**ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»**

**(СПбГУТ)**

Факультет «Информационных систем и технологий»

Кафедра «Автоматизации предприятий связи»

**Отчёт по лабораторным работам**

|  |
| --- |
| по дисциплине  **«Промышленные программно-вычислительные комплексы»** |

|  |
| --- |
| Выполнил:  студент группы ИСТ-941м  Колесов Д. С. |
| Принял:  Волынкин П. А. |
|  |
|

Санкт-Петербург

2020

**Задание №1**

**«Простая перестановка по ключу»**

**Описание шифра**

Шифром перестановки называется шифр, преобразования из которого изменяют только порядок следования символов исходного текста, но не изменяют их самих.

В простом шифре в качестве ключа используется прямоугольная таблица, в которой в первую строку записывают номера позиций, на которые необходимо совершить перестановку, а во вторую строку – номера символов для перестановок.

Пример:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |
| 4 | 5 | 2 | 3 | 1 |

Исходный текст: Текст

Выходной текст: стекТ

**Реализация шифра**

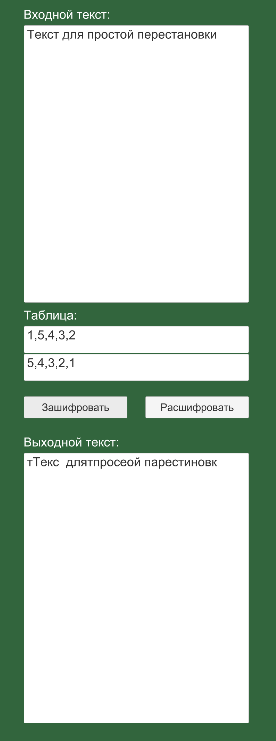


Рисунок 1. Реализация шифра простых перестановок по ключу

Реализация шифра простых перестановок по ключу показана на рисунке 1.

*Код программы:*

|  |
| --- |
| using System.Text;  using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using TMPro;  public class SingleKeyTranspositionCypher : MonoBehaviour  {  [SerializeField] TMP\_InputField inputField;  [SerializeField] TMP\_InputField keyField1;  [SerializeField] TMP\_InputField keyField2;  [SerializeField] TMP\_InputField outputField;  public void Encript()  {  if (!ValidateKey(out int keyArray1Length, out int[] keyIntArray1, keyField1.text))  return;  if (!ValidateKey(out int keyArray2Length, out int[] keyIntArray2, keyField2.text))  return;  if (keyArray1Length != keyArray2Length)  {  outputField.text = "Введён неправильный ключ";  return;  }  int[] keyIntArray = new int[keyArray1Length];  for (int i = 0; i < keyArray1Length; i++)  keyIntArray[keyIntArray1[i]] = keyIntArray2[i];  string textToEncript = inputField.text;  int textLength = textToEncript.Length;  int numberOfSplits = textLength % keyArray1Length == 0 ? textLength / keyArray1Length : textLength / keyArray1Length + 1;  StringBuilder sb = new StringBuilder();  for (int i = 0; i < numberOfSplits; i++)  {  for (int j = 0; j < keyArray1Length; j++)  {  if ((i \* keyArray1Length + keyIntArray[j]) >= textLength)  sb.Append(' ');  else sb.Append(textToEncript[i \* keyArray1Length + keyIntArray[j]]);  }  }  outputField.text = sb.ToString();  }  public void Decript()  {  if (!ValidateKey(out int keyArray1Length, out int[] keyIntArray1, keyField1.text))  return;  if (!ValidateKey(out int keyArray2Length, out int[] keyIntArray2, keyField2.text))  return;  if (keyArray1Length != keyArray2Length)  {  outputField.text = "Введён неправильный ключ";  return;  }  int[] reverseKeyIntArray = new int[keyArray1Length];  for (int i = 0; i < keyArray1Length; i++)  reverseKeyIntArray[keyIntArray2[i]] = keyIntArray1[i];  string textToEncript = inputField.text;  int textLength = textToEncript.Length;  int numberOfSplits = textLength % keyArray1Length == 0 ? textLength / keyArray1Length : textLength / keyArray1Length + 1;  StringBuilder sb = new StringBuilder();  for (int i = 0; i < numberOfSplits; i++)  {  for (int j = 0; j < keyArray1Length; j++)  {  if ((i \* keyArray1Length + reverseKeyIntArray[j]) >= textLength)  sb.Append(' ');  else sb.Append(textToEncript[i \* keyArray1Length + reverseKeyIntArray[j]]);  }  }  outputField.text = sb.ToString();  }  private bool ValidateKey(out int keyArrLength, out int[] keyArr, string keyText)  {  keyArrLength = 0;  keyArr = new int[0];  if (keyText.Length == 0)  {  outputField.text = "Введите ключ";  return false;  }  string[] keyArray = keyText.Split(new char[] { ' ', ',' }, System.StringSplitOptions.RemoveEmptyEntries);  int keyArrayLength = keyArray.Length;  int[] keyExistenceIntArray = new int[keyArrayLength];  int[] keyIntArray = new int[keyArrayLength];  for (int i = 0; i < keyArrayLength; i++)  {  int tempKey = 0;  if (int.TryParse(keyArray[i], out tempKey))  {  if (tempKey <= keyArrayLength && tempKey > 0)  {  keyExistenceIntArray[tempKey - 1] = 1;  keyIntArray[i] = tempKey - 1;  }  else  {  outputField.text = "Введён неверный ключ";  return false;  }  }  else  {  outputField.text = "Введён неверный ключ";  return false;  }  }  foreach (int keyExistence in keyExistenceIntArray)  if (keyExistence == 0)  {  outputField.text = "Введён неверный ключ";  return false;  }  keyArrLength = keyArrayLength;  keyArr = keyIntArray;  return true;  }  } |

**Задание №2**

**«Одиночная перестановка по ключу»**

**Описание шифра**

В этом шифре используется прямоугольная таблица, в которую сообщение записывается по строкам слева направо, затем, переставляются столбцы, и выписывается шифрограмма по горизонтали.

Пусть, ключом будет: **3, 4, 5, 2, 1**.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 3 | 4 | 5 | 2 | 1 |
| П | р | и | в | е |
| т |  | А | н | я |

Выходной текст: евПриянт А

**Реализация шифра**

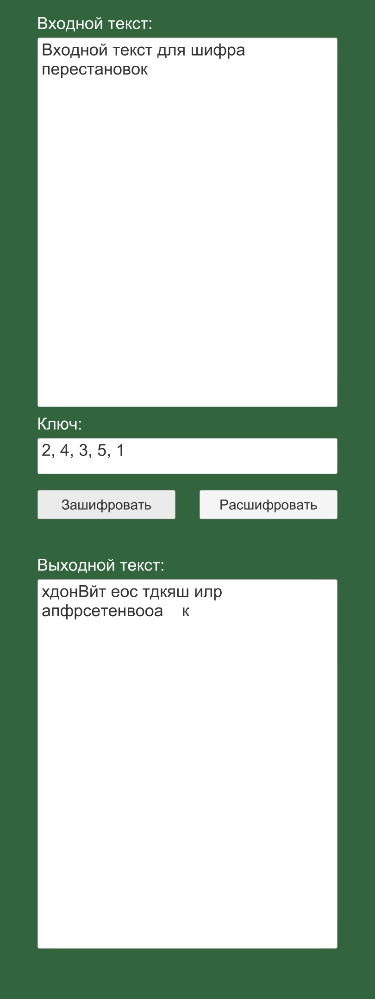


Рисунок 2. Реализация шифра одиночных перестановок по ключу

Реализация шифра одиночных перестановок по ключу показана на рисунке 2.

