and column on the board, model a snake. Each segment queue block has a next pointer which points in the direction the segments queue block moves. These next pointers allow the head and tail block of the snake body to be advanced easily around the specified dimensioned gaming screen.

Initializing display screen

```
def __init__(self, w=640, h=480):
    self.w = w
    self.h = h
    self.display = pygame.display.set_mode((self.w, self.h))
    pygame.display.set_caption('Snake')
    self.clock = pygame.time.Clock()
    self.reset()
```

Fig 16. Display

Initializing the game state

```
def reset(self):
        self.direction = Direction.RIGHT
        self.headblock = Point(self.w/2 ,self.h/2)
       self.snake = [self.head, Point(self.headblock.x-BLOCK_SIZE,
self.headblock.y),
                      Point(self.head.x-(2*BLOCK_SIZE), self.head.y)]
```

```
    self.score = 0
    self.food = None
    self._place_food()
```

Fig 17. Game state

Initializing food variable to random positions in the screen.

```
def _place_food(self):
        x = random.randint(0, (self.w-BLOCK_SIZE)//BLOCK_SIZE
)*BLOCK_SIZE
        y = random.randint(0, (self.h-BLOCK_SIZE )//BLOCK_SIZE
)*BLOCK_SIZE
        self.food = Point(x,y)
        if self.food in self.snake:
            self._place_food()
```

Fig 18. Place food

Defining play_step class to Collect user input, Move , Checking the game completiton ,Food movements, ui update and returning back score

```
def play_step(self):

        #1. Collect user input
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                quit()
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_LEFT:
```

```
                self.direction = Direction.LEFT
            elif event.key == pygame.K_RIGHT:
                self.direction = Direction.RIGHT
            elif event.key == pygame.K_UP:
                self.direction = Direction.UP
            elif event.key == pygame.K_DOWN:
                self.direction = Direction.DOWN


        # 2.MOVE
        self._move(self.direction)
        self.snake.insert(0, self.head)

        # 3. check if game finished
        game_over = False
        if self._is_collision():
            game_over = True
            return  game_over, self.score

        # 4.Place food or just move
        if self.head == self.food:
            self.score += 1
            self._place_food()
        else:
            self.snake.pop()

        #5. update UI and clock
        self._update_ui()
        self.clock.tick(SPEED)

        # 6.return game over and score
        game_over = False
        return  game_over, self.score


        pass

    def _is_collision(self):

        # hits boundary
        if self.headblock.x > self.w-BLOCK_SIZE or
self.headblock.x < 0 or self.headlock.y > self.h - BLOCK_SIZE:
            return True

        # hits itself
        if self.headblock in self.snake[1:]:
            return True

        return False

    def _update_ui(self):
        self.display.fill(BLACK)

        for pt in self.snake:
            pygame.draw.rect(self.display, BLUE1, pygame.Rect(pt.x,
pt.y, BLOCK_SIZE, BLOCK_SIZE))
            pygame.draw.rect(self.display, BLUE2,
pygame.Rect(pt.x+4, pt.y+4, 12,12))
```

```
        pygame.draw.rect(self.display, RED,
pygame.Rect(self.food.x,self.food.y,BLOCK_SIZE,BLOCK_SIZE
))


        text = font.render(" Score: " + str(self.score), True, WHITE)
        self.display.blit(text, [0, 0])
        pygame.display.flip()

    def _move(self,direction):
```

```
        x = self.head.x
        y = self.head.y

        if self.direction == Direction.RIGHT:
            x += BLOCK_SIZE
        elif self.direction == Direction.LEFT:
            x -= BLOCK_SIZE
        elif self.direction == Direction.DOWN:
            y += BLOCK_SIZE
        elif self.direction == Direction.UP:
            y -= BLOCK_SIZE

        self.head = Point(x,y)
```

Fig 19. Play step class

Reinforcement learning (RL) is a branch of machine learning that studies how software agents should operate in a given environment to maximise the concept of cumulative reward.Deep Q Learning extends Reinforcement Learning by predicting actions using a deep neural network.

After designing the snake gaming environment, another step to be used in Review III is to train a Neural Network with Reinforced Learning technique where I would be:

- Defining suitable observations while understanding the possibilities and challenges.
- Defining suitable rewards.
- Training neural networks with the gym environment.
- Discussion of the results

Since Reinforcement Learning is defined as an environment that can be seen as an interactive problem, requires to be solved in the best way possible.

A reward function is defined inside the environment to quantify success. The snake (agent) has access to the so-called observations, which provide data on the present state of the environment. It can then perform a specified action that returns the observation and scalar reward of the following environment's state. The agent's purpose is to maximise the total amount of rewards in the shortest amount of time possible.

