CSIT 696: Research Methods in Computing
Project Report

Project Title: Creating a 2048 Game and Implementing an AI Player
Student Name: Thilok Reddy Anugu

1. Description

1.1 Basic Information:
The project aimed to develop a 2048 game using the Pygame library and implement an AI player capable of making intelligent moves within the game environment.

1.2 Project Objectives:
The primary objectives were:

1. Design and implement a fully functional 2048 game using Pygame.
2. Develop an AI player using the Expectimax algorithm to play the 2048 game intelligently.
3. Evaluate and compare the performance of the AI player against human players.
4. Gain practical experience in game development, AI programming, and data analysis.

1.3 Description of the Data Set:
No external dataset was required. The game board state, player moves, and AI training data were generated within the program.

2. Design of the Project

2.1 Technique Methodology:
The methodology included game development, AI implementation, evaluation, and documentation.

- Game Development: Created the 2048 game using Pygame, focusing on game logic and user interaction.
- AI Implementation: Implemented the Expectimax algorithm for the AI player.
- Evaluation: Conducted performance evaluations, comparing the AI player against human players.
- Documentation: Maintained detailed documentation of code, algorithms, and project findings.

2.2 Implementation of the Project:
Implemented the 2048 game from scratch using Pygame. Developed an AI player using the Expectimax algorithm. Key components included game mechanics, user interface, and AI algorithms.

CODE

Game

```python
import pygame
from random import randint, choice
import colors as c
import math
import copy
import numpy as np

WIDTH, HEIGHT = 400, 500
FPS = 30

pygame.init()
window = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption('2048')

