1. Cách sử dụng game:

Các chức năng hiện tại trong game

- Tìm đường đi: Click chuột trái vào ô bất kì để xác định điểm đích(ko phải chướng ngại vật)

- Thêm hoặc xoá chướng ngại vật: Click chuột phải vào ô bất kì (nhân vật tự động tìm đường đi mới đến đích)

- DrawLine: Mô phỏng đường đi đến đích (Vẽ 1 đường màu xanh từ vị trí đang đứng đến đích)

- Bật tắt DrawLine: Phím C

- Bật tắt mô phỏng cách vận hành thuộc toán: phím spce(từng bước), phím enter(tự động)

- Chuyển đổi kiểu di chuyển: có 2 kiểu di chuyển, chuyển đổi bằng phím V

  + Kiểu 1: Chỉ đi thẳng và ngang

  + Kiểu 2: Thêm di chuyển theo đường chéo

2. Giải thích chung về trò chơi:

- Thuật toán sử dụng A*:

3. Giải thích chi tiết cách các chức năng vận hành

3.1 Lớp grid (lưới)

- Lớp này sẽ có các thuộc tính là chiều chiều dài lưới, chiều rộng lưới, độ lớn của ô, toạ độ bắt đầu của lưới và mảng 2 chiều đóng vai trò lưu trữ các nút(node)

```
private int width;
private int height;
private float cellSize;
private Vector3 originPosition;
private TGridObject[,] gridArray;
```

- Có thể bật tắt toạ độ ô bằng cách thay đổi giá trị showDebug (đang để là false)

```
47    bool showDebug = false;
```

3.2 Lớp PathNode(nút)

- Lớp này có các thuộc tính về vị trí, về giá trị gCost, hCost, fCost, trạng thái isWalkable(đã được duyệt hay chưa), cameFromNode lưu giữ node trước đó.

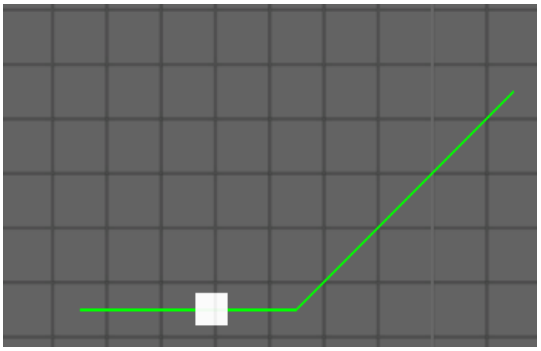
```
17  public class PathNode {
18
19      private Grid<PathNode> grid;
20      public int x;
21      public int y;
22
23      public int gCost;
24      public int hCost;
25      public int fCost;
26
27      public bool isWalkable;
28      public PathNode cameFromNode;
```

- fCost = gCost + hCost tính bằng phương thức CalculatorFCost

```
37      public void CalculateFCost() {
38          fCost = gCost + hCost;
39      }
```

3.3 Tìm đường đi:

- Trong lớp PathFinding có 2 chỉ số MOVE_STRAIGHT_COST và MOVE_DIAGONAL_COST thể hiện khoảng cách giữa 2 ô theo đường thẳng và theo đường chéo, với 2 ô cạnh nhau có khoảng cách là 10 và 2 ô chéo nhau có khoảng cách là 14. Các giá trị này được dùng để tính giá thành đường đi giữa 2 nút.



- Khởi tạo 3 thuộc tính gồm: grid, openList, closedList:

+ grid: lưới lưu trữ tất cả các node;

+ openList: lưu trữ các nút mở;

+ closedList: lưu trữ các nút đóng;

```
26      private Grid<PathNode> grid;
27      private List<PathNode> openList;
28      private List<PathNode> closedList;
```

- Trong phương thức FindPath() sẽ truyền vào các tham số là toạ độ vị trí bắt đầu và toạ độ vị trí kết thúc, khởi tạo vị trí bắt đầu(startNode) và vị trí kết thúc(endNode):

```
55    public List<PathNode> FindPath(int startX, int startY, int endX, int endY) {
56        PathNode startNode = grid.GetGridObject(startX, startY);
57        PathNode endNode = grid.GetGridObject(endX, endY);
```

