

|  |
| --- |
| **RAJALAKSHMI INSTITUTE OF TECHNOLOGY** |
| (An Autonomous Institution, Affiliated to Anna University, Chennai) |

**DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE**

**ACADEMIC YEAR 2025 - 2026**

**SEMESTER III**

**ARTIFICIAL INTELLIGENCE LABORATORY**

**MINI PROJECT REPORT**

|  |  |
| --- | --- |
| **REGISTER NUMBER** | 2117240070048 |
| **NAME** | DEEPAK D |
| **PROJECT TITLE** | Candy crush move prediction using minmax search |
| **DATE OF SUBMISSION** |  |
| **FACULTY IN-CHARGE** | **Mrs. M. Divya** |

**Signature of Faculty In-charge**

**INTRODUCTION**

**Artificial Intelligence (AI) enables computers to perform tasks that typically require human intelligence, such as decision-making, pattern recognition, and problem-solving. Within AI, search algorithms like Minimax help systems make optimal decisions in competitive environments.**

**Background Context**

**In games like Candy Crush, players need to choose moves strategically to maximize points and create chain reactions. The challenge lies in evaluating multiple possible moves and predicting outcomes several steps ahead.**

**PROBLEM STATEMENT**

**To develop an AI-based move recommendation system for a simplified version of Candy Crush that uses Minimax Search to evaluate possible moves and recommend the most optimal one that maximizes the score.**

**GOAL**

To implement an **AI algorithm (Minimax)** capable of evaluating and recommending the next best move in a Candy Crush–like grid.

* **Expected Result:** The AI suggests the best move based on score maximization.
* **Possibilities:** The system can be expanded to handle full Candy Crush mechanics or used in other match-3 puzzle games.

**THEORETICAL BACKGROUND**

**Candy Crush is a match-3 puzzle game where the player swaps adjacent candies to form lines of three or more identical candies. The objective is to achieve the highest score by triggering cascades and combos.**

**About the Algorithm**

**The Minimax algorithm is a recursive decision-making algorithm used in two-player games. It evaluates all possible moves and assumes that the opponent will also play optimally.**

**Working Principle**

* **The AI (maximizer) tries to maximize the score.**
* **The opponent (minimizer) represents unfavorable moves or randomness that reduces the score.**
* **The algorithm explores the game tree up to a certain depth and selects the move with the best minimax value.**

**Literature Survey**

1. **Minimax has been used widely in turn-based games like Chess, Tic-Tac-Toe, and Connect Four.**
2. **Candy Crush move recommendation problems often use heuristic search or reinforcement learning.**
3. **Heuristics for Candy Crush usually depend on score increase, combo length, and cascading potential.**

**Justification**

**Minimax is chosen because it can simulate decision-making under competitive or uncertain conditions — ideal for games with multiple possible moves.**

**ALGORITHM EXPLANATION WITH EXAMPLE**

**Consider a 3×3 Candy grid with three candy types (A, B, C).  
Each possible swap is simulated. If a swap results in 3 matching candies, those candies are cleared and points are added.  
The Minimax algorithm recursively explores possible moves for 2–3 levels and returns the move with the maximum expected score.**

