TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TPHCM
KHOA CÔNG NGHỆ THÔNG TIN

MÔN ĐỒ ÁN CÔNG NGHỆ THÔNG TIN

Đề Tài :

XÂY DỰNG GAME PUZZLE

BÁO CÁO ĐỒ ÁN

GIẢNG VIÊN HƯỚNG DẪN:

Ths.NGUYỄN MINH ĐẠO

LỚP PROJ215879_23_1_01CLC
THỰC HIỆN BỞI:
- SV: NGUYỄN THÀNH LỢI
- MSSV: 21110234

HỌC KỲ 1, NH: 2023-2024

ĐIỂM SỐ

Tiêu chí	Nội dung	Trình bày	Tổng
Điểm			

Nhận xét của giảng viên:

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

MỤC LỤC

A. GIỚI THIỆU ĐỒ ÁN

1. Giới thiệu đề tài "Xây dựng game Puzzle"

Đề tài "Xây dựng game Puzzle" là một chủ đề thú vị trong lĩnh vực phát triển phần mềm và game. Game Puzzle là một loại trò chơi tinh thần, thường đòi hỏi người chơi phải suy nghĩ logic, tìm kiếm giải pháp và sắp xếp các mảnh ghép để hoàn thành một bức tranh, một mô hình hoặc một nhiệm vụ cụ thể.

2. Luật chơi game Puzzle

- Mục tiêu: Cần sắp xếp các mảnh ghép sao cho chúng tạo thành hình ảnh hoàn chỉnh.
- Cách chơi:

+ Bắt đầu với mảnh puzzle xáo trộn.
+ Bạn chỉ có thể di chuyển một mảnh puzzle vào ô trống một cách ngang hoặc dọc bằng cách đẩy mảnh puzzle cùng với ô trống.
+ Cố gắng di chuyển các mảnh puzzle để hoàn thành hình ảnh ban đầu.

3. Mô tả đề tài

Game được xây dựng trên mô hình Winform C# với 2 chế độ chơi:

- Chế độ người chơi:

+ Người chơi xáo trộn các mảnh hình và xếp lại như hình mẫu bằng cách click vào các mảnh hình giáp với ô trống cho nó di chuyển vào ô trống.
+ Người chơi chiến thắng khi các mảnh hình đặt đúng vị trí của nó
+ Có tính thời gian và số bước đi của người chơi

- Chế độ máy chơi:

+ Khi người chơi không thể giải tiếp (mệt mỏi hoặc không thấy hướng giải) thì có thể sử dụng chế độ máy chơi để tìm lời giải
+ Ở chế độ này khi kết game sẽ vẫn hiển thị người chơi thắng nhưng có sử dụng tool
+ Máy chơi sẽ sử dụng các thuật toán: Going back (sử dụng stack để quay lui), BFS (giải bằng duyệt theo chiều rộng), DFS (giải bằng duyệt theo chiều sâu), IDS (giải bằng duyệt theo chiều sâu tăng dần với d=5)

Ngoài ra, Game còn triển khai trên mô hình Tkinter Python với 2 chế độ cũng như trên nhưng chỉ chơi trên puzzle 3x3

B. NỘI DUNG ĐỒ ÁN VỚI WINFORM C#

I. Chức năng của ứng dụng

1. Chức năng chung

1.1. Chức năng hiển thị ảnh

Chức năng này giúp game load ảnh một cách ngẫu nhiên lên form, ảnh sẽ được hiện lên 2 vị trí:

- Vị trí tại nơi hiển thị ảnh gốc giúp người chơi có thể so sánh ảnh mình đang ghép với ảnh gốc xem coi đã hoàn thành hay chưa
- Vị trí tại nơi người chơi sẽ xếp hình, tấm hình gốc sẽ bị chia thành 16 phần bằng nhau và gắn lên các button để người sử dụng có thể chơi

1.2. Chức năng xáo trộn các mảng hình

Chức năng này có thể giúp người sử dụng có thể xáo trộn ảnh một cách lộn xộn theo khung người chơi muốn chơi (3x3 hoặc 4x4)

1.3. Chức năng hiển thị số bước, thời gian chơi game

Chức năng này sẽ hiện thị lên số bước và thời gian người chơi chơi game (thời gian chỉ chính xác khi không sử dụng lời giải)

1.4. Chức năng Start Game

Chức năng này giúp cho người chơi có thể bắt đầu game và số bước và thời gian sẽ bắt đầu tại thời điểm Start Game

1.5. Chức năng End Game

Chức năng này giúp cho người chơi có thể kết thúc game và cho ra kết qua sau cùng. Tại thời điểm End Game, thời gian và số bước sẽ dừng lại hiển thị lên kết quả. Kết quả sẽ bao gồm: thời gian, số bước, số ô đặt đúng chỗ, có sử dụng lời giải hay không,...

2. Chức năng dành cho player chơi

2.1. Chức năng chơi game

Chức năng này giúp người chơi có thể chơi game trên form thông qua giao diện trò chơi, các event được lập trình trên các button mảng ảnh

2.2. Chức năng quay lại trạng thái trước

Chức năng này giúp cho người chơi có thể quay lại trạng thái trước đó khi đi sai lầm (tương tự như Undo)

3. Chức năng dành cho máy chơi (Tìm lời giải)

3.1. Chức năng tìm lời giải bằng Going Back

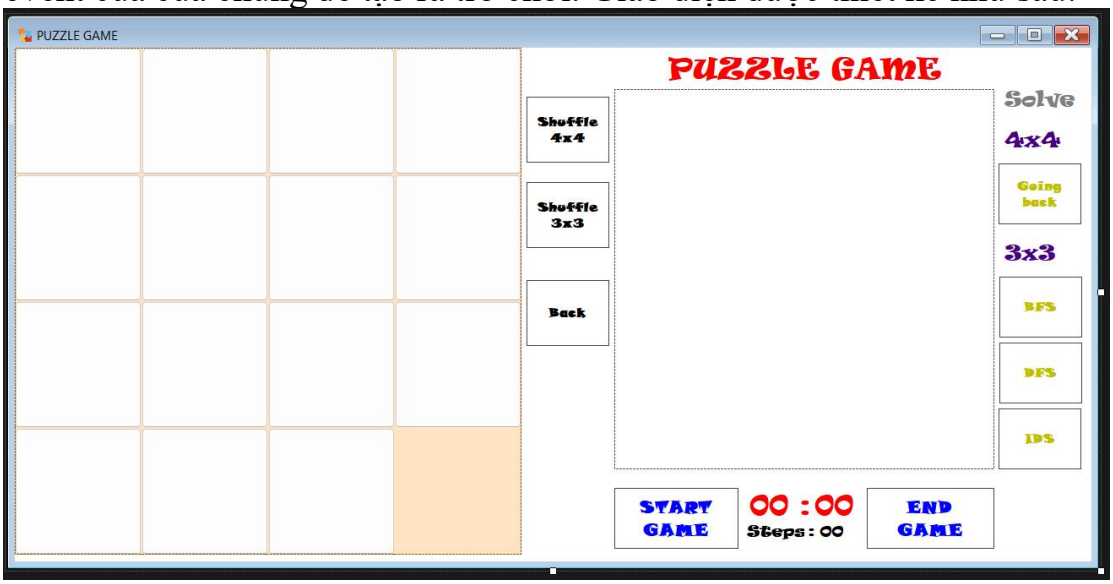
Chức năng này sẽ thực hiện quay ngược lại tất cả bước đi để tìm được lời giải. Chức năng này có thể áp dụng cho cả 3x3 và 4x4

3.2. Chức năng tìm lời giải bằng giải thuật BFS, DFS, IDS

Chức năng này sẽ thực hiện tìm ra lời giải cho bài toán qua các giải thuật tìm kiếm BFS, DFS, IDS. Chức này chỉ áp dụng cho mô hình 3x3

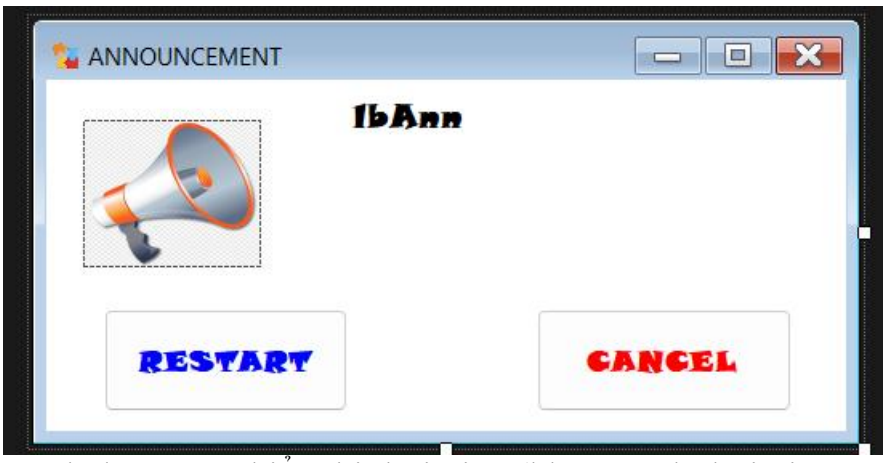
II. Thiết kế giao diện

Giao diện được thiết kế trên mô hình Winform sử dụng các Control cơ bản và event của của chung để tạo ra trò chơi. Giao diện dược thiết kế như sau:



- Panel ở nửa trái chứa 16 ô button, đây là khung vực trò chơi, người chơi có thể chơi bên khung vực này
- Các button ở nửa phải:

+ Shuffle 4x4 : Xáo trộn toàn bộ bức ảnh trong khung 4x4
+ Shuffle 3x3 : Chỉ xáo trộn phần 3x3 ở phần dưới bên phải của bức ảnh 4x4
+ Back : Quay lại bước trước đó
+ Going back : Giải puzzle bằng stack , có thể áp dụng cho giải cả 3x3 và 4x4
+ BFS : Giải bằng cách duyệt theo chiều rông, chỉ áp dụng giải 3x3
+ DFS : Giải bằng cách duyệt theo chiều sâu, chỉ áp dụng giải 3x3
+ IDS : Giải bằng cách duyệt theo chiều sau tăng dần với d = 5, chỉ áp dụng giải 3x3
+ START GAME : Bắt đầu game, các button trong panel sẽ có thể click được và thời gian, số bước di chuyển sẽ bắt đầu được tính
+ END GAME : Game sẽ được kết thúc và hiện lên kết quả, click restart để chơi lại hoặc click cancel để thoát khỏi chương trình



- Label “00:00” hiển thị thời gian (không tính thời gian máy chơi) và Label “Steps : 00” hiển thị số bước đi chuyển kể từ khi bấm Start (tính luôn chế độ máy chơi)
- PictureBox ở trung tâm bên nủa phải hiển thị hình ảnh gốc để xếp lại cho giống bức ảnh này.

III. Các lớp đối tượng sử dụng

1. Class PuzzleState

Lớp này sử dụng để lưu trạng thái của puzzle dưới dạng ma trận (mảng 2 chiều) và các thuộc tính của nó như là Move (độ sâu), path (lưu trữ đường đi đã đi qua)

```
class PuzzleState
{
    private int[][] puzzle;
    private int move = 0;
    private List<PuzzleState> path;

    public int[][] Puzzle { get => puzzle; set => puzzle = value; }
    public int Move { get => move; set => move = value; }
    internal List<PuzzleState> Path { get => path; set => path = value; }

    public PuzzleState(int[][] puzzle)
    {
        this.puzzle = puzzle;
        path = new List<PuzzleState>();
    }
    public PuzzleState(int[][] puzzle, int move)
    {
        this.puzzle = puzzle;
        path = new List<PuzzleState>();
        this.move = move;
    }
}
```

2. Class PuzzleSlove

Lớp này sử dụng để tìm lời giải cho Puzzle 3x3, trong lớp này có thuật toán lời giải BFS, DFS, IDS

- Các phương thức hỗ trợ hỗ trợ

+ Chuyển thành trạng thái puzzle về dạng string

```
    private string GetStringRepresentation(int[][] state)
    {
        string result = "";
        foreach (var row in state)
        {
            foreach (var item in row)
            {
```

```
                result += item + ",";
            }
        }
        return result.TrimEnd(',');
    }
```

+ Kiểm tra có đạt trạng thái đích chưa

```
    public bool IsGoalState(int[][] state)
    {
        int[][] goalState = new int[][] { new int[] { 1, 5, 9, 13 }, new int[]
{ 2, 6, 10, 14 }, new int[] { 3, 7, 11, 15 }, new int[] { 4, 8, 12, 0 } };

        for (int i = 0; i < 4; i++)
        {
            for (int j = 0; j < 4; j++)
            {
                if (state[i][j] != goalState[i][j])
                {
                    return false;
                }
            }
        }
```

+ Tìm các trạng thái kế tiếp

```
    public List<PuzzleState> GetNextStates(PuzzleState state)
    {
        List<PuzzleState> nextStates = new List<PuzzleState>();

        // Tìm vị trí của số 0 (ô trống)
        int zeroRow = -1;
        int zeroCol = -1;
        for (int i = 0; i < 4; i++)
        {
            for (int j = 0; j < 4; j++)
            {
                if (state.Puzzle[i][j] == 0)
                {
                    zeroRow = i;
                    zeroCol = j;
                    break;
                }
            }
            if (zeroRow != -1 && zeroCol != -1)
                break;
        }

        // Các hướng di chuyển: (dòng, cột)
        int[][] directions = { new int[] { 0, 1 }, new int[] { 0, -1 }, new int[]
{ 1, 0 }, new int[] { -1, 0 } };

        // Tạo các trạng thái mới bằng cách di chuyển số 0 đến các hướng có thể
        foreach (var direction in directions)
        {
            int newRow = zeroRow + direction[0];
            int newCol = zeroCol + direction[1];

            if (newRow > 0 && newRow < 4 && newCol > 0 && newCol < 4)
            {
                // Tạo một trạng thái mới bằng cách đổi chỗ số 0 và số kế bên
                int[][] newState = new int[4][];
                for (int i = 0; i < 4; i++)
                {
                    newState[i] = new int[4];
                    for (int j = 0; j < 4; j++)
                    {
                        newState[i][j] = state.Puzzle[i][j];
                    }
```

```
                    }

                    int temp = newState[zeroRow][zeroCol];
                    newState[zeroRow][zeroCol] = newState[newRow][newCol];
                    newState[newRow][newCol] = temp;
                    nextStates.Add(new PuzzleState(newState, state.Move + 1));
                }
            }

            return nextStates;
        }
```

- Phương thức sử dụng để giải puzzzle bằng thuật toán BFS

```
        public PuzzleState BFS_Puzzle(PuzzleState initialState)
        {
            HashSet<string> visited = new HashSet<string>();
            Queue<PuzzleState> queue = new Queue<PuzzleState>();
            string initialStateString = GetStringRepresentation(initialState.Puzzle);

            queue.Enqueue(initialState);
            visited.Add(initialStateString);

            while (queue.Count > 0)
            {
                PuzzleState current = queue.Dequeue();

                if (IsGoalState(current.Puzzle))
                {
                    return current;
                }

                List<PuzzleState> nextStates = GetNextStates(current);

                foreach (PuzzleState nextState in nextStates)
                {
                    string nextStateString =
GetStringRepresentation(nextState.Puzzle);

                    if (!visited.Contains(nextStateString))
                    {
                        visited.Add(nextStateString);
                        nextState.Path = current.Path.ToList();
                        nextState.Path.Add(nextState);
                        queue.Enqueue(nextState);
                    }
                }
            }

            return null;
        }
```

- Phương thức sử dụng để giải puzzzle bằng thuật toán DFS

```
        public PuzzleState DFS_Puzzle(PuzzleState initialState)
        {
            int MAX_DEPTH = 1000;
            HashSet<string> visited = new HashSet<string>();
            Stack<PuzzleState> stack = new Stack<PuzzleState>();
            string initialStateString = GetStringRepresentation(initialState.Puzzle);

            stack.Push(initialState);
            visited.Add(initialStateString);

            while (stack.Count > 0)
            {
```

```
                PuzzleState current = stack.Pop();

                if (IsGoalState(current.Puzzle))
                {
                    return current;
                }

                if (current.Move >= MAX_DEPTH)
                    continue;

                List<PuzzleState> nextStates = GetNextStates(current);

                foreach (PuzzleState nextState in nextStates)
                {
                    string nextStateString =
GetStringRepresentation(nextState.Puzzle);

                    if (!visited.Contains(nextStateString))
                    {
                        visited.Add(nextStateString);
                        nextState.Path = current.Path.ToList();
                        nextState.Path.Add(nextState);
                        stack.Push(nextState);
                    }
                }
            }

            return null;
        }
```

- Phương thức sử dụng để giải puzzzle bằng thuật toán IDS với d = 5

```
        public PuzzleState IDS_Puzzle(PuzzleState initialState, int depth_limit)
        {
            HashSet<string> visited = new HashSet<string>();
            Stack<PuzzleState> state_eq_limit = new Stack<PuzzleState>();
            string initialStateString = GetStringRepresentation(initialState.Puzzle);

            state_eq_limit.Push(initialState);
            visited.Add(initialStateString);
            while (state_eq_limit.Count > 0)
            {
                Stack<PuzzleState> stack = new Stack<PuzzleState>(new
Stack<PuzzleState>(state_eq_limit));
                state_eq_limit = new Stack<PuzzleState>();

                while (stack.Count > 0)
                {
                    PuzzleState current = stack.Pop();

                    if (IsGoalState(current.Puzzle))
                    {
                        return current;
                    }

                    List<PuzzleState> nextStates = GetNextStates(current);

                    foreach (PuzzleState nextState in nextStates)
                    {
                        string nextStateString =
GetStringRepresentation(nextState.Puzzle);

                        if (!visited.Contains(nextStateString) && nextState.Move <=
depth_limit)
                        {
                            visited.Add(nextStateString);
```

```
                    nextState.Path = current.Path.ToList();
                    nextState.Path.Add(nextState);
                    stack.Push(nextState);
                    if(nextState.Move == depth_limit)
                        state_eq_limit.Push(nextState);
                }
            }
        }
        depth_limit = depth_limit + 5;
    }

    return null;
}
```

3. Class ImageProcess

Lớp này sử dụng để hỗ trợ việc sử lý các event trên form như đổ hình lên, cắt hình, tính toán di chuyển hợp lý, kiểm tra win, ...