*Код программы:*

|  |
| --- |
| using System;  using System.Text;  using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using TMPro;  public class KeyTranspositionCypher : MonoBehaviour  {  [SerializeField] TMP\_InputField inputField;  [SerializeField] TMP\_InputField keyField;  [SerializeField] TMP\_InputField outputField;  public void Encript()  {  if (!ValidateKey(out int keyArrayLength, out int[] keyIntArray))  return;  string textToEncript = inputField.text;  int textLength = textToEncript.Length;  int numberOfRows = textLength % keyArrayLength == 0 ? textLength / keyArrayLength : textLength / keyArrayLength + 1;  StringBuilder sb = new StringBuilder();  for (int i = 0; i < numberOfRows; i++)  {  for (int j = 0; j < keyArrayLength; j++)  {  if ((i \* keyArrayLength + keyIntArray[j]) >= textLength)  sb.Append(' ');  else sb.Append(textToEncript[i \* keyArrayLength + keyIntArray[j]]);  }  }  outputField.text = sb.ToString().Trim();  }  public void Decript()  {  if (!ValidateKey(out int keyArrayLength, out int[] keyIntArray))  return;  int[] reverseKeyIntArray = new int[keyIntArray.Length];  for (int i = 0; i < keyIntArray.Length; i++)  reverseKeyIntArray[keyIntArray[i]] = i;  string textToEncript = inputField.text;  int textLength = textToEncript.Length;  int numberOfSplits = textLength % keyArrayLength == 0 ? textLength / keyArrayLength : textLength / keyArrayLength + 1;  StringBuilder sb = new StringBuilder();  for (int i = 0; i < numberOfSplits; i++)  {  for (int j = 0; j < keyArrayLength; j++)  {  if ((i \* keyArrayLength + reverseKeyIntArray[j]) >= textLength)  sb.Append(' ');  else sb.Append(textToEncript[i \* keyArrayLength + reverseKeyIntArray[j]]);  }  }  outputField.text = sb.ToString().Trim();  }  private bool ValidateKey(out int keyArrLength, out int[] keyArr)  {  keyArrLength = 0;  keyArr = new int[0];  string key = keyField.text;  if (key.Length == 0)  {  outputField.text = "Введите ключ";  return false;  }  bool isValidate;  if (char.IsDigit(key[0]))  isValidate = ValidateDigitalKey(out keyArrLength, out keyArr);  else isValidate = ValidateWordKey(out keyArrLength, out keyArr);  return isValidate;  }  private bool ValidateDigitalKey(out int keyArrLength, out int[] keyArr)  {  keyArrLength = 0;  keyArr = new int[0];  string key = keyField.text;  string[] keyArray = key.Split(new char[] { ' ', ',' }, System.StringSplitOptions.RemoveEmptyEntries);  int keyArrayLength = keyArray.Length;  int[] keyExistenceIntArray = new int[keyArrayLength];  int[] keyIntArray = new int[keyArrayLength];  for (int i = 0; i < keyArrayLength; i++)  {  int tempKey;  if (int.TryParse(keyArray[i], out tempKey))  {  if (tempKey <= keyArrayLength && tempKey > 0)  {  keyExistenceIntArray[tempKey - 1] = 1;  keyIntArray[i] = tempKey - 1;  }  else  {  outputField.text = "Введён неверный ключ";  return false;  }  }  else  {  outputField.text = "Введён неверный ключ";  return false;  }  }  foreach (int keyExistence in keyExistenceIntArray)  if (keyExistence == 0)  {  outputField.text = "Введён неверный ключ";  return false;  }  keyArrLength = keyArrayLength;  keyArr = keyIntArray;  return true;  }  private bool ValidateWordKey(out int keyArrLength, out int[] keyArr)  {  keyArrLength = 0;  keyArr = new int[0];  string key = keyField.text.ToLower();  int keyLength = key.Length;  int[] subKeyArray = new int[keyLength];  for (int i = 0; i < keyLength; i++)  {  subKeyArray[i] = key[i];  for (int j = 0; j < i; j++)  if (subKeyArray[j] == subKeyArray[i])  {  outputField.text = "Введён неверный ключ";  return false;  }  }  int[] sortKeyArray = subKeyArray.Clone() as int[];  Array.Sort(sortKeyArray);  for (int i = 0; i < keyLength; i++)  {  for (int j = 0; j < keyLength; j++)  {  if (subKeyArray[i] == sortKeyArray[j])  {  subKeyArray[i] = j;  break;  }  }  }  keyArrLength = keyLength;  keyArr = subKeyArray;  return true;  }  } |

**Задание №3**

**«Двойная перестановка по ключу»**

**Описание шифра**

При шифровании шифром двойной перестановки в таблицу по определённому маршруту записывается текст, затем переставляются столбцы и строки. Далее по определённому маршруту выписывается шифрограмма.

Пусть, ключами будут:

Горизонтальный: **3, 4, 5, 2, 1**.

Вертикальный: **3, 1, 2**.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 3 | 4 | 5 | 2 | 1 |
| 3 | Э | т | о |  | с |
| 1 | о | о | б | щ | е |
| 2 | н | и | е | . |  |

Выходной текст: ещооб .ниес Это

**Реализация шифра**

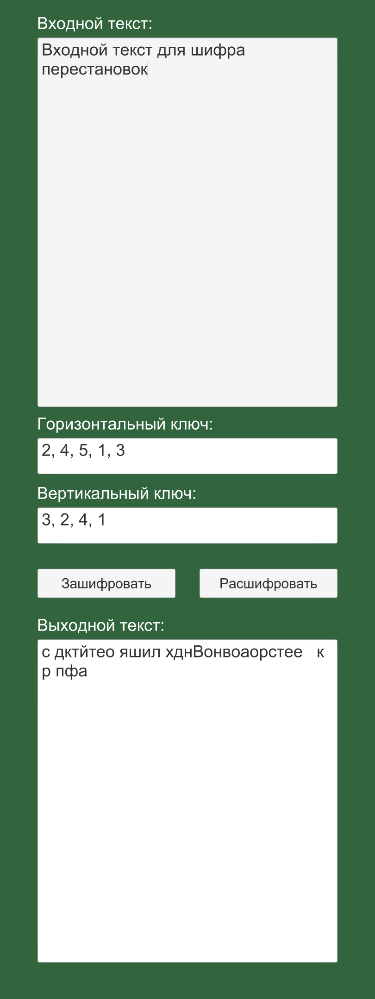


Рисунок 3. Реализация шифра двойных перестановок по ключу

Реализация шифра двойных перестановок по ключу показана на рисунке 3.

*Код программы:*

|  |
| --- |
| using System;  using System.Text;  using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using TMPro;  public class DoubleKeyTranspositionCypher : MonoBehaviour  {  [SerializeField] TMP\_InputField inputField;  [SerializeField] TMP\_InputField horizontalKeyField;  [SerializeField] TMP\_InputField verticalKeyField;  [SerializeField] TMP\_InputField outputField;  public void Encript()  {  if (!StaticValidator.ValidateWDKey(horizontalKeyField, outputField, out int horizontalKeyArrayLength, out int[] horizontalKeyIntArray))  return;  if (!StaticValidator.ValidateWDKey(verticalKeyField, outputField, out int verticalKeyArrayLength, out int[] verticalKeyIntArray))  return;  string textToEncript = inputField.text;  int textLength = textToEncript.Length;  int numberOfRows = textLength % horizontalKeyArrayLength == 0 ? textLength / horizontalKeyArrayLength : textLength / horizontalKeyArrayLength + 1;  int squareMultiplier = numberOfRows % verticalKeyArrayLength == 0 ? numberOfRows / verticalKeyArrayLength : numberOfRows / verticalKeyArrayLength + 1;  StringBuilder sb = new StringBuilder();  for (int m = 0; m < squareMultiplier; m++)  {  int quareShift = m \* horizontalKeyArrayLength \* verticalKeyArrayLength;  for (int i = 0; i < /\*numberOfRows\*/verticalKeyArrayLength; i++)  {  for (int j = 0; j < horizontalKeyArrayLength; j++)  {  if ((quareShift + verticalKeyIntArray[i] \* horizontalKeyArrayLength + horizontalKeyIntArray[j]) >= textLength)  sb.Append(' ');  else sb.Append(textToEncript[quareShift + verticalKeyIntArray[i] \* horizontalKeyArrayLength + horizontalKeyIntArray[j]]);  }  }  }  outputField.text = sb.ToString().Trim();  }  public void Decript()  {  if (!StaticValidator.ValidateWDKey(horizontalKeyField, outputField, out int horizontalKeyArrayLength, out int[] horizontalKeyIntArray))  return;  if (!StaticValidator.ValidateWDKey(verticalKeyField, outputField, out int verticalKeyArrayLength, out int[] verticalKeyIntArray))  return;  int[] horizontalReverseKeyIntArray = new int[horizontalKeyIntArray.Length];  for (int i = 0; i < horizontalKeyIntArray.Length; i++)  horizontalReverseKeyIntArray[horizontalKeyIntArray[i]] = i;  int[] verticalReverseKeyIntArray = new int[verticalKeyIntArray.Length];  for (int i = 0; i < verticalKeyIntArray.Length; i++)  verticalReverseKeyIntArray[verticalKeyIntArray[i]] = i;  string textToEncript = inputField.text;  int textLength = textToEncript.Length;  int numberOfRows = textLength % horizontalKeyArrayLength == 0 ? textLength / horizontalKeyArrayLength : textLength / horizontalKeyArrayLength + 1;  int squareMultiplier = numberOfRows % verticalKeyArrayLength == 0 ? numberOfRows / verticalKeyArrayLength : numberOfRows / verticalKeyArrayLength + 1;  StringBuilder sb = new StringBuilder();  for (int m = 0; m < squareMultiplier; m++)  {  int quareShift = m \* horizontalKeyArrayLength \* verticalKeyArrayLength;  for (int i = 0; i < /\*numberOfRows\*/verticalKeyArrayLength; i++)  {  for (int j = 0; j < horizontalKeyArrayLength; j++)  {  if ((quareShift + verticalReverseKeyIntArray[i] \* horizontalKeyArrayLength + horizontalReverseKeyIntArray[j]) >= textLength)  sb.Append(' ');  else sb.Append(textToEncript[quareShift + verticalReverseKeyIntArray[i] \* horizontalKeyArrayLength + horizontalReverseKeyIntArray[j]]);  }  }  }  outputField.text = sb.ToString().Trim();  }  } |

**Задание №4**

**«Маршрутная перестановка»**

**Описание шифра**

В маршрутных перестановках преобразования состоят в том, что отрезок открытого текста записывается в некоторую геометрическую фигуру по одной траектории, а выписывается – по другой.

