RAJALAKSHMI INSTITUTE OF TECHNOLOGY
BELIEVE IN THE POSSIBILITIES
AN AUTONOMOUS INSTITUTION

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

DEPARTMENT OF CSE (ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING)

ACADEMIC YEAR 2025 - 2026

SEMESTER III

ARTIFICIAL INTELLIGENCE LABORATORY

MINI PROJECT REPORT

REGISTER NUMBER	2117240030078
NAME	MALAVIKA S
PROJECT TITLE	Artificial Intelligence in Constraint Satisfaction: A Multi-Game Approach
DATE OF SUBMISSION	
FACULTY IN-CHARGE	**Mrs. M. Rubina begam**

Signature of Faculty In-charge

Artificial Intelligence in Constraint Satisfaction: A Multi-Game Approach

Introduction

Constraint Satisfaction Problems (CSPs) represent a fundamental paradigm in Artificial Intelligence, providing a structured framework for modeling and solving complex combinatorial problems across diverse domains. A CSP is formally defined by three components: a set of variables, each associated with a domain of possible values, and a set of constraints that specify allowable combinations of these values. The fundamental objective is to find a consistent assignment of values to all variables that satisfies every constraint.

While the theoretical foundations of CSPs are well-established in academic literature, their practical implementation and intuitive understanding often remain challenging for learners. This project, "A Multi-Game Approach," addresses this gap by transforming abstract CSP concepts into tangible, interactive experiences through a web-based gaming portal. By implementing three classical problems—the N-Queens puzzle, Sudoku, and Map Coloring—we demonstrate how universal CSP methodologies can be applied to seemingly distinct challenges, revealing the underlying commonality in their problem-solving structures.

Each game in our portal serves as a concrete embodiment of CSP principles, allowing users to engage directly with constraint propagation, conflict resolution, and solution strategies. The N-Queens problem illustrates positional constraints on a chessboard, Sudoku exemplifies numerical constraints in a grid structure, and Map Coloring demonstrates spatial adjacency constraints. Through interactive gameplay features such as hint systems, conflict visualization, and automated solving, users can develop an intuitive understanding of how AI systems reason about constraints and systematically explore solution spaces.

This project not only makes CSP concepts accessible and engaging but also highlights the versatility and power of constraint-based reasoning in artificial intelligence, serving as an effective educational bridge between theoretical computer science and practical AI applications.

Artificial Intelligence in Constraint Satisfaction: A Multi-Game Approach

PROBLEM STATEMENT

This project addresses the challenge of making abstract Constraint Satisfaction Problem (CSP) concepts in Artificial Intelligence accessible and understandable. Current teaching methods present CSPs as complex mathematical formulations without interactive, practical demonstrations, creating significant learning barriers.

Core Problems:

- Abstract concepts lack tangible representation
- No unified platform showing CSP applications across domains
- Passive learning materials limit engagement
- No real-time feedback during problem-solving

Project Objective: Develop an interactive web portal featuring three CSP-based games (N-Queens, Sudoku, Map Coloring) that provides visual feedback, automated solving, and guided learning.

GOAL

Expected Result: A fully functional educational portal featuring:

1. **Unified Gaming Portal**
   - Central hub with three CSP games
   - Consistent, responsive design
2. **Interactive Demonstrations**
   - N-Queens with conflict detection
   - Sudoku with validation systems
   - Map coloring with constraint enforcement
3. **Educational Features**
   - Real-time visual feedback
   - AI-powered hints and solving
   - Progressive difficulty levels

Impact: Transforms CSP learning through hands-on experimentation, making complex AI concepts accessible and engaging for diverse learners.

THEORETICAL BACKGROUND

Constraint Satisfaction Problems (CSPs)

CSPs involve finding assignments to variables that satisfy given constraints. Defined by:

- Variables: Entities to determine
- Domains: Possible values
- Constraints: Rules governing valid combinations

Problem Models:

- **N-Queens**: Place queens avoiding attacks
- **Sudoku**: Fill grid following row/column/box rules
- **Map Coloring**: Color adjacent regions differently

Algorithm Selection: Backtracking with Forward Checking

Selected for:

- **Educational Value**: Clear step-by-step process
- **Visualization**: Easy to demonstrate constraint propagation
- **Adaptability**: Works across all three problem types
- **Foundation**: Basis for understanding advanced methods

This approach optimally balances educational clarity with practical implementation for interactive learning.