- Thuộc tính biến stack được khai báo và sử dụng để lưu các bước đã đi qua trong quá trình chơi để back hoặc giải bằng cách Going Back

```
Stack<Point> stack = new Stack<Point>();

public Stack<Point> Stack { get { return stack; } }
```

- Phương thức để đẩy 1 điểm vào Stack

```
public void pushStack(Point p)
{
    stack.Push(p);
}
```

- Phương thức sử dụng cho việc thêm hình vào các button và xáo trộn (nếu cần); trả về vị trí trống

```
public Point ImageStart(Image original, ArrayList images, Panel pan, Point
entity, int num_loop, bool puzzle3x3)
{

    cropImageTomages(images, original, 600, 600);

    return AddImagesToButtons(images, pan, entity, num_loop, puzzle3x3);

}
```

- Phương thức cắt bức ảnh gốc (orginal) thành những mảng hình (images)

```
private void cropImageTomages(ArrayList images, Image orginal, int w, int h)
{
    Bitmap bmp = new Bitmap(w, h);

    Graphics g = Graphics.FromImage(bmp);

    g.DrawImage(orginal, 0, 0, w, h);

    g.Dispose();

    int movd = 0, movr = 0;

    for (int i = 0; i < 15; i++)
    {
        Bitmap piece = new Bitmap(150, 150);
```

```
                for (int j = 0; j < 150; j++)
                    for (int k = 0; k < 150; k++)
                        piece.SetPixel(j, k,
                            bmp.GetPixel(j + movr, k + movd));

                images.Add(piece);

                movd += 150;

                if (movd == 600)
                {
                    movd = 0;
                    movr += 150;
                }
            }
        }
```

- Phương thức thêm những mảnh hình vào các button trong panel, trả về vị trí trống

```
        private Point AddImagesToButtons(ArrayList images, Panel panel, Point s, int
num_loop, bool puzzle3x3)
        {
            int i = 14;
            int[] arr = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14 };

            foreach (Button b in panel.Controls)
            {
                if (i >= 0 && arr[i] < images.Count)
                {
                    b.Image = (Image)images[arr[i]];
                    i--;
                }
            }

            return Shuffle(num_loop, s, panel, puzzle3x3);
        }
```

- Phương thức sử dụng để xáo trộn bức hình trên khung trên khung tương ứng (4x4 nếu puzzle3x3 false và ngược lại nếu puzzle4x4 true), trả về vị trí trống

```
        private Point Shuffle(int num_loop, Point s, Panel panel, bool puzzle3x3)
        {
            for (int i = 0; i < num_loop; i++)
            {
                List<Point> list = new List<Point>();

                if (puzzle3x3 == false)
                {
                    if (s.Y - 150 >= 0)
                        list.Add(new Point(s.X, s.Y - 150));
                    if (s.Y + 150 < 600)
                        list.Add(new Point(s.X, s.Y + 150));
                    if (s.X - 150 >= 0)
                        list.Add(new Point(s.X - 150, s.Y));
                    if (s.X + 150 < 600)
                        list.Add(new Point(s.X + 150, s.Y));
                } else
                {
                    if (s.Y - 150 > 0)
                        list.Add(new Point(s.X, s.Y - 150));
                    if (s.Y + 150 < 600)
                        list.Add(new Point(s.X, s.Y + 150));
                    if (s.X - 150 > 0)
                        list.Add(new Point(s.X - 150, s.Y));
```

```
                if (s.X + 150 < 600)
                    list.Add(new Point(s.X + 150, s.Y));
            }

            Random random = new Random();
            int randomIndex = random.Next(0, list.Count);

            Point randomElement = list[randomIndex];
            SimulateButtonClick(randomElement, panel);

            s = randomElement;
        }
        return s;
    }
```

- Phương thức tìm và click vào button trong panel, trả về true (tìm thấy) và false(không tìm thấy)

```
    public bool SimulateButtonClick(Point buttonLocation, Panel panel)
    {
        foreach (Button b in panel.Controls)
        {
            if (b.Location == buttonLocation)
            {
                b.PerformClick();
                return true;
            }
        }
        return false;
    }
```

- Phương thức kiểm tra bức hình có đạt đúng chỗ hay chưa (check win), trả về số mảng ảnh đặt đúng chỗ

```
    public int CheckWin(ArrayList images, Panel panel)
    {
        int count = 0;
        try
        {
            foreach (Button b in panel.Controls)
            {
                if (!b.Enabled) return 0;
                int i = (b.Location.X / 150) * 4 + b.Location.Y / 150;
                if (b.Location.X == 450 && b.Location.Y == 450)
                    continue;
                else if (b.Image == (Image)images[i])
                    count++;
            }
        }
        catch (Exception ex)
        {
            MessageBoxGame messBox = new MessageBoxGame(ex.ToString());

            messBox.ShowDialog();
            return -1;
        }
        return count;
    }
```

- Phương thức sử dụng để giải bằng cách Going Back - quay lui; trả về vị trí trống

```
    public Point sweep(Point s, ArrayList images, Panel panel)
    {
        while (CheckWin(images, panel) != 15 && stack.Count >0)
        {
```

```
            s = stack.Pop();
            SimulateButtonClick(s, panel);
            Thread.Sleep(50);
        }
        return s;
    }
```

IV. Class FGame : Form

1. Các thuộc tính sử dụng

Các biến thuộc tính này lưu giữ những biến sử dụng trong suốt trò chơi

```
    Point EntityPoint;
    ArrayList images;
    string linkImg;
    ImageProcess imgProc = new ImageProcess();
    int sec;
    int min;
    int steps = 0;
    bool checkTool = false;
    int[][] puzzle = new int[4][];
    int puzz_n = 4;
    PuzzleSovle sovle = new PuzzleSovle();
```

2. Khởi tạo

Game khởi tạo ban đầu gắn vị trí trống tại gốc dưới bên phải

```
    public FGame()
    {
        EntityPoint.X = 450;
        EntityPoint.Y = 450;
        InitializeComponent();
    }
```

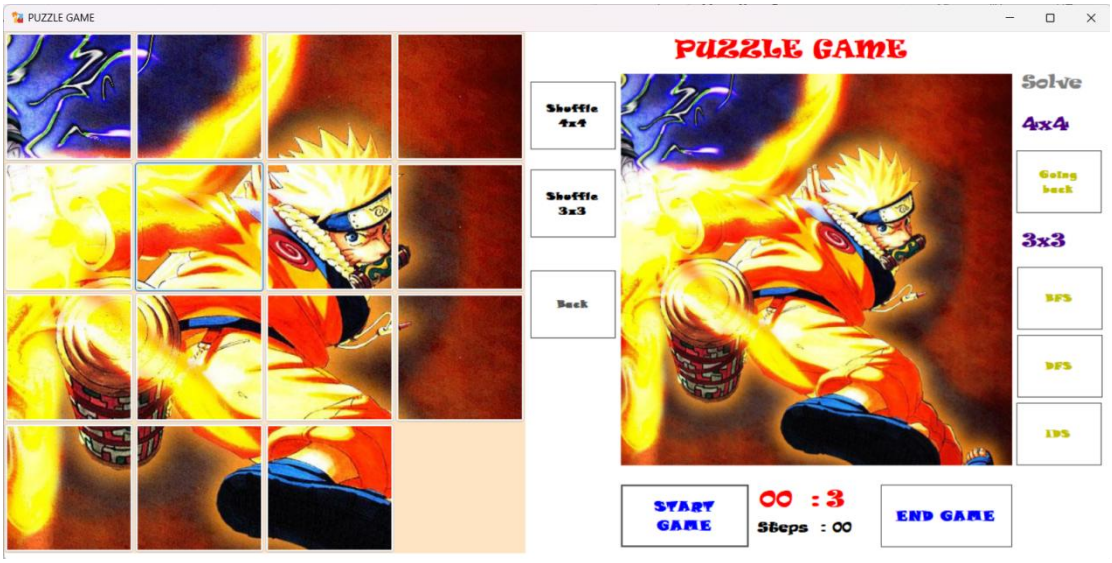
Load game khi mới chạy chương trình, đồng load hình logo (làm đẹp giao diện ban đầu)

```
    private void FGame_Load(object sender, EventArgs e)
    {
        pBOriginal.Image = Image.FromFile(Path.Combine(Application.StartupPath,
"images", "logo.jpg"));
        int x = 0;
        for (int y = 0; y < puzz_n; y++)
        {
            puzzle[y] = new int[puzz_n];
        }
        for (int i = 0; i < puzz_n; i++)
        {
            for (int j = 0; j < puzz_n; j++)
            {
                puzzle[j][i] = x;
                x++;
            }
        }
    }
```

3. Event btSGame_Click

Sự kiện này thực thi khi ta click vào Start Game để bắt đầu trò chơi

```
        private void btSGame_Click(object sender, EventArgs e)
        {
            checkTool = false;
            sec = 0;
            min = 0;
            steps = 0;
            lbSteps.Text = "00";
            btSolve.Enabled = true;
            btSolveBfs.Enabled = true;
            btSolveDfs.Enabled = true;
            btSolveIDS.Enabled = true;
            btShuffle.Enabled = true;
            btShuffle3x3.Enabled = true;
            imgProc.Stack.Clear();

            images = new ArrayList();

            foreach (Button b in panel.Controls)
                b.Enabled = true;

            Random rnd = new Random();
            int num = (rnd.Next()) % 5;
            linkImg = num.ToString() + ".jpg";

            pBOriginal.Image = Image.FromFile(Path.Combine(Application.StartupPath,
"images", linkImg));

            EntityPoint = imgProc.ImageStart(pBOriginal.Image, images, panel,
EntityPoint, 0, false);
            foreach (Button b in panel.Controls)
            {
                if (!b.Enabled) MessageBox.Show("Thông báo", "Lỗi !!");
                int i = b.Location.Y / 150;
                int j = b.Location.X / 150;
                puzzle[i][j] = Int32.Parse(b.Text);
            }

            timeCount.Start();

        }
```



Sau khi click, tất cả button sẽ có thể click được (trừ Back) và hình ảnh cũng sẽ được load lên trân giao diện. Đồng thời, thời gian giải bắt đầu được tính

4. Event button_Click

Sự kiện này sử dụng để di chuyển button trong panel khi ta click vào nó, nó được sử dụng giúp ta có thể xếp cho hình ở đúng vị trí

```
        private void button_Click(object sender, EventArgs e)
        {
            MoveButton((Button)sender);
            if (imgProc.Stack.Count <= 600)
                btBack.Enabled = false;
            else btBack.Enabled = true;
        }
```

- Hàm MoveButton hỗ trợ việc di chuyển đúng, tính toán các trường hợp di chuyển của button khi ta click vào button trong panel

```
        private void MoveButton(Button btn)
        {
            if (((btn.Location.X == EntityPoint.X - 150 || btn.Location.X ==
EntityPoint.X + 150) && btn.Location.Y == EntityPoint.Y)
                || ((btn.Location.Y == EntityPoint.Y - 150 || btn.Location.Y ==
EntityPoint.Y + 150) && btn.Location.X == EntityPoint.X))
```

```
            {
                puzzle[btn.Location.Y / 150][btn.Location.X / 150] = 0;
                puzzle[EntityPoint.Y / 150][EntityPoint.X / 150] =
Int32.Parse(btn.Text);
                Point swap = btn.Location;
                btn.Location = EntityPoint;
                EntityPoint = swap;
                imgProc.pushStack(EntityPoint);
                steps++;
                if (steps > 0)
                    lbSteps.Text = steps.ToString();
            }
        }
```

5. Event timeCount_Tick

Sự kiện này dùng để đếm thời gian trong quá trình giải

```
        private void timeCount_Tick(object sender, EventArgs e)
        {
            sec++;
            if (sec >= 60)
            {
                min++;
                lbmin.Text = min.ToString();
                sec = 0;
            }
            lbsec.Text = sec.ToString();
        }
```

6. Event btEGame_Click

Sự kiện này xuất hiện khi click EndGame, nó hiện ra thông báo kết thúc và hiển trị kết quả

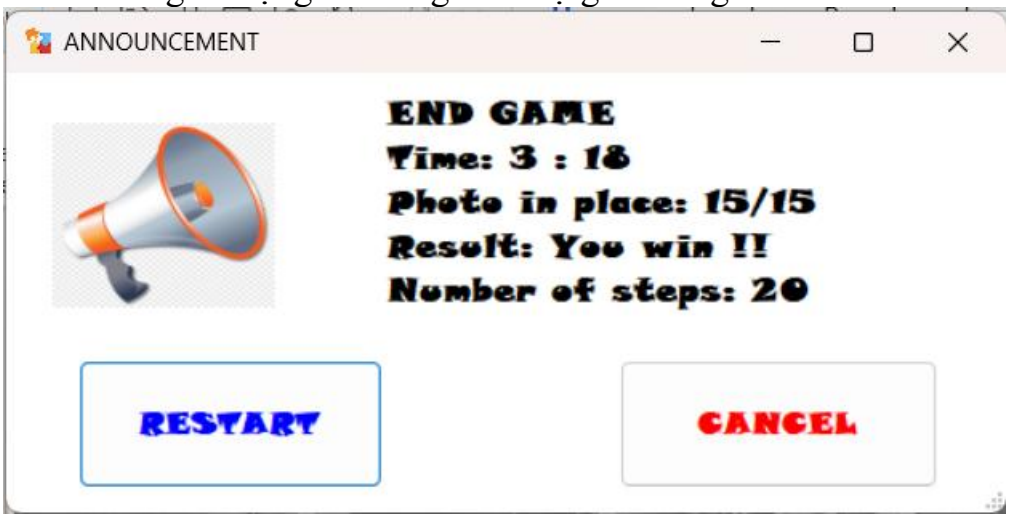
```
        private void btEGame_Click(object sender, EventArgs e)
        {
            MessageBoxGame messBox;
            timeCount.Stop();
            int count = imgProc.CheckWin(images, panel);
            foreach (Button b in panel.Controls)
                if (b.Enabled == true)
                    b.Enabled = false;
                else
                {
                    messBox = new MessageBoxGame("You haven't started yet!!");
                    messBox.ShowDialog();
                    return;
                }

            string result;
            if (count == 15)
                result = "You win !!";
            else
                result = "You fail !!";
            string Ann = "END GAME \n" +
                "Time: " + lbmin.Text + " : " + lbsec.Text + "\n" +
                "Photo in place: " + count + "/15 \n" +
                "Result: " + result + "\n" +
                "Number of steps: " + steps;
            if (checkTool)
                Ann += "\nUse a solution tool !!";
            messBox = new MessageBoxGame(Ann);

            messBox.ShowDialog();

            lbmin.Text = "00";
            lbsec.Text = "00";
```

```
            steps = -600;
            lbSteps.Text = "00";
            btBack.Enabled = false;
            btSolve.Enabled = false;
        }
```

Trong đó, messBox hỗ trợ cho việc thông báo kết quả được viết như sau

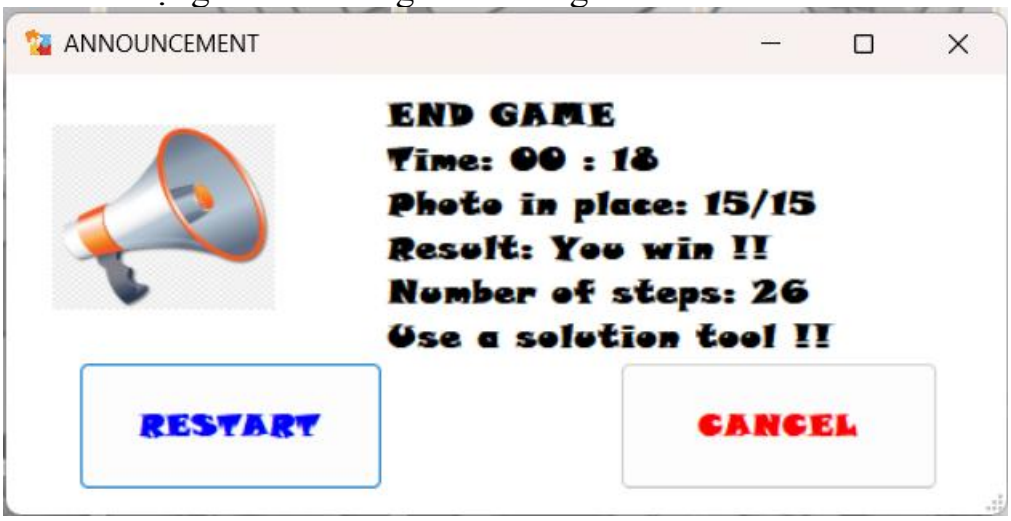
```
    public partial class MessageBoxGame : Form
    {
        public MessageBoxGame(string mess)
        {
            InitializeComponent();
            lbAnn.Text = mess;
        }

        private void btReS_Click(object sender, EventArgs e)
        {
            this.Close();
        }

        private void btCan_Click(object sender, EventArgs e)
        {
            Application.Exit();
        }
    }
```

