

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Timers;
5 using System.Text;
6 using System.Threading.Tasks;
7
8 namespace PlayfairCipher
9 {
10     class Program
11     {
12         protected static int origRow;
13         protected static int origCol;
14
15         protected static void WriteAt(string s, int x, int y)
16         {
17             try
18             {
19                 Console.SetCursorPosition(origCol + x, origRow + y);
20                 Console.Write(s);
21             }
22             catch (ArgumentOutOfRangeException e)
23             {
24                 Console.Clear();
25                 Console.WriteLine(e.Message);
26             }
27         }
28
29         static void Main(string[] args)
30         {
31             #region Encription
32             Console.Write("Podaj tekst jawny: ");
33             string publicText = Console.ReadLine().ToUpper().Replace(" ", "");
34             char[] publicTextCharTab = new char[publicText.Length];
35             publicTextCharTab = publicText.ToArray();
36             Console.Write("Podaj klucz szyfrujący: ");
37             string key = Console.ReadLine().ToUpper().Replace(" ", "");
38             char[] keyCharTab = new char[key.Length];
39             keyCharTab = key.ToList().Distinct().ToArray();
40             PlayfairTab(keyCharTab);
41             publicTextCharTab = AdditionalLetter(publicTextCharTab);
42             int[] typeOfCodingTab = new int[publicTextCharTab.Length / 2];
43             typeOfCodingTab = TypeOfCoding(Coordinates(PlayfairTab
44                 (keyCharTab), publicTextCharTab));
45             int[] encryptedCoordinate = new int[Coordinates(PlayfairTab
46                 (keyCharTab), publicTextCharTab).Length];
47             encryptedCoordinate = PlayfairCypherEncription(PlayfairTab
48                 (keyCharTab), Coordinates(PlayfairTab(keyCharTab),
49                 publicTextCharTab), typeOfCodingTab);
50             string encryptedString = new string(EncryptedCoordinatesToCharTab
51                 (encryptedCoordinate, PlayfairTab(keyCharTab)));
52             char[] EncryptedCharTab = new char[EncryptedCoordinatesToCharTab
53                 (encryptedCoordinate, PlayfairTab(keyCharTab)).Length];
54             EncryptedCharTab = EncryptedCoordinatesToCharTab
55                 (encryptedCoordinate, PlayfairTab(keyCharTab));
56             Console.WriteLine("\nKODOWANIE: ");
```

```

50         Random rnd = new Random();
51
52         Timer aTimer = new System.Timers.Timer();
53         aTimer.Interval = 50;
54
55         // Hook up the Elapsed event for the timer.
56         DateTime t = DateTime.Now;
57         aTimer.Elapsed += (sender, e) => OnTimedEvent(sender, e,      ↗
            EncryptedCharTab, t);
58
59         // Have the timer fire repeated events (true is the default)
60         aTimer.AutoReset = true;
61
62         // Start the timer
63         aTimer.Enabled = true;
64
65         Console.ReadLine();
66
67         #endregion
68
69         #region Decription
70
71         bool isCorrect = false;
72         do
73         {
74             Console.WriteLine("Wybierz \n 1) Aby odszyfrować pierwotne      ↗
                hasło \n 2) Aby odszyfrować dowolne hasło");
75             int choice = Int32.Parse(Console.ReadLine());
76             if (choice==1)
77             {
78                 isCorrect = true;
79                 string decriptedString = new string      ↗
                    (DecriptedCoordinatesToCharTab(PlayfairCypherDecription      ↗
                    (PlayfairTab(keyCharTab), CoordinatesDecription      ↗
                    (PlayfairTab(keyCharTab), EncryptedCharTab),      ↗
                    TypeOfCodingDecripted(CoordinatesDecription(PlayfairTab      ↗
                    (keyCharTab), EncryptedCharTab))), PlayfairTab      ↗
                    (keyCharTab)));
80                 Console.WriteLine(decriptedString);
81             }
82             else
83             {
84                 if (choice==2)
85                 {
86                     Console.Write("Podaj hasło do odszyfrowania: ");
87                     string encryptedStringDecoding = Console.ReadLine();
88                     char[] encryptedTextDecoding = new char      ↗
                        [encryptedStringDecoding.Length];
89                     isCorrect = true;
90                     string decriptedString = new string      ↗
                        (DecriptedCoordinatesToCharTab(PlayfairCypherDecription      ↗
                        (PlayfairTab(keyCharTab), CoordinatesDecription      ↗
                        (PlayfairTab(keyCharTab), EncryptedCharTab),      ↗
                        TypeOfCodingDecripted(CoordinatesDecription(PlayfairTab      ↗
                        (keyCharTab), EncryptedCharTab))), PlayfairTab      ↗
                        (keyCharTab)));

```

