通識計算機程式設計期末考參考解答

6/19/2020

1. 參考圖1的 UML 類別圖，撰寫*C#*類別 **Puck** 及應用此一類別的主程式，模仿冰上曲棍球所用圓碟外形球在冰上的移動：(假設 **using System;** 敘述已經包含於程式之中)

(a) 宣告私有成員變數 **radius**、**x**、**y**，三者都是 **float** 型別。其中 **radius** 代表圓碟半徑，(**x,y**)是球中心的位置座標。(3%)

Ans.

```
class Puck
{
   private float radius;
   private float x;
   private float y;
}
```

(b) 宣告並實作公有建構式，輸入參數 **radius**、**x**、**y** 為 **float** 型別，與成員變數同名。 (3%)

Ans.

```
public Puck(float radius, float x, float y)
{
   this.radius = radius;
   this.x = x;
   this.y = y;
}
```

(c) 宣告並實作公有屬性(property) **xPosition** 及 **yPosition**，分別以 **get** 傳回成員變數 **x** 和 **y** 之值。注意不要實作 **set** 的部分。 (3%)

Ans.

```
public float xPosition
{
   get { return x; }
}

public float yPosition
{
   get { return y; }
}
```

(d) 宣告並實作私有成員函式 **xTranslate**，其輸入為 **float** 參數 **delta_x**，用以將球中心的 *x* 座標，平移(translate) **delta_x**，成為原先的 **x** 加上 **delta_x**。這模擬圓碟球沿著 *x* 方向的運動。用類似的寫法，完成另一私有成員函式 **yTranslate** ，模擬圓碟球沿 *y* 方向的運動。(3%)

Ans.

```
private void xTranslate(float delta_x)
{
    x += delta_x;
}

private void yTranslate(float delta_y)
{
    y += delta_y;
}
```

(e) 宣告並實作公有成員函式 **Move**，其參數為 *x* 和 *y* 方向的速率 **x_speed** 與 **y_speed**，以及時間增加量 **delta_t**。運用中間變數 **delta_x = x_speed * delta_t** 和 **delta_y = y_speed * delta_t**，配合函式 **xTranslate** 和 **yTranslate**，將圓碟球中心座標由(**x, y**)移到(**x+delta_x, y+delta_y**)。(3%)

Ans.

```
public void Move(
   float x_speed, float y_speed, float delta_t)
{
    float delta_x = x_speed * delta_t;
    float delta_y = y_speed * delta_t;
    xTranslate(delta_x);
    yTranslate(delta_y);
}
```

(f) 撰寫一段主程式，建立一個半徑 **1**，位在原點的圓碟球物件 **puck**。接著用一個 **for** 迴圈，讓迴圈變數 **n** 由 **0** 到 **2**，對應時間從 **0** 到 **2*delta_t**，**delta_t=0.01**。同時使 **puck** 位置沿 *x* 方向，以 **1.0** 的速率運動(*y* 方向的速率為 **0**)。在迴圈內加上 **Console.WriteLine** 敘述，使程式執行時的螢幕輸出如圖 2 所示。(3%)

Ans.

```
class Program
{
    static void Main(string[] args)
    {
        Puck puck = new Puck(1.0f, 0.0f, 0.0f);
        for(int n = 0; n < 3; ++n)
        {
            puck.Move(1.0f, 0.0f, 0.01f);
            Console.WriteLine("n = {0}, position =({1}, {2})",
                n, puck.xPosition, puck.yPosition);
        }
        Console.WriteLine();

        Console.WriteLine("按 enter 或 retuen 鍵結束");
        Console.ReadLine();
    }
}
```

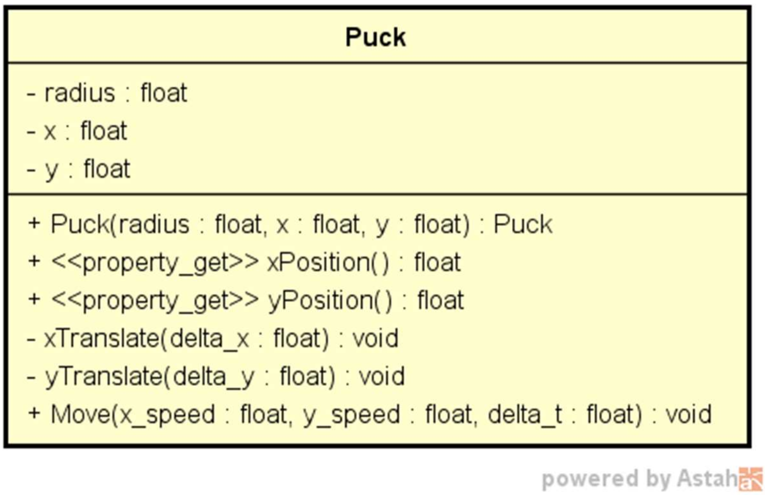
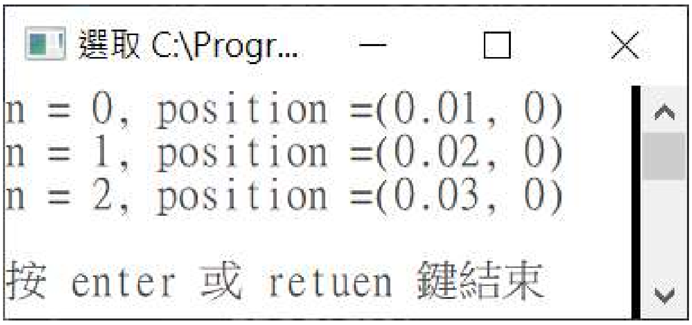

Puck
- radius : float
- x : float
- y : float
+ Puck(radius : float, x : float, y : float) : Puck
+ <<property_get>> xPosition() : float
+ <<property_get>> yPosition() : float
- xTranslate(delta_x : float) : void
- yTranslate(delta_y : float) : void
+ Move(x_speed : float, y_speed : float, delta_t : float) : void
powered by Astah

圖 1. 圓碟球對應的 UML 類別圖



```
n = 0, position =(0.01, 0)
n = 1, position =(0.02, 0)
n = 2, position =(0.03, 0)

按 enter 或 retuen 鍵結束
```

圖 2. 第 1 題程式碼，執行時的主控台螢幕畫面

2. 參考圖3的 UML 類別圖，撰寫*C#*介面 **IoTObject** 與類別 **Thermometer** 及應用此一類別的主程式，模仿物聯網(Internet of Things，IoT)物件及溫度計：(假設 **using System;** 敘述已經包含於程式之中)

(a) 宣告介面 **IoTObject**，包含成員函式 **SetUp** 及 **Loop**，代表物件建立及連續運作。(3%)

Ans.

```
interface IoTObject
{
    void SetUp();
    void Loop();
}
```

(b) 宣告類別 **Thermometer**，實作介面 **IoTObject**。同時宣告私有 **float** 成員變數 **temperature**、**int** 成員變數 **n**、**Random** 物件 **random**。不必實作 **Thermometer** 的建構式。 (3%)

Ans.

```
class Thermometer : IoTObject
{
    float temperature;
    int n;
    Random random;
}
```

(c) 實作 **Thermometer** 成員函式 **SetUp**、**Loop**、**Tick**。 **SetUp** 的實際功能應該包含建立與其他物聯網物件的聯繫，此處我們只在其中初始化 **random**，並設 **n = 0**。**Loop** 則應該每隔一個很小的時間間距，就量一次溫度。這裡我們讓它呼叫私有成員函式 **Tick**，在 **Tick** 中讓 **n** 加 **1**，並且以 **random** 隨機產生一個溫度，模仿溫度計的運作。(3%)

Ans.

```
public void SetUp()
{
    // determine spatial positions
    // Set transmission rate
    // set communication protocol
    random = new Random(777);
    n = 0;
}

public void Loop()
{
    // measure temperature every delta_t
    Tick();
}

private void Tick()
{
    ++n;
    float dev = (float) random.NextDouble();
    temperature = 30.0f + 5.0f*(dev-0.5f);
}
```

(d) 假設一共擺放 5 個溫度計。寫一個主程式，令 **n** 由 **0** 變化到 **4**，計算並輸出每一個 **n** 值對應的 5 個溫度計所測溫度的平均。程式執行時的主控

台螢幕畫面如圖 4。假定屬性 **Temperature** 已經寫好。(3%)