- Khi không sử dụng tìm lời giải và tự giải thắng

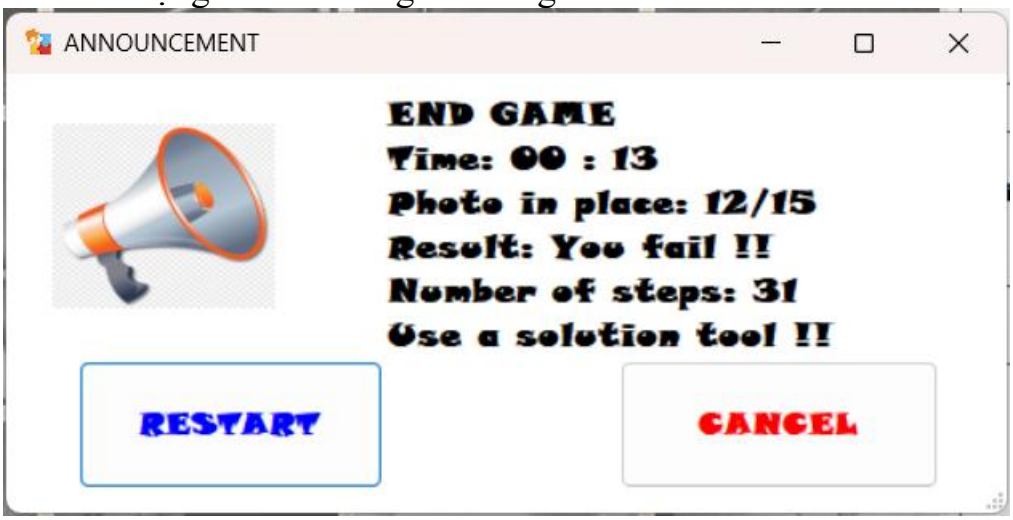


- Khi không sử dụng tìm lời giải và tự giải không ra (thua)

- Khi sử dụng tool tìm lời giải và thắng



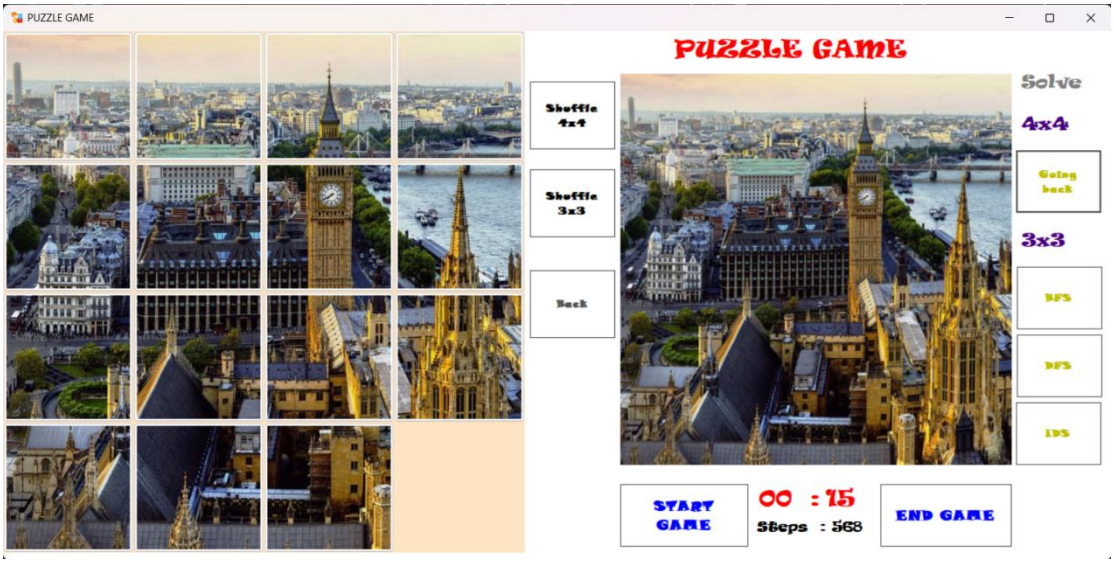
- Khi sử dụng tool tìm lời giải nhưng vẫn thua



7. Event btSolve_Click

Event này xuất hiện khi click vào Going Back để tìm lời giải, lời giải này được tìm bằng cách quay lui, và lời giải khoảng trên 500 bước. Trong sự kiện này sẽ gọi lại phương thức sweep trong ImageProcess để tìm lời giải.

```
private void btSolve_Click(object sender, EventArgs e)
{
    try
    {
        EntityPoint = imgProc.sweep(EntityPoint, images, panel);
        checkTool = true;
    }
    catch (Exception ex)
    {
        MessageBoxGame messBox = new MessageBoxGame(ex.ToString());

        messBox.ShowDialog();
    }
}
```



8. Event btBack_Click

Event này xuất hiện khi click vào Back để quay ngược lại bước trước đó

```
private void btBack_Click(object sender, EventArgs e)
{
    Point s = imgProc.Stack.Pop();
    imgProc.SimulateButtonClick(s, panel);
}
```

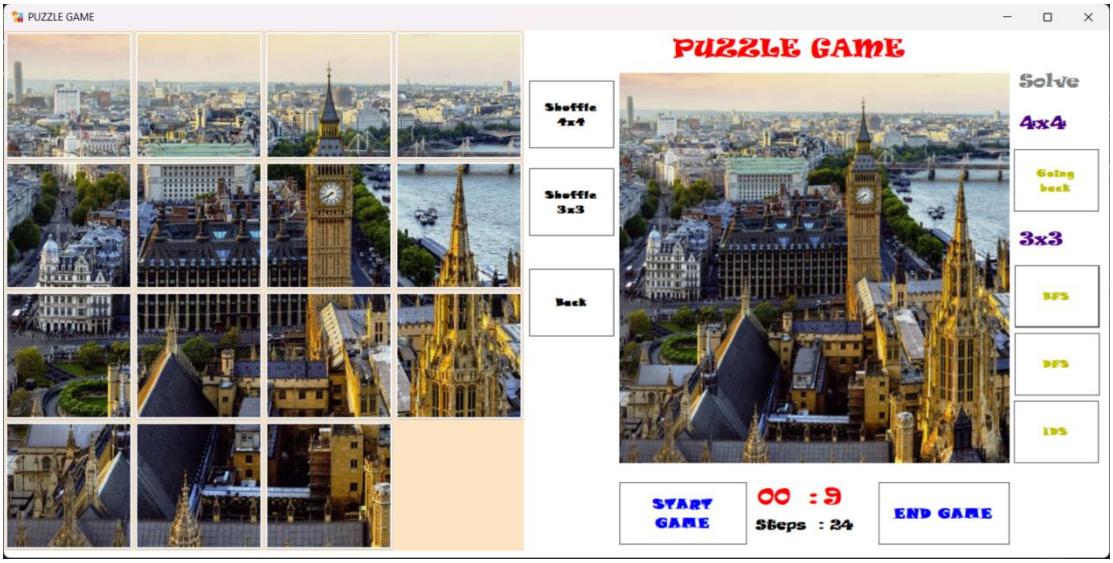
9. Event btSolveBfs_Click

Phương thức sử dụng hỗ trợ cho việc chuyển dời button

```
private void Move(int i, int j)
{
    foreach (Button b in panel.Controls)
    {
        if (b.Location.X / 150 == j && b.Location.Y / 150 == i)
        {
            MoveButton(b);
            return;
        }
    }
}
```

Event này xuất hiện khi click vào BFS để tìm lời giải cho puzzle

```
private void btSolveBfs_Click(object sender, EventArgs e)
{
    PuzzleState listpuzzle = sovle.BFS_Puzzle(new PuzzleState(puzzle));
    foreach (PuzzleState p in listpuzzle.Path)
    {
        for (int i = 0; i < puzz_n; i++)
        {
            for (int j = 0; j < puzz_n; j++)
            {
                Console.Write(p.Puzzle[i][j] + " ");
                if (p.Puzzle[i][j] == 0)
                {
                    Move(i, j);
                    Thread.Sleep(50);
                    break;
                }
            }
            Console.WriteLine();
        }
        Console.WriteLine();
    }
    checkTool = true;
}
```

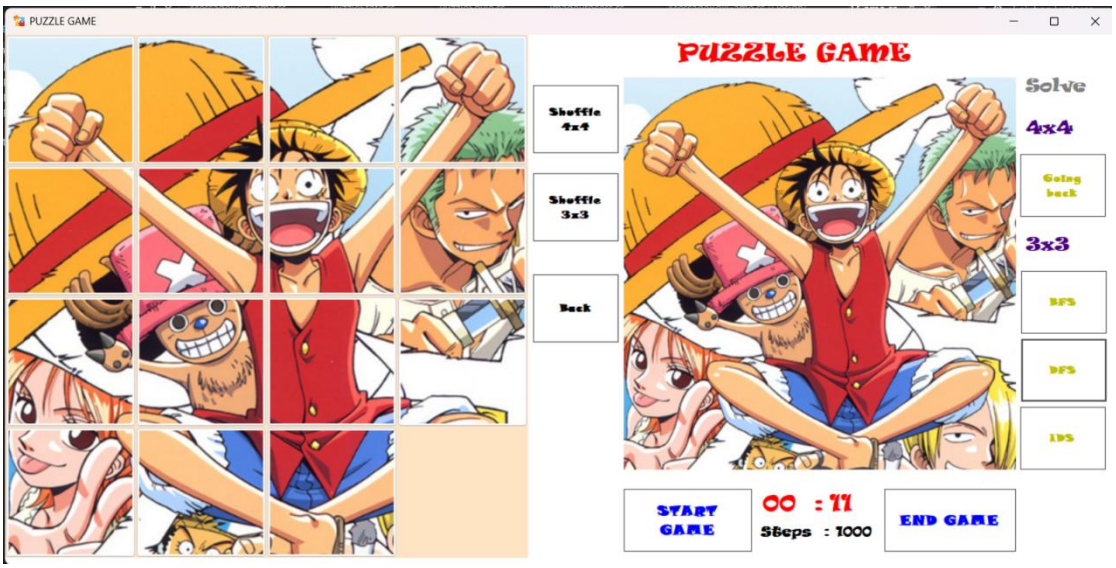


Trong Event có sử dụng phương thức BFS_Puzzle trong PuzzleSlove để tìm lời giải cho Puzzle

10. Event btSolveDfs_Click

Event này xuất hiện khi click vào DFS để tìm lời giải cho puzzle

```
        private void btSolveDfs_Click(object sender, EventArgs e)
        {
            PuzzleState listpuzzle = sovle.DFS_Puzzle(new PuzzleState(puzzle));
            if (listpuzzle is null)
            {
                MessageBox.Show("SỐ BƯỚC DI CHUYỂN ĐÃ LỚN HƠN
1000 !!!","Sorry",MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            foreach (PuzzleState p in listpuzzle.Path)
            {
                for (int i = 0; i < puzz_n; i++)
                {
                    for (int j = 0; j < puzz_n; j++)
                    {
                        Console.Write(p.Puzzle[i][j] + " ");
                        if (p.Puzzle[i][j] == 0)
                        {
                            Move(i, j);
                            Thread.Sleep(50);
                            break;
                        }
                    }
                    Console.WriteLine();
                }
                Console.WriteLine();
            }
            checkTool = true;
        }
```

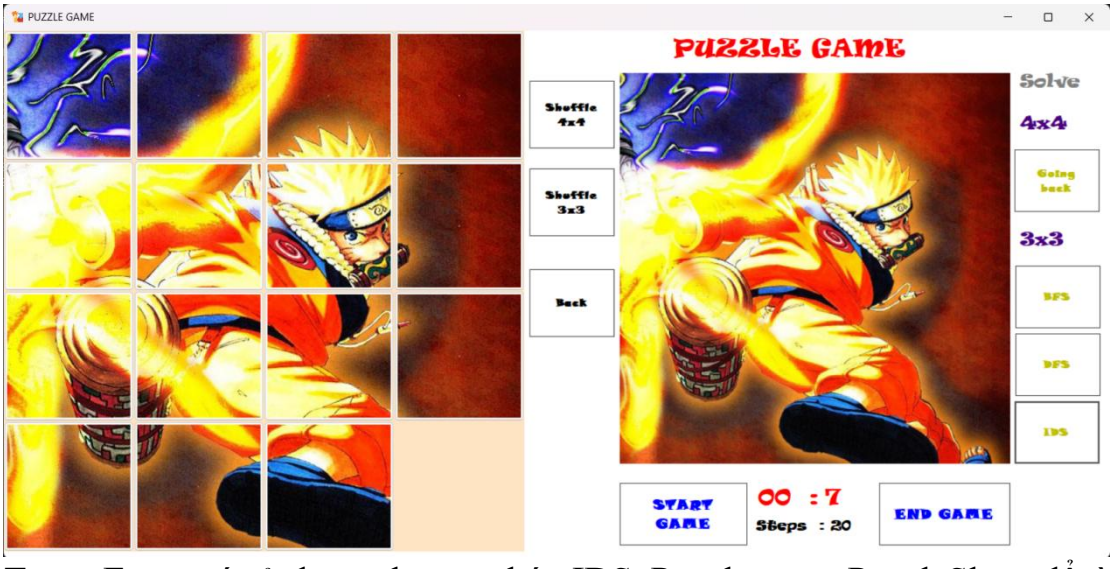


Trong Event có sử dụng phương thức DFS_Puzzle trong PuzzleSlove để tìm lời giải cho Puzzle

11. Event btSolveIDS_Click

Event này xuất hiện khi click vào IDS (d=5) để tìm lời giải cho puzzle

```
private void btSolveIDS_Click(object sender, EventArgs e)
{
    PuzzleState listpuzzle = sovle.IDS_Puzzle(new PuzzleState(puzzle),5);

    foreach (PuzzleState p in listpuzzle.Path)
    {
        for (int i = 0; i < puzz_n; i++)
        {
            for (int j = 0; j < puzz_n; j++)
            {
                Console.Write(p.Puzzle[i][j] + " ");
                if (p.Puzzle[i][j] == 0)
                {
                    Move(i, j);
                    Thread.Sleep(50);
                    break;
                }
            }
            Console.WriteLine();
        }
        Console.WriteLine();
    }
    checkTool = true;
}
```

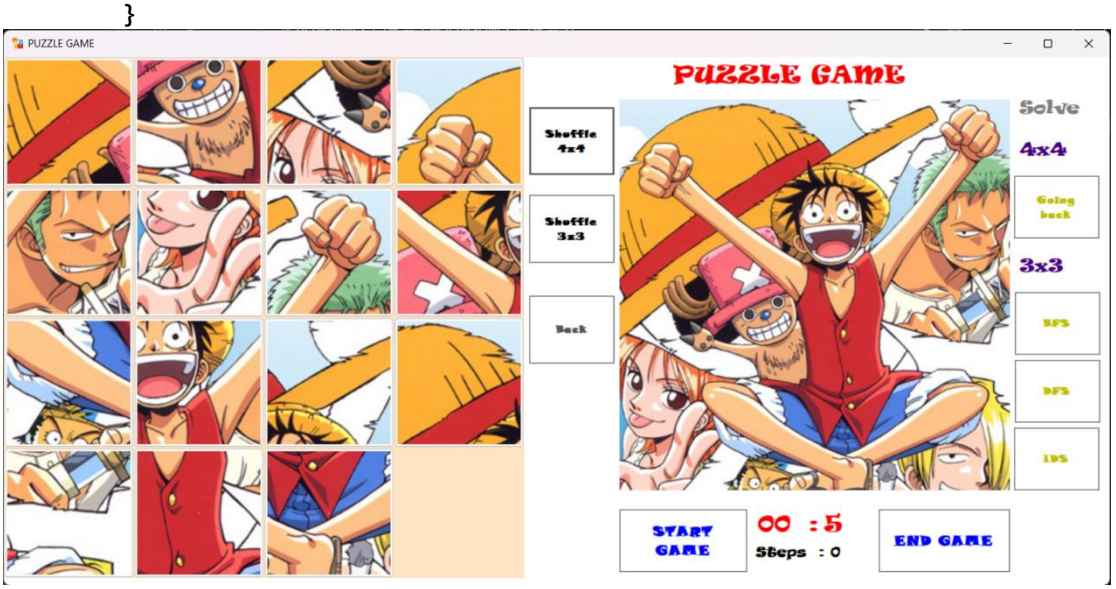


Trong Event có sử dụng phương thức IDS_Puzzle trong PuzzleSlove để tìm lời giải cho Puzzle

12. Event btShuffle_Click

Event này sử dụng để xáo trộn các button trong panel puzzle để tạo ra một hình bị sắp lộn xộn các mảnh hình (trên phạm vi 4x4)

```
        private void btShuffle_Click(object sender, EventArgs e)
        {
            imgProc.Stack.Clear();
            int tempsteps = steps;
            EntityPoint = imgProc.ImageStart(pBOriginal.Image, images, panel,
EntityPoint, 600, false);
            steps = tempsteps;
            lbSteps.Text = steps.ToString();
        }
```