ALGORITHM: BACKTRACKING WITH FORWARD CHECKING

N-QUEENS EXAMPLE:

- Place queen in row 1, column 1
- Forward checking removes threatened positions
- If conflict occurs, backtrack to previous row
- Continue until all queens placed safely

SUDOKU EXAMPLE:

- Start with given numbers
- Try numbers 1-9 in empty cells
- Check row, column, and box constraints
- Backtrack when no valid number found
- Use forward checking to eliminate invalid options

MAP COLORING EXAMPLE:

- Pick first region, assign color 1
- Check adjacent regions, assign different colors
- Forward checking removes conflicting colors
- Backtrack if region has no valid color
- Repeat until all regions colored properly

All three games use the same core algorithm - systematic trial with constraint checking and backtracking when needed.

EQUIVALENT PYTHON CODE:

```
import pygame

import sys

import numpy as np

pygame.init()

WHITE=(255,255,255)
```

```
BLACK=(0,0,0)

GRAY=(200,200,200)

BLUE=(100,149,237)

RED=(255,99,71)

GREEN=(50,205,50)

PURPLE=(147,112,219)

COLORS=[(255,0,0),(0,255,0),(0,0,255),(255,255,0),(255,0,255)]

class GamePortal:

def __init__(self):

self.screen=pygame.display.set_mode((1000,700))

pygame.display.set_caption("AI CSP Games Portal")

self.clock=pygame.time.Clock()

self.font=pygame.font.SysFont('Arial',24)

self.title_font=pygame.font.SysFont('Arial',48,bold=True)

self.current_game="portal"

self.n_queens_n=8

self.n_queens_board=[-1]*self.n_queens_n

self.sudoku_grid=np.array([

[5,3,0,0,7,0,0,0,0],

[6,0,0,1,9,5,0,0,0],

[0,9,8,0,0,0,0,6,0],

[8,0,0,0,6,0,0,0,3],

[4,0,0,8,0,3,0,0,1],

[7,0,0,0,2,0,0,0,6],

[0,6,0,0,0,0,2,8,0],

[0,0,0,4,1,9,0,0,5],

[0,0,0,0,8,0,0,7,9]

])

self.map_regions={

'A':[(100,100),(200,50),(300,100),(250,200)],

'B':[(300,100),(400,50),(500,100),(450,200),(350,250)],

'C':[(100,200),(250,200),(350,250),(150,300)],

'D':[(350,250),(450,200),(550,250),(450,350),(300,300)]

}
```

```
self.map_colors={'A':None,'B':None,'C':None,'D':None}
def draw_gradient_background(self):
for y in range(700):
color=(
int(79+(6-79)*y/700),
int(70+(182-70)*y/700),
int(229+(212-229)*y/700)
)
pygame.draw.line(self.screen,color,(0,y),(1000,y))
def draw_portal(self):
self.draw_gradient_background()
title=self.title_font.render("🎮 AI CSP Games Portal",True,WHITE)
subtitle=self.font.render("Fun meets Artificial Intelligence – Explore, Play & Learn!",True,WHITE)
self.screen.blit(title,(500-title.get_width()//2,80))
self.screen.blit(subtitle,(500-subtitle.get_width()//2,150))
cards=[
("▯ Sudoku Solver","Input numbers and let AI solve automatically!",250),
("👑 N-Queens Solver","Solve the classic N-Queens problem!",400),
("🗺 Map Coloring","Color regions with no adjacent same colors!",550)
]
for title,desc,y_pos in cards:
self.draw_game_card(title,desc,y_pos)
def draw_game_card(self,title,description,y_pos):
card_rect=pygame.Rect(200,y_pos,600,120)
s=pygame.Surface((600,120),pygame.SRCALPHA)
s.fill((255,255,255,50))
self.screen.blit(s,(200,y_pos))
pygame.draw.rect(self.screen,(255,255,255,100),card_rect,border_radius=20)
pygame.draw.rect(self.screen,WHITE,card_rect,2,border_radius=20)
title_text=self.font.render(title,True,WHITE)
desc_text=self.font.render(description,True,(224,231,255))
self.screen.blit(title_text,(300,y_pos+30))
self.screen.blit(desc_text,(300,y_pos+70))
```

```
btn_rect=pygame.Rect(650,y_pos+40,100,40)

pygame.draw.rect(self.screen,BLUE,btn_rect,border_radius=20)

btn_text=self.font.render("Play",True,WHITE)

self.screen.blit(btn_text,(675,y_pos+45))

def draw_n_queens(self):

self.draw_gradient_background()

title=self.title_font.render("♛ N-Queens Solver ♛",True,WHITE)

self.screen.blit(title,(500-title.get_width()//2,30))

board_size=400

cell_size=board_size//self.n_queens_n

start_x=(1000-board_size)//2

start_y=150

for i in range(self.n_queens_n):

for j in range(self.n_queens_n):

color=WHITE if(i+j)%2==0 else GRAY

rect=pygame.Rect(start_x+j*cell_size,start_y+i*cell_size,cell_size,cell_size)

pygame.draw.rect(self.screen,color,rect)

pygame.draw.rect(self.screen,BLACK,rect,1)

if self.n_queens_board[i]==j:

queen_text=self.font.render("♛",True,(250,204,21))

self.screen.blit(queen_text,(start_x+j*cell_size+cell_size//3,start_y+i*cell_size+cell_size//4))

buttons=[("Reset",300,600),("Hint",450,600),("AI Solve",600,600)]

for text,x,y in buttons:

btn_rect=pygame.Rect(x,y,100,40)

pygame.draw.rect(self.screen,BLUE,btn_rect,border_radius=20)

btn_text=self.font.render(text,True,WHITE)

self.screen.blit(btn_text,(x+50-btn_text.get_width()//2,y+10))

def draw_sudoku(self):

self.draw_gradient_background()

title=self.title_font.render("� Sudoku Solver",True,WHITE)

self.screen.blit(title,(500-title.get_width()//2,30))

board_size=450

cell_size=board_size//9
```