Ans.

```
class Program
{
   static void Main(string[] args)
   {
      const int N_THERMOMETERS = 10;
      Thermometer[]thermometers =
         new Thermometer[N_THERMOMETERS];
      for(int i = 0; i < N_THERMOMETERS; ++i)
      {
         thermometers[i] = new Thermometer();
         thermometers[i].SetUp();
      }
      const int N_TIME_STEPS = 5;
      float average_temperatures;
      for(int n = 0; n < N_TIME_STEPS; ++n)
      {
         average_temperatures = 0.0f;
         float sum_temperatures = 0.0f;
         for(int i = 0; i < N_THERMOMETERS; ++i)
         {
            thermometers[i].Loop();
            sum_temperatures += thermometers[i].Temperature;
         }
         average_temperatures =
             sum_temperatures/(float) (N_THERMOMETERS);
         Console.WriteLine("n = {0}, average temperature = {1}",
             n, average_temperatures);
      }

      Console.WriteLine("按 enter 或 retuen 鍵結束");
      Console.ReadLine();
   }
}
```

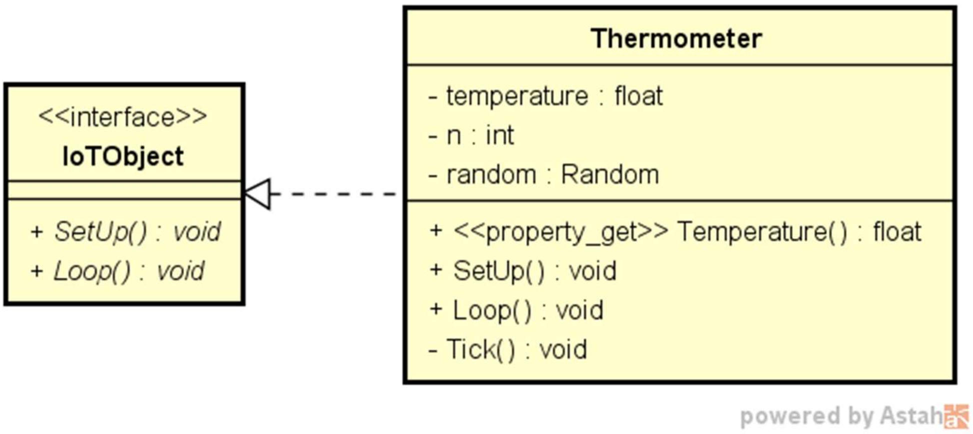
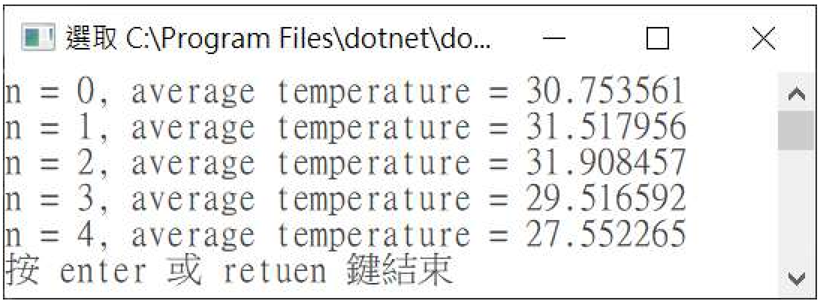


圖 3. 第 2 題概念對應的 UML 類別圖



```
n = 0, average temperature = 30.753561
n = 1, average temperature = 31.517956
n = 2, average temperature = 31.908457
n = 3, average temperature = 29.516592
n = 4, average temperature = 27.552265
按 enter 或 retuen 鍵結束
```

圖 4. 第 2 題程式碼，執行時的主控台螢幕畫面

3. 找出以下程式片段之錯誤，說明錯誤原因，並盡量以最簡潔、保留原意，又不易再出錯的方式更正。

(a) (3%) 一個語法錯誤

```
class StatisticalFunctions {
  public double Normal(double x, double mu, double sigma)  {
    double z = (x - mu)/sigma;
    return Math.Exp(-z*z/2.0);
  }
}
class Program {
  static void Main(string[] args) {
     double y = StatisticalFunctions.Normal(0.5, 0.0, 1.0);
  }
}
```

Ans.

以類別名稱呼叫其成員函式，此一成員函式必須宣告為 **static**，而且函式中不能用到類別的非 **static** 成員變數。本題用意是收集統計常用的函式到一個類別之中，類別中不必宣告使用成員變數，就像是 **C#**中的類別 **Math**。所以本題較好的修改是將函式 **Normal** 宣告為 **static**。原程式更正如下:

```
class StatisticalFunctions {
public static double Normal(double x, double mu, double sigma)  {
    double z = (x - mu)/sigma;
    return Math.Exp(-z*z/2.0);
  }
 }
 class Program {
  static void Main(string[] args) {
     double y = StatisticalFunctions.Normal(0.5, 0.0, 1.0);
  }
 }
```

(b) (3%) 一個語法錯誤

```
class IntegerPair {
  private int i1;
  private int i2;
  public IntegerPair(int i1, int i2) {
    this.i1 = i1;
    this.i2 = i2;
  }
  public int Ratio() {
    int result = ((i1 == 0)? 0 : i2/i1);
    return;
  }
  public double Ratio() {
   double result =
     ((i1 == 0)? 0.0 : (double)(i1)/(double)(i2));
   return;
  }
}
```

Ans.

在 C# 語言中的函式多載(overloading)，函式以其 signature 分辨。函式的 signature 包含函式名稱及其呼叫參數的個數，對應型別與順序，但不包括其 return 的型別。本題的兩個 **Ratio** 函式，函式名稱及參數完全相同，所以 signature 一致，視為相同，變成函式重複宣告，造成錯誤。參酌題意，兩個函式應該改用不同名稱。程式更正的一例如下:

```
public int ~~Ratio~~ IntRatio()
{
    int result = ((i1 == 0)? 0 : i2/i1);
    return result;
}

public double ~~Ratio~~ DoubleRatio()
{
    double result =
       ((i1 == 0)? 0.0 : (double)(i1)/(double)(i2));
    return result;
}
```

(c) (3%) 一個語法錯誤。

```
class Animal {
  private string species;
  public Animal(string species) {
    this.species = species;
  }
}
class Ape : Animal {
  private string name;
  public Ape(string name) : base("Primate") {
    this.name = name;
  }
  public void Print() {
    Console.WriteLine(
      "An ape, " + name + ", is of species " + species);
  }
}
```

Ans.

類別 **Animal** 中的成員變數 **species** 宣告為 **private**，其子類別 **Ape** 也不能夠直接使用。考慮題意，Animal 的子類別應該要能夠取用 species，所以只要將 species 宣告為 protected 就可以。更正的程式如下:

```
class Animal {
  ~~private~~ protected string species;
  public Animal(string species) {
    this.species = species;
  }
}
class Ape : Animal {
  private string name;
  public Ape(string name) :  base("Primate") {
    this.name = name;
  }
  public void Print() {
    Console.WriteLine(
      "An ape, " + name + ", is of species " + species);
  }
}
```

(d) (3%) 一種語義錯誤 (以 **Debug.Assert** 敘述的要求為準)

```
class GameObject  {
  private int id;
  public GameObject(){}
  public int ID  {
     set {id = value;}
     get {return id;}
 }
 public GameObject(GameObject game_object) {
     id = game_object.id;
 }
}
class Program {
  static void Main(string[] args) {
    GameObject go1 = new GameObject();
```

```
        go1.ID = 1234;
        GameObject go2 = go1;
        go2.ID = 5678;
        Debug.Assert(go1.ID == 1234);
    }
}
```

Ans.

*C#*語言中，物件設值 **go2 = go1**，僅將 **go1** 的參考設為 **go2** 的參考，因此 **go1** 和 **go2** 經過相同的參考，具備完全相同的內容。當 **go2** 的成員變數 **id** 改為 **5678** 時，**go1** 的成員變數 **id** 也變成 **5678**，使 **Debug.Assert(go1.ID == 1234)** 不成立。解決的方法，是不要直接將物件設值(shallow copy)，而要將內容複製，並產生新的物件參考(deep copy)。因此，比較徹底的解法，就是使用 copy constructor。程式更正如下:

```
class GameObject
{
    private int id;
    public GameObject()
    {}

    public int ID
    {
        set {id = value;}
        get {return id;}
    }

    // Copy Constructor
    public GameObject(GameObject game_object)
    {
        id = game_object.id;
    }
}

class Program
{
    static void Main(string[] args)
    {
```

```
    GameObject go1 = new GameObject();
    go1.ID = 1234;
    ~~GameObject go2 = go1;~~
    GameObject go2 = new GameObject(go1);
    go2.ID = 5678;
    Debug.Assert(go1.ID == 1234);
  }
}
```

(e) (3%) 一種語法錯誤

```
abstract class Chatbot  {
  string message;
  string response;
  abstract public void SpeechToText();
  abstract public void Response();
  abstract public void TextToSpeech();
}
class Zenbot : Chatbot {
  public void SpeechToText() {
    Console.Write("Listening to a speaker, and");
    Console.WriteLine("transform the voice into text message");
  }
  public void Response() {
    Console.WriteLine("Generating a response");
  }
  public void TextToSpeech() {
    Console.WriteLine("Transform the response into voice");
  }
}
```

Ans.