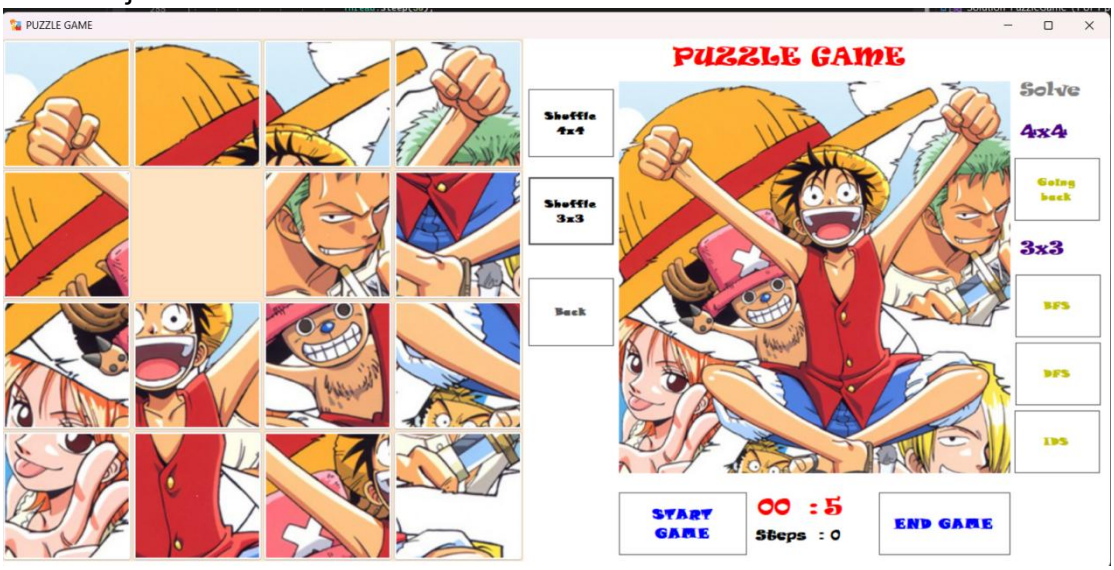


13. Event btShuffle3x3_Click

Event này sử dụng để xáo trộn các button phần 3x3 ở dưới bên phải trong panel puzzle để tạo ra một hình bị sắp lộn xộn các mảng hình (trên phạm vi 3x3)

```
        private void btShuffle3x3_Click(object sender, EventArgs e)
        {
            imgProc.Stack.Clear();
            int tempsteps = steps;
```

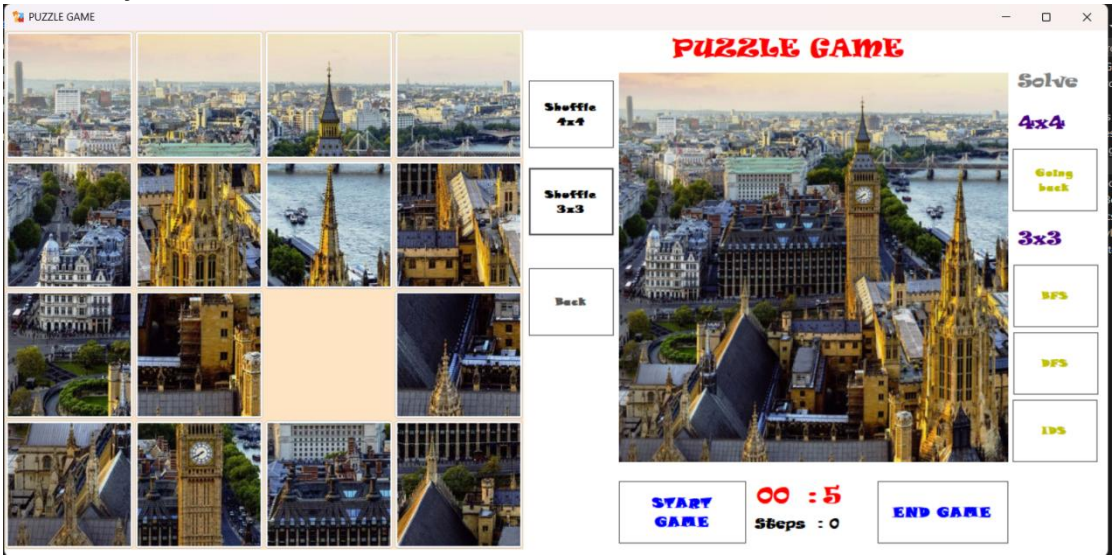
```
            EntityPoint = imgProc.ImageStart(pBOriginal.Image, images, panel,
EntityPoint, 600, true);
            steps = tempsteps;
            lbSteps.Text = steps.ToString();
        }
```

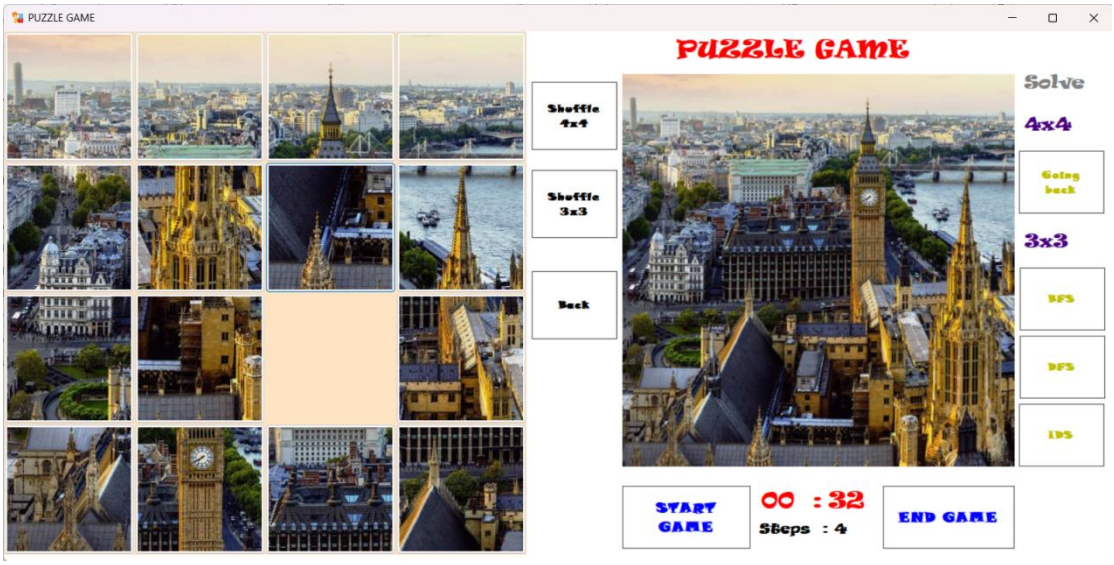


14. Event lbSteps_TextChanged

Event này sử dụng khi số bước bằng 0 thì Back sẽ khóa và lớn hơn 0 thì Back mở khóa

```
        private void lbSteps_TextChanged(object sender, EventArgs e)
        {
            int step = int.Parse(lbSteps.Text);
            if (step == 0)
                btBack.Enabled = false;
            else if (step > 0)
                btBack.Enabled = true;
        }
```




PUZZLE GAME
PUZZLE GAME
Shuffle 4x4
Shuffle 3x3
Back
Solve
4x4
Going back
3x3
BFS
DFS
IDS
START GAME
00 : 32
Steps : 4
END GAME

C. NỘI DUNG ĐỒ ÁN VỚI TKINTER PYTHON

I. Chức năng của ứng dụng

1. Chức năng chung

1.1. Chức năng hiển thị ảnh

Chức năng này giống với mô hình winform C#, nhưng ở mô hình python có thể cho người chơi chọn hình ảnh

1.2. Chức năng xáo trộn các mảng hình

Chức năng này giống với mô hình winform C#

1.3.Chức năng hiển thị số bước

Chức năng này giống với mô hình winform C#

1.4. Chức Reset

Chức năng này sẽ trả lại trạng thái xáo trộn ban đầu (khi mới click xáo trộn)

2. Chức năng dành cho player chơi

Chức năng chơi game : Giúp người chơi có thể chơi game trên form thông qua giao diện trò chơi

3. Chức năng dành cho máy chơi (Tìm lời giải)

3.1. Chức năng tìm lời giải bằng giải thuật BFS, DFS, IDS, UCS, A, Beam Search, Hill Climbing*

Chức năng này sẽ thực hiện tìm ra lời giải cho bài toán qua các giải thuật tìm kiếm BFS, DFS, IDS, UCS, A*, Beam Search, Hill Climbing

3.2. Chức năng hiển thị số trạng thái máy duyệt

Chức năng này sẽ cho xem số trạng thái mà máy đã duyệt qua để tìm lời giải cho trò chơi

II. Thư viện sử dụng

```
import tkinter as tk
import random
import time
from collections import deque
from PIL import Image, ImageTk
from tkinter import filedialog
from tkinter import messagebox
import cv2
```

III. Các hàm hỗ trợ

Sử dụng để kiểm tra xáo trộn puzzle coi có hợp lệ không (có thể giải được không)

```
def count_inversions(puzzle):
    # Flatten the puzzle into a 1D list
    flattened_puzzle = [item for row in puzzle for item in row]

    # Count inversions
    inversions = 0
    for i in range(len(flattened_puzzle)):
        for j in range(i+1, len(flattened_puzzle)):
            if flattened_puzzle[i] and flattened_puzzle[j] and flattened_puzzle[i] >
flattened_puzzle[j]:
                inversions += 1

    return inversions

def is_solvable(puzzle):
    # Check if the puzzle is solvable based on the inversion count
    inversions = count_inversions(puzzle)
```

```
    return inversions % 2 == 0
```

IV. Lớp PuzzleState

Lớp này sử dụng cho việc giải puzzle

```
class PuzzleState:
    def __init__(self, puzzle, moves=0, path = [], cost = 0):
        self.puzzle = puzzle
        self.moves = moves
        self.path = path
        self.cost = cost
    def setHeuristic(self, heu):
        self.Heuristic = heu
    def Priority(self):
        return self.cost + self.Heuristic
```

V. Lớp PuzzleGame

1. Khởi tạo ban đầu

```
 def __init__(self, master):
        self.points = 0
        self.steps = 0
        self.puzzle_pieces_root = None
        self.puzzle_pieces = None
        self.master = master
        self.master.title("Puzzle Game")
        self.master.geometry("850x490")
        self.create_puzzle()
        self.empty_cell = (2, 2)  # Empty cell initially at the bottom-right
        self.create_widgets()
        self.update_display()

        self.master.bind('<Up>', self.move_by_key)
        self.master.bind('<Down>', self.move_by_key)
        self.master.bind('<Left>', self.move_by_key)
        self.master.bind('<Right>', self.move_by_key)
```

Khởi tạo các biến thuộc tính cần sử dụng :

+ `self.steps :đếm số bước đi`
+ `self.puzzle_pieces_root : lưu các mảnh ảnh ban đầu (lúc chưa xáo trộn)`
+ `self.puzzle_pieces : lưu các mảnh ảnh tức thời (có thể đã bị xáo trộn)`
+ `self.master : Giao diện window chính`

a. Phương thức self.create_puzzle()

```
  def create_puzzle(self):
        # Create a solvable puzzle
        self.puzzle = [[0]*3 for _ in range(3)]
        nums = list(range(1, 9))
        for i in range(3):
            for j in range(3):
                if nums:
                    self.puzzle[i][j] = nums.pop(0)
```

Khởi tạo puzzle lưu ở dạng ma trận (mảng 2 chiều) có giá trị là trạng thái đích như sau:

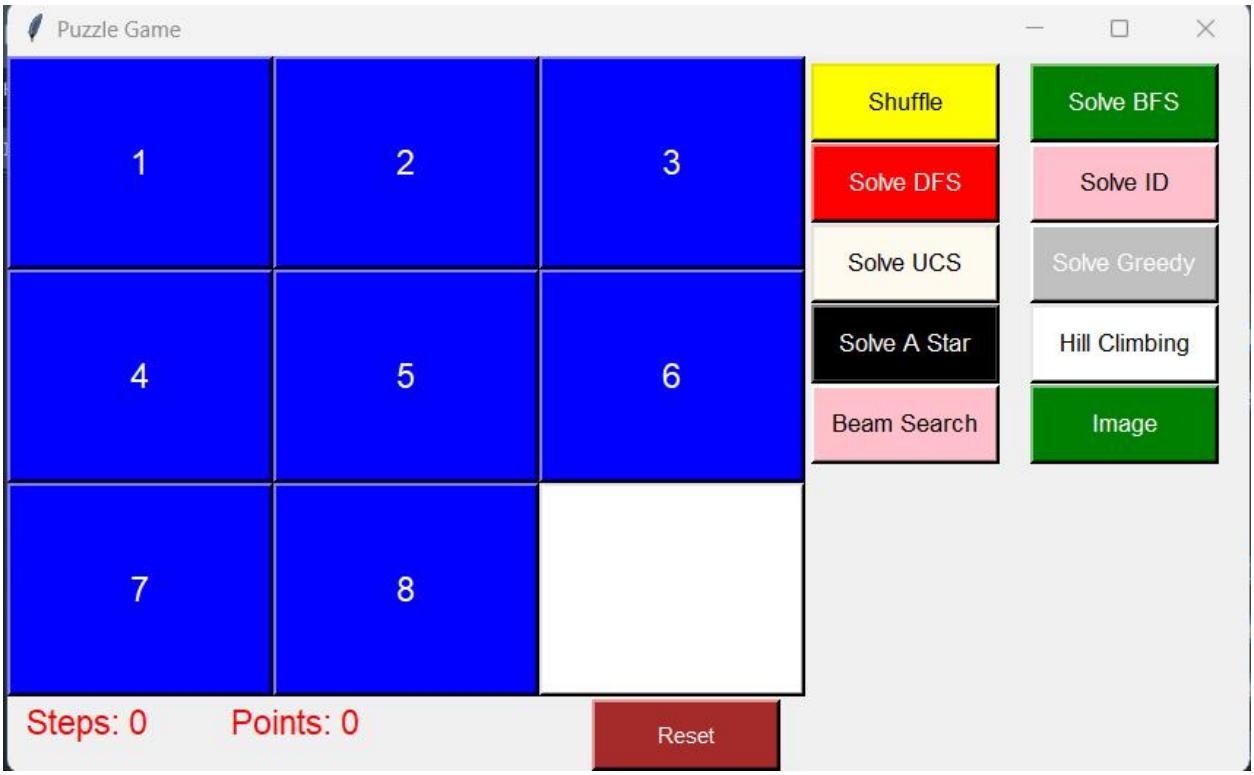
1	2	3
4	5	6
7	8	X

b. Phương thức **self.create_widgets()**

```
    def create_widgets(self):
        self.buttons = [[tk.Button(self.master, text=str(self.puzzle[i][j]),
width=10, height=4,  font=('Arial', 14),bg='blue', fg='white', relief=tk.RAISED,
bd=3, padx=20, pady=10,
                         activebackground="#8bbaff", highlightbackground="#a5b1c2",
                        highlightcolor="#a5b1c2", highlightthickness=2, command=lambda
i=i, j=j: self.move(i, j)) for j in range(3)] for i in range(3)]
        for i in range(3):
            for j in range(3):
                self.buttons[i][j].grid(row=i, column=j)
        self.buttons[2][2].configure(bg='white', fg='white')

        btshuffle= tk.Button(self.master,text="Shuffle", width=13, height=2,
font=('Arial',10),bg='yellow', fg='black', relief=tk.RAISED, bd=3,
command=self.shuffle_puzzle)
        btshuffle.place(x=550, y=5)
        btsolvebfs= tk.Button(self.master,text="Solve BFS", width=13, height=2,
font=('Arial',10),bg='green', fg='white', relief=tk.RAISED, bd=3,
command=self.slovebfs)
        btsolvebfs.place(x=700, y=5)
        btsolvedfs= tk.Button(self.master,text="Solve DFS", width=13, height=2,
font=('Arial',10),bg='red', fg='white', relief=tk.RAISED, bd=3,
command=self.slovedfs)
        btsolvedfs.place(x=550, y=60)
        btsolveID= tk.Button(self.master,text="Solve ID", width=13, height=2,
font=('Arial',10),bg='pink', fg='black', relief=tk.RAISED, bd=3,
command=self.sloveID)
        btsolveID.place(x=700, y=60)
        btsolveUCS= tk.Button(self.master,text="Solve UCS", width=13, height=2,
font=('Arial',10),bg='floralwhite', fg='black', relief=tk.RAISED, bd=3,
command=self.sloveucs)
        btsolveUCS.place(x=550, y=115)
        btsolveGreedy= tk.Button(self.master,text="Solve Greedy", width=13, height=2,
font=('Arial',10),bg='silver', fg='white', relief=tk.RAISED, bd=3,
command=self.slovegreedy)
        btsolveGreedy.place(x=700, y=115)
        btsolveAStar= tk.Button(self.master,text="Solve A Star", width=13, height=2,
font=('Arial',10),bg='black', fg='white', relief=tk.RAISED, bd=3,
command=self.sloveAStar)
        btsolveAStar.place(x=550, y=170)
        btsolveHillClimbing= tk.Button(self.master,text="Hill Climbing", width=13,
height=2,  font=('Arial',10),bg='white', fg='black', relief=tk.RAISED, bd=3,
command=self.sloveHillClimbing)
        btsolveHillClimbing.place(x=700, y=170)
        btsolveBeamSearch= tk.Button(self.master,text="Beam Search", width=13,
height=2,  font=('Arial',10),bg='pink', fg='black', relief=tk.RAISED, bd=3,
command=self.sloveBeamSearch)
        btsolveBeamSearch.place(x=550, y=225)
        btImg= tk.Button(self.master,text="Image", width=13, height=2,
font=('Arial', 10),bg='green', fg='white', relief=tk.RAISED, bd=3, command= lambda:
self.Image(3,3))
```

```
        btImg.place(x=700, y=225)
        btReset= tk.Button(self.master,text="Reset", width=13, height=2,
font=('Arial', 9),bg='brown', fg='white', relief=tk.RAISED, bd=3, command=
self.reset)
        btReset.place(x=400, y=440)

        self.lblSteps = tk.Label(self.master,text="Steps: {}".format(self.steps),
font=('Arial', 14), fg='red')
        self.lblSteps.place(x=10, y=440)

        self.lblPoints = tk.Label(self.master,text="Points: {}".format(self.points),
font=('Arial', 14), fg='red')
        self.lblPoints.place(x=150, y=440)
```