Пример маршрутной перестановки на таблице:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Э | т | о |  | л |
| и | ш | ь |  | п |
| р | и | м | е | р |

Выходной текст: лпр еоьмтшиЭир

**Реализация шифра**

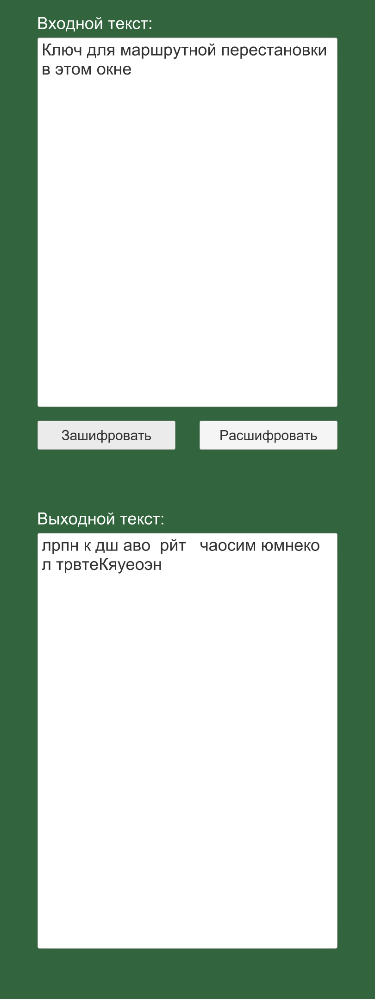


Рисунок 4. Реализация шифра маршрутной перестановки

Реализация шифра маршрутной перестановки показана на рисунке 4.

*Код программы:*

|  |
| --- |
| using System.Text;  using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using TMPro;  public class RouteTranspositionCypher : MonoBehaviour  {  [SerializeField] TMP\_InputField inputField;  [SerializeField] TMP\_InputField outputField;  public void Encript()  {  string textToEncript = inputField.text;  int textLength = textToEncript.Length;  int oneSideDemention = (int)Mathf.Sqrt(textLength);  if (oneSideDemention \* oneSideDemention != textLength)  oneSideDemention++;  StringBuilder sb = new StringBuilder();  for (int i = oneSideDemention - 1; i >= 0; i--)  {  for (int j = 0; j < oneSideDemention; j++)  {  if ((j \* oneSideDemention + i) < textLength)  sb.Append(textToEncript[j \* oneSideDemention + i]);  else sb.Append(' ');  }  }  outputField.text = sb.ToString();  }  public void Decript()  {  string textToDecript = inputField.text;  int textLength = textToDecript.Length;  int oneSideDemention = (int)Mathf.Sqrt(textLength);  if (oneSideDemention \* oneSideDemention != textLength)  oneSideDemention++;  StringBuilder sb = new StringBuilder();  for (int i = 0; i < oneSideDemention; i++)  {  for (int j = oneSideDemention - 1; j >= 0; j--)  {  if ((j \* oneSideDemention + i) < textLength)  sb.Append(textToDecript[j \* oneSideDemention + i]);  else sb.Append(' ');  }  }  outputField.text = sb.ToString().Trim();  }  } |

**Задание №5**

**«Поворотная решётка Кардано»**

**Описание шифра**

Решётка – квадрат клеток, некоторые из которых вырезаны (в том числе, возможно, и имеющие общую сторону или вершину). Клетки должны иметь такой размер, чтобы в каждую помещалась ровно одна буква. Вырезанные клетки должны располагаться таким образом, чтобы никакие две из них не оказывались в одном и том же месте при поворотах решётки.

Чтобы зашифровать сообщение, нужно разместить решётку на бумаге и вписать текст в вырезанные клетки, затем повернуть решётку на 90° и вписать следующую часть и т. д.

Пусть, исходным текстом будет: Это сообщение!

Поворотная решётка выглядит следующим образом:

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 4 | 2 | 1 |
| 2 | 3 | 3 | 4 |
| 4 | 3 | 3 | 2 |
| 1 | 2 | 4 | 1 |

Выходной текст:

|  |  |  |  |
| --- | --- | --- | --- |
| Э | ! | с | т |
| о | щ | е |  |
|  | н | и | о |
| о | б |  |  |

Или: «Э!стоще ниооб »

**Реализация шифра**

Реализация шифра поворотной решётки Кардано показана на рисунке 5.

*Код программы:*

|  |
| --- |
| using System;  using System.Text;  using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using TMPro;  public class TurningCardanGrilleCypher : MonoBehaviour  {  [SerializeField] TMP\_InputField inputField;  [SerializeField] TMP\_InputField outputField;  public void Encript()  {  string textToEncript = inputField.text;  int textLength = textToEncript.Length;  int[] grid = GetCardanGrilleOrder(0);  int numberOfLettersInGrid = grid.Length;  int numberOfLettersInRow = (int) Mathf.Sqrt(grid.Length);  int numberOfGrids = textLength % numberOfLettersInGrid == 0 ? textLength / numberOfLettersInGrid : textLength / numberOfLettersInGrid + 1;  StringBuilder sb = new StringBuilder();  for (int i = 0; i < numberOfGrids; i++)  {  for (int j = 0; j < numberOfLettersInGrid; j++)  {  if (j != 0 && j % numberOfLettersInRow == 0)  sb.Append("\n");  if ((i \* numberOfLettersInGrid + grid[j]) < textLength)  sb.Append(textToEncript[i \* numberOfLettersInGrid + grid[j]]);  else sb.Append(' ');  }  if (i != numberOfGrids - 1)  sb.Append("\n\n");  }  outputField.text = sb.ToString();  }  public void Decript()  {  string textToDecript = string.Join("", inputField.text.Split(new char[] { '\n' }, StringSplitOptions.RemoveEmptyEntries));  int textLength = textToDecript.Length;  int[] grid = GetCardanGrilleOrder(0);  int numberOfLettersInGrid = grid.Length;  int numberOfGrids = textLength % numberOfLettersInGrid == 0 ? textLength / numberOfLettersInGrid : textLength / numberOfLettersInGrid + 1;  int[] reverseGrid = new int[numberOfLettersInGrid];  for (int i = 0; i < numberOfLettersInGrid; i++)  reverseGrid[grid[i]] = i;  StringBuilder sb = new StringBuilder();  for (int i = 0; i < numberOfGrids; i++)  {  for (int j = 0; j < numberOfLettersInGrid; j++)  {  if ((i \* numberOfLettersInGrid + reverseGrid[j]) < textLength)  sb.Append(textToDecript[i \* numberOfLettersInGrid + reverseGrid[j]]);  else sb.Append(' ');  }  }  outputField.text = sb.ToString().Trim();  }  private int[] GetCardanGrilleOrder(int value)  {  switch (value)  {  default:  return new int[16] { 1, 4, 10, 15, 2, 7, 9, 12, 0, 5, 11, 14, 3, 6, 8, 13 };  }  }  } |

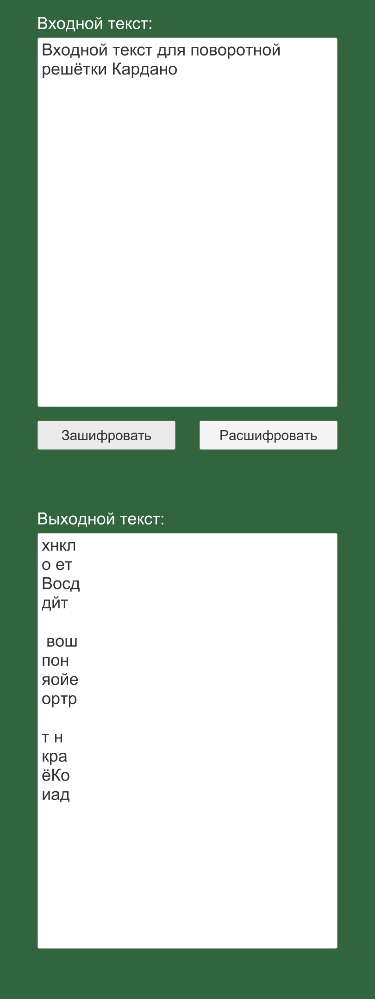


Рисунок 5. Реализация шифра поворотной решётки Кардано

**Задание №6**

**«Магический квадрат»**

**Описание шифра**

Магический квадрат — это квадратная таблица , заполненная числами таким образом, что сумма чисел в каждой строке, каждом столбце и на обеих диагоналях одинакова. Если в квадрате равны суммы чисел только в строках и столбцах, то он называется полумагическим. Нормальным называется магический квадрат, заполненный натуральными числами от до .