```

91         Console.WriteLine(decriptedString);
92     }
93     else
94     {
95         isCorrect = false;
96         Console.WriteLine("Nie poprawnie wybrana opcja.");
97     }
98 }
99 } while (!isCorrect);
100
101 #endregion
102 }
103
104 #region EncryptionFunction
105
106 private static void OnTimedEvent(Object source,
107     System.Timers.ElapsedEventArgs e, char[] encryptedCharTab, DateTime t)
108 {
109     TimeSpan dtts = new TimeSpan(t.Day, t.Hour, t.Minute, t.Second);
110     int dttsTempSec = (int)dtts.TotalSeconds;
111     TimeSpan ts = new TimeSpan(e.SignalTime.Day, e.SignalTime.Hour,
112         e.SignalTime.Minute, e.SignalTime.Second);
113     int tempSec = (int)ts.TotalSeconds;
114     t = t.Add(ts);
115
116     Random rnd = new Random();
117     for (int i = tempSec - dttsTempSec; i < encryptedCharTab.Length; i++)
118     {
119         WriteAt(((char)rnd.Next(65, 90)).ToString(), i, 4);
120     }
121     if (tempSec - dttsTempSec < encryptedCharTab.Length)
122     {
123         WriteAt(encryptedCharTab[tempSec - dttsTempSec].ToString(),
124             tempSec - dttsTempSec, 4);
125     }
126 }
127
128 public static char[,] PlayfairTab(char[] keyCharTab)
129 {
130     char[] AlphabetTab = new char[26];
131     AlphabetTab[0] = 'A';
132     for (int i = 1; i < AlphabetTab.Length; i++)
133     {
134         AlphabetTab[i] = (char)((int)AlphabetTab[0] + i);
135     }
136     char[] PlayfairTab1D = new char[keyCharTab.Length +
137         AlphabetTab.Length];
138     PlayfairTab1D = keyCharTab.Concat(AlphabetTab).ToArray().ToList()
139         ().Distinct().ToArray().Where(x => x != 'J').ToArray();
140     char[,] PlayfairTab2D = new char[5, 5];
141     for (int i = 0; i < 5; i++)
142     {
143         for (int j = 0; j < 5; j++)

```

```
140         {
141             PlayfairTab2D[i, j] = PlayfairTab1D[5 * i + j];
142         }
143     }
144     return PlayfairTab2D;
145 }
146
147 public static char[] AdditionalLetter(char[] publicTextCharTab)
148 {
149     int index = -2;
150     do
151     {
152         index = ReturnDoubleLetterIndex(publicTextCharTab);
153         if (index != -1)
154         {
155             List<char> tempCharList = publicTextCharTab.ToList();
156             tempCharList.Insert(index, 'X');
157             publicTextCharTab = tempCharList.ToArray();
158         }
159     } while (index != -1);
160     if (publicTextCharTab.Length % 2 == 1)
161     {
162         List<char> tempCharList = publicTextCharTab.ToList();
163         tempCharList.Insert(publicTextCharTab.Length, 'X');
164         publicTextCharTab = tempCharList.ToArray();
165     }
166     return publicTextCharTab;
167 }
168
169 public static int ReturnDoubleLetterIndex(char[] publicTextCharTab)
170 {
171     for (int i = 0; i < publicTextCharTab.Length; i += 2)
172     {
173         if (i + 1 < publicTextCharTab.Length)
174         {
175             if (publicTextCharTab[i] == publicTextCharTab[i + 1])
176             {
177                 return i + 1;
178             }
179         }
180     }
181     return -1;
182 }
183
184 public static int[] Coordinates(char[,] playfairTab, char[] publicTextCharTab)
185 {
186     int[] coordinates = new int[(publicTextCharTab.Length) * 2];
187     for (int k = 0; k < publicTextCharTab.Length; k++)
188     {
189         for (int i = 0; i < 5; i++)
190         {
191             for (int j = 0; j < 5; j++)
192             {
193                 if (publicTextCharTab[k] == playfairTab[i, j])
194                 {
```