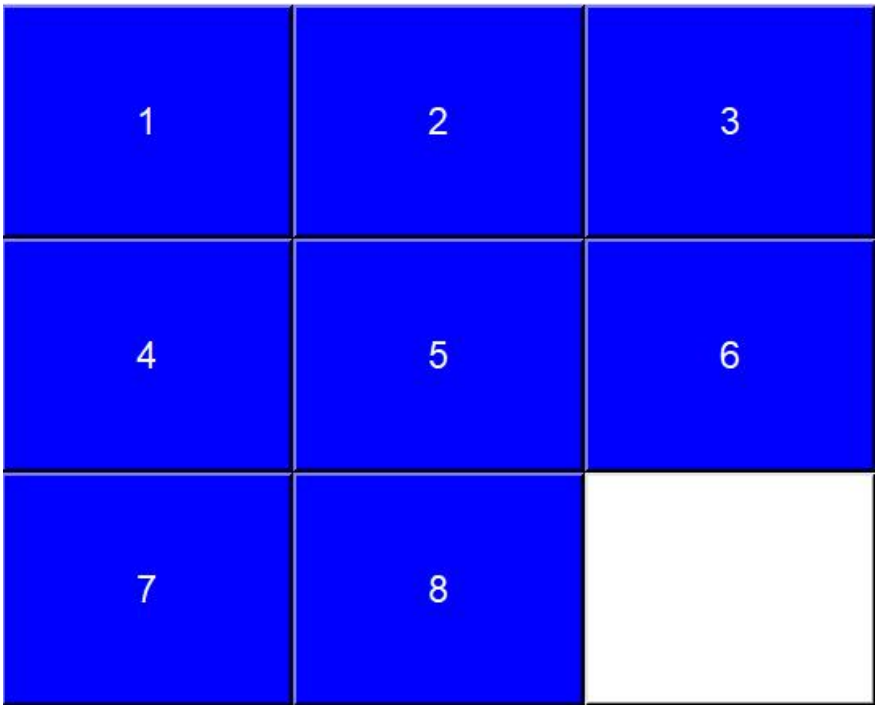
Khởi tạo Controls trên giao diện , bao gồm bảng xếp hình puzzle chứa các button, Các button tùy chọn , và cả label hiện bước đi



c. Phương thức **self.update_display()**

```
    def update_display(self):
        self.lblSteps.config(text="Steps: {}".format(self.steps))
        for i in range(3):
            for j in range(3):
                num = self.puzzle[i][j]
                text = str(num) if num != 0 else ""

                if self.puzzle_pieces is not None:
                    self.buttons[i][j].config(text=text, bg='blue', fg='white',
image= self.puzzle_pieces[i][j])
                else:
                    self.buttons[i][j].config(text=text, bg='blue', fg='white')
                if num == 0:
                    self.empty_cell = (i,j)
                    self.buttons[i][j].configure(bg='white', fg='white')
```

Cập nhật puzzle trên giao diện theo puzzle dưới dạng ma trận đã khởi tạo trước đó, mỗi giá trị trong ma trận được được gắn giá trị text cho button



d. Phương thức **self.move_by_key()**

```
    def move_by_key(self, event):
        row, col = self.empty_cell

        if event.keysym == 'Up' and row < 2:
            self.move(row + 1, col)
        elif event.keysym == 'Down' and row > 0:
            self.move(row - 1, col)
        elif event.keysym == 'Left' and col < 2:
            self.move(row, col + 1)
        elif event.keysym == 'Right' and col > 0:
            self.move(row, col - 1)
```

Bắt sự kiện khi click phím di chuyển trái, phải, lên, xuống thay cho click button

e. Phương thức `self.move(row, col)`

```
    def move(self, i, j):
        if (i, j) == self.empty_cell:
            return

        row, col = self.empty_cell

        # Check if the clicked cell is adjacent to the empty cell
        if (i == row and abs(j - col) == 1) or (j == col and abs(i - row) == 1):
            self.steps = self.steps + 1
            # Swap the clicked cell with the empty cell
            self.puzzle[row][col], self.puzzle[i][j] = self.puzzle[i][j],
self.puzzle[row][col]
            self.empty_cell = (i, j)
            if self.puzzle_pieces is not None:
                tempImg = self.puzzle_pieces[i][j]
                self.puzzle_pieces[i][j] = self.puzzle_pieces[row][col]
                self.puzzle_pieces[row][col] = tempImg
                self.buttons[i][j].config(image=self.puzzle_pieces[i][j])
                self.buttons[row][col].config(image=self.puzzle_pieces[row][col])
            self.buttons[i][j].configure(bg='white', fg='white')
            self.buttons[row][col].configure(bg='blue', fg='white')
            self.update_display()
        print(self.puzzle)
```

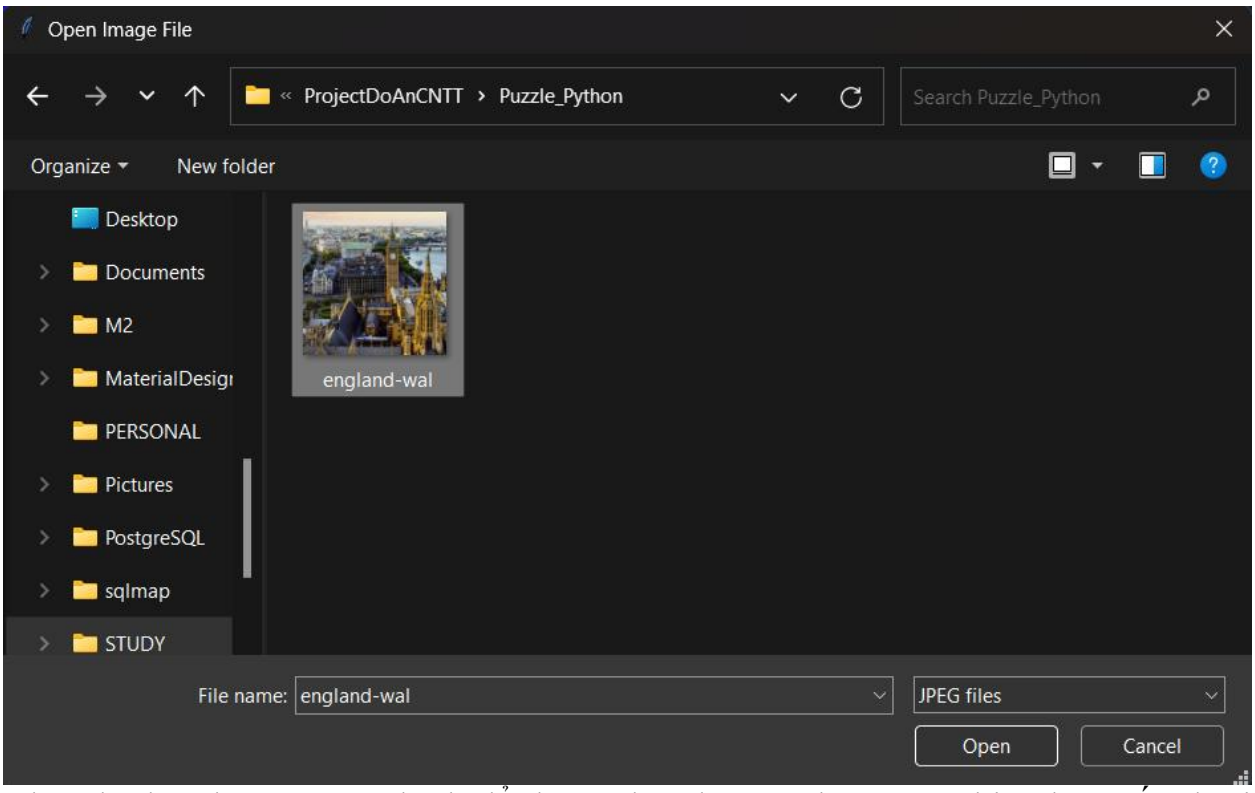
Tạo hiệu ứng button di chuyển trên giao diện

2. Button lựa chọn Image

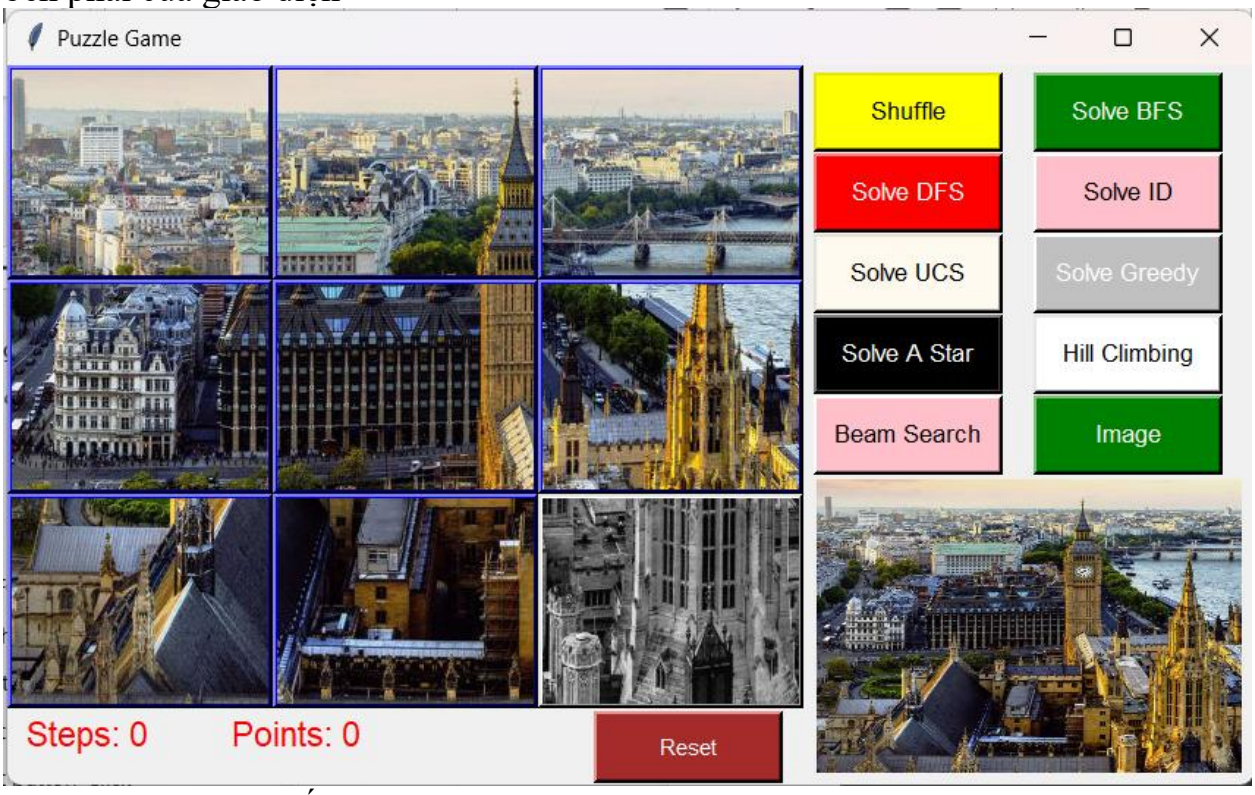
```
     def Image(self,rows,cols):
        # Load hình ảnh
        input_file = filedialog.askopenfilename(title="Open Image File",
                                                filetypes=(("JPEG files", "*.jpg"),
("PNG files", "*.png")))
        if input_file:
            self.ShowImg(input_file)
            self.image = Image.open(input_file)
            self.image = self.image.resize((600, 600))  # Điều chỉnh kích thước hình
ảnh nếu cần
            self.puzzle_pieces = self.create_puzzle_pieces(rows, cols)
            self.puzzle_pieces_root = self.puzzle_pieces
            # đặt các button
            for i in range(rows):
                for j in range(cols):
                    self.buttons[i][j].config(image=self.puzzle_pieces[i][j],
width=171, height=136)
        else:
            messagebox.showerror("Sorry","Không tìm thấy file của bạn !!")

    def create_puzzle_pieces(self, rows, cols):
        puzzle_width, puzzle_height = self.image.size
        piece_width = puzzle_width // cols
        piece_height = puzzle_height // rows

        puzzle_pieces = []

        for i in range(rows):
            row_pieces = []
            for j in range(cols):
                left = j * piece_width
                upper = i * piece_height
                right = left + piece_width
                lower = upper + piece_height
                piece_image = self.image.crop((left, upper, right, lower))
                if self.puzzle[i][j] == 0:
                    piece_image = piece_image.convert("L")
                piece_photo = ImageTk.PhotoImage(piece_image)
                row_pieces.append(piece_photo)
            puzzle_pieces.append(row_pieces)

        return puzzle_pieces
```

Phương thức có chức năng chọn hình từ filedialog và hiển thị lên form và trên các button puzzle

Mở filedialog chọn hình



Chọn hình ảnh và open , hình để được chèn lên các button và hiện bên gốc dưới bên phải của giao diện



Hình bị xám là ô trống trong puzzle (hay vị trí có giá trị 0 hay (X) trong ma trận puzzle)

3. Hàm đếm trạng thái đã duyệt

Thêm Points trên giao diện để đếm số trạng thái đã duyệt

```
        self.lblPoints = tk.Label(self.master,text="Points: {}".format(self.points),
font=('Arial', 14), fg='red')
```

```
        self.lblPoints.place(x=150, y=440)
```

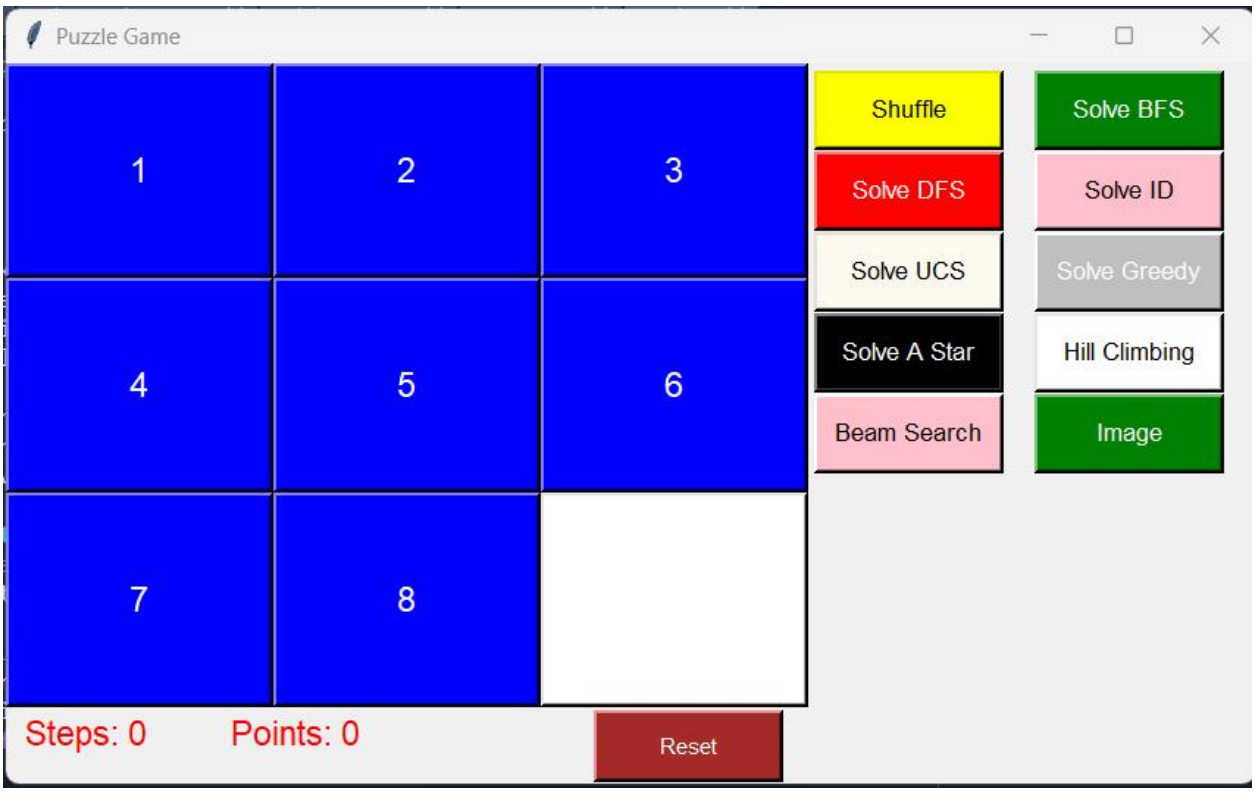
Hàm update lại Points mỗi lần duyệt thêm đỉnh mới

```
    def update_points(self):
        self.points = self.points + 1
        self.lblPoints.config(text="Points: {}".format(self.points))
        self.master.update()
```