```

```python
class Game:
    def __init__(self, window):
        self.window = window
        self.matrix = [[0]*4 for _ in range(4)]    # The matrix that holds the values
        self.cells = []    # Store data about tiles and text to draw on the screen
        self.score = [0,0]   # List to store the score in first index and data to draw in second position
        self.fontEngine = pygame.font.SysFont(c.SCORE_LABEL_FONT, 45)
        self.over = [False, False]   # First index stores whether the game is over. Second index stores whether game is lost or won
        self.startGame()

    def startGame(self):
        ''' Entry point for the game. Executes every time a new board is made '''

        # Adding two random tiles to the matrix
        row, col = randint(0,3), randint(0,3)
        self.matrix[row][col] = 2
        while self.matrix[row][col] != 0:
            row, col = randint(0,3), randint(0,3)
        self.matrix[row][col] = 2

        # To populate self.cells list with required data to draw
        for i in range(1,5):
            row = []
            for j in range(4):
                rect = pygame.Rect(10+j*100, 10+i*100, 80, 80)
                textRect, textSurface = None, None
                # For walrus operator fix uncomment the two lines below and comment the third line
                # x = self.matrix[i-1][j]
                # if x != 0:
                if (x:=self.matrix[i-1][j]) != 0:
                    textSurface = self.fontEngine.render(str(x), True, c.CELL_NUMBER_COLORS[x])
                    textRect = textSurface.get_rect()
                    textRect.center = rect.center
                row.append({
                    "rect": rect,
                    "textRect": textRect,
                    "textSurface": textSurface
                })
            self.cells.append(row)

        # To populate self.score with required data to draw
```

```
        # To populate self.score with required data to draw
        scoreSurface = pygame.font.SysFont(c.SCORE_LABEL_FONT, 50).render('Score : ', True, (0,0,0))
        scoreRect = scoreSurface.get_rect()
        scoreRect.top = 25
        self.score[1] = [scoreSurface, scoreRect]

    def addNewTile(self):
        ''' Adds a new tile to the matrix '''
        row, col = randint(0,3), randint(0,3)
        while self.matrix[row][col] != 0:
            row, col = randint(0,3), randint(0,3)
        self.matrix[row][col] = choice([2,2,2,2,4])

    def horMoveExists(self):
        ''' Checks whether a horizontal move exists or not '''
        for i in range(4):
            for j in range(3):
                if self.matrix[i][j+1] == self.matrix[i][j]:
                    return True
        return False

    def verMoveExists(self):
        ''' Checks whether a vertical move exists or not '''
        for i in range(3):
            for j in range(4):
                if self.matrix[i+1][j] == self.matrix[i][j]:
                    return True
        return False

    def gameOver(self):
        ''' Checks whether the game is over or not '''
        if any(2048 in row for row in self.matrix):
            self.over = [True, True]
        if not any(0 in row for row in self.matrix) and not self.horMoveExists() and not self.verMoveExists():
            self.over = [True, False]

    def updateTiles(self):
```

```
91      def updateTiles(self):
92          ''' Updates self.cells with the new data when something changes it's position on the board '''
93          for i in range(4):
94              for j in range(4):
95                  # For walrus operator fix uncomment the two lines below and comment the third line
96                  # x = self.matrix[i][j]
97                  # if x != 0:
98                  if (x:=self.matrix[i][j]) != 0:
99                      textSurface = self.fontEngine.render(str(x), True, c.CELL_NUMBER_COLORS[x])
100                     textRect = textSurface.get_rect()
101                     textRect.center = self.cells[i][j]['rect'].center
102                     self.cells[i][j]['textRect'] = textRect
103                     self.cells[i][j]['textSurface'] = textSurface
104                 elif x == 0:
105                     self.cells[i][j]['textRect'] = None
106                     self.cells[i][j]['textSurface'] = None
107
108     def stack(self):
109         ''' Stacks all the elements to the left of the matrix '''
110         new_matrix = [[0]*4 for _ in range(4)]
111         for i in range(4):
112             position = 0
113             for j in range(4):
114                 if self.matrix[i][j] != 0:
115                     new_matrix[i][position] = self.matrix[i][j]
116                     position += 1
117         self.matrix = new_matrix
118
119     def combine(self):
120         ''' Combines two elements if they are of same value into one and updates the matrix '''
121         for i in range(4):
122             for j in range(3):
123                 x = self.matrix[i][j]
124                 if x != 0 and x == self.matrix[i][j+1]:
125                     self.matrix[i][j] *= 2
126                     self.matrix[i][j+1] = 0
127                     self.score[0] += self.matrix[i][j]
128
129     def reverse(self):
```

```
129     def reverse(self):
130         ''' Mirrors the matrix. Ex. [[2,4,8,8],...] will give [[8,8,4,2],...] '''
131         new_matrix = []
132         for row in self.matrix:
133             new_matrix.append(row[::-1])
134         self.matrix = new_matrix
135
136     def transpose(self):
137         ''' Takes the transpose of matrix. Ref : https://www.geeksforgeeks.org/program-to-find-transpose-of-a-matrix/ '''
138         new_matrix = [[0]*4 for _ in range(4)]
139         for i in range(4):
140             for j in range(4):
141                 new_matrix[j][i] = self.matrix[i][j]
142         self.matrix = new_matrix
143
144     def scs(self):
145         ''' Helper function to stack, combine and stack '''
146         oldmatrix = self.matrix
147         self.stack()
148         self.combine()
149         self.stack()
150         return oldmatrix
151
152     def aug(self):
153         ''' Helper function to add new tile, updating tiles and checking whether game is over '''
154         self.addNewTile()
155         self.updateTiles()
156         self.gameOver()
157
158     def clone(self):
```

```
    def clone(self):
        """Create a deep clone of the Game object."""
        new_game = Game(self.window)

        new_game.matrix = [row[:] for row in self.matrix]
        new_game.score = self.score.copy()
        new_game.over = self.over.copy()

        new_game.cells = []
        for i in range(4):
            row = []
            for j in range(4):
                cell = self.cells[i][j]
                new_cell = {
                    "rect": pygame.Rect(cell['rect']),
                    "textRect": None if cell['textRect'] is None else pygame.Rect(cell['textRect']),
                    "textSurface": None if cell['textSurface'] is None else cell['textSurface'].copy(),
                }
                row.append(new_cell)
            new_game.cells.append(row)

        return new_game

    def left(self):
        oldmatrix = self.scs()
        if oldmatrix == self.matrix:
            return
        self.aug()

    def right(self):
        oldmatrix = self.matrix
        self.reverse()
        self.scs()
        self.reverse()
        if oldmatrix == self.matrix:
            return
        self.aug()

    def up(self):
```

```
    def up(self):
        oldmatrix = self.matrix
        self.transpose()
        self.scs()
        self.transpose()
        if oldmatrix == self.matrix:
            return
        self.aug()

    def down(self):
        oldmatrix =self.matrix
        self.transpose()
        self.reverse()
        self.scs()
        self.reverse()
        self.transpose()
        if oldmatrix == self.matrix:
            return
        self.aug()

    def reset(self):
        ''' Resets the game by calling the constructor '''
        self.__init__(self.window)

