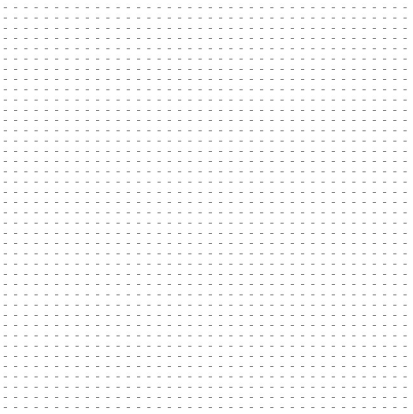
課題：I235 1st Report　　言語：C#(ConsoleApplication)

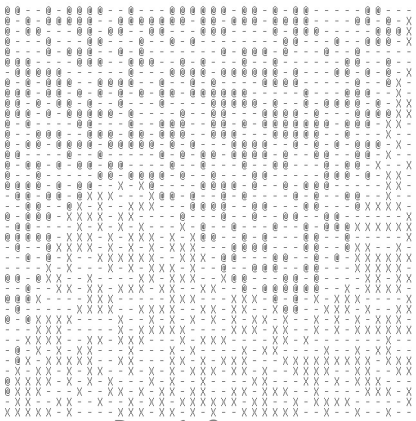
氏名：GAO,YouWei

学生番号：s1910092

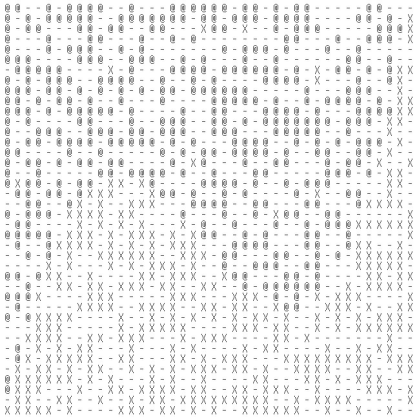
提出日：2019/04/26



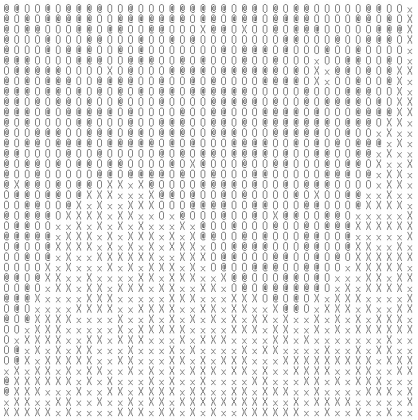
Demo1-1.png



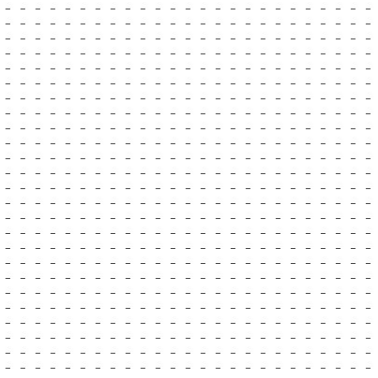
Demo1-2.png



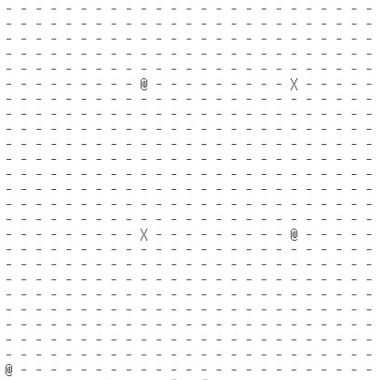
Demo1-3.png



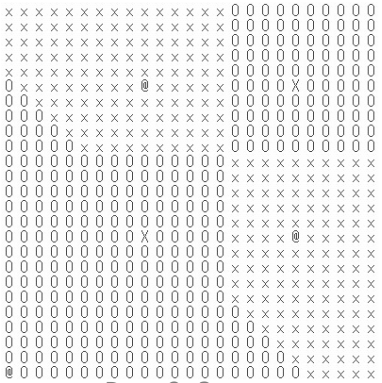
Demo1-4.png



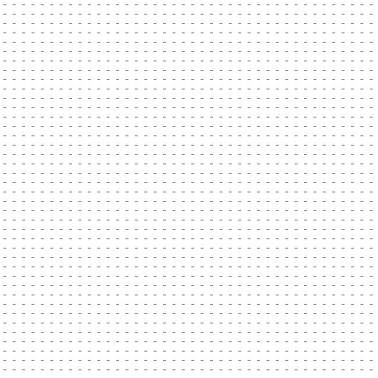
Demo2-1.png



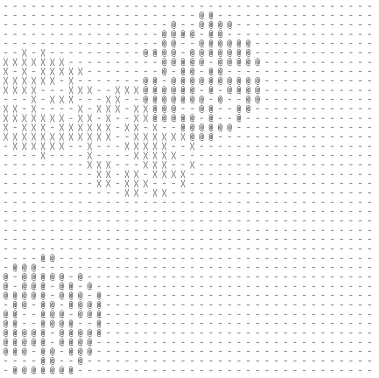
Demo2-2.png



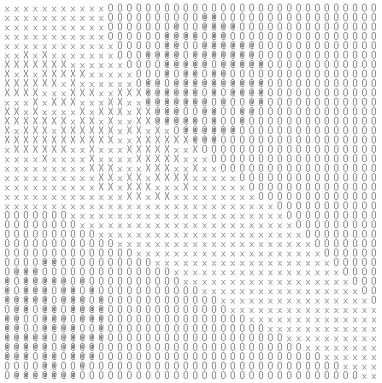
Demo2-3.png



Demo3-1.png



Demo3-2.png



Demo3-3.png

```
1  namespace s1910092
2  {
3      partial class NearestNeighbor
4      {
5          static void Main(string[] args)
6          {
7              //ランダムで(幅20-40f、高さ20-40f)空間を生成する
8              SquareFrame demo1 = new SquareFrame().RandomCreat(20, 40, 20, 40)
9              .Print()//Demo1-1.png
10             //ランダムな(0-1f,0-1f,密度50%)ベジェ曲線を生成する
11             .RandomAddPoint_Bezier(1f, 1f, 0.5f)
12             .Print()//Demo1-2.png
13             //ランダムな「X」点20個生成する(ノイズ)
14             .RandomAddPoint_Point(20, MARK.X)
15             .Print()//Demo1-3.png
16             .Run_kNN(3)//k-NN(3-NN)を実行する
17             .Print();//Demo1-4.png
18
19
20             //(幅5f、高さ5f)空間を生成する(解像度0.2f)
21             SquareFrame demo2 = new SquareFrame(5f, 5f, 0.2f)
22             .Print()//Demo2-1.png
23             .AddGivenPoint(0.2f, 0.2f, MARK.a)//座標(0.2f,0.2f)に「@」を生成す
               る
24             .AddGivenPoint(2, 2, MARK.X)//座標(2,2)に「X」を生成する
25             .AddGivenPoint(2, 4, MARK.a)//座標(2,4)に「@」を生成する
26             .AddGivenPoint(4, 2, MARK.a)//座標(4,2)に「@」を生成する
27             .AddGivenPoint(4, 4, MARK.X)//座標(4,4)に「X」を生成する
28             .Print()//Demo2-2.png
29             .Run_kNN(3)//k-NN(3-NN)を実行する
30             .Print();//Demo2-3.png
31
32
33             //(幅40f、高さ40f)空間を生成する(解像度default：1f)
34             SquareFrame demo3 = new SquareFrame(40, 40)
35             .Print()//Demo3-1.png
36             //ランダムな(焦点間距離2-7f、半径6-9f、密度70%、@)楕円区域を生成す
               る
37             .RandomAddPoint_Oval(2, 7, 6, 9, 0.7f, MARK.a)
38             .RandomAddPoint_Oval(2, 7, 6, 9, 0.7f, MARK.X)//ランダム楕円*2
39             .RandomAddPoint_Oval(2, 7, 6, 9, 0.7f, MARK.a)//ランダム楕円*3
40             .RandomAddPoint_Oval(2, 7, 6, 9, 0.7f, MARK.X)//ランダム楕円*4
41             .Print()//Demo3-2.png
42             .Run_kNN(3)//k-NN(3-NN)を実行する
43             .Print();//Demo3-3.png
44         }
45     }
46 }
```

```
1  namespace s1910092
2  {
3      using System;
4      using System.Linq;
5      using System.Collections.Generic;
6      partial class NearestNeighbor
7      {
8          enum MARK : byte { NULL, a, X, o, x }
9          static private string[] STRINGS = { " -", " @", " X", " O", " x" };
10         class SquareFrame
11         {
12             private float width_0;
13             private float height_0;
14             private float DISPLAY_RESOLUTION_F;
15             private Random ran = new Random();
16             public class Lattice
17             {
18                 public MARK mark = MARK.NULL;
19                 public Vector coord;
20                 public Lattice next_lattice;
21                 public Lattice(Vector coord, Lattice next_lattice)
22                 {
23                     this.coord = coord;
24                     this.next_lattice = next_lattice;
25                 }
26                 public void Print()
27                 {
28                     Console.Write(STRINGS[(int)mark]);
29                 }