Definition of action space

The idea here is to create an array by basic mathematical logic that the snake will try learn by itself the reinforced learning. The values of each array cell are continuous and must be in the range of [-1,1].

Different codes for generating snake agent AI are-

- **Agent Module:**

```
import Arsonist
 import arbitrary
 import numpy as np
 from collections import deque
 from snakegame import SnakeGameAI, Direction, Point
 from model importLinear_Qnet, QTrainer
 from coadjutor import plot
 = 100_000
 = 1000
 LR = 0.001
 class Agent
 def, init, ( tone)
 = 0
 = 0#randomness
 = 0.9#discount rate
 = deque (maxlen = MAX_MEMORY)#popleft ()
 = Linear_Qnet ()
self.trainer = QTrainer (self.model, lr = LR, gamma = self.gamma)
 defget_state ( tone, game)
 head = game.snake (0)
 = Point (head.x-20,head.y)
 = Point (head.x 20,head.y)
 = Point (head.x,head.y-20)
 = Point (head.x,head.y 20)
 = game.direction == Direction.LEFT
 = game.direction == Direction.RIGHT
 = game.direction == Direction.UP
 = game.direction == Direction.DOWN
```

```
 state = (
 in straight
 (dir_r andgame.is_collision (point_r)) or
 (dir_l andgame.is_collision (point_l)) or
 (dir_u andgame.is_collision (point_u)) or
 (dir_d andgame.is_collision (point_d)),
#Danger in Right
 (dir_u andgame.is_collision (point_r)) or
 (dir_d andgame.is_collision (point_l)) or
 (dir_l andgame.is_collision (point_u)) or
 (dir_r andgame.is_collision (point_d)),
#Danger in Left
 (dir_d andgame.is_collision (point_r)) or
 (dir_u andgame.is_collision (point_l)) or
 (dir_r andgame.is_collision (point_u)) or
 (dir_l andgame.is_collision (point_d)),
#Move direction
,
dir_r,
dir_u,
dir_d,
#Food position
game.food.x>game.head.x,
game.food.ygame.food.y>game.head.y,
)
 returnnp.array ( state, dtype = int)
 def remember (, done)
self.memory.append ( (, done))#pop left ifMAX_MEMORY reached
 pass
 deftrain_long_memory ( tone)
```

```
 if len (self.memory)>BATCH_SIZE
 = random.sample (self.memory,BATCH_SIZE)#list of tuples
 differently
 = self.memory
 countries, conduct, prices,next_states, dones = zip (*mini_sample)
self.trainer.train_step ( countries, conduct, prices,next_states, dones)
 deftrain_short_memory ( tone, state, action, price,next_state, done)
self.trainer.train_step ( state, action, price,next_state, done)
 defget_action ( tone, state)
self.epsilon = 80-self.n_games
 = ()
 ifrandom.randint () move = random.randint ()
final_move ( move) = 1
 additional
 state0 = torch.tensor ( state, dtype = torch.float)
 vaticination = self.model (state0)
 move = torch.argmax ( vaticination). item ()
 ( move) = 1
 returnfinal_move
 def train ()
 = ()
 =( )
 = 0
 record = 0
 agent = Agent ()
 game = SnakeGameAI ()
 while True
 state
 = agent.get_state ( game)
 move
```

```
 = agent.get_action (state_old)
 move and get bew state
 price, done, score = game.play_step (final_move)
 = agent.get_state ( game)
 short memory
 (state_old,final_move, price,state_new, done)
#remember
 (state_old,final_move, price,state_new, done)
 if done
 long memory, plot results obtaoned
game.reset ()
 = 1
 ()
 if score> record
 record = score
.model.save ()
 print (' Game',agent.n_games,' Score', score,' Records', record)
plot_scores. appen ( score)
 = score
 = total_score/agent.n_games
 plot (plot_scores,plot_mean_scores)
#plot
 pass
 if, name, == ', main,'
 train ()
```

• Helper Module

```
 importmatplotlib.pylot as plt
 from IPython import display
 ()
 def plot ( scores,mean_scores)
```

```
display.clear_output ( delay = True)
 (plt.gcf ())
 ()
 (' Training.')
 (' Number of Games')
 (' Score')
 ( scores)
 (mean_scores)
 (ymin = 0)
 (len ( scores)-1, scores (-1), str ( scores (-1)))
plt.text (len (mean_scores)-1,mean_scores (-1), str (mean_scores (-1)))
plt.text (len (mean_scores)-1,mean_scores (-1), str (mean_scores (-1)))
```