Подобные квадраты используется для вписывания шифруемого текста по приведенной в них нумерации.

Пусть, исходным текстом будет: Сообщение

Поворотная решётка выглядит следующим образом:

|  |  |  |
| --- | --- | --- |
| 2 | 7 | 6 |
| 9 | 5 | 1 |
| 4 | 3 | 8 |

Выходной текст:

|  |  |  |
| --- | --- | --- |
| о | н | е |
| е | щ | С |
| б | о | и |

Или: «онеещСбои»

**Реализация шифра**

Реализация шифра магическим квадратом показана на рисунке 6.

*Код программы:*

|  |
| --- |
| using System;  using System.Text;  using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using TMPro;  public class MagicSquareCypher : MonoBehaviour  {  [SerializeField] TMP\_InputField inputField;  [SerializeField] TMP\_InputField outputField;  private static readonly int[] magicSquare = new int[16] {11,0,3,4,1,14,2,9,15,5,12,7,10,6,13,8};  public void Encript()  {  string textToEncript = inputField.text;  int textLength = textToEncript.Length;  int[] grid = magicSquare;  int numberOfLettersInGrid = grid.Length;  int numberOfLettersInRow = (int)Mathf.Sqrt(grid.Length);  int numberOfGrids = textLength % numberOfLettersInGrid == 0 ? textLength / numberOfLettersInGrid : textLength / numberOfLettersInGrid + 1;  StringBuilder sb = new StringBuilder();  for (int i = 0; i < numberOfGrids; i++)  {  for (int j = 0; j < numberOfLettersInGrid; j++)  {  if (j != 0 && j % numberOfLettersInRow == 0) sb.Append("\n");  if ((i \* numberOfLettersInGrid + grid[j]) < textLength)  sb.Append(textToEncript[i \* numberOfLettersInGrid + grid[j]]);  else sb.Append(' ');  }  if (i != numberOfGrids - 1) sb.Append("\n\n");  }  outputField.text = sb.ToString();  }  public void Decript()  {  string textToDecript = string.Join("", inputField.text.Split(new char[] { '\n' }, StringSplitOptions.RemoveEmptyEntries));  int textLength = textToDecript.Length;  int[] grid = magicSquare;  int numberOfLettersInGrid = grid.Length;  int numberOfGrids = textLength % numberOfLettersInGrid == 0 ? textLength / numberOfLettersInGrid : textLength / numberOfLettersInGrid + 1;  int[] reverseGrid = new int[numberOfLettersInGrid];  for (int i = 0; i < numberOfLettersInGrid; i++) reverseGrid[grid[i]] = i;  StringBuilder sb = new StringBuilder();  for (int i = 0; i < numberOfGrids; i++)  {  for (int j = 0; j < numberOfLettersInGrid; j++)  {  if ((i \* numberOfLettersInGrid + reverseGrid[j]) < textLength)  sb.Append(textToDecript[i \* numberOfLettersInGrid + reverseGrid[j]]);  else sb.Append(' ');  }  }  outputField.text = sb.ToString().Trim();  }  } |

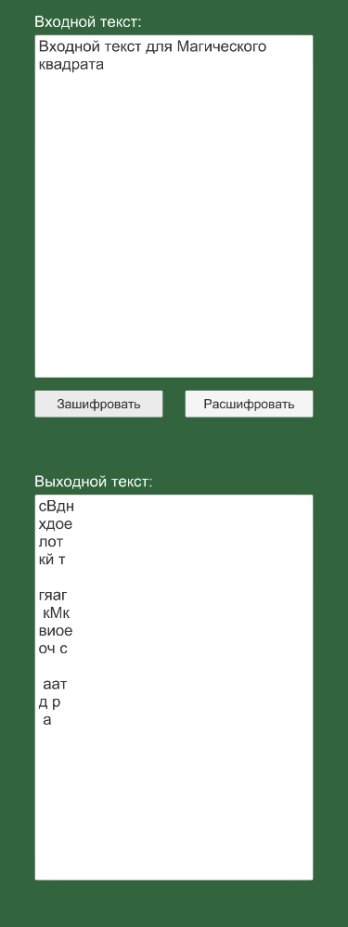


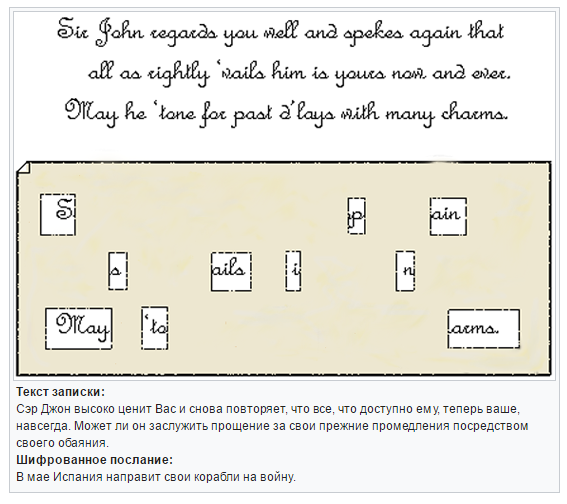
Рисунок 6. Реализация шифра магическим квадратом

**Задание №7**

**«Шифр Ришелье»**

**Описание шифра**

Для реализации шифра Ришелье, лист с решёткой Кардано накладывался на лист бумаги, и в окна вписывался текст, содержащий закодированную информацию. Далее сетка убиралась и в промежутки между текстом документа вписывались произвольные слова или набор букв, с целью запутать того, для кого сообщения не было предназначено:



**Реализация шифра**

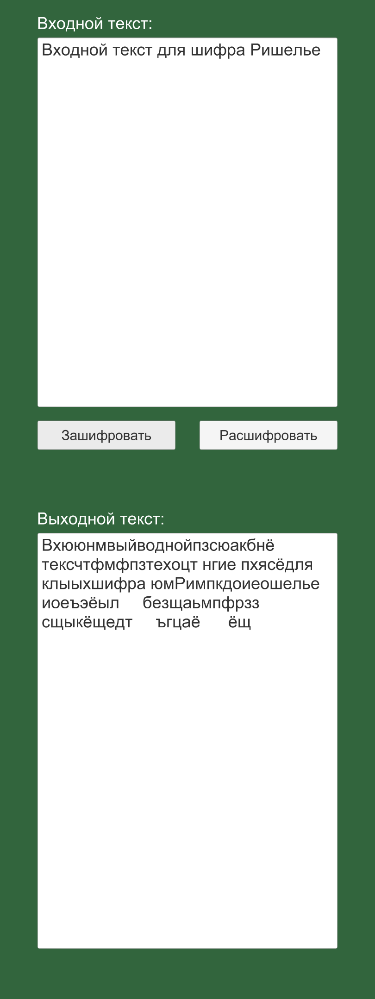


Рисунок 7. Реализация шифра Ришелье

Реализация шифра Ришелье показана на рисунке 7.