```
195         coordinates[2 * k] = i;
196         coordinates[2 * k + 1] = j;
197     }
198 }
199 }
200 }
201 return coordinates;
202 }
203
204 public static int[] TypeOfCoding(int[] coordinates)
205 {
206     int[] TypeOfCodingTab = new int[coordinates.Length / 4];
207     for (int i = 0; i < coordinates.Length / 4; i++)
208     {
209         if (4 * i + 3 < coordinates.Length)
210         {
211             if (coordinates[4 * i] == coordinates[4 * i + 2])
212             {
213                 TypeOfCodingTab[i] = 2;
214             }
215             else
216             {
217                 if (coordinates[4 * i + 1] == coordinates[4 * i + 3])
218                 {
219                     TypeOfCodingTab[i] = 1;
220                 }
221                 else
222                 {
223                     TypeOfCodingTab[i] = 3;
224                 }
225             }
226         }
227     }
228     return TypeOfCodingTab;
229 }
230
231 public static int[] PlayfairCypherEncription(char[,] playfairTab, int [] coordinates, int[] typeOfCodingTab)
232 {
233     for (int i = 0; i < typeOfCodingTab.Length; i++)
234     {
235         if (typeOfCodingTab[i] == 1)
236         {
237             coordinates[4 * i] = (coordinates[4 * i] + 1) % 5;
238             coordinates[4 * i + 2] = (coordinates[4 * i + 2] + 1) % 5;
239         }
240         else
241         {
242             if (typeOfCodingTab[i] == 2)
243             {
244                 coordinates[4 * i + 1] = (coordinates[4 * i + 1] + 1) % 5;
245                 coordinates[4 * i + 3] = (coordinates[4 * i + 3] + 1) % 5;
246             }
247             else
```

```
248         {
249             int temp = 0;
250             temp = coordinates[4 * i];
251             coordinates[4 * i] = coordinates[4 * i + 2];
252             coordinates[4 * i + 2] = temp;
253         }
254     }
255 }
256 return coordinates;
257 }
258
259 public static char[] EncryptedCoordinatesToCharTab(int[] encryptedCoordinates, char[,] playfairCharTab)
260 {
261     char[] encryptedCharTab = new char[encryptedCoordinates.Length / 2];
262     for (int i = 0; i < encryptedCoordinates.Length/2; i++)
263     {
264         encryptedCharTab[i] = playfairCharTab[encryptedCoordinates[2*i], encryptedCoordinates[2*i + 1]];
265     }
266     return encryptedCharTab;
267 }
268
269 #endregion
270
271 #region DecriptionFunction
272
273 public static int[] CoordinatesDecription(char[,] playfairTab, char[] encryptedCharTab)
274 {
275     int[] coordinatesDecripted = new int[(encryptedCharTab.Length) * 2];
276     for (int k = 0; k < encryptedCharTab.Length; k++)
277     {
278         for (int i = 0; i < 5; i++)
279         {
280             for (int j = 0; j < 5; j++)
281             {
282                 if (encryptedCharTab[k] == playfairTab[i, j])
283                 {
284                     coordinatesDecripted[2 * k] = i;
285                     coordinatesDecripted[2 * k + 1] = j;
286                 }
287             }
288         }
289     }
290     return coordinatesDecripted;
291 }
292
293 public static int[] TypeOfCodingDecripted(int[] coordinatesDecripted)
294 {
295     int[] TypeOfCodingTabDecripted = new int
296         [coordinatesDecripted.Length / 4];
297     for (int i = 0; i < coordinatesDecripted.Length / 4; i++)
298     {
```