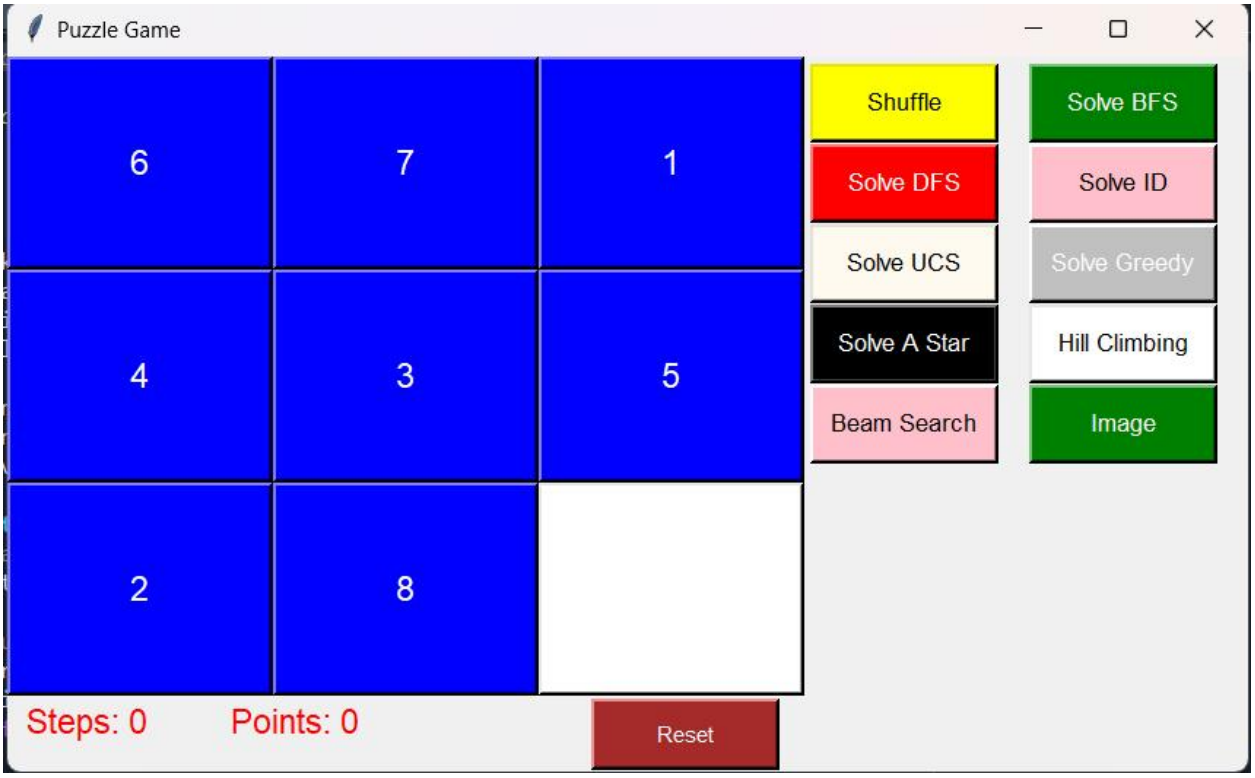
4. Button lựa chọn Shuffle

```
    def shuffle_puzzle(self):
        while(True):
            self.puzzle = [[0]*3 for _ in range(3)]
            nums = list(range(1, 9))
            random.shuffle(nums)
            for i in range(3):
                for j in range(3):
                    if nums:
                        self.puzzle[i][j] = nums.pop(0)

            if is_solvable(self.puzzle):
                break
        self.puzzle_shuffle_root = [row[:] for row in self.puzzle]

        if self.puzzle_pieces is not None:
            img_temp_pieces = [row[:] for row in self.puzzle_pieces_root]
            for i in range(3):
                for j in range(3):
                    v = self.puzzle[i][j] - 1
                    if v != -1:
                        x = v % 3
                        y = v // 3
                    else:
                        x = 2
                        y = 2
                    img_temp_pieces[i][j]= self.puzzle_pieces_root[y][x]
            self.puzzle_pieces = img_temp_pieces
            self.puzzle_pieces_shuffle_root = [row[:] for row in img_temp_pieces]
        self.steps = 0
        self.points = -1
        self.update_points()
        self.update_display()
```

Phương thức có chức năng xáo trộn puzzle (cả ma trận và puzzle giao diện) đồng thời lưu lại giá trị xáo vào `self.puzzle_shuffle_root` và `self.puzzle_pieces_shuffle_root` để sử dụng cho chức năng reset
Khi chưa đổ hình mà bấm Shuffle



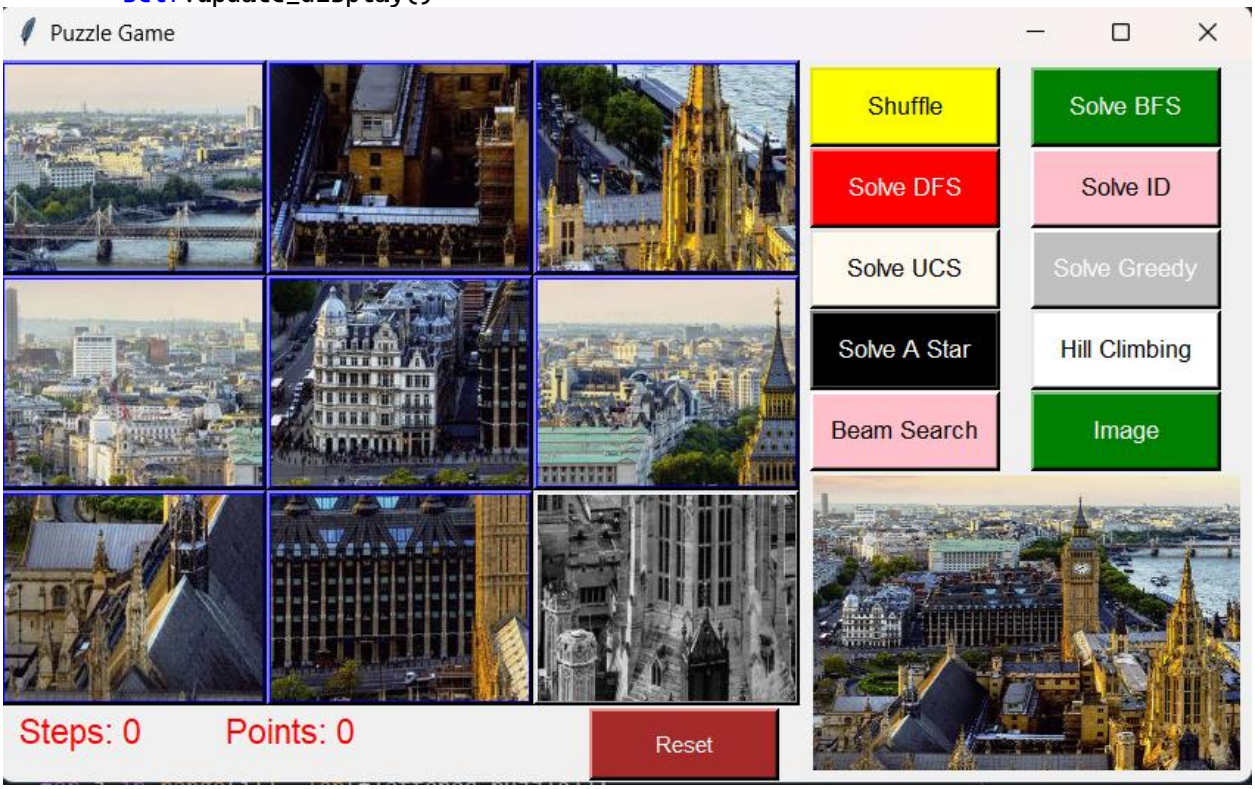
Khi đổ hình mà bấm Shuffle



5. Button lựa chọn Reset

```
    def reset(self):
        self.puzzle = [row[:] for row in self.puzzle_shuffle_root]
        if self.puzzle_pieces is not None:
            self.puzzle_pieces = [row[:] for row in self.puzzle_pieces_shuffle_root]
        self.steps = 0
        self.points = -1
        self.update_points()
```

```
        self.update_display()
```



Phương thức có chức năng này sẽ copy `self.puzzle_shuffle_root` và `self.puzzle_pieces_shuffle_root` đã lưu trước và thực hiện update lại

6. Button lựa chọn Solve BFS

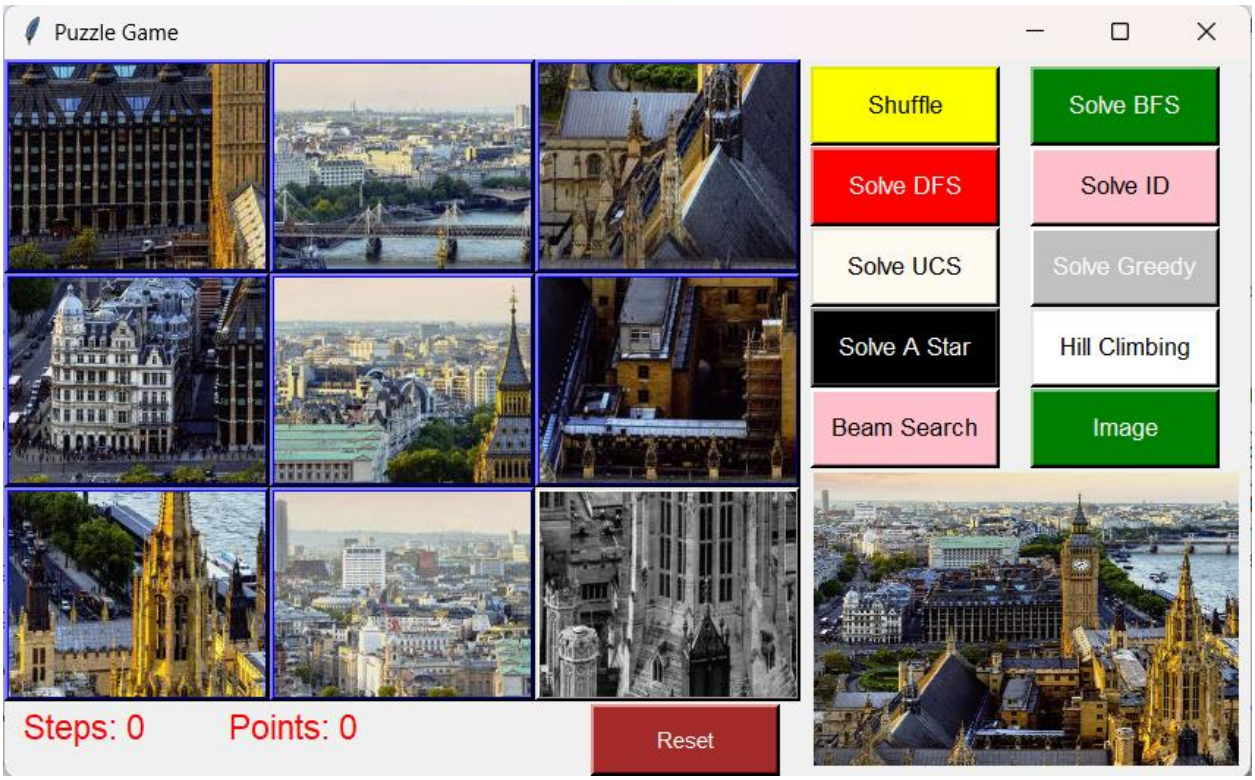
- Chọn Solve BFS sẽ gọi phương thức sau:

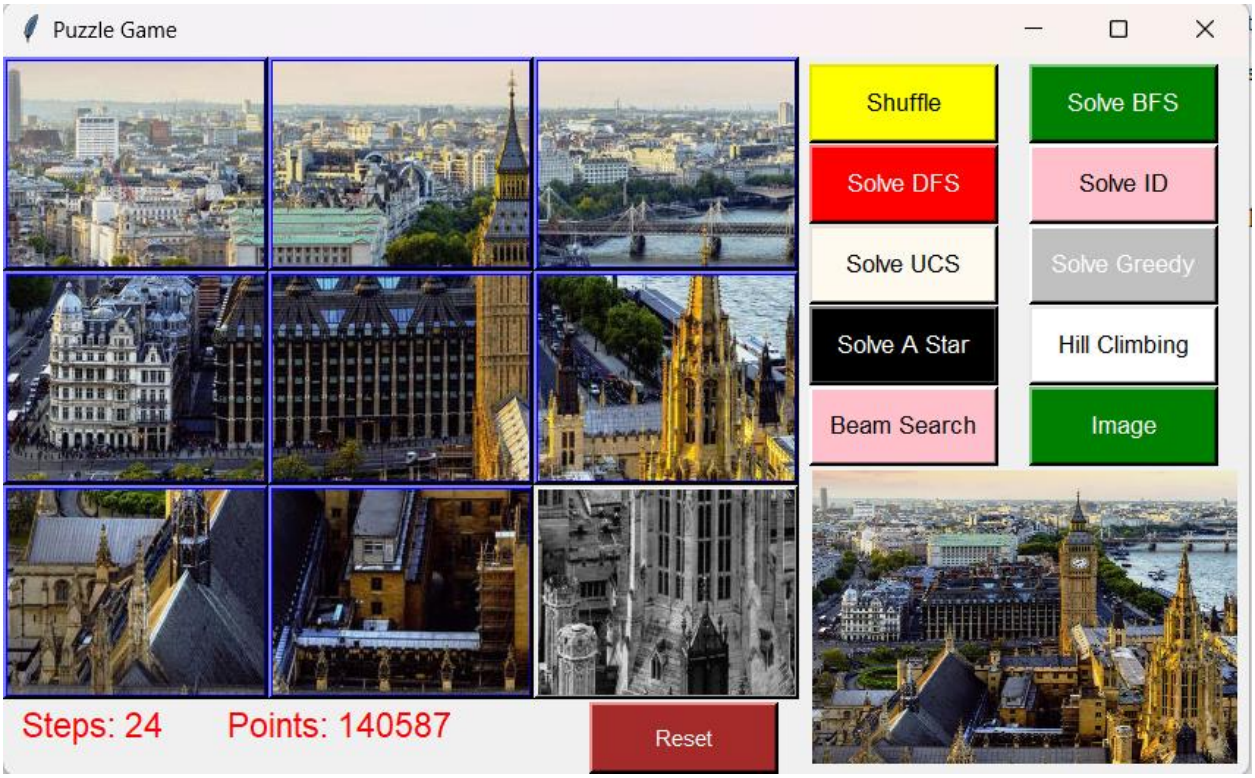
```
    def slovebfs(self):
        t=0;
        solution = self.bfs_puzzle(PuzzleState(self.puzzle))
        for step, state in enumerate(solution.path):
            print(f"Step {step}:")
            for row in state:
                for i in range(3):
                    for j in range(3):
                        num = state[i][j]
                        if(num == 0):
                            t=1
                            self.move(i, j)
                            self.master.update()
                            break
                if t == 1:
                        t = 0
                        break
            time.sleep(0.1)

            print()
```

- Phương thức thực hiện chính và tìm đường đi cho lời giải puzzle bằng phương pháp BFS

```
    # Hàm thực hiện BFS để giải puzzle
    def bfs_puzzle(self,initial_state):
        queue = deque([initial_state])

        visited = set()  # Lưu trữ các trạng thái đã xem
        visited.add(tuple(map(tuple, initial_state.puzzle)))
        self.update_points()
        while queue:
            current_state = queue.popleft()
```

```
            if self.is_goal_state(current_state.puzzle):
                return current_state  # Nếu đạt được trạng thái mục tiêu, trả về kết
quả

            # Sinh các trạng thái kế tiếp và thêm vào queue
            for move in self.generate_moves(current_state):
                if tuple(map(tuple, move.puzzle)) not in visited:
                    queue.append(move)
                    visited.add(tuple(map(tuple, move.puzzle)))
                    move.path = current_state.path + [move.puzzle]
                    self.update_points()

        return None
```





7. Button lựa chọn Solve DFS

- Chọn Solve DFS sẽ gọi phương thức sau:

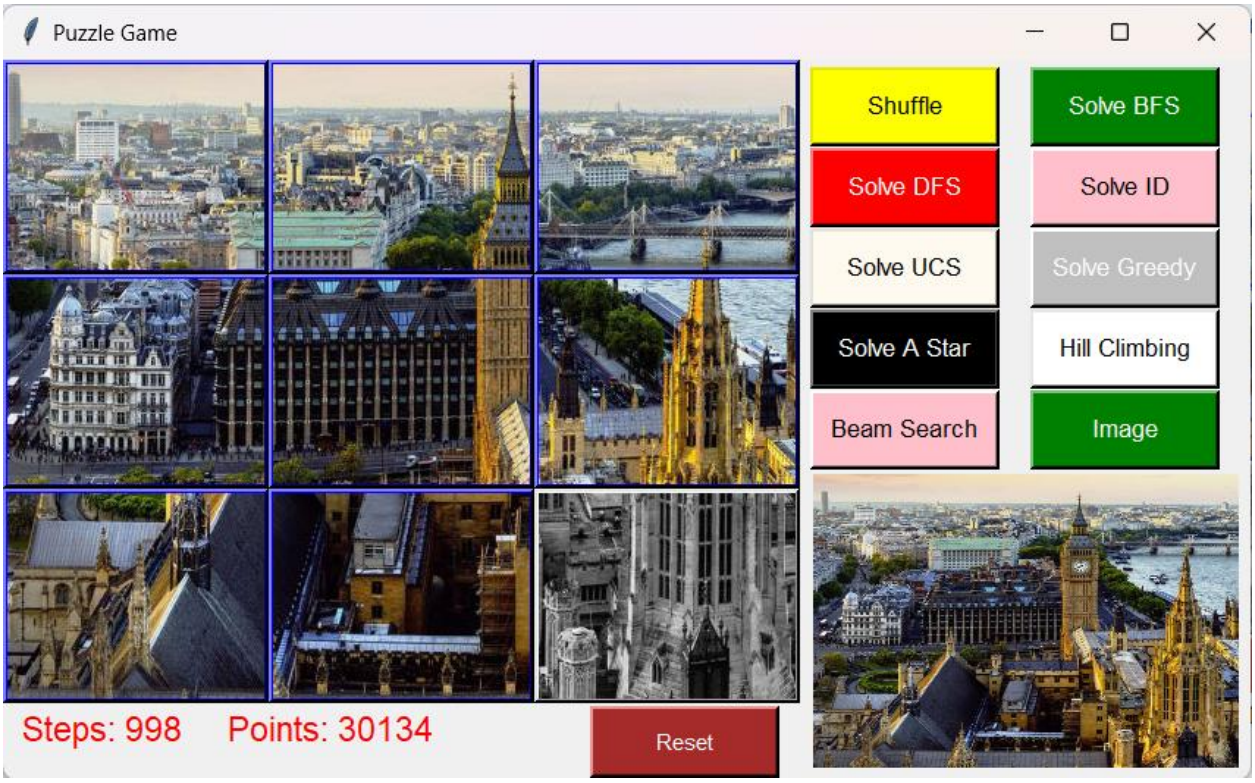
```
    def slovedfs(self):
        result = self.dfs_search(PuzzleState(self.puzzle))
        if result is None:
            messagebox.showerror("Sorry", "Số bước lớn hơn 1000 nên không thể tìm
được !!")
            return
        t = 0
        for step, state in enumerate(result.path):
            print(f"Step {step}:")
            for row in state:
                for i in range(3):
                    for j in range(3):
                        num = state[i][j]
                        if(num == 0):
                            t=1
                            self.move(i, j)
                            self.master.update()
                            break
                if t == 1:
                        t = 0
                        break
            time.sleep(0.1)
```