抽象類別中的抽象成員函式，相當於虛擬(**virtual**)成員函式，所以其子類別實作這些函式時，必須加上關鍵字 **override**。程式更正如下:

```
abstract class Chatbot
{
    string message;
    string response;
    abstract public void SpeechToText();
    abstract public void Response();
    abstract public void TextToSpeech();
}

class Zenbot : Chatbot
{
    public override void SpeechToText()
    {
      Console.Write("Listening to a speaker, and");
      Console.WriteLine("transform the voice into text message");
    }

    public override void Response()
    {
        Console.WriteLine("Generating a response");
    }

    public override void TextToSpeech()
    {
        Console.WriteLine("Transform the response into voice");
    }
}
```

4. 試寫出下列程式的輸出。本題修改自 https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-3-tic-tac-toe-ai-finding-optimal-move/ (12%)

```
using System;
namespace Problem4 {
 enum Status {
   WIN, LOSE, NEUTRAL
 }
 struct Move {
   public int i0;
   public int j0;
   public Move(int i0, int j0) {
    this.i0 = i0;
    this.j0 = j0;
  }
}
class Board {
  private char[,] board;
  public Board() {
    board = new char[3, 3]
      { {' ', ' ', ' '},
        {' ', ' ', ' '},
        {' ', ' ', ' '} };
     }
  public bool IsVacant(int i, int j) {
    bool result = (board[i, j] == ' ');
    return result;
  }
  public bool HasVacancies() {
    bool hasVacancies = false;
    for (int i = 0; i < 3; ++i)  {
      for (int j = 0; j < 3; ++j) {
        if (board[i, j] == ' ') {
          hasVacancies = true;
          break;
        }
      }
    }
    return hasVacancies;
  }


  public bool PlaceMove(char piece, int i, int j) {
    bool success = false;
    if(IsVacant(i, j)) {
      board[i, j] = piece;
      success = true;
    }
    return success;
  }
```

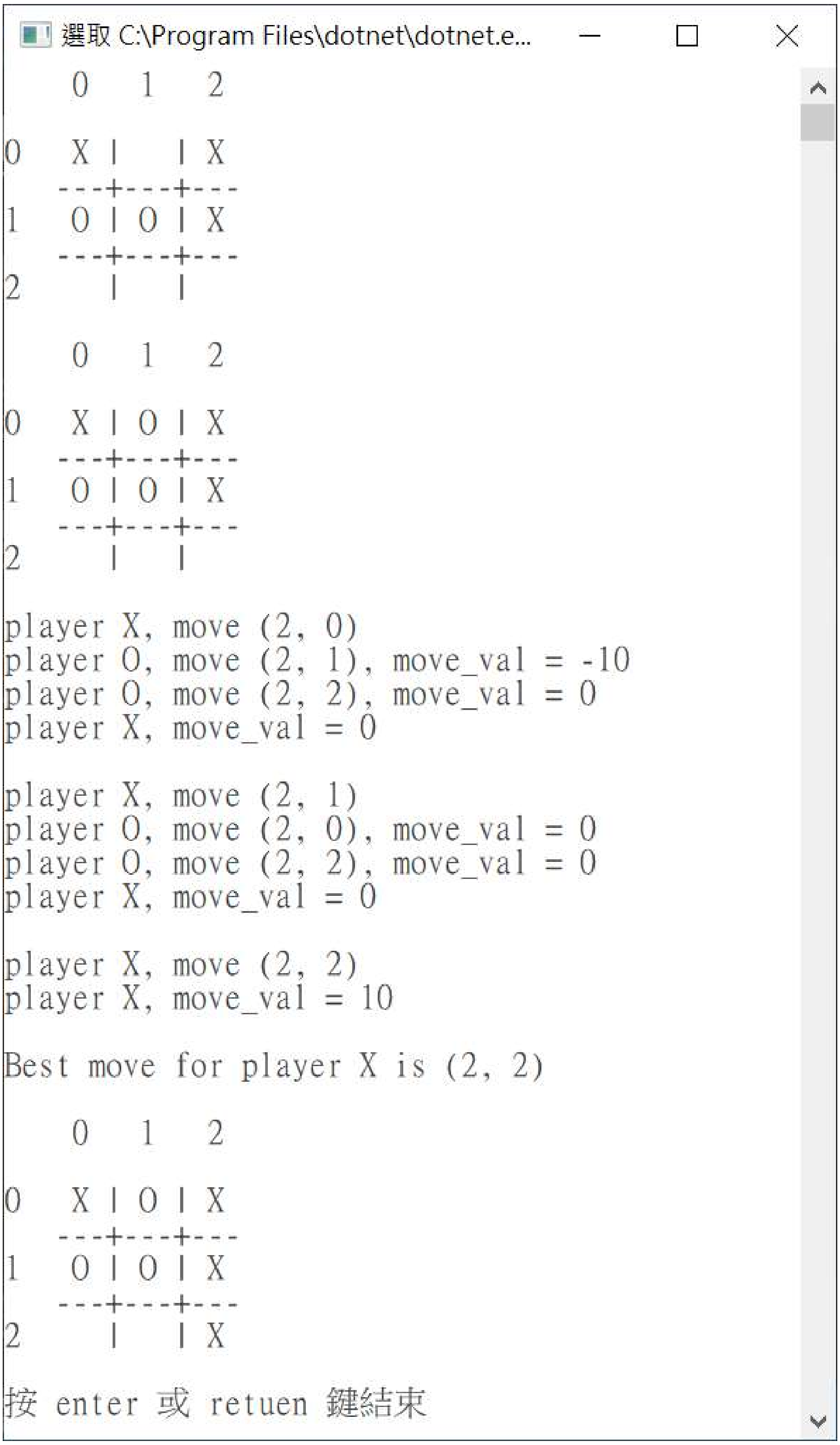
```
public void UndoMove(int i, int j) {
  board[i, j] = ' ';
}
public Status Check(char piece) {
   Status status = Status.NEUTRAL;
   for(int i = 0; i < 3; ++i) {
     if( (board[i, 0] == board[i, 1]) &&
          (board[i, 1] == board[i, 2]) ) {
       status = (board[i, 0] == piece) ?
        Status.WIN : Status.LOSE;
       break;
     }
    }
    if(status != Status.NEUTRAL) return status;
    for(int j = 0; j < 3; ++j){
     if( (board[0, j] == board[1, j]) &&
          (board[1, j] == board[2, j]) ) {
       status = (board[0, j] == piece) ?
        Status.WIN : Status.LOSE;
       break;
     }
    }
    if(status != Status.NEUTRAL) return status;
    if( (board[0, 0] == board[1, 1]) &&
        (board[1, 1] == board[2, 2]) )  {
      status = (board[0, 0] == piece) ?
        Status.WIN : Status.LOSE;
    }
    if(status != Status.NEUTRAL) return status;

    if( (board[0, 2] == board[1, 1]) &&
         (board[1, 1] == board[2, 0]) ) {
      status = (board[0, 2] == piece) ?
        Status.WIN : Status.LOSE;
      }
      return status;
    }
    public int Evaluate(char piece) {
      Status status = Check(piece);
      int score = 0;
      if(status == Status.WIN) {
         score = 10;
      } else if(status == Status.LOSE) {
         score = -10;
      } else {
         score = 0;
      }
        return score;
    }
    public void Display() {
      Console.WriteLine("    0   1   2 ");
      Console.WriteLine("              ");
      Console.WriteLine("0   {0} | {1} | {2} ",
          board[0, 0], board[0, 1], board[0, 2]);
      Console.WriteLine("   ---+---+---");
      Console.WriteLine("1   {0} | {1} | {2} ",
          board[1, 0], board[1, 1], board[1, 2]);
      Console.WriteLine("   ---+---+---");
      Console.WriteLine("2   {0} | {1} | {2} ",
          board[2, 0], board[2, 1], board[2, 2]);
      Console.WriteLine("              ");
    }
}
class Player {
    private char piece;
    private Board board;
    public Player(char piece, Board board) {
      this.piece = piece;
      this.board = board;
    }
    public char OpponentPiece() {
      char opponent_piece = 'O';
      if(piece == 'O') {
        opponent_piece = 'X';
      }
      return opponent_piece;
```

```
}
private int PlayerTurn() {
  int score = board.Evaluate(piece);
  if (score == 10) return score;
  if (score == -10) return score;
  if (!board.HasVacancies()) return 0;
  char opponent_piece = OpponentPiece();
  int best = -1000;
  for(int i = 0; i < 3; ++i) {
    for(int j = 0; j < 3; ++j) {
      bool success = board.PlaceMove(opponent_piece, i, j);
      if(success) {
        Console.Write("player {0}, move ({1}, {2}), ",
          opponent_piece, i, j);
        int opp_move_val = OpponentTurn();
        best = Math.Max(best, opp_move_val);
        board.UndoMove(i, j);
        Console.WriteLine("move_val = " + opp_move_val);
      }
    }
  }
  return best;
}
private int OpponentTurn() {
  char opponent_piece = OpponentPiece();
  int score = board.Evaluate(opponent_piece);
  return -score;
}


public Move BestMove() {
  int best_val = -1000;
  Move best_move = new Move(-1, -1);
  for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
      bool success = board.PlaceMove(piece, i, j);
      if(success) {
        Console.WriteLine(
            "player {0}, move ({1}, {2})", piece, i, j);
          int move_val = PlayerTurn();
          if (move_val > best_val) {
            best_move = new Move(i, j);
            best_val = move_val;
          }
          board.UndoMove(i, j);
          Console.WriteLine("player {0}, move_val = {1} ",
            piece,  move_val);
          Console.WriteLine();
        }
      }
    }
    return best_move;
   }
}
class Program {
   static void Main(string[] args) {
     Board board = new Board();
     Player player_X = new Player('X', board);
     board.PlaceMove('X', 0, 0);
     board.PlaceMove('O', 1, 1);
     board.PlaceMove('X', 1, 2);
     board.PlaceMove('O', 1, 0);
     board.PlaceMove('X', 0, 2);
     board.Display();
     board.PlaceMove('O', 0, 1);
     board.Display();
     Move best_move = player_X.BestMove();
     board.PlaceMove('X', best_move.i0, best_move.j0);
     Console.WriteLine("Best move for player X is ({0}, {1}) ",
          best_move.i0, best_move.j0);
     Console.WriteLine();
     board.Display();
     Console.WriteLine("按 enter 或 retuen 鍵結束");
     Console.ReadLine();
   }
 }
```