+ Gán giá trị vào startNode với gCost = 0, hCost = khoảng cách ước tính đến endNode, và tính fCost = gCost + hCost

```
76        startNode.gCost = 0;
77        startNode.hCost = CalculateDistanceCost(startNode, endNode);
78        startNode.CalculateFCost();
```

```
37    public void CalculateFCost() {
38        fCost = gCost + hCost;
39    }
```

- Bắt đầu duyệt qua các node trong openList:

+ Đầu tiên lấy gán currentNode = node có fCost thấp nhất:

```
83        while (openList.Count > 0) {
84            PathNode currentNode = GetLowestFCostNode(openList);
```

```
200   private PathNode GetLowestFCostNode(List<PathNode> pathNodeList) {
201       PathNode lowestFCostNode = pathNodeList[0];
202       for (int i = 1; i < pathNodeList.Count; i++) {
203           if (pathNodeList[i].fCost < lowestFCostNode.fCost) {
204               lowestFCostNode = pathNodeList[i];
205           }
206       }
207       return lowestFCostNode;
208   }
```

+ Nếu currentNode = endNode thì return Path

+ openList xoá đi currentNode

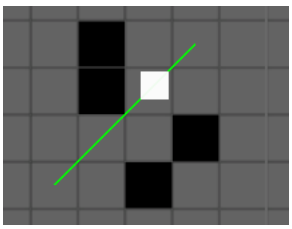
+ closedList thêm currentNode vào

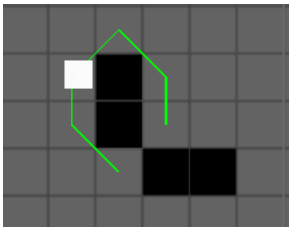
+ tìm kiếm node hàng xóm của currentNode để cho vào openList (Ở đây sẽ bỏ qua những node hàng xóm đã được duyệt)

+ gán giá trị cho node hàng xóm

```
102           int tentativeGCost = currentNode.gCost + CalculateDistanceCost(currentNode, neighbourNode);
103           if (tentativeGCost < neighbourNode.gCost) {
104               neighbourNode.cameFromNode = currentNode;
105               neighbourNode.gCost = tentativeGCost;
106               neighbourNode.hCost = CalculateDistanceCost(neighbourNode, endNode);
107               neighbourNode.CalculateFCost();
```

- Tìm kiếm node hàng xóm: Tính theo 2 kiểu gồm chỉ đi thẳng và có thể đi chéo(trường hợp đi chéo sẽ xét thêm điều kiện không đi qua được vị trí ở góc chéo nếu có 2 ô chặn ở 2 đỉnh:





```
121     private List<PathNode> GetNeighbourList(PathNode currentNode) {
122         List<PathNode> neighbourList = new List<PathNode>();
123 
124         if (currentNode.x - 1 >= 0) {
125             // Left
126             neighbourList.Add(GetNode(currentNode.x - 1, currentNode.y));
127             // Left Down
128             if (currentNode.y - 1 >= 0 && isWalkDiagonally)
129             {
130                 if (GetNode(currentNode.x - 1, currentNode.y).isWalkable || GetNode(currentNode.x, currentNode.y - 1).isWalkable)
131                 {
132                     neighbourList.Add(GetNode(currentNode.x - 1, currentNode.y - 1));
133                 }
134             }
135 
136             // Left Up
137             if (currentNode.y + 1 < grid.GetHeight() && isWalkDiagonally)
138             {
139                 if (GetNode(currentNode.x - 1, currentNode.y).isWalkable || GetNode(currentNode.x, currentNode.y + 1).isWalkable)
140                 {
141                     neighbourList.Add(GetNode(currentNode.x - 1, currentNode.y + 1));
142                 }
143             }
144 
145         }
```

```
146         if (currentNode.x + 1 < grid.GetWidth()) {
147             // Right
148             neighbourList.Add(GetNode(currentNode.x + 1, currentNode.y));
149             // Right Down
150             if (currentNode.y - 1 >= 0 && isWalkDiagonally)
151             {
152                 if (GetNode(currentNode.x + 1, currentNode.y).isWalkable || GetNode(currentNode.x, currentNode.y - 1).isWalkable)
153                 {
154                     neighbourList.Add(GetNode(currentNode.x + 1, currentNode.y - 1));
155                 }
156             }
157             // Right Up
158             if (currentNode.y + 1 < grid.GetHeight() && isWalkDiagonally)
159             {
160                 if (GetNode(currentNode.x + 1, currentNode.y).isWalkable || GetNode(currentNode.x, currentNode.y + 1).isWalkable)
161                 {
162                     neighbourList.Add(GetNode(currentNode.x + 1, currentNode.y + 1));
163                 }
164             }
165         }
166         // Down
167         if (currentNode.y - 1 >= 0) neighbourList.Add(GetNode(currentNode.x, currentNode.y - 1));
168         // Up
169         if (currentNode.y + 1 < grid.GetHeight()) neighbourList.Add(GetNode(currentNode.x, currentNode.y + 1));
170 
171         return neighbourList;
172     }
```