*Код программы:*

|  |
| --- |
| using System;  using System.Text;  using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using TMPro;  public class RichelieuCypher : MonoBehaviour  {  [SerializeField] TMP\_InputField inputField;  [SerializeField] TMP\_InputField outputField;  public void Encript()  {  string textToEncript = inputField.text;  int textLength = textToEncript.Length;  int numberOfLettersInBlock;  int[] grid = GetRichelieuGrille(out numberOfLettersInBlock);  int numberOfGrids = textLength % numberOfLettersInBlock == 0 ? textLength / numberOfLettersInBlock : textLength / numberOfLettersInBlock + 1;  char[] charSet = { '?' };  if (textToEncript != null && textToEncript.Length > 0)  {  if ((int)textToEncript[0] > 1000)  charSet = Alphabets.russianWithSpace;  else charSet = Alphabets.englishWithSpace;  }  System.Random random = new System.Random();  int iterator = 0;  StringBuilder sb = new StringBuilder();  while (iterator < textLength)  {  for (int i = 0; i < numberOfGrids; i++)  {  for (int j = 0; j < grid.Length; j++)  {  if (iterator < textToEncript.Length)  {  if (grid[j] == 1)  {  sb.Append(textToEncript[iterator]);  iterator++;  }  else  sb.Append(charSet[random.Next(0, charSet.Length)]);  }  else  {  if (grid[j] == 1)  sb.Append(' ');  else sb.Append(charSet[random.Next(0, charSet.Length)]);  }  }  }  }  outputField.text = sb.ToString();  }  public void Decript()  {  string textToEncript = inputField.text;  int textLength = textToEncript.Length;  int[] grid = GetRichelieuGrille(out \_);  StringBuilder sb = new StringBuilder();  for (int i = 0; i < textLength; i++)  {  int gridI = i % grid.Length;  if (grid[gridI] == 1)  sb.Append(textToEncript[i]);  }  outputField.text = sb.ToString().Trim();  }  private int[] GetRichelieuGrille(out int numberOfFreeHoles)  {  int[] grid = new int[70];  int free = 0;  System.Random random = new System.Random(70);  bool holes = true;  int iterator = 0;  int randNumber = random.Next(1, 7);  while (iterator < grid.Length)  {  if (holes)  {  grid[iterator] = 1;  free++;  }  else grid[iterator] = 0;  iterator++;  randNumber--;  if (randNumber <= 0) {  if (holes)  randNumber = random.Next(5, 15);  else randNumber = random.Next(1, 7);  holes = !holes;  }  }  numberOfFreeHoles = free;  return grid;  }  } |

**Задание №8**

**«Шифр Порты»**

**Описание шифра**

Шифр Порты является биграммным шифром. В таблицу наверху горизонтально и слева вертикально записывается стандартный алфавит. В ячейках таблицы можно записать любые числа, буквы или символы, при условии, что содержимое ни одной из ячеек не повторяется. Каждая биграмма на выходе заменяется на соответствующее значение ячейки для этой биграммы.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **А** | **Б** | **В** | **Г** | **Д** | **Е** | **Ё** | **Ж** | **З** | **И** | **Й** |  |
| **А** | r0000 | r0001 | r0002 | r0003 | r0004 | r0005 | r0006 | r0007 | r0008 | r0009 | r0010 |  |
| **Б** | r0100 | r0101 | r0102 | r0103 | r0104 | r0105 | r0106 | r0107 | r0108 | r0109 | r0110 |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |

**Реализация шифра**

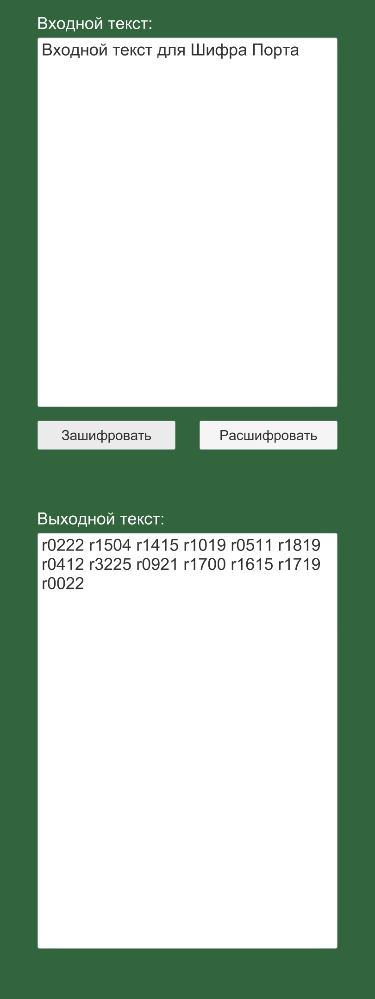


Рисунок 8. Реализация шифра Порты

Реализация шифра Порты показана на рисунке 8.

*Код программы:*

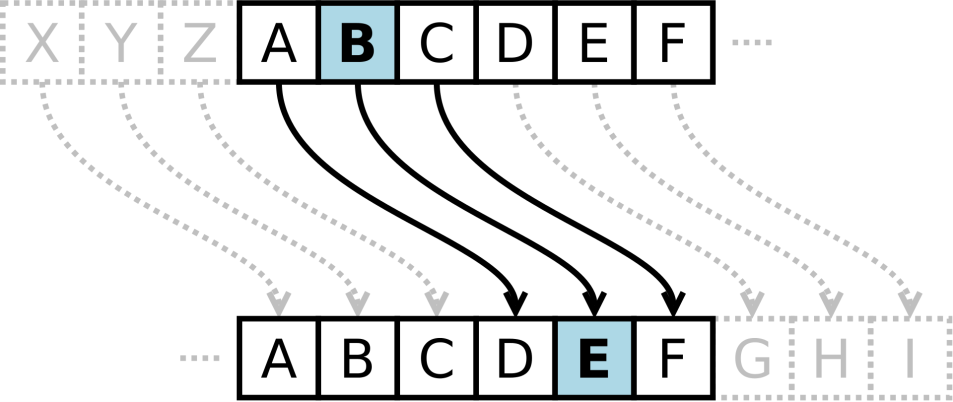
|  |
| --- |
| using System;  using System.Text;  using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using TMPro;  class PortaCypher : MonoBehaviour  {  [SerializeField] TMP\_InputField inputField;  [SerializeField] TMP\_InputField outputField;    public void Encript()  {  string textToEncript = string.Join("", inputField.text.ToLower().Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries));  if (textToEncript.Length > 0)  {  Dictionary<string, List<char>> alphabets = new Dictionary<string, List<char>>();  alphabets.Add("r", new List<char>(Alphabets.russian));  alphabets.Add("e", new List<char>(Alphabets.english));  CharEnumerator inCE = textToEncript.GetEnumerator();  StringBuilder sb = new StringBuilder();  char tempChar;  KeyValuePair<string, char>? firstChar = null;  bool moving = inCE.MoveNext();  while (moving || firstChar.HasValue)  {  if (moving)  {  tempChar = inCE.Current;  bool findList = false;  List<char> tempList;  foreach (string key in alphabets.Keys)  {  if (alphabets.TryGetValue(key, out tempList))  {  if (tempList.Contains(tempChar))  {  findList = true;  if (firstChar.HasValue)  {  if (firstChar.Value.Key == key)  {  sb.Append(key);  sb.AppendFormat("{0:00}", tempList.IndexOf(firstChar.Value.Value));  sb.AppendFormat("{0:00}", tempList.IndexOf(tempChar));  sb.Append(" ");  firstChar = null;  break;  }  else  {  outputField.text = "Неправильный входной текст";  return;  }  }  else  {  firstChar = new KeyValuePair<string, char>(key, tempChar);  }  }  }  }  if (!findList)  {  outputField.text = "Неправильный входной текст";  return;  }  }  else  {  if (firstChar.Value.Key == "r")  tempChar = 'х';  else tempChar = 'x';  if (alphabets.TryGetValue(firstChar.Value.Key, out List<char> tempList))  {  sb.Append(firstChar.Value.Key);  sb.AppendFormat("{0:00}", tempList.IndexOf(firstChar.Value.Value));  sb.AppendFormat("{0:00}", tempList.IndexOf(tempChar));  }  firstChar = null;  }  moving = inCE.MoveNext();  }  outputField.text = sb.ToString().Trim();  }  }  public void Decript()  {  string[] textToEncript = inputField.text.Split(' ');  Dictionary<string, List<char>> alphabets = new Dictionary<string, List<char>>();  alphabets.Add("r", new List<char>(Alphabets.russian));  alphabets.Add("e", new List<char>(Alphabets.english));  StringBuilder sb = new StringBuilder();  foreach (string pair in textToEncript)  {  List<char> tempList;  if (alphabets.TryGetValue(pair[0].ToString(), out tempList))  {  if (int.TryParse((pair[1].ToString() + pair[2].ToString()), out int it1) && int.TryParse((pair[3].ToString() + pair[4].ToString()), out int it2))  {  sb.Append(tempList[it1]);  sb.Append(tempList[it2]);  }  else  {  outputField.text = "Неправильный входной текст";  return;  }  }  else  {  outputField.text = "Неправильный входной текст";  return;  }  }  outputField.text = sb.ToString().Trim();  }  } |

**Задание №9**

**«Шифр Цезаря»**

**Описание шифра**

Шифр Цезаря – это вид шифра подстановки, в котором каждый символ в открытом тексте заменяется символом, находящимся на некотором постоянном числе позиций левее или правее него в алфавите, в соответствии с ключом. Например, в шифре со сдвигом вправо на 3, А была бы заменена на Г, Б станет Д, и так далее.