}

Ans.

參見以下主控台畫面截圖：



```
    0   1   2

0   X |   | X
   ---+---+---
1   O | O | X
   ---+---+---
2     |   |

    0   1   2

0   X | O | X
   ---+---+---
1   O | O | X
   ---+---+---
2     |   |

player X, move (2, 0)
player O, move (2, 1), move_val = -10
player O, move (2, 2), move_val = 0
player X, move_val = 0

player X, move (2, 1)
player O, move (2, 0), move_val = 0
player O, move (2, 2), move_val = 0
player X, move_val = 0

player X, move (2, 2)
player X, move_val = 10

Best move for player X is (2, 2)

    0   1   2

0   X | O | X
   ---+---+---
1   O | O | X
   ---+---+---
2     |   | X

按 enter 或 retuen 鍵結束
```

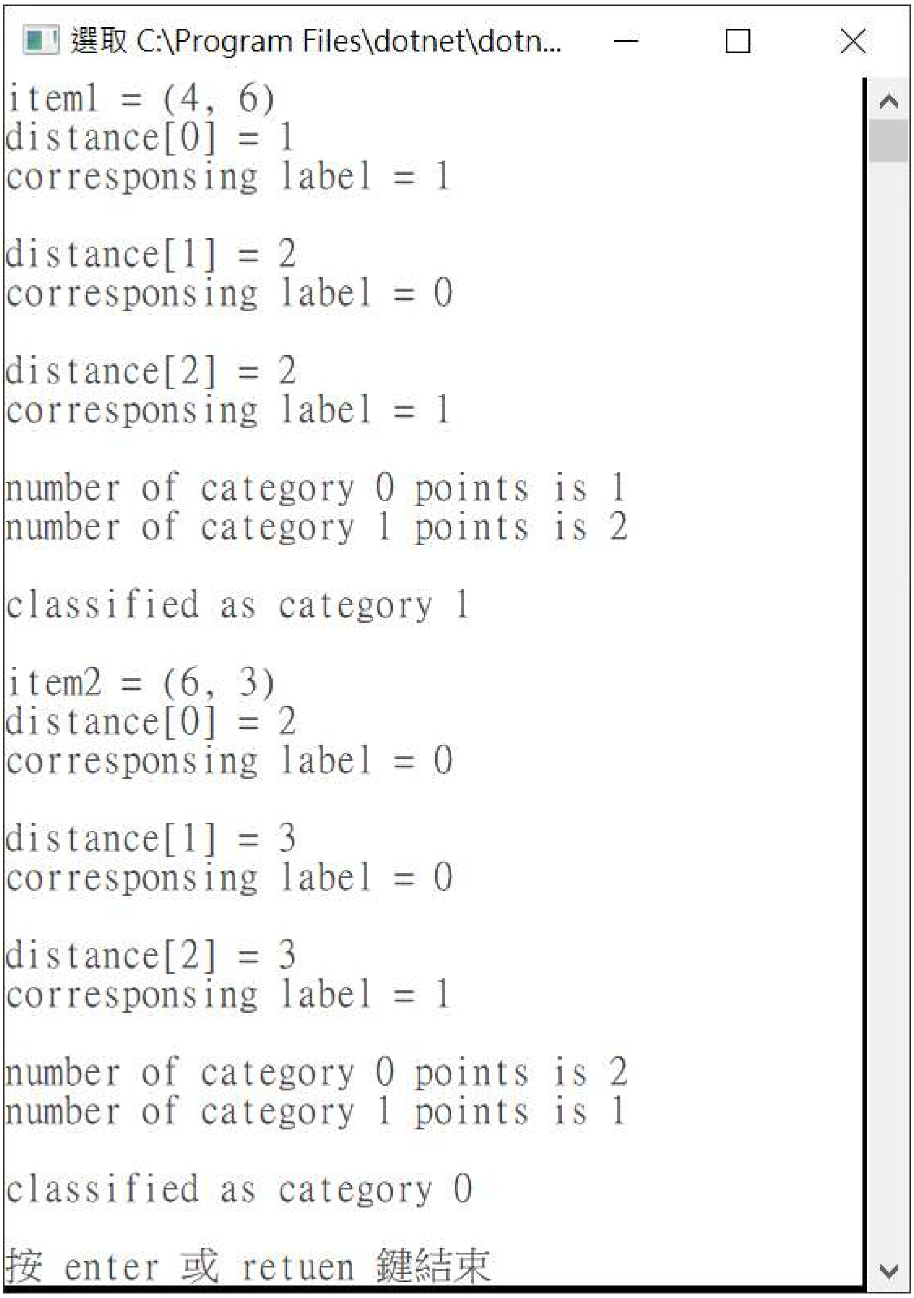
5.試寫出下列程式的輸出。本題修改自 https://docs.microsoft.com/zh-tw/archive/msdn-magazine/2019/may/test-run-weighted-k-nn-classification-using-csharp (12%)

```
using System;
namespace Problem5 {
  struct FeatureSpacePoint {
    public int feature1;
    public int feature2;
    public FeatureSpacePoint(int feature1, int feature2) {
      this.feature1 = feature1;
      this.feature2 = feature2;
    }
  }
  struct Data {
    public int feature1;
    public int feature2;
    public int label;
    public Data(int feature1, int feature2, int label) {
      this.feature1 = feature1;
      this.feature2 = feature2;
      this.label = label;
    }
  }
  class BinaryClassifier {
    int n_data;
    int k;
    Data[] data_set;

    public BinaryClassifier(int k) {
      this.k = k;
    }
    public void Train(Data[] data) {
      n_data = data.Length;
      data_set = new Data [n_data];
      for(int n = 0; n < n_data; ++n) {
        data_set[n].feature1 = data[n].feature1;
        data_set[n].feature2 = data[n].feature2;
    data_set[n].label = data[n].label;
  }
}
public int
  ManhattanDistance(FeatureSpacePoint item, Data data) {
  int dist = Math.Abs(item.feature1 - data.feature1) +
             Math.Abs(item.feature2 - data.feature2);
  return dist;
}
public void ComputeDistancesAndSetLabels(
  FeatureSpacePoint item, int[] distances, int[] labels) {
    for(int n = 0; n < n_data; ++n) {
      distances[n] = ManhattanDistance(item, data_set[n]);
      labels[n] = data_set[n].label;
    }
  }
  private int Vote(int[] distances, int[] labels) {
    Array.Sort(distances, labels);
    for(int n = 0; n < k; ++n) {
      Console.WriteLine("distance[{0}] = {1}", n, distances[n]);
      Console.WriteLine("corresponsing label = " + labels[n]);
      Console.WriteLine();
    }
    int n_label_0 = 0;
    int n_label_1 = 0;
    for(int n = 0; n < k; ++n) {
      if(labels[n] == 0) {
        ++n_label_0;
      } else {
        ++n_label_1;
      }
    }
    Console.WriteLine(
      "number of category 0 points is " + n_label_0);
    Console.WriteLine(
      "number of category 1 points is " + n_label_1);
    Console.WriteLine();
    int label = (n_label_0 > n_label_1) ? 0 : 1;
    return label;
  }
  public int Category(FeatureSpacePoint item) {
    int[] distances = new int[n_data];
    int[] labels = new int[n_data];
    ComputeDistancesAndSetLabels(item, distances, labels);
    int label = Vote(distances, labels);
    return label;
  }
}
class Program {
  static void Main(string[] args) {
    Data[] data = new Data[10];
    data[0] = new Data(2, 4, 0);
    data[1] = new Data(3, 3, 0);
    data[2] = new Data(3, 5, 0);
    data[3] = new Data(4, 1, 0);
    data[4] = new Data(4, 7, 1);
    data[5] = new Data(5, 2, 0);
    data[6] = new Data(6, 6, 1);
    data[7] = new Data(7, 8, 1);
    data[8] = new Data(8, 5, 1);
    data[9] = new Data(8, 7, 1);
    BinaryClassifier cls = new BinaryClassifier(3);
    cls.Train(data);
    FeatureSpacePoint item1 = new FeatureSpacePoint(4, 6);
    Console.WriteLine("item1 = ({0}, {1})",
      item1.feature1, item1.feature2);
    int label1 = cls.Category(item1);
    Console.WriteLine("classified as category " + label1);
    Console.WriteLine();
    FeatureSpacePoint item2 = new FeatureSpacePoint(6, 3);
    Console.WriteLine("item2 = ({0}, {1})",
      item2.feature1, item2.feature2);
    int label2 = cls.Category(item2);
    Console.WriteLine("classified as category " + label2);
    Console.WriteLine();
    Console.WriteLine("按 enter 或 retuen 鍵結束");
```

```
            Console.ReadLine();
          }
        }
      }
```