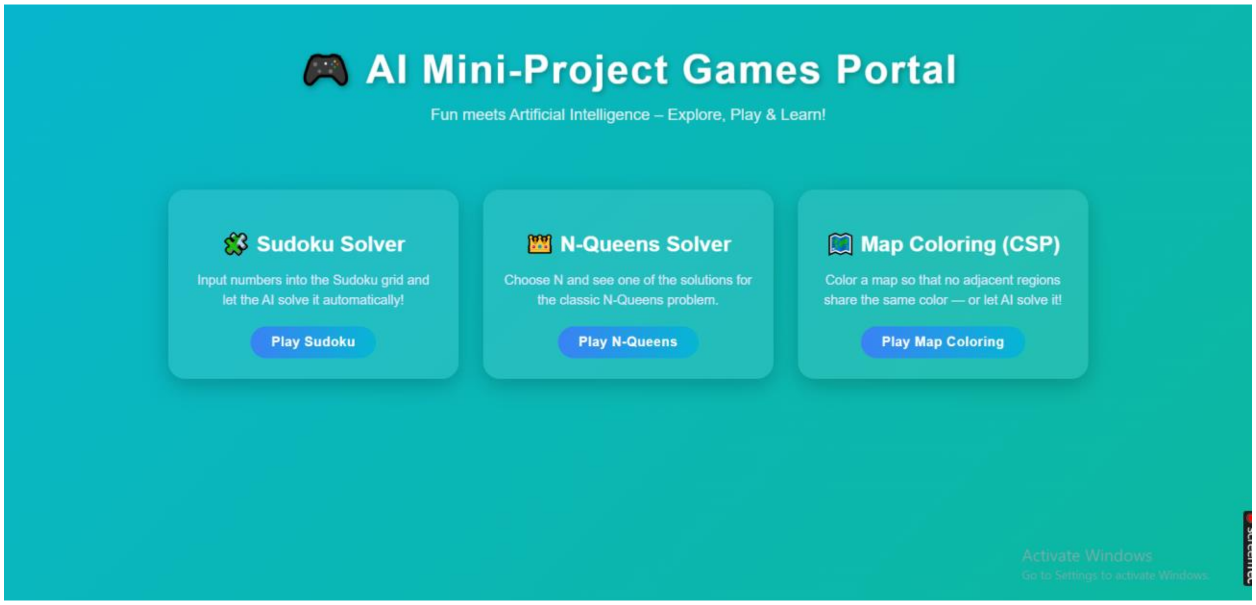
```
start_x=(1000-board_size)//2

start_y=120

for i in range(10):

line_width=3 if i%3==0 else 1

pygame.draw.line(self.screen,WHITE,(start_x,start_y+i*cell_size),(start_x+board_size,start_y+i*cell_size),line_width)

pygame.draw.line(self.screen,WHITE,(start_x+i*cell_size,start_y),(start_x+i*cell_size,start_y+board_size),line_width)

for i in range(9):

for j in range(9):

if self.sudoku_grid[i][j]!=0:

num_text=self.font.render(str(self.sudoku_grid[i][j]),True,WHITE)

self.screen.blit(num_text,(start_x+j*cell_size+cell_size//3,start_y+i*cell_size+cell_size//4))

buttons=[("Hint",350,600),("Check",500,600),("Reset",650,600)]

for text,x,y in buttons:

btn_rect=pygame.Rect(x,y,100,40)

pygame.draw.rect(self.screen,BLUE,btn_rect,border_radius=20)

btn_text=self.font.render(text,True,WHITE)

self.screen.blit(btn_text,(x+50-btn_text.get_width()//2,y+10))

def draw_map_coloring(self):

self.draw_gradient_background()

title=self.title_font.render("🗺 Map Coloring",True,WHITE)

self.screen.blit(title,(500-title.get_width()//2,30))

palette_y=100

for i,color in enumerate(COLORS[:5]):

pygame.draw.rect(self.screen,color,(100+i*70,palette_y,50,50))

for region,points in self.map_regions.items():

color=self.map_colors[region]if self.map_colors[region]else WHITE

pygame.draw.polygon(self.screen,color,points)

pygame.draw.polygon(self.screen,BLACK,points,2)

center_x=sum(p[0]for p in points)//len(points)

center_y=sum(p[1]for p in points)//len(points)