3.4 Nhân vật di chuyển(lớp CharacterMove)

- Thay đổi tốc độ di chuyển

```
6   public class CharacterMove : MonoBehaviour
7   {
8       private const float speed = 40f;
```

- Trong hàm update sẽ tự động di chuyển nhân vật theo path và cập nhật path mới nếu click chuột trái

```
16      private void Update()
17      {
18          HandleMovement();
19
20          if (Input.GetMouseButtonDown(0))
21          {
22              SetTargetPosition(UtilsClass.GetMouseWorldPosition());
23          }
24      }
```

- Trong hàm di chuyển có thể thay đổi hướng nhân vật bằng cách lấy giá trị x, y của moveDir, giá trị này thay đổi trong khoảng 0 – 1

```
29      private void HandleMovement()
30      {
31          if (pathVectorList != null)
32          {
33              Vector3 targetPosition = pathVectorList[currentPathIndex];
34              if (Vector3.Distance(transform.position, targetPosition) > 1f)
35              {
36                  Vector3 moveDir = (targetPosition - transform.position).normalized;
37
38                  Debug.Log("Toạ độ di chuyển: " + moveDir);
39                  float distanceBefore = Vector3.Distance(transform.position, targetPosition);
40                  transform.position = transform.position + moveDir * speed * Time.deltaTime;
41              }
42              else
43              {
```

```
[17:35:14] Toạ độ di chuyển: (1.00, 0.00, 0.00)
UnityEngine.Debug:Log (object)
[17:35:14] Toạ độ di chuyển: (1.00, 0.00, 0.00)
UnityEngine.Debug:Log (object)
[17:35:14] Toạ độ di chuyển: (1.00, 0.00, 0.00)
UnityEngine.Debug:Log (object)
[17:35:14] Toạ độ di chuyển: (1.00, 0.00, 0.00)
UnityEngine.Debug:Log (object)
[17:35:14] Toạ độ di chuyển: (1.00, 0.00, 0.00)
UnityEngine.Debug:Log (object)
[17:35:14] Toạ độ di chuyển: (0.73, 0.68, 0.00)
UnityEngine.Debug:Log (object)
[17:35:14] Toạ độ di chuyển: (0.73, 0.68, 0.00)
UnityEngine.Debug:Log (object)
```

3.5 Lớp Testing

- Lớp này chứa các hành động chức năng như bật tắt các đường vẽ, thay đổi kiểu đi, vẽ đường đến đích bằng cách nhận input từ bàn phím và chuột

```
Unity Message | 0 references
private void Update() {
    if (Input.GetMouseButtonDown(0)) {
        endPoint = UtilsClass.GetMouseWorldPosition();
        drawLineEndPoint();
        characterPathfinding.SetTargetPosition(endPoint);
    }

    if (Input.GetMouseButtonDown(1)) {
        mouseWorldPosition = UtilsClass.GetMouseWorldPosition();
        pathfinding.GetGrid().GetXY(mouseWorldPosition, out int x, out int y);
        pathfinding.GetNode(x, y).SetIsWalkable(!pathfinding.GetNode(x, y).isWalkable);
        drawLineEndPoint();
        characterPathfinding.SetTargetPosition(endPoint);
    }

    if(Input.GetKeyDown(KeyCode.C))
    {
        isDraw = !isDraw;
    }

    if(Input.GetKeyDown(KeyCode.V))
    {
        pathfinding.ReDiagonally();
    }
}
```