Ans.

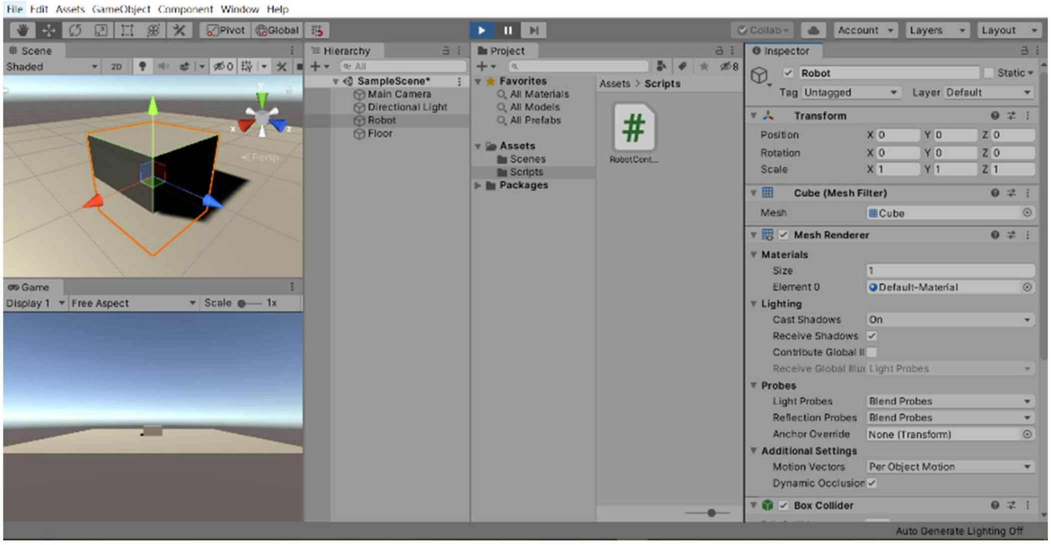
參見以下主控台畫面截圖：



```
item1 = (4, 6)
distance[0] = 1
corresponsing label = 1

distance[1] = 2
corresponsing label = 0

distance[2] = 2
corresponsing label = 1

number of category 0 points is 1
number of category 1 points is 2

classified as category 1

item2 = (6, 3)
distance[0] = 2
corresponsing label = 0

distance[1] = 3
corresponsing label = 0

distance[2] = 3
corresponsing label = 1

number of category 0 points is 2
number of category 1 points is 1

classified as category 0

按 enter 或 retuen 鍵結束
```

6. 依據以下描述，完成某一 Unity *C#* 腳本程式。 (6%)

程式描述：如圖 5，已建立好一個正方形碟狀物，稱為 **Robot**，代表第 7 題所討論的簡化吸塵器機器人(vacuum cleaner robot)。試撰寫其對應 *C#* 腳本，讓此一碟狀物，啟動遊戲後，如圖 6 所示，沒有外加推拉力量，仍能以設定的速率參數 **speed**，自行沿 *x* 軸(橫向)移動。



Robot ☑ □ Static
Tag Untagged Layer Default

Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Cube (Mesh Filter)
Mesh Cube

☑ Mesh Renderer
Materials
Size 1
Element 0 Default-Material
Lighting
Cast Shadows On
Receive Shadows ☑
Contribute Global Il □
Receive Global Illur Light Probes
Probes
Light Probes Blend Probes
Reflection Probes Blend Probes
Anchor Override None (Transform)
Additional Settings
Motion Vectors Per Object Motion
Dynamic Occlusion ☑

☑ Box Collider

圖 5. 在 Unity Editor 所建立的簡化吸塵器機器人內容

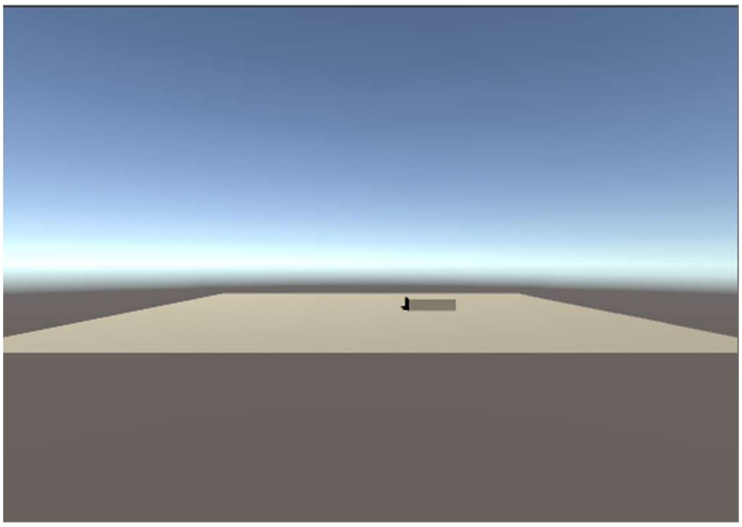
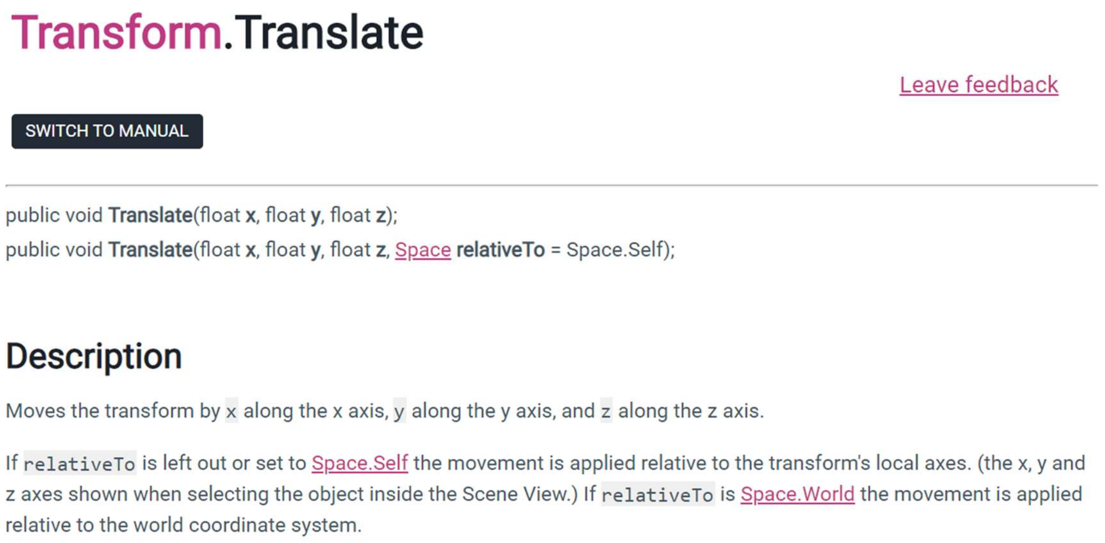