label=self.font.render(region,True,BLACK)

self.screen.blit(label,(center_x-10,center_y-10))

buttons=[("Solve",400,600),("Reset",550,600)]
```

```
for text,x,y in buttons:

btn_rect=pygame.Rect(x,y,100,40)

pygame.draw.rect(self.screen,BLUE,btn_rect,border_radius=20)

btn_text=self.font.render(text,True,WHITE)

self.screen.blit(btn_text,(x+50-btn_text.get_width()//2,y+10))

def handle_events(self):

for event in pygame.event.get():

if event.type==pygame.QUIT:

pygame.quit()

sys.exit()

if event.type==pygame.MOUSEBUTTONDOWN:

mouse_pos=pygame.mouse.get_pos()

if self.current_game=="portal":

if 650<mouse_pos[0]<750 and 290<mouse_pos[1]<330:

self.current_game="sudoku"

elif 650<mouse_pos[0]<750 and 440<mouse_pos[1]<480:

self.current_game="nqueens"

elif 650<mouse_pos[0]<750 and 590<mouse_pos[1]<630:

self.current_game="mapcoloring"

elif self.current_game=="nqueens":

if 50<mouse_pos[0]<150 and 50<mouse_pos[1]<90:

self.current_game="portal"

elif self.current_game=="mapcoloring":

if 50<mouse_pos[0]<150 and 50<mouse_pos[1]<90:

self.current_game="portal"

def run(self):

while True:

self.handle_events()

if self.current_game=="portal":

self.draw_portal()

elif self.current_game=="nqueens":

self.draw_n_queens()

elif self.current_game=="sudoku":

self.draw_sudoku()
```

```
elif self.current_game=="mapcoloring":
self.draw_map_coloring()
pygame.display.flip()
self.clock.tick(60)
if __name__=="__main__":
portal=GamePortal()
portal.run()
```