- Phương thức thực hiện chính và tìm đường đi cho lời giải puzzle bằng phương pháp DFS

```
    def dfs_search(self, initial_state, MAX_DEPTH = 1000):
        stack = [(initial_state)]  # Stack để lưu trữ các trạng thái cần xem xét

        visited = set()  # Lưu trữ các trạng thái đã xem
        self.update_points()

        while stack:
            current_state = stack.pop()  # Lấy trạng thái hiện tại từ stack

            if self.is_goal_state(current_state.puzzle):
```

```
                return current_state  # Nếu đạt được trạng thái mục tiêu, trả về kết
quả

            if current_state.moves >= MAX_DEPTH: #giúp không bị treo máy
                continue

            # Sinh các trạng thái kế tiếp và thêm vào stack
            for move in self.generate_moves(current_state):
                if tuple(map(tuple, move.puzzle)) not in visited:
                    stack.append(move)
                    visited.add(tuple(map(tuple, move.puzzle)))
                    move.path = current_state.path + [move.puzzle]
                    self.update_points()
        return None
```

- Do DFS một hướng đi và duyệt đến cùng một đầu của nhánh trước khi quay lại và thử một hướng khác. Để tránh bị treo máy nên thêm MAX_DEPTH để số bước không vượt quá nhiều



8. Button lựa chọn Solve ID

- Chọn Solve ID (d = 5)sẽ gọi phương thức sau:

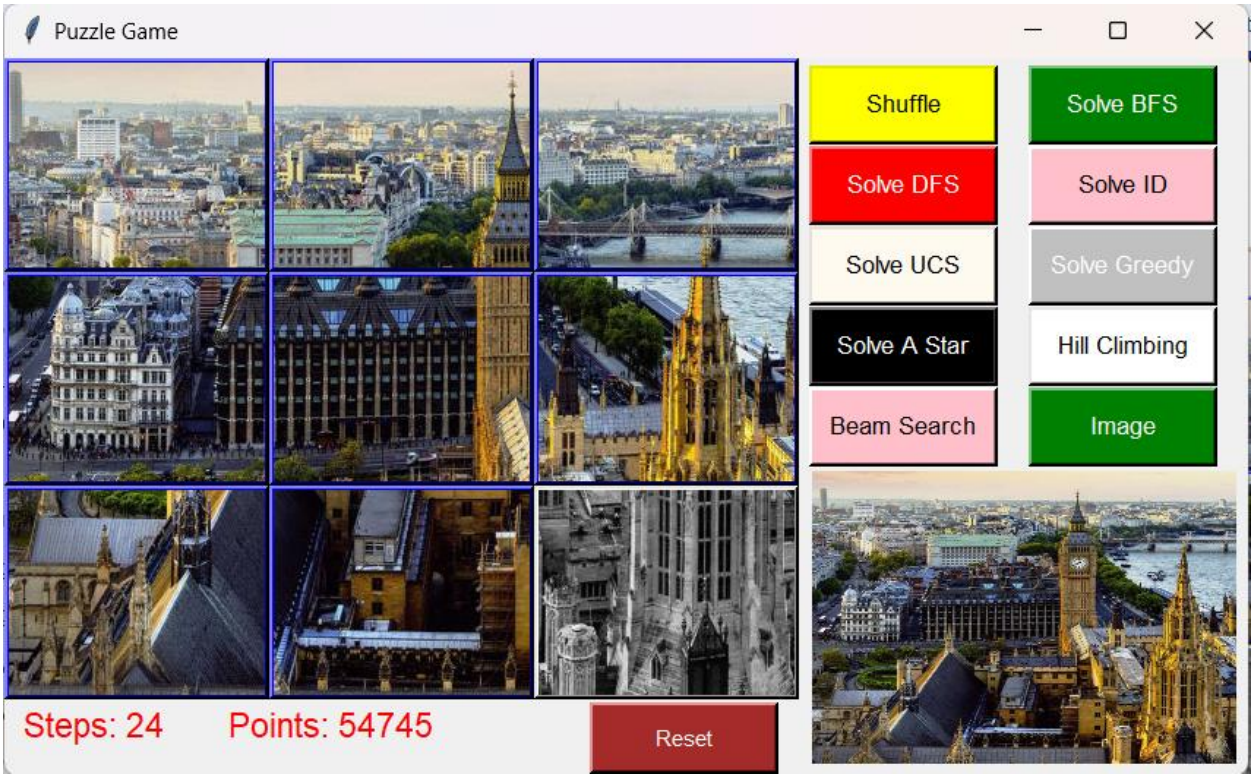
```
    def sloveID(self):
        d = 5
        t = 0
        result = self.ID_search(PuzzleState(self.puzzle),d)
        if result is not None:
            for step, state in enumerate(result.path):
                print(f"Step {step}:")
                for row in state:
                    for i in range(3):
                        for j in range(3):
                            num = state[i][j]
                            if(num == 0):
                                t=1
                                self.move(i, j)
                                self.master.update()
```

```
                            break
                    if t == 1:
                        t = 0
                        break
            time.sleep(0.1)
```

- Phương thức thực hiện chính và tìm đường đi cho lời giải puzzle bằng phương pháp ID

```
    def ID_search(self, initial_state, depth_limit):
        visited = set()  # Lưu trữ các trạng thái đã xem

        state_eq_limit = [initial_state]

        while state_eq_limit:
            stack = state_eq_limit
            state_eq_limit = []
            while stack:
                current_state = stack.pop()  # Lấy trạng thái hiện tại từ stack

                if self.is_goal_state(current_state.puzzle):
                    return current_state  # Nếu đạt được trạng thái mục tiêu, trả về
kết quả

                # Sinh các trạng thái kế tiếp và thêm vào stack
                for move in self.generate_moves(current_state):
                    if tuple(map(tuple, move.puzzle)) not in visited and move.moves
<= depth_limit:
                        stack.append(move)
                        visited.add(tuple(map(tuple, move.puzzle)))
                        self.update_points()
                        move.path = current_state.path + [move.puzzle]
                        if move.moves == depth_limit:
                            state_eq_limit.append(move)
            depth_limit = depth_limit + 5

        return None
```

- ID tăng dần độ sâu nên sẽ tìm được lời giải tối ưu hơn DFS



9. Button lựa chọn Solve UCS

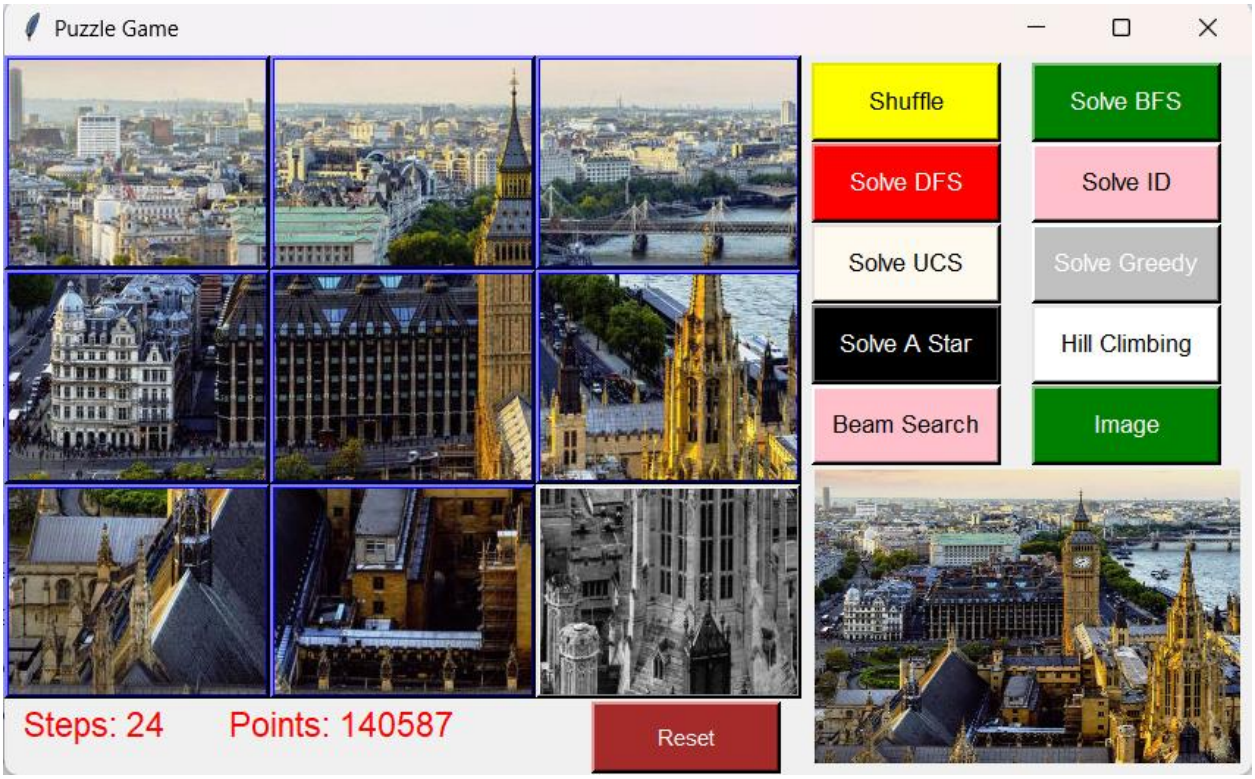
- Chọn Solve UCS sẽ gọi phương thức sau:

```
    def sloveucs(self):
        t=0;
        solution = self.ucs_puzzle(PuzzleState(self.puzzle))
        print("Cost: {}".format(solution.cost))
        for step, state in enumerate(solution.path):
            print(f"Step {step}:")
            for row in state:
                print(row)
                for i in range(3):
                    for j in range(3):
                        num = state[i][j]
                        if(num == 0):
                            t=1
                            self.move(i, j)
                            self.master.update()
                            break
                if t == 1:
                        t = 0
                        break
            time.sleep(0.1)

            print()
```

- Phương thức thực hiện chính và tìm đường đi cho lời giải puzzle bằng phương pháp UCS

```
    def ucs_puzzle(self,initial_state):
        queue = deque([initial_state])

        visited = set()  # Lưu trữ các trạng thái đã xem
        visited.add(tuple(map(tuple, initial_state.puzzle)))
        self.update_points()

        while queue:
            current_state = queue.popleft()

            if self.is_goal_state(current_state.puzzle):
                return current_state  # Nếu đạt được trạng thái mục tiêu, trả về kết
quả

            # Sinh các trạng thái kế tiếp và thêm vào queue
            for move in self.generate_moves(current_state):
                if tuple(map(tuple, move.puzzle)) not in visited:
                    queue.append(move)
                    visited.add(tuple(map(tuple, move.puzzle)))
                    self.update_points()
                    move.path = current_state.path + [move.puzzle]
                    move.cost = current_state.cost + 1
            sorted_queue_list = sorted(list(queue), key=lambda item: item.cost)
            queue = deque(sorted_queue_list)
        return None
```

- UCS sắp xếp lại hàng đợi theo chi phí, có thể sẽ thực hiện chậm hơn BFS do có hàm sort và số bước đi sẽ có thể không tối ưu hoặc tối ưu



10. Button lựa chọn Best first search (Greedy search)

- Hàm tính Heuristic

```
 def Heuristic(self,puzzle):
        h = 0
        for x in range(3):
            for y in range(3):
                v = puzzle[x][y] - 1
                if v == -1 :
                    v = 8
                y_g = v % 3
                x_g = v //3
                h = h + abs(x_g - x) + abs(y_g - y)
        return h
```

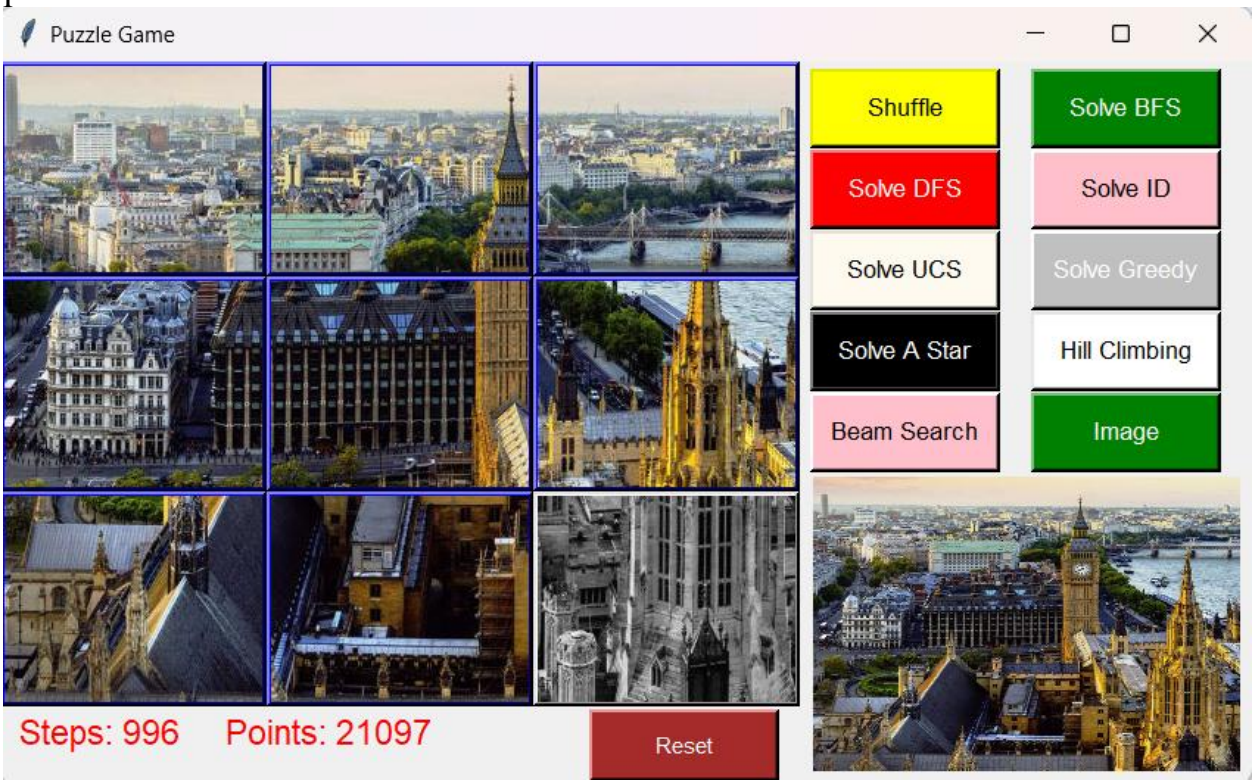
- Chọn Solve Greedy sẽ gọi phương thức sau:

```
    def slovegreedy(self):
        result = self.greedy_search(PuzzleState(self.puzzle))
        if result is None:
            messagebox.showerror("Sorry", "Số bước lớn hơn 1000 nên không thể tìm
dược !!")
            return
        t = 0
        for step, state in enumerate(result.path):
            print(f"Step {step}:")
            for row in state:
                for i in range(3):
                    for j in range(3):
                        num = state[i][j]
                        if(num == 0):
                            t=1
                            self.move(i, j)
                            self.master.update()
                            break
                if t == 1:
                        t = 0
                        break
            time.sleep(0.1)
```

- Phương thức thực hiện chính và tìm đường đi cho lời giải puzzle bằng phương pháp Greedy

```
    def greedy_search(self, initial_state, MAX_DEPTH = 1000):
            stack = [(initial_state)]  # Stack để lưu trữ các trạng thái cần xem xét

            visited = set()  # Lưu trữ các trạng thái đã xem

            while stack:
                current_state = stack.pop()  # Lấy trạng thái hiện tại từ stack

                if self.is_goal_state(current_state.puzzle):
                    return current_state  # Nếu đạt được trạng thái mục tiêu, trả về
kết quả

                if current_state.moves >= MAX_DEPTH: #giúp không bị treo máy
                    continue

                schedule = self.generate_moves(current_state)
                schedule = sorted(schedule , key=lambda x: self.Heuristic(x.puzzle),
reverse=True)
                # Sinh các trạng thái kế tiếp và thêm vào stack
                for move in schedule:
                    print(self.Heuristic(move.puzzle))
                    if tuple(map(tuple, move.puzzle)) not in visited:
                        stack.append(move)
                        visited.add(tuple(map(tuple, move.puzzle)))
                        self.update_points()
                        move.path = current_state.path + [move.puzzle]
                        move.setHeuristic(self.Heuristic(move.puzzle))
                print()

            return None
```