圖 6. 第 6 題程式執行時的畫面截圖。圖中方塊會自行往右移動。

加入 **Robot** 的原始 Unity *C#* 腳本框架 RobotController.cs 如下，請加入必要的敘述。

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class RobotController : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }
}
```

提示： Unity **scene** 中的每一個物件，都具備 **Transform** 元件（component，類似成員物件），用來平移（translate）、旋轉（rotate）、縮放（scale）物件位置或外形。在 Unity 的說明文件 https://docs.unity3d.com/ScriptReference/Transform.Translate.html，**Translate** 的 API 說明如圖 7，提供參考。



Transform.Translate

Leave feedback

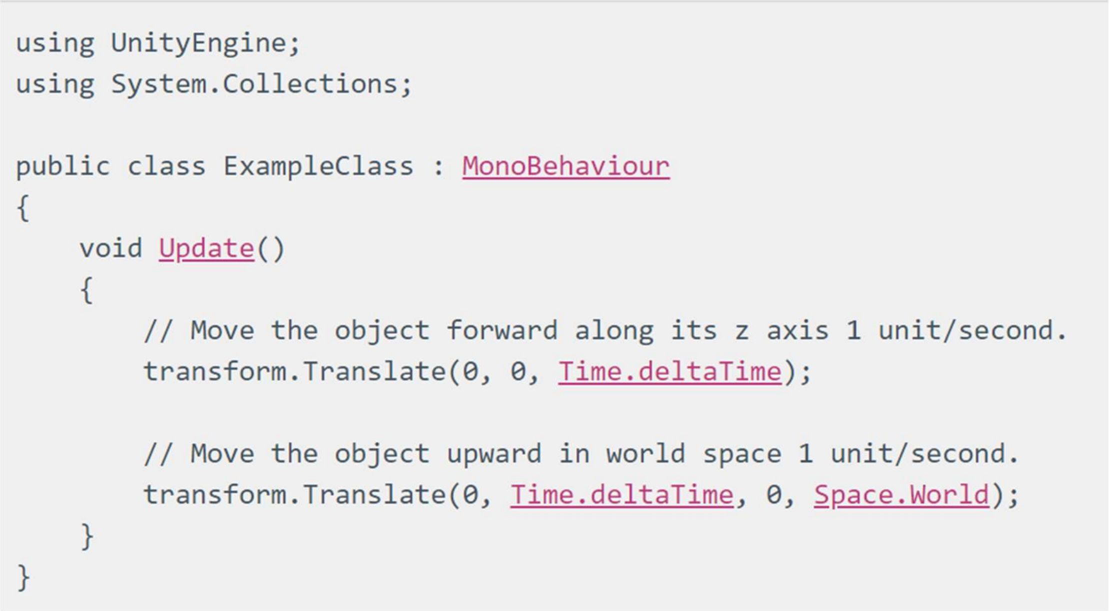
SWITCH TO MANUAL

public void **Translate**(float **x**, float **y**, float **z**);
public void **Translate**(float **x**, float **y**, float **z**, Space **relativeTo** = Space.Self);

Description

Moves the transform by `x` along the x axis, `y` along the y axis, and `z` along the z axis.

If `relativeTo` is left out or set to Space.Self the movement is applied relative to the transform's local axes. (the x, y and z axes shown when selecting the object inside the Scene View.) If `relativeTo` is Space.World the movement is applied relative to the world coordinate system.



```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour
{
    void Update()
    {
        // Move the object forward along its z axis 1 unit/second.
        transform.Translate(0, 0, Time.deltaTime);

        // Move the object upward in world space 1 unit/second.
        transform.Translate(0, Time.deltaTime, 0, Space.World);
    }
}
```

圖 7. Unity 對於 **Transform.Translate** 的說明文件。取自 https://docs.unity3d.com/ScriptReference/Transform.Translate.html

Ans.

程式列表如下:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class RobotController : MonoBehaviour
{
   public float speed;

   // Start is called before the first frame update
   void Start()
   {
   }

   // Update is called once per frame
   void Update()
   {
      transform.Translate(speed * Time.deltaTime, 0, 0);
   }
}
```

7.掃地機器人如 Roomba® (圖 8)，已經是相當普遍的小家電了。其實掃地機器人是一個自動行走的真空吸塵器，所以稱為真空吸塵器機器人(Vacuum Cleaner Robot)比較適當。學術界關於這種單純用途機器人的討論，手邊資料顯示最早可能是源自 S. Russell and Peter Norvig, *Artificial Intelligence : A Modern Approach*, Prentice-Hall Inc., New Jersey, USA, 1995. 一書第 2 章的習題及第 3 章的討論。該書後來的版本，相關的討論就刪減甚多了。在 Russel 與 Norvig 的書中，特別指出此種簡化的真空吸塵器機器人，關鍵的問題是路徑搜尋(route finding)；與一些學術與實際世界中的重要問題密切相關，例如旅行銷售員問題(travelling salesperson problem)，超大型積體電路佈局(VLSI layout)，一般機器人導航(robot navigation)、工廠機器人產品組裝(assembly sequencing)等。

圖 8. 真空吸塵器機器人 Roomba。取自 https://shopping.udn.com/mall/cus/cat/Cc1c02.do?dc_cargxuid_0=U010396697&gclid=CjwKCAjw26H3BRB2EiwAy32zheCKFCWC3n8z5RU-Wbnuh-6UP6YUXg4HfbxnWn20vPJ8KGOuy2CxnBoCHnYQAvD_BwE

顯然，產品化的真空吸塵器機器人，需要處理的實際問題，相當麻煩瑣碎。我們這裡聚焦到先前所說的路徑搜尋問題，其他的技術細節省略。有興趣可以參考 https://www.cnet.com/news/appliance-science-how-robotic-vacuums-navigate/ 以及 https://robotics.stackexchange.com/questions/628/what-algorithm-should-i-implement-to-program-a-room-cleaning-robot 和 https://www.techhive.com/article/3281014/how-a-robot-vacuum-navigates-your-home.html 的概略說明。

對於真空吸塵器機器人的路徑規劃(path planning)演算法的介紹與比較，在 T. Edwards 和 J. Sörme 的學士學位論文 *A Comparison of Path Planning Algorithms for Robotic Vacuum Cleaners*, KTH Royal Institute of Technology, School of Electrical Engineering and Computer Science, Stockholm, Sweden, 2018. (https://kth.diva-portal.org/smash/get/diva2:1214422/FULLTEXT01.pdf) 有較為簡單清楚的解說。該論文將真空吸塵器機器人的路徑規劃演算法分成三類：隨機亂步(random walk based algorithm)、螺旋(spiral algorithm)、蛇狀(snaking and wall follow algorithm)，其概念如圖 9 所示。

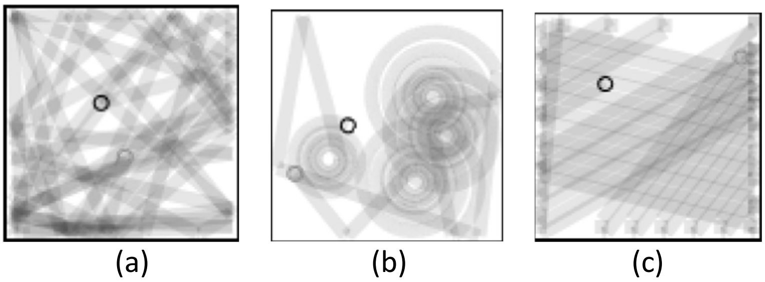


圖 9. 真空吸塵器機器人的路徑規劃：(a) random walk (b) spiral (c) snaking。取自 T. Edwards and J. Sörme, *A Comparison of Path Planning Algorithms for Robotic Vacuum Cleaners*, KTH Royal Institute of Technology, School of Electrical Engineering and Computer Science, Stockholm, Sweden, 2018. (https://kth.diva-portal.org/smash/get/diva2:1214422/FULLTEXT01.pdf)