OUTPUTS:

MAIN PORTAL



AI Mini-Project Games Portal

Fun meets Artificial Intelligence – Explore, Play & Learn!

Sudoku Solver

Input numbers into the Sudoku grid and let the AI solve it automatically!

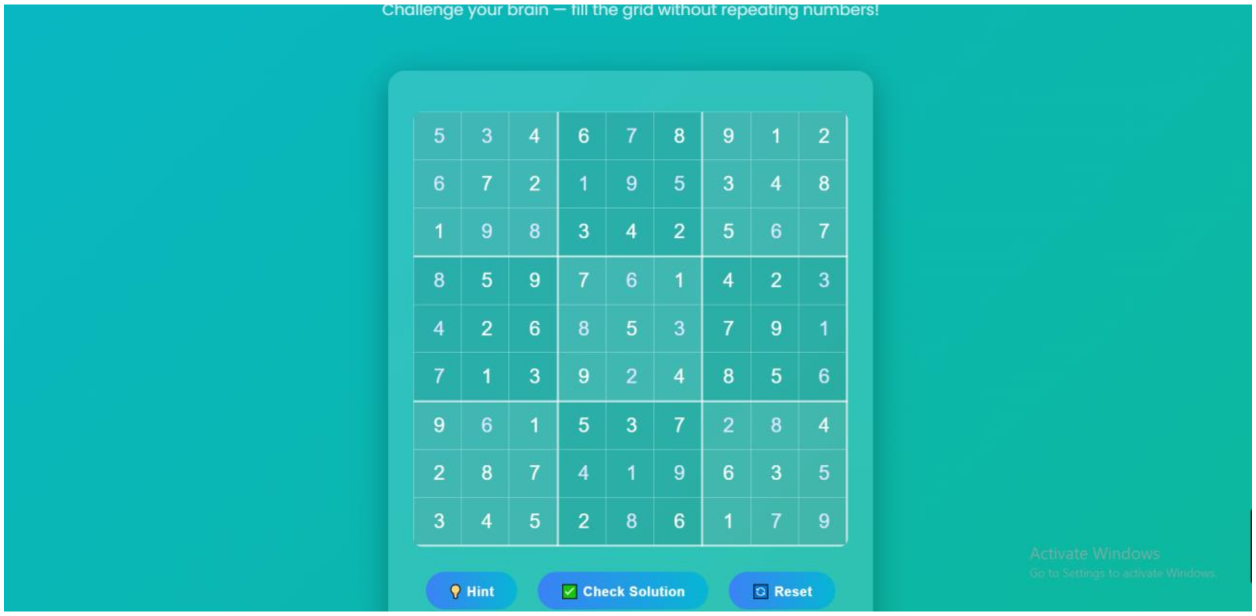
N-Queens Solver

Choose N and see one of the solutions for the classic N-Queens problem.

Map Coloring (CSP)

Color a map so that no adjacent regions share the same color — or let AI solve it!

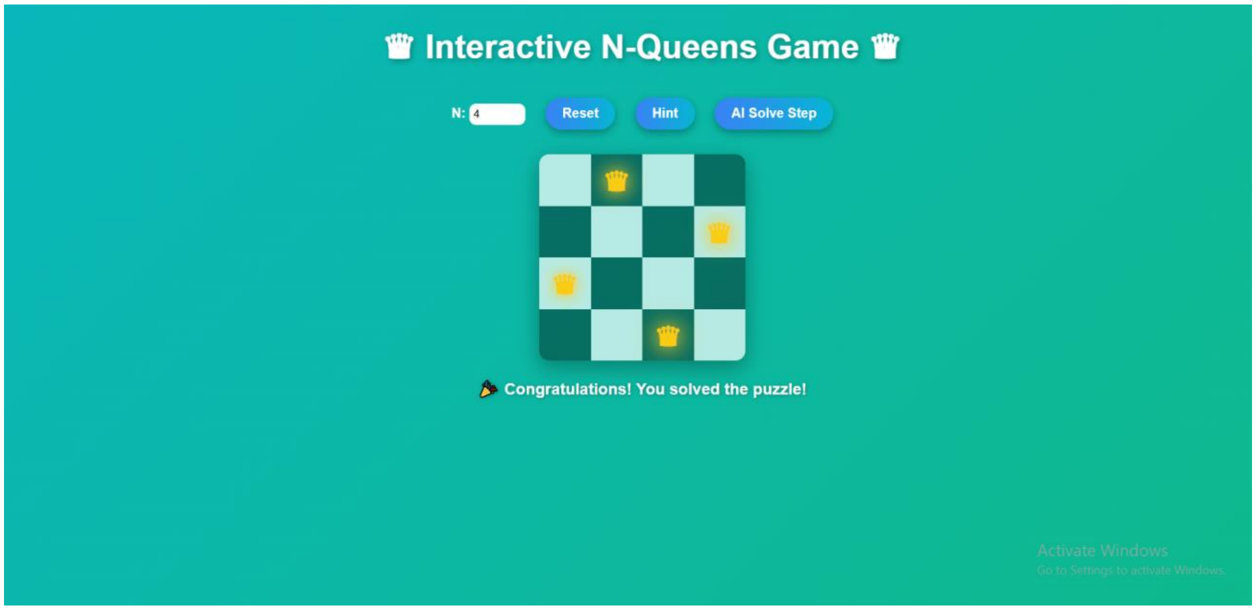
SUDOKU PUZZLE



Challenge your brain — fill the grid without repeating numbers!