30                 public void PrintLine()
31                 {
32                     Console.WriteLine(STRINGS[(int)mark]);
33                 }
34             }
35             Lattice first_lattice;
36             Lattice lattice_pointer = null;
37             public Dictionary<Vector, Lattice> lattice_dic = new
                 Dictionary<Vector, Lattice>();
38             public List<Vector> givenpoint_list = new List<Vector>();
39             public SquareFrame(float width, float height, float
                 DISPLAY_RESOLUTION_F)
40             {
41                 this.DISPLAY_RESOLUTION_F = DISPLAY_RESOLUTION_F;
42                 width_0 = width - DISPLAY_RESOLUTION_F;
43                 height_0 = height - DISPLAY_RESOLUTION_F;
44                 for (float j = 0f; j <= height_0; j = (float)Math.Round(j +
                   DISPLAY_RESOLUTION_F, 2))
45                 {
46                     for (float i = width_0; i >= 0f; i = (float)Math.Round(i -
                      DISPLAY_RESOLUTION_F, 2))
47                     {
48                         if (lattice_pointer == null)
49                         {
50                             first_lattice = lattice_pointer = new Lattice(new
                    Vector(i, j), null);
51                         }
```

```
52                          else
53                          {
54                              first_lattice = new Lattice(new Vector(i, j),
                        lattice_pointer);
55                              lattice_pointer = first_lattice;
56                          }
57                          lattice_dic.Add(new Vector(i, j), lattice_pointer);
58                      }
59                  }
60              }
61              public SquareFrame(float width, float height) : this(width,
                  height, 1f) { }
62              public SquareFrame()
63              {
64                  //pls "new SquareFrame().RandomCreat(...)"
65              }
66              public SquareFrame RandomCreat(int width_min, int width_max, int
                  hight_min, int hight_max)
67              {
68                  return new SquareFrame(ran.Next(width_min, width_max),
                      ran.Next(hight_min, hight_max));
69              }
70              public SquareFrame RandomCreat()
71              {
72                  return RandomCreat(6, 9, 6, 9);
73              }
74              public SquareFrame Print()
75              {
76                  lattice_pointer = first_lattice;
77                  while (lattice_pointer != null)
78                  {
79                      if (lattice_pointer.coord.x == width_0)
                        lattice_pointer.PrintLine();
80                      else lattice_pointer.Print();
81                      lattice_pointer = lattice_pointer.next_lattice;
82                  }
83                  Console.WriteLine();
84                  return this;
85              }
86              public SquareFrame AddGivenPoint(float x, float y, MARK mark)
87              {
88                  x -= DISPLAY_RESOLUTION_F;
89                  y -= DISPLAY_RESOLUTION_F;
90                  givenpoint_list.RemoveAll((item_1 => item_1.x == x && item_1.y
                       == y));
91                  givenpoint_list.Add(new Vector(x, y));
92                  if (lattice_dic.ContainsKey(new Vector(x, y)))
93                  {
94                      lattice_dic[new Vector(x, y)].mark = mark;
95                  }
96                  return this;
97              }
98              public SquareFrame RandomAddPoint_Oval(
99                  float fixed_points_distances_min,