這樣的路徑規劃，還是複雜了一點。所以我們進一步簡化：

1) 機器人外形改為正方形碟狀。

2) 房間設為矩形，四面牆分別在東(East，x 軸正向)、西(West，x 軸負向)、南(South，z 軸負向)，北(North，z 軸正向)。房間西南角設為座標系原點(所以地板上任一點的 x, z 座標均為正)。注意，為配合第 6 題的 Unity 程式，此處使用 x-z-y 的左手座標系。

3) 房間地板依照機器人形狀，劃分為 n_rows 乘 n_columns 個小方格(cell)。東西向的格子構成列(row)，南北向的格子構成行(column)。

4) 房間內無家俱，且每一小格均在「髒」(dirty)的狀態。機器人在某個小方格吸地(suck)後，那個小方格的狀態就改為「清潔」(clean)。

5) 機器人的頭部指向(heading)，限制為東、西、南、北四個方向。

6) 機器人頭部有一個感知器(sensor)，可以判斷頭部指向的下一個 cell，是否在牆外，亦即：下一步會不會撞到牆。

7) 機器人可以執行的動作包括: 感知障礙物(obstacle sensed)、前進(move)、吸地(suck)、原地左轉(turn left)、原地右轉(turn right)、原地迴轉(reverse)。

8) 機器人可以知道自己的位置座標(position)。

接著，要比較兩種機器人的清潔能力。假定兩種機器人，分別採用簡化的 random walk 及 snaking 演算法。令 n_rows = n_columns = 10，讓兩種機器人中心位置都從座標(5.5, 5.5)開始運作，觀察機器人移動(move)100 次之後，狀態為「清潔」的方格數佔全部方格數的比例，據此判斷何種路徑搜尋的演算法較佳。

簡化 random walk 及 snaking 在每一回合的演算法如下：

演算法 random walk

```
while ObstacleSensed {
  m = a new random integer
  if m % 2 == 0 {
    TurnRight
  } else {
    TurnLeft
  }
}
Move
Suck
```

要點是碰到障礙時，隨機決定左右轉，否則就依原來方向直走。典型的機器人中心路徑如圖 10(a)，顯然碰到牆後，就只能貼著牆壁走，沒有甚麼效率。

演算法 **snaking**

```
if !ObstacleSensed{
  Move
  Suck
} else {
  if last_turn == RIGHT {
    TurnLeft
    last_turn = LEFT
  } else {
    TurnRight
    last_turn = RIGHT
  }
  if !ObstacleSensed {
    Move
    Suck
    if last_turn == RIGHT {
      TurnRight
      last_turn = RIGHT
    } else {
      TurnLeft
      last_turn = LEFT
    }
  } else {
    // last_turn does not matter
    Reverse
  }
}
```

重點是需轉彎時，除非迴轉，要參考前一次是向左轉，還是向右轉。典型機器人中心路徑如圖 10(b)，效率比 random walk 高很多。

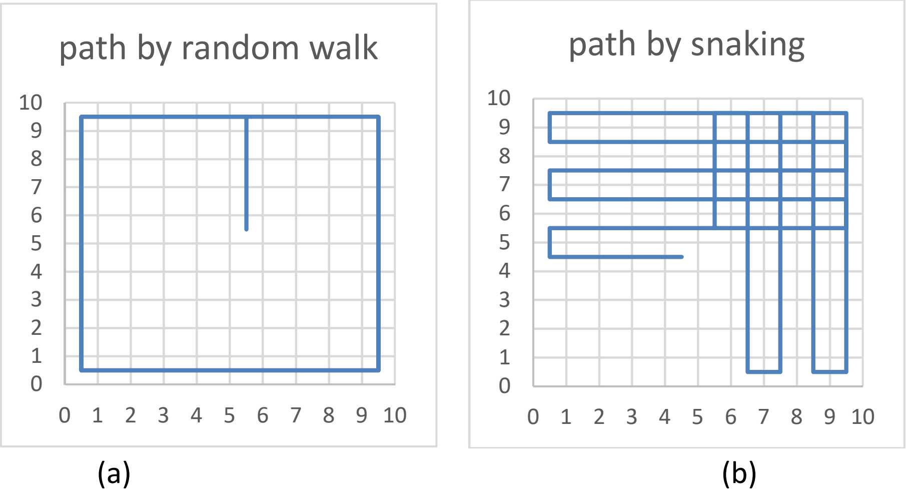
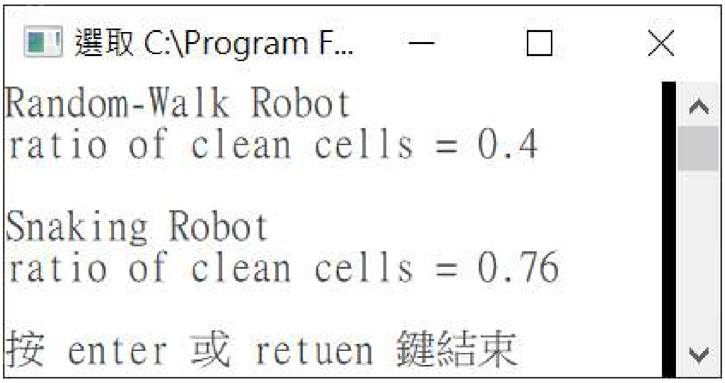


圖 10. 機器人中心移動路徑。(a) random walk。(b) snaking。

本題希望你參照以上說明，完成並測試兩種機器人模擬的效能。程式執行時的主控台畫面，應如圖 11 所示。



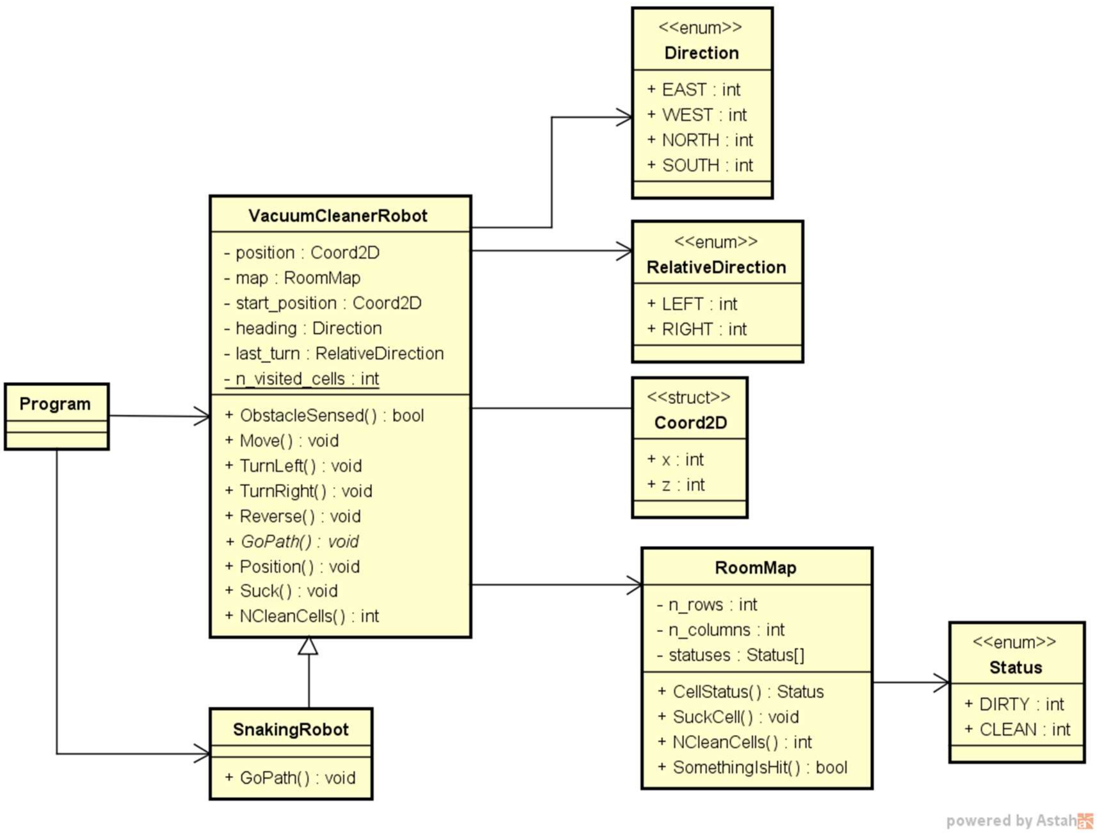
```
Random-Walk Robot
ratio of clean cells = 0.4

Snaking Robot
ratio of clean cells = 0.76

按 enter 或 retuen 鍵結束
```

圖 11. 第 7 題程式執行時的主控台螢幕畫面示意

本題滿分 25 分，全部程式集中寫成一個大 **Main** 函式，不區分其他函式者，最高得 18 分；善用函式者，最高得 20 分；能利用虛擬碼或 UML 類別圖思考，適當劃分類別(class)者，最高得 22 分；善用類別繼承與多型(polymorphism)者，最高得 25 分。 (25%)

Ans.