**IMPLEMENTATION AND CODE**

**import random**

**import copy**

**# Initialize board**

**CANDIES = ['R', 'G', 'B'] # Red, Green, Blue**

**BOARD\_SIZE = 5**

**def generate\_board():**

**return [[random.choice(CANDIES) for \_ in range(BOARD\_SIZE)] for \_ in range(BOARD\_SIZE)]**

**def print\_board(board):**

**for row in board:**

**print(' '.join(row))**

**print()**

**# Check and clear matches**

**def clear\_matches(board):**

**score = 0**

**cleared = [[False]\*BOARD\_SIZE for \_ in range(BOARD\_SIZE)]**

**# Horizontal matches**

**for i in range(BOARD\_SIZE):**

**for j in range(BOARD\_SIZE-2):**

**if board[i][j] == board[i][j+1] == board[i][j+2]:**

**cleared[i][j] = cleared[i][j+1] = cleared[i][j+2] = True**

**score += 10**

**# Vertical matches**

**for j in range(BOARD\_SIZE):**

**for i in range(BOARD\_SIZE-2):**

**if board[i][j] == board[i+1][j] == board[i+2][j]:**

**cleared[i][j] = cleared[i+1][j] = cleared[i+2][j] = True**

**score += 10**

**# Clear matched candies**

**for i in range(BOARD\_SIZE):**

**for j in range(BOARD\_SIZE):**

**if cleared[i][j]:**

**board[i][j] = random.choice(CANDIES)**

**return score**

**# Generate all possible moves**

**def get\_possible\_moves(board):**

**moves = []**

**for i in range(BOARD\_SIZE):**

**for j in range(BOARD\_SIZE):**

**if j+1 < BOARD\_SIZE:**

**moves.append(((i,j),(i,j+1))) # swap right**

**if i+1 < BOARD\_SIZE:**

**moves.append(((i,j),(i+1,j))) # swap down**

**return moves**

**def make\_move(board, move):**

**new\_board = copy.deepcopy(board)**

**(x1, y1), (x2, y2) = move**

**new\_board[x1][y1], new\_board[x2][y2] = new\_board[x2][y2], new\_board[x1][y1]**

**score = clear\_matches(new\_board)**

**return new\_board, score**

**# Minimax algorithm**

**def minimax(board, depth, is\_maximizing):**

**if depth == 0:**

**return 0**

**moves = get\_possible\_moves(board)**

**if is\_maximizing:**

**best\_score = float('-inf')**

**for move in moves:**

**new\_board, score = make\_move(board, move)**

**total\_score = score + minimax(new\_board, depth-1, False)**

**best\_score = max(best\_score, total\_score)**

**return best\_score**

**else:**

**worst\_score = float('inf')**

**for move in moves:**

**new\_board, score = make\_move(board, move)**

**total\_score = score - minimax(new\_board, depth-1, True)**

**worst\_score = min(worst\_score, total\_score)**

**return worst\_score**

**def recommend\_move(board, depth=2):**

**best\_move = None**

**best\_score = float('-inf')**

**for move in get\_possible\_moves(board):**

**new\_board, score = make\_move(board, move)**

**total\_score = score + minimax(new\_board, depth-1, False)**

**if total\_score > best\_score:**

**best\_score = total\_score**

**best\_move = move**

**return best\_move, best\_score**

**# Main program**

**board = generate\_board()**

**print("Initial Board:")**

**print\_board(board)**

**move, score = recommend\_move(board)**

**print(f"Recommended Move: Swap {move} with expected score gain {score}")**

**OUTPUT**

A black screen with white text

AI-generated content may be incorrect.

**RESULTS AND FUTURE ENHANCEMENT**

**Results**

* The Minimax algorithm successfully recommends the best move for the current grid.
* Demonstrates decision-making capability using heuristic evaluation.

**Future Enhancements**

* Integrate a GUI-based Candy Crush board.
* Add cascading mechanics (chain reactions).
* Use **Alpha-Beta Pruning** to optimize search performance.
* Extend to a full-sized game board for real-time recommendations.

|  |  |
| --- | --- |
| **Git Hub Link of the project** | **https://github.com/deepak1257d-hue/AI-mini-project** |

**REFERENCES**

 Russell, S., & Norvig, P. (2021). *Artificial Intelligence: A Modern Approach*. Pearson.

 GeeksforGeeks – *Minimax Algorithm in Artificial Intelligence*.

 Towards Data Science – *Understanding the Minimax Algorithm*.

 Candy Crush Saga Mechanics – King Games Official Documentation.

 GitHub Repository Examples of Match-3 AI Implementations.