100                 float fixed_points_distances_max,
101                 float r_min,
```

```
102                 float r_max,
103                 float probability,
104                 MARK mark)
105             {
106                 float r =
107                     (float)ran.NextDouble()
108                     + (r_max - r_min)
109                     + r_min;
110                 float f_2 =
111                     (float)ran.NextDouble()
112                     * (fixed_points_distances_max -
                    fixed_points_distances_min)
113                     + fixed_points_distances_min;
114                 if (ran.Next(0, 1) > 0) f_2 *= -1;
115                 Vector f1 = new Vector((float)ran.NextDouble() * width_0,
                      (float)ran.NextDouble() * height_0);
116                 Vector f2 = new Vector(f1.x + f_2, f1.y);
117                 foreach (var item in lattice_dic)
118                 {
119                     if (((item.Key - f1).Length() + (item.Key - f2).Length())
                    < r * r)
120                     {
121                         if (givenpoint_list.Contains(item.Key))
122                         {
123                             givenpoint_list.RemoveAll((item_1 => item_1 ==
                    item.Key));
124                         }
125                         item.Value.mark = MARK.NULL;
126                         if ((float)ran.NextDouble() < probability)
127                         {
128                             givenpoint_list.Add(item.Key);
129                             item.Value.mark = mark;
130                         }
131                     }
132                 }
133                 return this;
134             }
135             public SquareFrame RandomAddPoint_Point(int number, MARK mark)
136             {
137                 List<Vector> add_list = new List<Vector>();
138                 Vector ran_v = new Vector();
139                 while (number > 0)
140                 {
141                     do
142                     {
143                         ran_v.x = (float)Math.Round(
144                             ran.Next(0, (int)Math.Round(width_0 /
                    DISPLAY_RESOLUTION_F))
145                             * DISPLAY_RESOLUTION_F, 2);
146                         ran_v.y = (float)Math.Round(
147                             ran.Next(0, (int)Math.Round(height_0 /
                    DISPLAY_RESOLUTION_F))
148                             * DISPLAY_RESOLUTION_F, 2);
149                     } while (add_list.Contains(ran_v));
150                     add_list.Add(ran_v);
151                     number--;
```

```
152                 }
153                 foreach (var item in add_list)
154                 {
155                     givenpoint_list.RemoveAll((item_1 => item_1 == item));
156 
157                     if (lattice_dic.ContainsKey(item))
158                     {
159                         givenpoint_list.Add(item);
160                         lattice_dic[item].mark = mark;
161                     }
162                 }
163                 return this;
164             }
165             public Vector GetBezierPoint(float t, float ran_1, float ran_2)
166             {
167                 Vector p_0 = new Vector(0f, 0f);
168                 Vector p_1 = p_0; p_1.y += 2 * ran_1 * height_0;
169                 Vector p_3 = new Vector(width_0, height_0);
170                 Vector p_2 = p_3; p_2.y -= 2 * ran_2 * height_0;
171                 Vector p_r = p_0 * (float)Math.Pow((1f - t), 3)
172                     + p_1 * 3 * t * (float)Math.Pow((1f - t), 2)