- Phương pháp Greedy sắp xếp thứ tự theo hàm Heuristic trước khi vào Stack nên phương pháp này sẽ nhanh hơn so với DFS nhưng sẽ không tối ưu trong giải puzzle



11. Button lựa chọn A* search

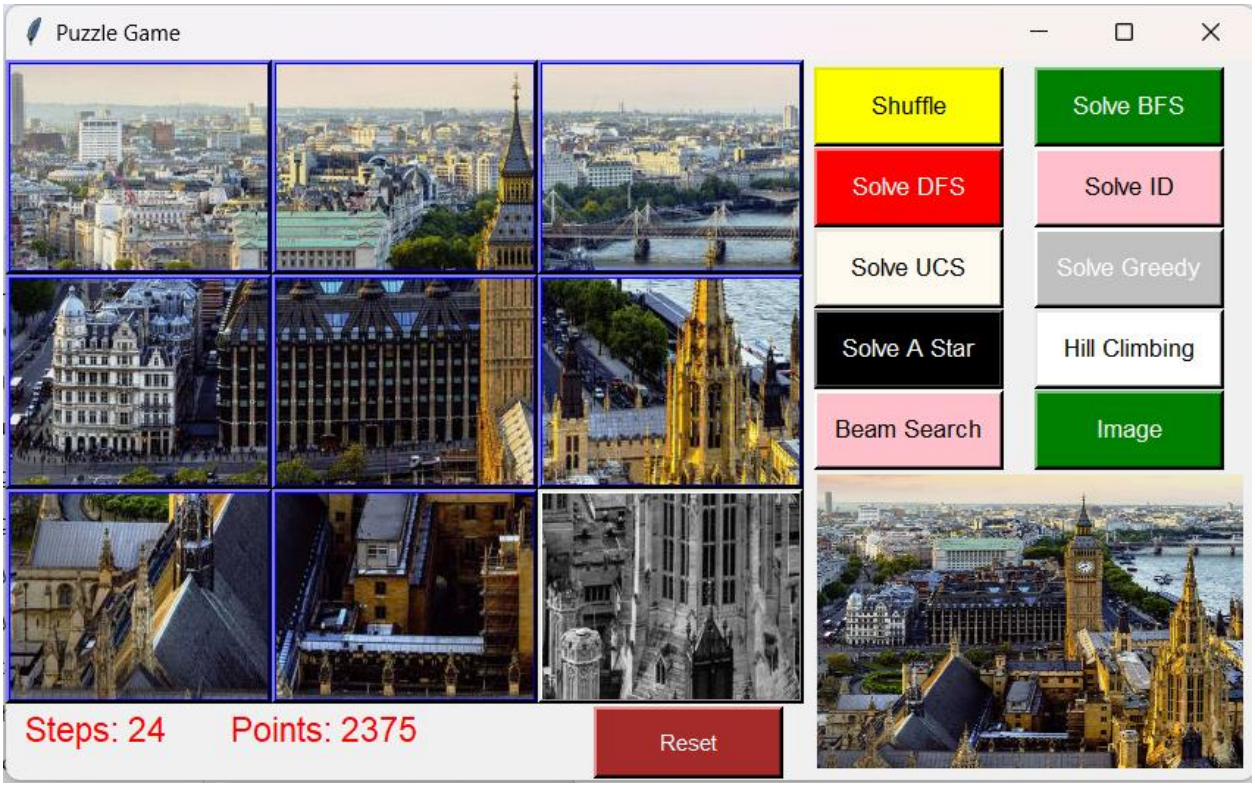
- Chọn Solve A Star sẽ gọi phương thức sau:

```
    def sloveAStar(self):
        t=0;
        solution = self.AStar_search(PuzzleState(self.puzzle))
        print("Priority: {}".format(solution.Heuristic))
        for step, state in enumerate(solution.path):
            print(f"Step {step}:")
            for row in state:
                print(row)
                for i in range(3):
                    for j in range(3):
                        num = state[i][j]
                        if(num == 0):
                            t=1
                            self.move(i, j)
                            self.master.update()
                            break
                if t == 1:
                        t = 0
                        break
            time.sleep(0.1)

            print()
```

- Phương thức thực hiện chính và tìm đường đi cho lời giải puzzle bằng phương pháp A Star

```
    def AStar_search(self, initial_state):
        queue = deque([initial_state])

        visited = set()  # Lưu trữ các trạng thái đã xem
        visited.add(tuple(map(tuple, initial_state.puzzle)))
        self.update_points()
        while queue:
            current_state = queue.popleft()

            if self.is_goal_state(current_state.puzzle):
                return current_state  # Nếu đạt được trạng thái mục tiêu, trả về kết
quả

            # Sinh các trạng thái kế tiếp và thêm vào queue
            for move in self.generate_moves(current_state):
                if tuple(map(tuple, move.puzzle)) not in visited:
                    queue.append(move)
                    visited.add(tuple(map(tuple, move.puzzle)))
                    self.update_points()
                    move.path = current_state.path + [move.puzzle]
                    move.cost = current_state.cost + 1
                    move.setHeuristic(self.Heuristic(move.puzzle))

            sorted_queue_list = sorted(list(queue), key=lambda item: item.Priority())
            queue = deque(sorted_queue_list)
        return None
```

- Phương pháp A Star sắp xếp thứ tự hàng đợi theo độ ưu tiên = Heuristic + Cost nên tốc độ thực hiện sẽ nhanh hơn và tìm đường đi sẽ tối ưu



12. Button lựa chọn Hill Climbing

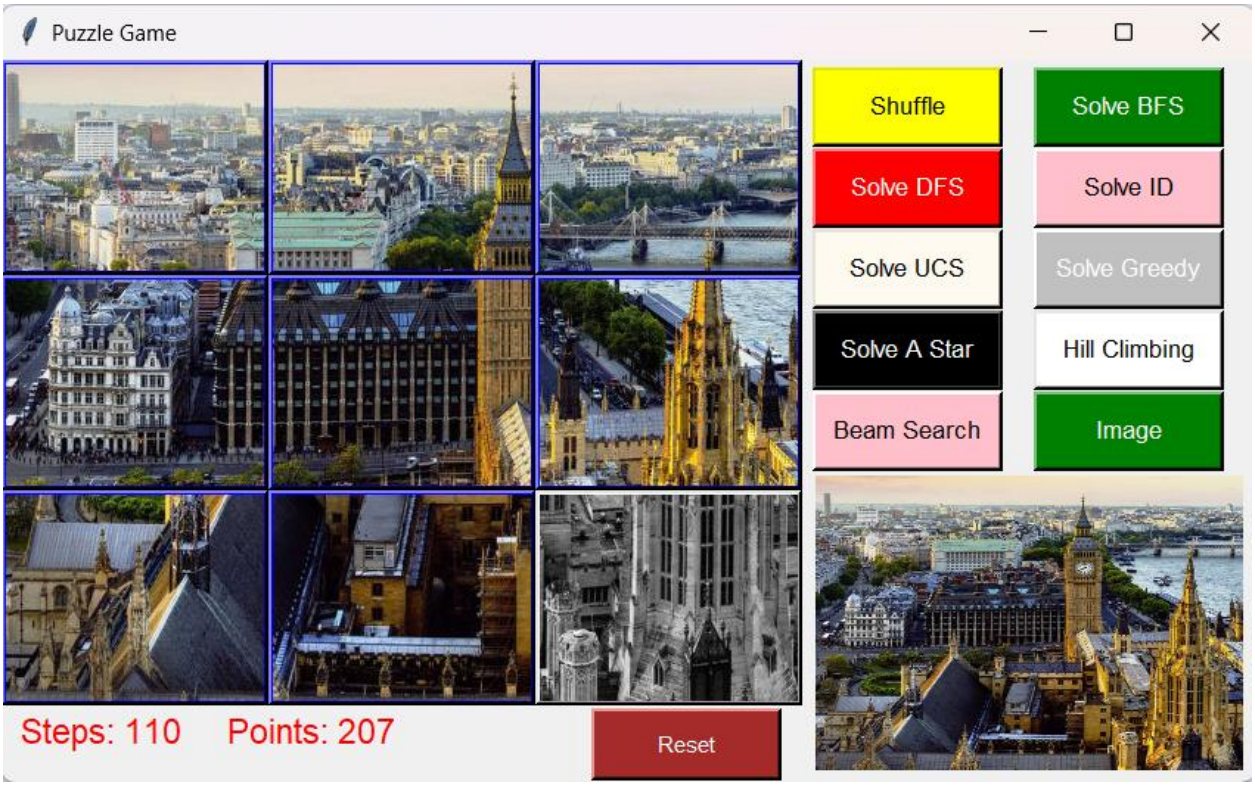
- Chọn Hill Climbing sẽ gọi phương thức sau:

```
    def sloveHillClimbing(self):
        result = self.HillClimbing(PuzzleState(self.puzzle))
        if result is None:
            messagebox.showerror("Sorry", "Không thể tìm được path với Hill
Climbing !!")
            return
        t = 0
        for step, state in enumerate(result.path):
            print(f"Step {step}:")
            for row in state:
                for i in range(3):
                    for j in range(3):
                        num = state[i][j]
                        if(num == 0):
                            t=1
                            self.move(i, j)
                            self.master.update()
                            break
                if t == 1:
                        t = 0
                        break
            time.sleep(0.1)
```

- Phương thức thực hiện chính và tìm đường đi cho lời giải puzzle bằng phương pháp Hill Climbing

```
    def HillClimbing(self, initial_state):
        stack_initial = [(initial_state)]
        visited = set()  # Lưu trữ các trạng thái đã xem
        self.update_points()
        while stack_initial:
            stack = [stack_initial.pop()]
            while stack:
                current_state = stack.pop()  # Lấy trạng thái hiện tại từ stack

                if self.is_goal_state(current_state.puzzle):
```

```
                return current_state  # Nếu đạt được trạng thái mục tiêu, trả về
kết quả

            # Sinh các trạng thái kế tiếp và thêm vào stack
            for move in self.generate_moves(current_state):
                if tuple(map(tuple, move.puzzle)) not in visited:
                    visited.add(tuple(map(tuple, move.puzzle)))
                    move.path = current_state.path + [move.puzzle]
                    if self.Heuristic(move.puzzle) <
self.Heuristic(current_state.puzzle):
                        stack.append(move)
                    else:
                        stack_initial.append(move)
                    self.update_points()

        return None
```



13. Button lựa chọn Beam Search

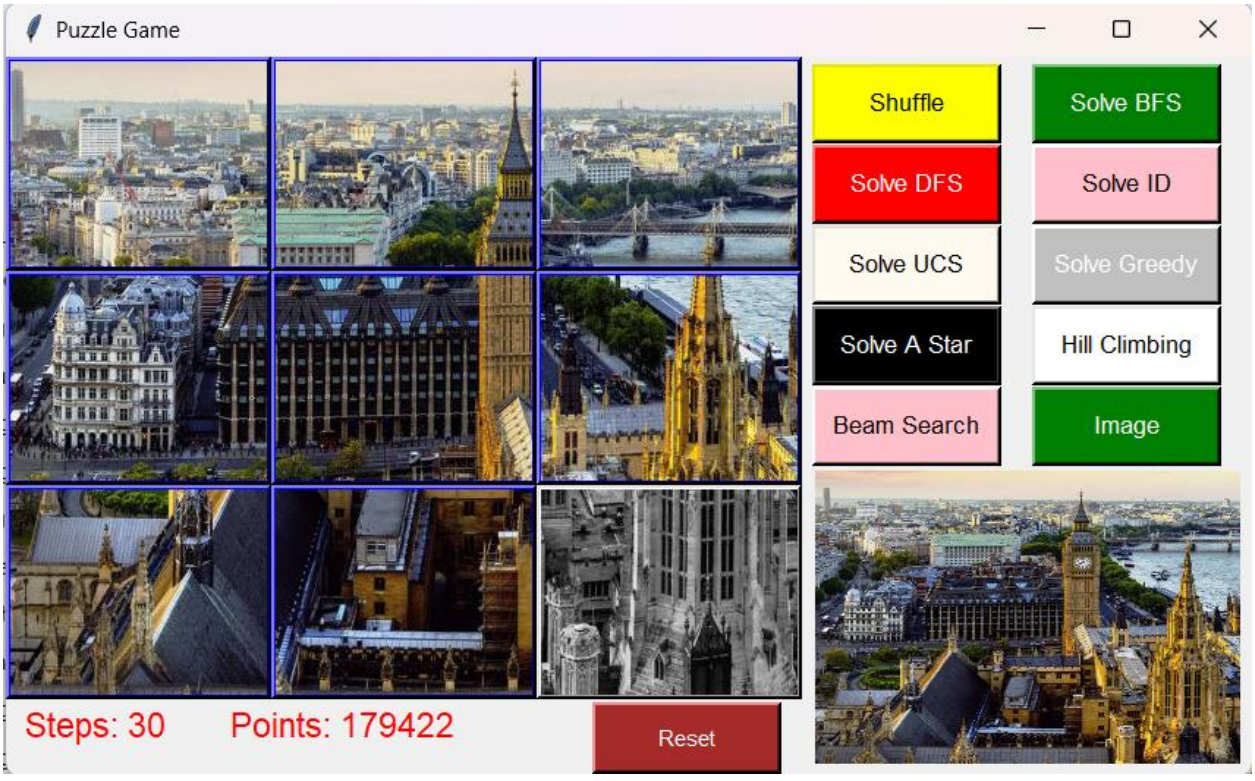
- Chọn Beam Search sẽ gọi phương thức sau:

```
    def sloveBeamSearch(self):
        result = self.BeamSearch(PuzzleState(self.puzzle),2)
        if result is None:
            messagebox.showerror("Sorry", "Không thể tìm được path với Beam
Search !!")
            return
        t = 0
        for step, state in enumerate(result.path):
            print(f"Step {step}:")
            for row in state:
                for i in range(3):
                    for j in range(3):
                        num = state[i][j]
                        if(num == 0):
                            t=1
                            self.move(i, j)
                            self.master.update()
                            break
                if t == 1:
                        t = 0
```

```
                    break
            time.sleep(0.1)
```

- Phương thức thực hiện chính và tìm đường đi cho lời giải puzzle bằng phương pháp Beam Search

```
    def BeamSearch(self, initial_state, k):
        queue = deque([initial_state])

        visited = set()  # Lưu trữ các trạng thái đã xem
        visited.add(tuple(map(tuple, initial_state.puzzle)))
        self.update_points()
        while queue:
            k_loop = k
            current_state = queue.popleft()

            if self.is_goal_state(current_state.puzzle):
                return current_state  # Nếu đạt được trạng thái mục tiêu, trả về kết
quả

            schedule = self.generate_moves(current_state)
            schedule = sorted(schedule , key=lambda x: self.Heuristic(x.puzzle),
reverse=True)
            # Sinh các trạng thái kế tiếp và thêm vào queue
            for move in schedule:
                if tuple(map(tuple, move.puzzle)) not in visited:
                    visited.add(tuple(map(tuple, move.puzzle)))
                    move.path = current_state.path + [move.puzzle]
                    self.update_points()
                    if k_loop > 0:
                        queue.append(move)
                        k_loop = k_loop - 1


        return None
```



D. KẾT LUẬN

1. Tự đánh giá kết quả

Sản phẩm cuối cùng nhìn chung đã đáp ứng được những yêu cầu cơ bản của một trò chơi Puzzle là chế độ người chơi. Ngoài ra còn có thể tìm lời giải cho puzzle bằng cách sử dụng các thuật toán và để cho máy chơi.

Tự đánh giá về ưu điểm và nhược điểm:

- Ưu điểm:

+ Giao diện thiết kế hợp lý và màu sắc hài hòa

+ Mã nguồn được viết tương đối chặt chẽ và có chia thành từ phần khác nhau

+ Chức năng của trò chơi tương đối khá đầy đủ đáp ứng nhu cầu của một trò chơi

+ Áp dụng được các thuật toán để tìm lời giải cho trò chơi

+ Sử dụng 2 ngôn ngữ lập trình: C#, Python

- Nhược điểm:

+ Chưa tách biệt được hoàn toàn phần giao diện với phần logic

+ Chưa thể tối ưu hóa được hoàn toàn code mã nguồn

2. Định hướng phát triển

Game Puzzle này còn có thể phát triển thêm các hướng sau:

- Nâng cấp giao diện
- Thêm chế độ chơi đối kháng và chơi online qua internet
- Có thể áp dụng tất cả thuật toán tìm lời giải trên cả puzzle 4x4

3. Lời kết

Đồ án "Xây dựng game Puzzle" là sự vận dụng kết hợp giữ các môn học khác nhau như: Lập trình hướng đối tượng, Lý thuyết đồ thị, Lập trình trên windows, Cấu trúc dữ liệu và giải thuật, ... để hoàn thành trò chơi

Ứng dụng trò chơi giúp hỗ trợ cho việc tư duy logic, tạo ra không gian xếp hình đa dạng giúp trẻ em lẫn người lớn có thể phát triển tư duy của mình. Ngoài ra, còn có thể nâng cao khả năng tưởng tượng không gian, khả năng dự doán và tìm tồi. Puzzle vừa có thể giải trí, vừa có thể tăng khả năng tư duy và phù hợp với mọi lứa tuổi.

==========HẾT==========