**Реализация шифра**

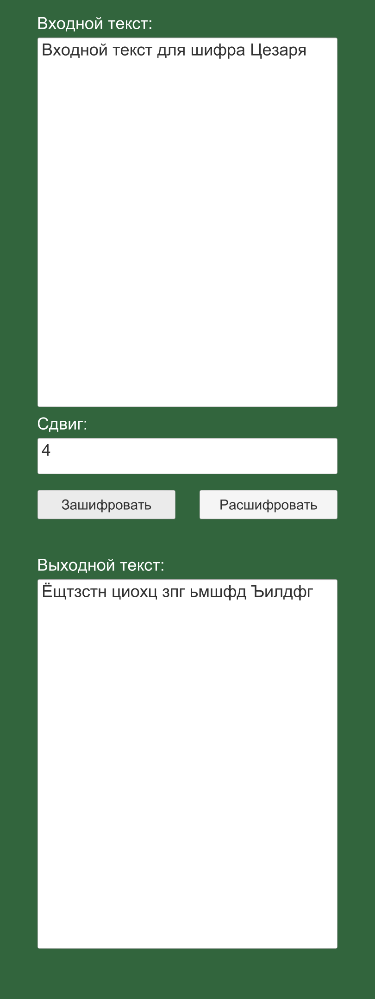


Рисунок 9. Реализация шифра Цезаря

Реализация шифра Цезаря показана на рисунке 9.

Блок-схема шифра Цезаря показана на рисунке 10.

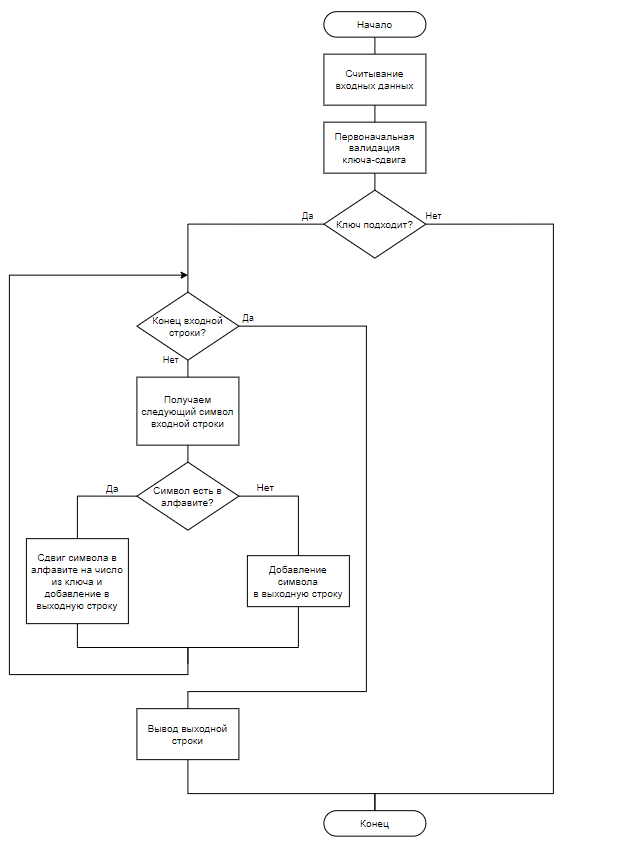


Рисунок 10. Блок-схема шифра Цезаря

*Код программы:*

|  |
| --- |
| using System;  using System.Text;  using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using TMPro;  public class CesarCypher : MonoBehaviour  {  [SerializeField] TMP\_InputField inputField;  [SerializeField] TMP\_InputField shiftField;  [SerializeField] TMP\_InputField outputField;  public void Encript()  {  DoCesar(int.Parse(shiftField.text));  }  public void Decript()  {  DoCesar(-int.Parse(shiftField.text));  }  private void DoCesar(int shift)  {  string textToEncript = inputField.text;  if (shift != 0)  {  if (textToEncript.Length > 0)  {  CharEnumerator inCE = textToEncript.GetEnumerator();  StringBuilder sb = new StringBuilder();  List<List<char>> listOfAlphabets = new List<List<char>>();  listOfAlphabets.Add(new List<char>(Alphabets.russian));  listOfAlphabets.Add(new List<char>(Alphabets.russianBig));  listOfAlphabets.Add(new List<char>(Alphabets.english));  listOfAlphabets.Add(new List<char>(Alphabets.englishBig));  char tempChar;  while (inCE.MoveNext())  {  tempChar = inCE.Current;  int tempShift;  bool findList = false;  foreach (List<char> alphabet in listOfAlphabets)  {  if (alphabet.Contains(tempChar))  {  tempShift = shift;  while (tempShift < 0)  tempShift += alphabet.Count;  tempShift = tempShift % alphabet.Count;  sb.Append(alphabet[(alphabet.IndexOf(tempChar) + tempShift) % alphabet.Count]);  findList = true;  break;  }  }  if (!findList)  sb.Append(tempChar);  }  outputField.text = sb.ToString();  }  else outputField.text = "";  }  else outputField.text = textToEncript;  }  } |

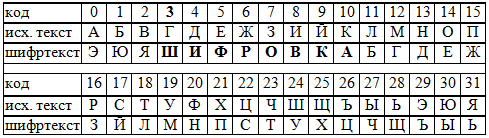
**Задание №10**

**«Шифр Цезаря с ключом»**

**Описание шифра**

Выбирается некий сдвиг, и слово или короткую фразу в качестве ключевого слова. Желательно, чтобы все буквы ключевого слова были различными. Пусть выбраны слово «Шифровка» в качестве ключевого слова и сдвиг равный 3. Ключевое слово записывается под буквами алфавита, начиная с буквы, числовой код которой совпадает с выбранным сдвигом.

Оставшиеся буквы алфавита подстановки записываются после ключевого слова в алфавитном порядке. Теперь мы имеем подстановку для каждой буквы произвольного сообщения.



**Реализация шифра**

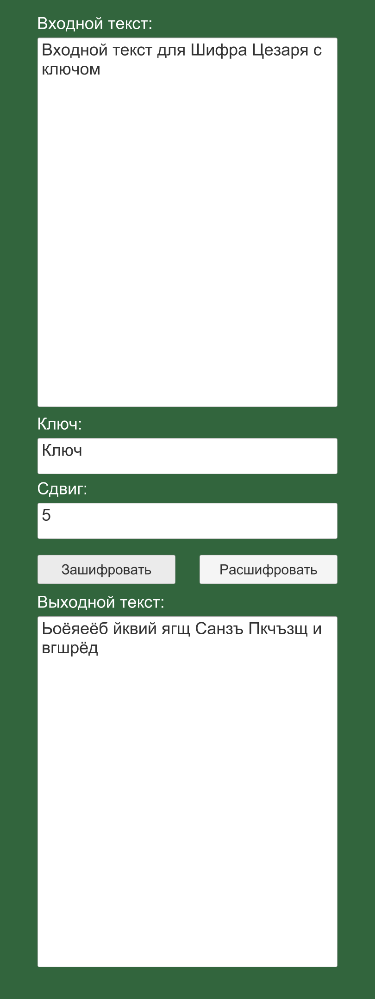


Рисунок 11. Реализация шифра Цезаря с ключом

Реализация шифра Цезаря с ключом показана на рисунке 11.