173                     + p_2 * 3 * (float)Math.Pow(t, 2) * (1f - t)
174                     + p_3 * (float)Math.Pow(t, 3);
175                 return p_r;
176             }
177             public SquareFrame RandomAddPoint_Bezier(float p1_max, float
                  p2_max, float probability)
178             {
179                 p1_max *= (float)ran.NextDouble();
180                 p2_max *= (float)ran.NextDouble();
181                 givenpoint_list.Clear();
182                 foreach (var item in lattice_dic)
183                 {
184                     item.Value.mark = MARK.NULL;
185                     if (item.Key.y > GetBezierPoint(item.Key.x / width_0,
                        p1_max, p2_max).y)
186                     {
187                         if (ran.NextDouble() < probability)
188                         {
189                             item.Value.mark = MARK.a;
190                             givenpoint_list.Add(item.Key);
191                         }
192                     }
193                     else
194                     {
195                         if (ran.NextDouble() < probability)
196                         {
197                             item.Value.mark = MARK.X;
198                             givenpoint_list.Add(item.Key);
199                         }
200                     }
201                 }
202                 return this;
203             }
204             public SquareFrame Run_kNN(ushort k)
205             {
```

```
206                 lattice_pointer = first_lattice;
207                 while (lattice_pointer != null)
208                 {
209                     if (lattice_pointer.mark == MARK.NULL)
210                     {
211                         List<Vector> distance_list = new List<Vector>();
212                         foreach (var item in givenpoint_list)
213                         {
214                             distance_list.Add(item - lattice_pointer.coord);
215                         }
216                         if (distance_list.Count() >= k)
217                         {
218                             distance_list.Sort();
219                             Dictionary<MARK, uint> mark_num_dic = new
                    Dictionary<MARK, uint>();
220                             for (int i = k - 1; i >= 0; i--)
221                             {
222                                 Lattice l_target = lattice_dic
                    [lattice_pointer.coord + distance_list[i]];
223                                 if (mark_num_dic.ContainsKey(l_target.mark))
224                                 {
225                                     mark_num_dic[l_target.mark]++;
226                                 }
227                                 else
228                                 {
229                                     mark_num_dic.Add(l_target.mark, 1);
230                                 }
231                             }
232                             MARK mark_max = MARK.NULL;
233                             uint mark_max_uint = 0;
234                             foreach (var item in mark_num_dic)
235                             {
236                                 if (item.Value >= mark_max_uint)
237                                 {
238                                     mark_max_uint = item.Value;
239                                     mark_max = item.Key;
240                                 }
241                             }
242                             lattice_pointer.mark = (MARK)((int)(mark_max) +
                    2);
243                         }
244                     }