• **Model Module**

```
 from re import X
 import arsonist
 importtorch.nn as nn
 importtorch.optim as optim
 importtorch.nn.functional as F
 import zilches
 classLinear_Qnet (nn.Module)
 def, init, (,hidden_size,output_size)
 super ()., init, ()
 = nn.linear (input_size,hidden_size)
self.linear2 = nn.linear (hidden_size,output_size)
 def forward ( tone, x)
 x = F.relu (self.linear1 (x))
 x = self.linear2 (x)
 return x
```

```
def save ( tone,file_name = 'model.pth')
 = './ model'
 if notos.path.exists (model_folder_path)
 (model_folder_path)
 = os.path.join (model_folder_path,file_name)
torch.save (self.state_dict (),file_name)
 class Qtrainer
 def, init, ( tone, model, lr, gamma)
self.lr = lr
 = gamma
 = model
 = optim.Adam (model.parameters (), lr = self.lr)
 = nn.MSELoss ()
 deftrain_step ( tone, state, action, price,next_state, done)
 state = torch.tensor ( state, dtype = torch.float)
 = torch.tensor (next_state, dtype = torch.long)
 action = torch.tensor ( action, dtype = torch.long)
 price = torch.sensor ( price, dtype = torch.float)
 if len (state.shape) == 1
 (1, x)
 state = torch.unsqueeze ( state,)
next_state = torch.unsqueeze (next_state, 0)
 action = torch.unsqueeze ( action,)
 price = torch.unsqueeze ( price,)
 done = ( done,)
# 1 prognosticated Q values with current state
 pred = self.model ( state)
 target = pred.clone ()
 for idx in range (len ( done))
 = price (idx)
```

if not done (idx)

= price (idx)self.gamma *torch.max (self.model (next_state (idx)))

target (idx) (torch.argmax (action). item ()) = Q_new

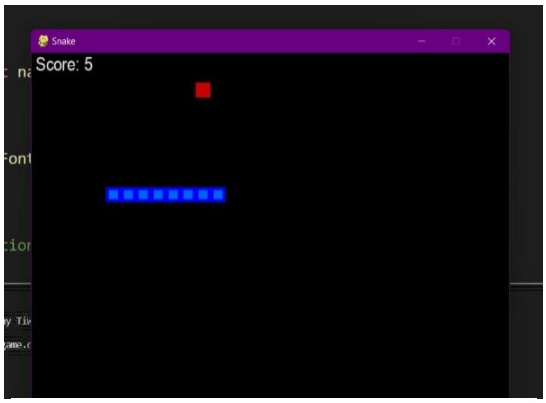
2. r y * maximum (next_predicted Q) value

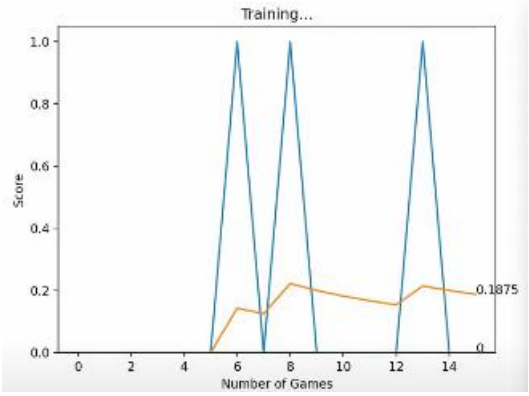
()

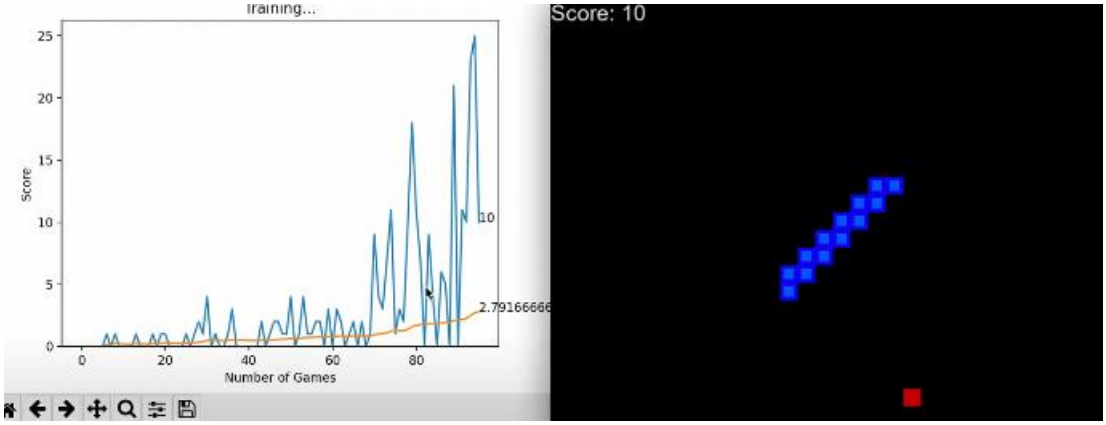
loss = self.criterion (target, pred)

loss.backward ()

()







The snake game was successfully implemented using Linked List concepts and RL where the snake is trying to improve the score by itself and the plot is also showing the same.