*Код программы:*

|  |
| --- |
| using System;  using System.Text;  using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using TMPro;  public class CesarCypherWithKey : MonoBehaviour  {  [SerializeField] TMP\_InputField inputField;  [SerializeField] TMP\_InputField keyField;  [SerializeField] TMP\_InputField shiftField;  [SerializeField] TMP\_InputField outputField;  public void Encript()  {  int shift = shiftField.text.Length == 0 ? 0 : - int.Parse(shiftField.text);  string key = keyField.text;  string textToEncript = inputField.text;  if (key != null && key.Length != 0)  {  List<char> currentAlphabet;  if (!ValidateKey(key, out currentAlphabet))  outputField.text = "Неправильный ключ";  else if (shift != 0)  {  if (textToEncript.Length > 0)  {  CharEnumerator inCE = textToEncript.GetEnumerator();  StringBuilder sb = new StringBuilder();  List<char> transKey = new List<char>();  foreach (char letter in key)  if (!transKey.Contains(char.ToLower(letter)) && letter != ' ')  transKey.Add(char.ToLower(letter));  foreach (char letter in currentAlphabet)  if (!transKey.Contains(letter))  transKey.Add(letter);  int tempShift = shift;  while (tempShift < 0)  tempShift += transKey.Count;  tempShift = tempShift % transKey.Count;  char tempChar;  while (inCE.MoveNext())  {  tempChar = inCE.Current;  if (currentAlphabet.Contains(char.ToLower(tempChar)))  sb.Append(char.IsLower(tempChar) ? transKey[(currentAlphabet.IndexOf(char.ToLower(tempChar)) + tempShift) % transKey.Count] : char.ToUpper(transKey[(currentAlphabet.IndexOf(char.ToLower(tempChar)) + tempShift) % transKey.Count]));  else sb.Append(tempChar);  }  outputField.text = sb.ToString();  }  else outputField.text = "";  }  else outputField.text = textToEncript;  }  else outputField.text = "Введите ключ";  }  public void Decript()  {  int shift = shiftField.text.Length == 0 ? 0 : int.Parse(shiftField.text);  string key = keyField.text;  string textToEncript = inputField.text;  if (key != null && key.Length != 0)  {  List<char> currentAlphabet;  if (!ValidateKey(key, out currentAlphabet))  outputField.text = "Неправильный ключ";  else if (shift != 0)  {  if (textToEncript.Length > 0)  {  CharEnumerator inCE = textToEncript.GetEnumerator();  StringBuilder sb = new StringBuilder();  List<char> transKey = new List<char>();  foreach (char letter in key)  if (!transKey.Contains(char.ToLower(letter)) && letter != ' ')  transKey.Add(char.ToLower(letter));  foreach (char letter in currentAlphabet)  if (!transKey.Contains(letter))  transKey.Add(letter);  int tempShift = shift;  while (tempShift < 0)  tempShift += transKey.Count;  tempShift = tempShift % transKey.Count;  char[] reverseTransKey = new char[transKey.Count];  for (int i = 0; i < transKey.Count; i++)  {  reverseTransKey[currentAlphabet.IndexOf(transKey[i])] = currentAlphabet[(i + tempShift) % transKey.Count];  }  transKey = new List<char>(reverseTransKey);  char tempChar;  while (inCE.MoveNext())  {  tempChar = inCE.Current;  if (currentAlphabet.Contains(char.ToLower(tempChar)))  sb.Append(char.IsLower(tempChar) ? transKey[currentAlphabet.IndexOf(char.ToLower(tempChar))] : char.ToUpper(transKey[currentAlphabet.IndexOf(char.ToLower(tempChar))]));  else sb.Append(tempChar);  }  outputField.text = sb.ToString();  }  else outputField.text = "";  }  else outputField.text = textToEncript;  }  else outputField.text = "Введите ключ";  }  private bool ValidateKey(string key, out List<char> alphabet)  {  string loverKey = string.Join("", key.ToLower().Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries));  List<char> russian = new List<char>(Alphabets.russian);  List<char> english = new List<char>(Alphabets.english);  List<char> currentAlphabet;  if (russian.Contains(loverKey[0]))  {  currentAlphabet = russian;  alphabet = russian;  }  else if (english.Contains(loverKey[0]))  {  currentAlphabet = english;  alphabet = english;  }  else  {  alphabet = null;  return false;  }  for (int i = 0; i < loverKey.Length; i++)  {  if (!currentAlphabet.Contains(loverKey[i]))  return false;  }  return true;  }  } |

**Задание №11**

**«Аффинный шифр»**

**Описание шифра**

В аффинном шифре каждой букве алфавита размера ставится в соответствие число из диапазона . Затем, при помощи модульной арифметики, для каждой буквы вычисляется номер шифрующей её буквы по формуле:

, где

– размер алфавита, а пара и – ключ шифра. Значение должно быть выбрано таким, что и – взаимно простые числа.

Расшифровка происходит по формуле

, где

– обратное к число по модулю .

**Реализация шифра**

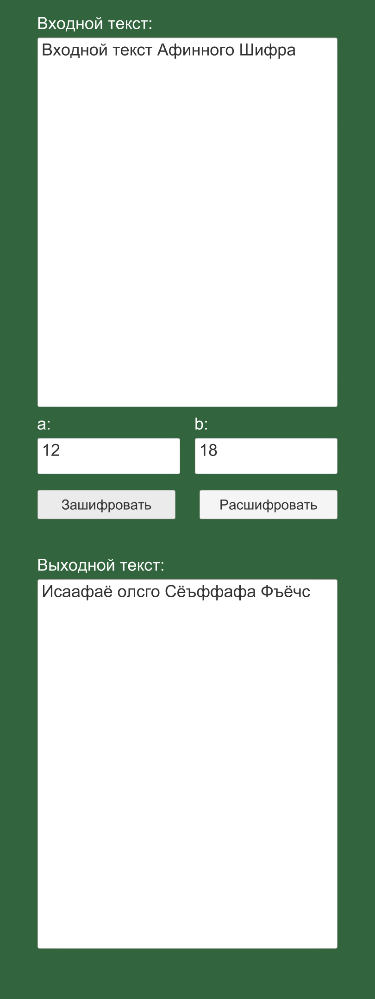


Рисунок 12. Реализация Аффинного шифра

Реализация Аффинного шифра показана на рисунке 12.

*Код программы:*

|  |
| --- |
| using System;  using System.Text;  using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using TMPro;  class AffineCypher : MonoBehaviour  {  [SerializeField] TMP\_InputField inputField;  [SerializeField] TMP\_InputField keyAField;  [SerializeField] TMP\_InputField keyBField;  [SerializeField] TMP\_InputField outputField;  public void Encript()  {  int keyA = keyAField.text.Length == 0 ? 0 : int.Parse(keyAField.text);  int keyB = keyBField.text.Length == 0 ? 0 : int.Parse(keyBField.text);  string textToEncript = inputField.text;  List<char> currentAlphabet;  if (ValidateKey(keyA, ref textToEncript, out currentAlphabet))  {  CharEnumerator inCE = textToEncript.GetEnumerator();  StringBuilder sb = new StringBuilder();  char tempChar;  int alphabetLenth = currentAlphabet.Count;  while (inCE.MoveNext())  {  tempChar = inCE.Current;  if (currentAlphabet.Contains(char.ToLower(tempChar)))  {  sb.Append(char.IsLower(tempChar) ? currentAlphabet[(keyA \* currentAlphabet.IndexOf(char.ToLower(tempChar)) + keyB) % alphabetLenth] : char.ToUpper(currentAlphabet[(keyA \* currentAlphabet.IndexOf(char.ToLower(tempChar)) + keyB) % alphabetLenth]));  }  else sb.Append(tempChar);  }  outputField.text = sb.ToString();  }  else  {  if (currentAlphabet == null)  outputField.text = "Ошибка считывания текста.";  else outputField.text = "Неправильный ключ. Переменная 'a' не должна нацело делиться на " + currentAlphabet.Count;  }  }  public void Decript()  {  int keyA = keyAField.text.Length == 0 ? 0 : int.Parse(keyAField.text);  int keyB = keyBField.text.Length == 0 ? 0 : int.Parse(keyBField.text);  string textToEncript = inputField.text;  List<char> currentAlphabet;  if (ValidateKey(keyA, ref textToEncript, out currentAlphabet))  {  CharEnumerator inCE = textToEncript.GetEnumerator();  StringBuilder sb = new StringBuilder();  char tempChar;  int alphabetLenth = currentAlphabet.Count;  int aMinus = CalculateAMinus(keyA, alphabetLenth);  while (inCE.MoveNext())  {  tempChar = inCE.Current;  if (currentAlphabet.Contains(char.ToLower(tempChar)))  {  sb.Append(char.IsLower(tempChar) ? currentAlphabet[(aMinus \* (currentAlphabet.IndexOf(char.ToLower(tempChar)) + alphabetLenth - keyB)) % alphabetLenth] : char.ToUpper(currentAlphabet[(aMinus \* (currentAlphabet.IndexOf(char.ToLower(tempChar)) + alphabetLenth - keyB)) % alphabetLenth]));  }  else sb.Append(tempChar);  }  outputField.text = sb.ToString();  }  else  {  if (currentAlphabet == null)  outputField.text = "Ошибка считывания текста.";  else outputField.text = "Неправильный ключ. Переменная 'a' не должна нацело делиться на " + currentAlphabet.Count;  };  }  private bool ValidateKey(int keyA, ref string text, out List<char> alphabet)  {  List<char> russian = new List<char>(Alphabets.russian);  List<char> english = new List<char>(Alphabets.english);  List<char> currentAlphabet;  if (russian.Contains(char.ToLower(text[0])))  {  currentAlphabet = russian;  alphabet = russian;  }  else if (english.Contains(char.ToLower(text[0])))  {  currentAlphabet = english;  alphabet = english;  }  else  {  alphabet = null;  return false;  }  if (keyA != 1 && (keyA < 1 || currentAlphabet.Count % keyA == 0))  return false;  return true;  }  private int CalculateAMinus(int a, int m)  {  int aMinus = 1;  while (((a \* aMinus) % m) != 1)  aMinus++;  return aMinus;  }  } |

**Задание №12**

**«Шифр Виженера»**

**Описание шифра**

Шифр Виженера состоит из последовательности нескольких шифров Цезаря с различными значениями сдвига. Для зашифровывания может использоваться таблица алфавитов, называемая квадрат Виженера. На каждом этапе шифрования используются различные алфавиты, выбираемые в зависимости от символа ключевого слова. Например, предположим, что исходный текст имеет такой вид:

|  |
| --- |
| ATTACKATDAWN |

Человек, посылающий сообщение, записывает ключевое слово («LEMON») циклически до тех пор, пока его длина не будет соответствовать длине исходного текста:

|  |
| --- |
| LEMONLEMONLE |

Первый символ исходного текста ("A") зашифрован последовательностью L, которая является первым символом ключа. Первый символ зашифрованного текста ("L") находится на пересечении строки L и столбца A в таблице Виженера. Точно так же для второго символа исходного текста используется второй символ ключа; то есть второй символ зашифрованного текста ("X") получается на пересечении строки E и столбца T. Остальная часть исходного текста шифруется подобным способом.

|  |
| --- |
| Исходный текст: ATTACKATDAWN  Ключ: LEMONLEMONLE  Зашифрованный текст: LXFOPVEFRNHR |

**Реализация шифра**

Реализация шифра Виженера с ключом показана на рисунке 13.