245                     lattice_pointer = lattice_pointer.next_lattice;
246                 }
247                 return this;
248             }
249         }
250         struct Vector : IComparable<Vector>
251         {
252             public float x;
253             public float y;
254             public float length;
255             public int ran_i;
256             public float Length()
257             {
258                 return (x * x) + (y * y);
```

```
259             }
260             public Vector(float x, float y)
261             {
262                 this.x = (float)Math.Round(x, 2);
263                 this.y = (float)Math.Round(y, 2);
264                 length = (x * x) + (y * y);
265                 ran_i = new Random().Next();
266             }
267             public static bool operator ==(Vector v1, Vector v2)
268             {
269                 if (v1.x == v2.x && v1.y == v2.y)
270                     return true;
271                 return false;
272             }
273             public static bool operator !=(Vector v1, Vector v2)
274             {
275                 if (v1.x != v2.x || v1.y != v2.y)
276                     return true;
277                 return false;
278             }
279             public static Vector operator -(Vector v1, Vector v2)
280             {
281                 return new Vector(
282                     (float)Math.Round(v1.x - v2.x, 2)
283                     , (float)Math.Round(v1.y - v2.y, 2));
284             }
285             public static Vector operator +(Vector v1, Vector v2)
286             {
287                 return new Vector(
288                     (float)Math.Round(v1.x + v2.x, 2)
289                     , (float)Math.Round(v1.y + v2.y, 2));
290             }
291             public static Vector operator *(Vector v1, float f2)
292             {
293                 return new Vector(
294                     (float)Math.Round(v1.x * f2, 2)
295                     , (float)Math.Round(v1.y * f2, 2));
296             }
297             public override string ToString()
298             {
299                 return x.ToString() + "," + y.ToString();
300             }
301             public override int GetHashCode()
302             {
303                 return ToString().GetHashCode();
304             }
305             public override bool Equals(object obj)
306             {
307                 if (obj == null || GetType() != obj.GetType())
308                     return false;
309                 Vector v2 = (Vector)obj;
310                 if (x == v2.x && y == v2.y) return true;
311                 else return false;
312             }
313             public int CompareTo(Vector other)
314             {
```

```
315                     if (length > other.length)
316                     {
317                         return 1;
318                     }
319                     else if (length == other.length)
320                     {
321                         if (ran_i > other.ran_i)
322                         {
323                             return 1;
324                         }
325                         else if (ran_i == other.ran_i)
326                         {
327                             return 0;
328                         }
329                         else
330                         {
331                             return -1;
332                         }
333                     }
334                     else
335                     {
336                         return -1;
337                     }
338                 }
339             }
340 
341         }
342     }
343 
```