Snake Game output

Code link : https://github.com/Vibhmay04/Projects.git

REFERENCES

[1] Rong Zhou, Eric A. Hansen, Breadth-first heuristic search, Artificial Intelligence, Volume 170, Issues 4–5,2006, Pages 385-408, ISSN 0004-3702

[2] Kang, Xiyu, Yiqi Wang, and Yanrui Hu. "Research on different heuristics for minimax algorithm insight from connect-4 game." *Journal of Intelligent Learning Systems and Applications* 11.02 (2019): 15.

[3] Nasa, Rijul, et al. "Alpha-beta pruning in mini-max algorithm–an optimized approach for a connect-4 game." *Int. Res. J. Eng. Technol* (2018): 1637-1641.

[4] Buffet, Olivier, et al. "Optimistic heuristics for Minesweeper." *Advances in Intelligent Systems and Applications-Volume 1*. Springer, Berlin, Heidelberg, 2013. 199-207.

[5] Mossel, Elchanan. "The Minesweeper game: percolation and complexity." *Combinatorics, Probability and Computing* 11.5 (2002): 487-499.

[6] Tingting, Wu, et al. "Analysis of single-agent game: Case study using minesweeper." *2020 International Conference on Advanced Information Technologies (ICAIT)*. IEEE, 2020.

[7] Vomlelová, Marta, and Jirı Vomlel. "Applying Bayesian networks in the game of Minesweeper." *Proceedings of the Twelfth Czech-Japan Seminar on Data Analysis and Decision Making under Uncertainty*. 2009.

[8] Robles, David, and Simon M. Lucas. "A simple tree search method for playing Ms. Pac-Man." *2009 IEEE Symposium on Computational Intelligence and Games*. IEEE, 2009.

[9] DeNero, John, and Dan Klein. "Teaching introductory artificial intelligence with pac-man." *First AAAI Symposium on Educational Advances in Artificial Intelligence*. 2010.

[10] Grivokostopoulou, Foteini, Isidoros Perikos, and Ioannis Hatzilygeroudis. "An educational game for teaching search algorithms." *International Conference on Computer Supported Education*. Vol. 3. SCITEPRESS, 2016.
[11] Wirth, Nathan, and Marcus Gallagher. "An influence map model for playing Ms. Pac-Man." *2008 IEEE Symposium on Computational Intelligence and Games*. IEEE, 2008.
[12] Duro, Joao António, and José Valente de Oliveira. "Particle swarm optimization applied to the chess game." *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*. IEEE, 2008.
[13] Wolfe, David. "Snakes in domineering games." *Theoretical computer science* 119.2 (1993): 323-329.
[14] Job, Dhanya, and Varghese Paul. "Recursive backtracking for solving 9* 9 Sudoku puzzle." *Bonfring International Journal of Data Mining* 6.1 (2016): 07-09.
[15] Kamal, Snigdha, Simarpreet Singh Chawla, and Nidhi Goel. "Detection of Sudoku puzzle using image processing and solving by Backtracking, Simulated Annealing and Genetic Algorithms: A comparative analysis." *2015 third international conference on image information processing (ICIIP)*. IEEE, 2015.
[16] Maji, Arnab Kumar, and Rajat Kumar Pal. "Sudoku solver using minigrid based backtracking." *2014 IEEE International Advance Computing Conference (IACC)*. IEEE, 2014.
[17] Berggren, Patrik, and David Nilsson. "A study of Sudoku solving algorithms." *Royal Institute of Technology, Stockholm* (2012).
[18] Hasanah, Novrindah Alvi, et al. "A Comparative Study: Ant Colony Optimization Algorithm and Backtracking Algorithm for Sudoku Game." *2020 International Seminar on Application for Technology of Information and Communication (iSemantic)*. IEEE, 2020.
[19] Wei, Zhepei, et al. "Autonomous agents in Snake game via deep reinforcement learning." *2018 IEEE International Conference on Agents (ICA)*. IEEE, 2018.
[20] Sridhar, Prashanth. *Implementation of the environment of a practical work in a computational intelligence course*. BS thesis. Universitat Politècnica de Catalunya, 2019.
[21] Becerra, David J. *Algorithmic approaches to playing minesweeper*. Diss. 2015.
[22] Golan, Shahar. "Minesweeper strategy for one mine." *Applied Mathematics and Computation* 232 (2014): 292-302.
[23] Iyer, Rohit, Amrish Jhaveri, and Krutika Parab. "A review of Sudoku solving using patterns." *Int. J. Sci. Res. Publ* 3.5 (2013): 1-4.
[24] Holdsworth, Jason J. "The nature of breadth-first search." (1999).