```

Used numpy matrix to represent the grid in the game

UI:

```
def draw(window, matrix, cells, score, over):
    ''' Single function to populate the board with the elements '''
    # Background and Score label
    window.fill(c.GRID_COLOR)
    window.blit(score[1][0], score[1][1])
    # Score
    scoreSurface = pygame.font.SysFont(c.SCORE_LABEL_FONT, 50).render(str(score[0]), True, (0,0,0))
    scoreRect = scoreSurface.get_rect()
    scoreRect.top = 25
    scoreRect.left = score[1][1].right + 10
    window.blit(scoreSurface, scoreRect)
    # Cells
    for i in range(4):
        for j in range(4):
            cell = cells[i][j]
            # For walrus operator fix uncomment the two lines below and comment the third line
            # x = matrix[i][j]
            # if x != 0:
            if (x:=matrix[i][j]) != 0:
                pygame.draw.rect(window, c.CELL_COLORS[x], cell['rect'])
                window.blit(cell['textSurface'], cell['textRect'])
            elif x == 0:
                pygame.draw.rect(window, c.EMPTY_CELL_COLOR, cell['rect'])
    # Game Over
    if over[0] and over[1]:
        gameOverSurface = pygame.font.SysFont(c.SCORE_LABEL_FONT, 25).render('2048 Completed. Ctrl + q to reset', True, (0,0,0))
        gameOverRect = gameOverSurface.get_rect()
        gameOverRect.center = (WIDTH//2, HEIGHT//2)
        pygame.draw.rect(window,(255,255,255),gameOverRect)
        window.blit(gameOverSurface, gameOverRect)
    if over[0] and not over[1]:
        gameOverSurface = pygame.font.SysFont(c.SCORE_LABEL_FONT, 25).render('No moves left. Ctrl + q to reset', True, (0,0,0))
        gameOverRect = gameOverSurface.get_rect()
        gameOverRect.center = (WIDTH//2, HEIGHT//2)
        pygame.draw.rect(window,(255,255,255),gameOverRect)
        window.blit(gameOverSurface, gameOverRect)

    pygame.display.update()

```

AI:

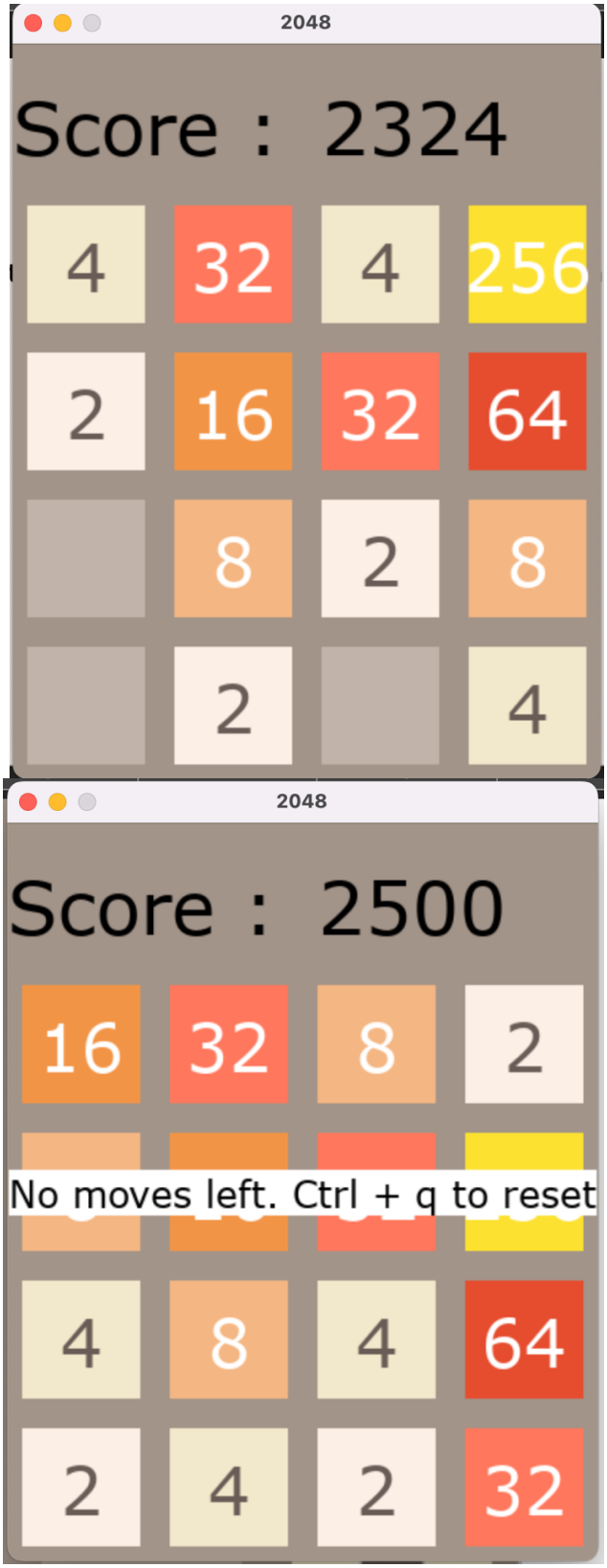
```
class AI:
    def __init__(self, game):
        self.game = game

    def get_move(self):
        moves = ["left", "right", "up", "down"]
        scores = []

        for move in moves:
            cloned_game = self.game.clone()
            getattr(cloned_game, move)()
            score = self.expectimax(cloned_game, depth=2, is_maximizing=False)
            scores.append(score)

        sorted_moves = [move for _, move in sorted(zip(scores, moves), reverse=True)]

        for move in sorted_moves:
            if self.is_valid_move(move):
                return move

        return choice(moves)

    def is_valid_move(self, move):
        cloned_game = self.game.clone()
        getattr(cloned_game, move)()
        return cloned_game.matrix != self.game.matrix
```

```python
    def expectimax(self, game, depth, is_maximizing):
        if depth == 0 or game.over[0]:
            return self.evaluate_board(game)

        if is_maximizing:
            max_score = float('-inf')
            for move in ["left", "right", "up", "down"]:
                cloned_game = self.game.clone()
                getattr(cloned_game, move)()
                score = self.expectimax(cloned_game, depth - 1, False)
                max_score = max(max_score, score)
               # print(f"{move}: {score}")
            return max_score
        else:
            empty_cells = [(i, j) for i in range(4) for j in range(4) if game.matrix[i][j] == 0]
            total_score = 0

            for i, j in empty_cells:
                cloned_game_2 = self.game.clone()
                cloned_game_4 = self.game.clone()

                cloned_game_2.matrix[i][j] = 2
                cloned_game_4.matrix[i][j] = 4

                total_score += 0.9 * self.expectimax(cloned_game_2, depth - 1, True)
                total_score += 0.1 * self.expectimax(cloned_game_4, depth - 1, True)

            if not empty_cells:
                return total_score

            return total_score / len(empty_cells)


    def evaluate_board(self, game):
        return game.score[0] + np.max(game.matrix)
```