構想如以下之類別圖:



程式列表如下:

```
// Program.cs
using System;
namespace Problem7
{
    class Program
    {
        static void Main(string[] args)
        {
            int n_rows = 10;
            int n_columns = 10;
            Coord2D start_position = new Coord2D(5.5, 5.5);
            Direction heading = Direction.NORTH;
            Console.WriteLine("Random-Walk Robot");
            VacuumCleanerRobot robot1 = new VacuumCleanerRobot(
                n_rows, n_columns, start_position, heading
            );
            const int MAX_N_TIME_STEPS = 100;
            for(int n = 0; n < MAX_N_TIME_STEPS; ++n)
            {
                robot1.GoPath();
            }
            int n_clean_cells = robot1.NCleanCells();
            double ratio_of_clean_cells =
                (double) (n_clean_cells) / (double) (n_rows*n_columns);
            Console.WriteLine(
                "ratio of clean cells = {0}", ratio_of_clean_cells);
            Console.WriteLine();

            n_rows = 10;
            n_columns = 10;
            start_position = new Coord2D(5.5, 5.5);
            heading = Direction.NORTH;
            Console.WriteLine("Snaking Robot");
            SnakingRobot robot2 = new SnakingRobot(
                n_rows, n_columns, start_position, heading
            );
            for(int n = 0; n < MAX_N_TIME_STEPS; ++n)
            {
                robot2.GoPath();
            }
            n_clean_cells = robot2.NCleanCells();
            ratio_of_clean_cells =
                (double) (n_clean_cells) / (double) (n_rows*n_columns);
            Console.WriteLine(
                "ratio of clean cells = {0}", ratio_of_clean_cells);
            Console.WriteLine();

            Console.WriteLine("按 enter 或 retuen 鍵結束");
             Console.ReadLine();
        }
    }
}
```

```
// RoomMap.cs
using System;

namespace Problem7
{
   public enum Status
   {
      DIRTY,
      CLEAN
   }

   public class RoomMap
   {
      private int n_rows;
      private int n_columns;
      private Status[,] statuses;

      public RoomMap(int n_rows, int n_columns)
      {
         this.n_rows = n_rows;
         this.n_columns = n_columns;
         statuses = new Status [n_rows, n_columns];
         for(int i = 0; i < n_rows; ++i)
         {
            for(int j = 0; j < n_columns; ++j)
            {
               statuses[i, j]=Status.DIRTY;
            }
         }
      }

      public Status CellStatus(Coord2D pos)
      {
         int i = (int) pos.z;
         int j = (int) pos.x;
         return statuses[i, j];
      }
```

```
public void SuckCell(Coord2D pos)
{
    int i = (int) pos.z;
    int j = (int) pos.x;
    statuses[i, j] = Status.CLEAN;
}

public int NCleanCells()
{
    int n_clean_cells = 0;
    foreach (Status status in statuses)
    {
        if(status == Status.CLEAN)
        {
            ++n_clean_cells;
        }
    }
    return n_clean_cells;
}

public bool SomeThingIsHit(Coord2D pos, Direction heading)
{
    bool result = false;

    switch(heading)
    {
        case Direction.EAST:
        result = (pos.x + 1.0 >= (double)(n_columns));
        break;

        case Direction.WEST:
        result = (pos.x - 1.0 <= 0.0);
        break;

        case Direction.NORTH:
        result = (pos.z + 1.0 >= (double)(n_rows));
        break;
```

```
                case Direction.SOUTH:
                result = (pos.z - 1.0 <= 0.0);
                break;

                default:
                Console.WriteLine("Should not be here");
                break;
            }
            return result;
        }
    }
}

// VacuumCleanerRobots.cs
using System;

namespace Problem7
{
    public enum Direction
    {
        EAST,
        WEST,
        NORTH,
        SOUTH
    }

    public enum RelativeDirection
    {
        LEFT,
        RIGHT
    }

    public struct Coord2D
    {
        public double x;
        public double z;

        public Coord2D(double x, double z)
    {
        this.x = x;
        this.z = z;
    }
}
public class VacuumCleanerRobot
{
    private Coord2D position;
    private RoomMap map;
    private Coord2D start_position;
    private Direction heading;

    private Random random;

    public VacuumCleanerRobot(
        int n_rows, int n_columns,
        Coord2D start_position,
        Direction heading)
    {
        map = new RoomMap(n_rows, n_columns);
        this.start_position = start_position;
        this.heading = heading;
        position.x = start_position.x;
        position.z = start_position.z;
        Suck();
        random = new Random(777);
    }

    public bool ObstacleSensed()
    {
        return map.SomeThingIsHit(position, heading);
    }

    public void Move()
    {
        switch(heading)
        {
            case Direction.EAST:
```

```
        position.x += 1.0;
        break;

        case Direction.WEST:
        position.x -= 1.0;
        break;

        case Direction.NORTH:
        position.z += 1.0;
        break;

        case Direction.SOUTH:
        position.z -= 1.0;
        break;

        default:
        Console.WriteLine("Should not be here");
        break;
    }
}

public void TurnLeft()
{
    switch(heading)
    {
        case Direction.EAST:
        heading = Direction.NORTH;
        break;

        case Direction.WEST:
        heading = Direction.SOUTH;
        break;

        case Direction.NORTH:
        heading = Direction.WEST;
        break;

        case Direction.SOUTH:
            heading = Direction.EAST;
            break;
        }
    }

    public void TurnRight()
    {
        switch(heading)
        {
            case Direction.EAST:
            heading = Direction.SOUTH;
            break;

            case Direction.WEST:
            heading = Direction.NORTH;
            break;

            case Direction.NORTH:
            heading = Direction.EAST;
            break;

            case Direction.SOUTH:
            heading = Direction.WEST;
            break;
        }
    }

    public void Reverse()
    {
        switch(heading)
        {
            case Direction.EAST:
            heading = Direction.WEST;
            break;

            case Direction.WEST:
            heading = Direction.EAST;
            break;
        case Direction.NORTH:
        heading = Direction.SOUTH;
        break;

        case Direction.SOUTH:
        heading = Direction.NORTH;
        break;
    }
}

public void Suck()
{
    map.SuckCell(position);
}

public virtual void GoPath()
{
    while(ObstacleSensed())
    {
        int rand = random.Next();
        if(rand % 2 == 0)
        {
            TurnRight();
        }
        else
        {
            TurnLeft();
        }
    }
    Move();
    Suck();
}
public Coord2D Position()
{
    return position;
}
```

```
        public int NCleanCells()
        {
            int result = map.NCleanCells();
            return result;
        }
    }
}



// SnakingRobot.cs
using System;

namespace Problem7
{
    public class SnakingRobot : VacuumCleanerRobot
    {
        private RelativeDirection last_turn;
        public SnakingRobot(int n_rows, int n_columns,
            Coord2D start_position,
            Direction heading) :
            base(n_rows, n_columns, start_position, heading)
            {
               last_turn = RelativeDirection.LEFT;
            }

        public override void GoPath()
        {
             if(!ObstacleSensed())
             {
                Move();
                Suck();
            }
            else
            {
                if(last_turn == RelativeDirection.RIGHT)
                {
                    TurnLeft();
                    last_turn = RelativeDirection.LEFT;
                }
                else
                {
                    TurnRight();
                    last_turn = RelativeDirection.RIGHT;
                }

                if(!ObstacleSensed())
                {
                    Move();
                    Suck();

                    if(last_turn == RelativeDirection.RIGHT)
                    {
                        TurnRight();
                        last_turn = RelativeDirection.RIGHT;
                    }
                    else
                    {
                        TurnLeft();
                        last_turn = RelativeDirection.LEFT;
                    }
                }
                else
                {
                    // last_turn does not matter here
                    Reverse();
                }
            }
        }
    }
}
```