```
298         if (4 * i + 3 < coordinatesDecripted.Length)
299         {
300             if (coordinatesDecripted[4 * i] == coordinatesDecripted[4 * i + 2])
301             {
302                 TypeOfCodingTabDecripted[i] = 2;
303             }
304             else
305             {
306                 if (coordinatesDecripted[4 * i + 1] == coordinatesDecripted[4 * i + 3])
307                 {
308                     TypeOfCodingTabDecripted[i] = 1;
309                 }
310                 else
311                 {
312                     TypeOfCodingTabDecripted[i] = 3;
313                 }
314             }
315         }
316     }
317     return TypeOfCodingTabDecripted;
318 }
319
320 public static int[] PlayfairCypherDecription(char[,] playfairTab, int [] coordinatesDecripted, int[] typeOfCodingTabDecripted)
321 {
322     for (int i = 0; i < typeOfCodingTabDecripted.Length; i++)
323     {
324         if (typeOfCodingTabDecripted[i] == 1)
325         {
326             if ((coordinatesDecripted[4 * i] - 1) % 5 < 0)
327             {
328                 coordinatesDecripted[4 * i] = ((coordinatesDecripted[4 * i] - 1) % 5) + 5;
329             }
330             else
331             {
332                 coordinatesDecripted[4 * i] = ((coordinatesDecripted[4 * i] - 1) % 5);
333             }
334             if ((coordinatesDecripted[4 * i + 2] - 1) % 5 < 0)
335             {
336                 coordinatesDecripted[4 * i + 2] = ((coordinatesDecripted[4 * i + 2] - 1) % 5) + 5;
337             }
338             else
339             {
340                 coordinatesDecripted[4 * i + 2] = (coordinatesDecripted[4 * i + 2] - 1) % 5;
341             }
342         }
343     }
344     else
345     {
346         if (typeOfCodingTabDecripted[i] == 2)
```

```

347         {
348             if ((coordinatesDecrypted[4 * i + 1] - 1) % 5 < 0)
349             {
350                 coordinatesDecrypted[4 * i + 1] =
351                 ((coordinatesDecrypted[4 * i + 1] - 1) % 5) + 5;
352             }
353             else
354             {
355                 coordinatesDecrypted[4 * i + 1] =
356                 ((coordinatesDecrypted[4 * i + 1] - 1) % 5);
357             }
358             if ((coordinatesDecrypted[4 * i + 3] - 1) % 5 < 0)
359             {
360                 coordinatesDecrypted[4 * i + 3] =
361                 ((coordinatesDecrypted[4 * i + 3] - 1) % 5) + 5;
362             }
363             else
364             {
365                 coordinatesDecrypted[4 * i + 3] =
366                 (coordinatesDecrypted[4 * i + 3] - 1) % 5;
367             }
368             }
369             }
370             }
371             }
372             }
373             }
374             return coordinatesDecrypted;
375         }
376
377     public static char[] DecryptedCoordinatesToCharTab(int[]
378     coordinatesDecrypted, char[,] playfairCharTab)
379     {
380         char[] decryptedCharTab = new char[coordinatesDecrypted.Length /
381         2];
382         for (int i = 0; i < coordinatesDecrypted.Length / 2; i++)
383         {
384             decryptedCharTab[i] = playfairCharTab[coordinatesDecrypted[2 *
385             i], coordinatesDecrypted[2 * i + 1]];
386         }
387         return decryptedCharTab;
388     }
389 }
390

```