AI Implementation:

```
326  def main():
327      ''' Main entry point for the program '''
328      running = True
329      clock = pygame.time.Clock()
330      game = Game(window)
331      ai = AI(game)
332
333      while running:
334          clock.tick(FPS)
335
336          draw(window, game.matrix, game.cells, game.score, game.over)
337
338          for event in pygame.event.get():
339              if event.type == pygame.QUIT:
340                  running = False
341                  break
342
343              if event.type == pygame.KEYUP:
344                  if event.key == pygame.K_LEFT:
345                      game.left()
346                  if event.key == pygame.K_RIGHT:
347                      game.right()
348                  if event.key == pygame.K_UP:
349                      game.up()
350                  if event.key == pygame.K_DOWN:
351                      game.down()
352                  if event.key == pygame.K_q and pygame.key.get_mods() & pygame.KMOD_CTRL and game.over:
353                      game.reset()
354
355          ai_move = ai.get_move()
356          getattr(game,ai_move)()
357
358
359      pygame.quit()
360      quit()
361
362  if __name__ == "__main__":
363      main()
364
```

UI:


2048
Score : 2324
4 32 4 256
2 16 32 64
8 2 8
2 4
2048
Score : 2500
16 32 8 2
No moves left. Ctrl + q to reset
4 8 4 64
2 4 2 32

2.3 Evaluation of the Project:
Compared AI player performance against human players, considering factors such as high scores and win rates.

3. Future Roadmap

As the initial phase of the project has been successfully completed, paving the way for a functional 2048 game and an AI player, the future roadmap will focus on further enhancements, optimizations, and potential extensions. The key areas for future development are outlined below:

3.1. Algorithm Optimization:
- Conduct a comprehensive analysis of the Expectimax algorithm's performance.
- Explore advanced AI techniques, such as neural networks, to potentially improve decision-making.
- Optimize algorithm parameters to strike a balance between exploration and exploitation.

3.2. Enhanced User Interface:
- Improve the user interface by adding more features, animations, and visual feedback.
- Implement a scoring system that provides detailed insights into the AI player's performance.
- Allow users to customize game settings and difficulty levels.

3.3. Machine Learning Integration:
- Investigate the integration of machine learning techniques for the AI player's training.
- Explore reinforcement learning approaches to enable the AI player to adapt and learn from gameplay experiences.

3.4. User Experience Research:
- Conduct user experience research to gather feedback on the game's design, difficulty, and overall enjoyment.
- Use feedback to make iterative improvements to both the game and the AI player.

4. Conclusion

Based on the test results, the following observations were made:
- The Expectimax algorithm effectively guided the AI player in making strategic moves within the 2048 game.
- Thorough testing and iterative refinement led to an AI player capable of competitive and strategically adept gameplay.

- The project achieved its objectives of developing a functional 2048 game and implementing an AI player capable of intelligent gameplay.
- The AI though failed to win the game showed significant performance and strategy in it’s gameplay

This project provided valuable insights into game development, AI algorithms, and the iterative process of optimization. The documentation ensures future reference and knowledge sharing. The successful implementation of the 2048 game and AI player demonstrates the application of research methods in computing to solve real-world problems.