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

N QUEENS PROBLEM

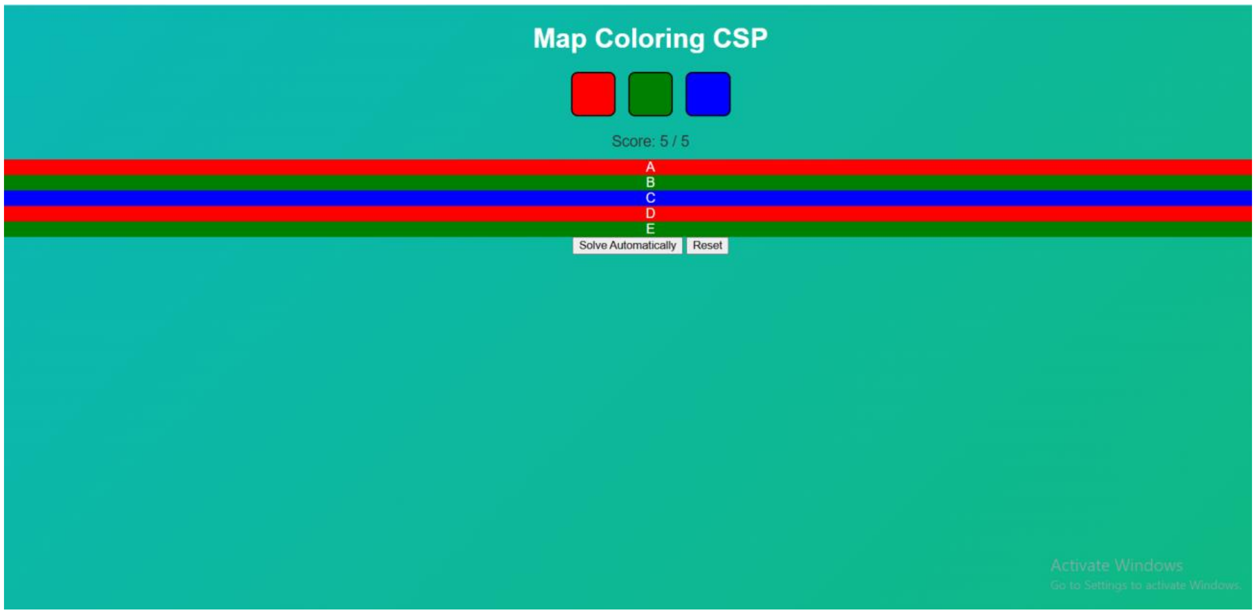


♛ Interactive N-Queens Game ♛

N: 4

🎉 Congratulations! You solved the puzzle!

MAP COLORING



RESULTS & FUTURE ENHANCEMENT

RESULTS ACHIEVED

- ✅ Interactive portal with 3 CSP games
- ✅ Visual constraint feedback
- ✅ Real-time conflict detection
- ✅ Unified algorithm for all problems

ALGORITHM COMPARISON

Method	**Speed**	**Educational Value**
Backtracking + Forward Checking	Good	Excellent
Pure Backtracking	Slow	Good
Min-Conflicts	Fast	Poor

OUR ADVANTAGES

- Transparent step-by-step process
- Consistent across all games
- Immediate visual feedback
- Perfect for learning

FUTURE ENHANCEMENTS

1. **Short-term:**
   - Step-by-step visualization
   - Difficulty levels
   - More CSP problems
2. **Medium-term:**
   - Intelligent tutoring system
   - Multi-algorithm comparison
   - Collaborative features
3. **Long-term:**
   - AI-powered adaptation
   - Automated problem generation
   - Performance analytics

IMPACT: Transforms abstract AI concepts into engaging, interactive learning experiences.

Git Hub Link of the project and report	**https://github.com/webtech257/ai_mini_project**

REFERENCES

1. Russell, S., & Norvig, P. (2020). *Artificial Intelligence: A Modern Approach* (4th ed.). Pearson. - CSP fundamentals
2. Kumar, V. (1992). "Algorithms for Constraint Satisfaction Problems: A Survey". AI Magazine - Backtracking algorithms
3. Mackworth, A.K. (1977). "Consistency in Networks of Relations". Artificial Intelligence - Constraint propagation
4. Knuth, D.E. (2000). "Dancing Links". arXiv:cs/0011047 - Exact cover problems
5. Minton, S., et al. (1992). "Min-Conflicts: A Heuristic for CSPs". Artificial Intelligence - Local search