*Код программы:*

|  |
| --- |
| using System;  using System.Text;  using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using TMPro;  public class VigenereCypher : MonoBehaviour  {  [SerializeField] TMP\_InputField inputField;  [SerializeField] TMP\_InputField keyField;  [SerializeField] TMP\_InputField outputField;  public void Encript()  {  if (keyField.text.Length != 0)  {  List<int> key = ValidateKey(keyField.text);  if (key != null)  DoCesar(key, true);  else outputField.text = "Неправильный ключ";  }  else outputField.text = inputField.text;  }  public void Decript()  {  if (keyField.text.Length != 0)  {  List<int> key = ValidateKey(keyField.text);  if (key != null)  DoCesar(key, false);  else outputField.text = "Неправильный ключ";  }  else outputField.text = inputField.text;  }  private void DoCesar(List<int> key, bool encript)  {  string textToEncript = inputField.text;  if (textToEncript.Length > 0)  {  CharEnumerator inCE = textToEncript.GetEnumerator();  StringBuilder sb = new StringBuilder();  List<List<char>> listOfAlphabets = new List<List<char>>();  listOfAlphabets.Add(new List<char>(Alphabets.russian));  listOfAlphabets.Add(new List<char>(Alphabets.russianBig));  listOfAlphabets.Add(new List<char>(Alphabets.english));  listOfAlphabets.Add(new List<char>(Alphabets.englishBig));  int keyLength = key.Count;  int currentKey = 0;  char tempChar;  while (inCE.MoveNext())  {  tempChar = inCE.Current;  int tempShift;  bool findList = false;  foreach (List<char> alphabet in listOfAlphabets)  {  if (alphabet.Contains(tempChar))  {  tempShift = encript ? key[currentKey] : -key[currentKey];  while (tempShift < 0)  tempShift += alphabet.Count;  tempShift = tempShift % alphabet.Count;  sb.Append(alphabet[(alphabet.IndexOf(tempChar) + tempShift) % alphabet.Count]);  findList = true;  if (++currentKey >= keyLength)  currentKey = 0;  break;  }  }  if (!findList)  sb.Append(tempChar);  }  outputField.text = sb.ToString();  }  else outputField.text = "";  }  private List<int> ValidateKey(string key)  {  key = string.Join("", key.Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries));  List<int> intKey = new List<int>();  List<List<char>> listOfAlphabets = new List<List<char>>();  listOfAlphabets.Add(new List<char>(Alphabets.russian));  listOfAlphabets.Add(new List<char>(Alphabets.russianBig));  listOfAlphabets.Add(new List<char>(Alphabets.english));  listOfAlphabets.Add(new List<char>(Alphabets.englishBig));  for (int i = 0; i < key.Length; i++)  {  bool find = false;  foreach(List<char> alphabet in listOfAlphabets)  {  if (alphabet.Contains(key[i]))  {  intKey.Add(alphabet.IndexOf(key[i]));  find = true;  break;  }  }  if (!find)  return null;  }  return intKey;  }  } |

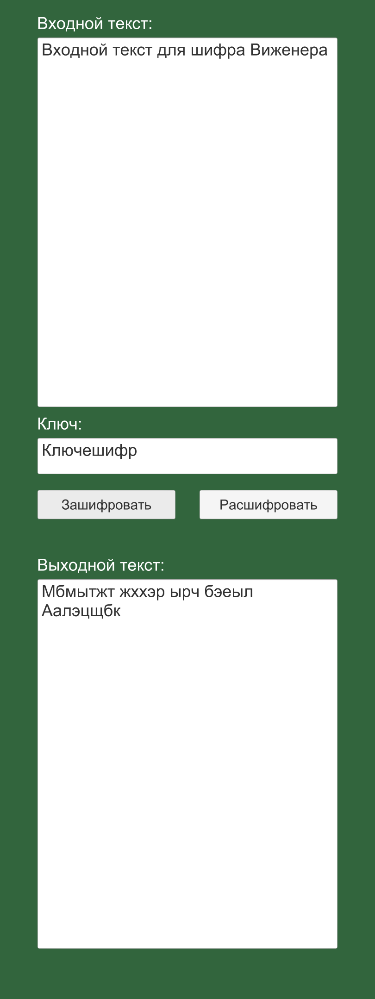


Рисунок 13. Реализация шифра Виженера

**Задание №13**

**«Шифр Гронсфельда»**

**Описание шифра**

Шифр Гронсфельда очень схож с шифром Виженера, однако ключом выступает не слово, а набор цифр.

**Реализация шифра**

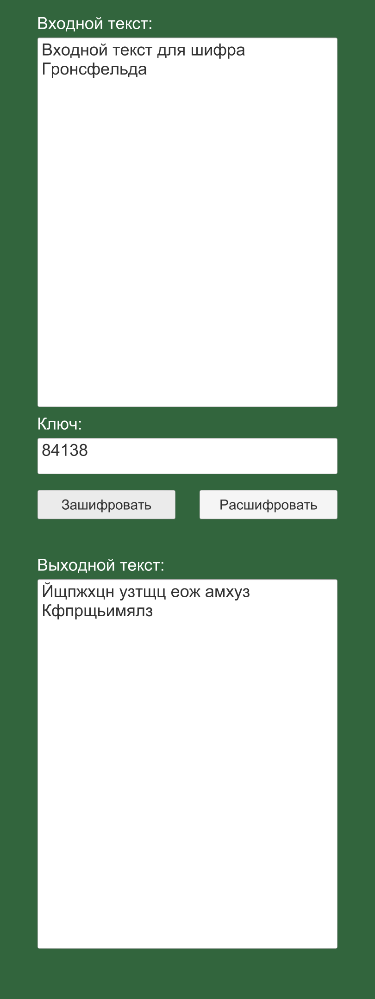


Рисунок 14. Реализация шифра Гронсфельда

Реализация шифра Гронсфельда показана на рисунке 14.

*Код программы:*

|  |
| --- |
| using System;  using System.Text;  using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using TMPro;  public class GronsfeldCypher : MonoBehaviour  {  [SerializeField] TMP\_InputField inputField;  [SerializeField] TMP\_InputField keyField;  [SerializeField] TMP\_InputField outputField;  public void Encript()  {  if (keyField.text.Length != 0)  {  List<int> key = ValidateKey(keyField.text);  if (key != null)  DoCesar(key, true);  }  else outputField.text = inputField.text;  }  public void Decript()  {  if (keyField.text.Length != 0)  {  List<int> key = ValidateKey(keyField.text);  if (key != null) DoCesar(key, false);  }  else outputField.text = inputField.text;  }  private void DoCesar(List<int> key, bool encript)  {  string textToEncript = inputField.text;  if (textToEncript.Length > 0)  {  CharEnumerator inCE = textToEncript.GetEnumerator();  StringBuilder sb = new StringBuilder();  List<List<char>> listOfAlphabets = new List<List<char>>();  listOfAlphabets.Add(new List<char>(Alphabets.russian));  listOfAlphabets.Add(new List<char>(Alphabets.russianBig));  listOfAlphabets.Add(new List<char>(Alphabets.english));  listOfAlphabets.Add(new List<char>(Alphabets.englishBig));  int keyLength = key.Count;  int currentKey = 0;  char tempChar;  while (inCE.MoveNext())  {  tempChar = inCE.Current;  int tempShift;  bool findList = false;  foreach (List<char> alphabet in listOfAlphabets)  {  if (alphabet.Contains(tempChar))  {  tempShift = encript ? key[currentKey] : -key[currentKey];  while (tempShift < 0) tempShift += alphabet.Count;  tempShift = tempShift % alphabet.Count;  sb.Append(alphabet[(alphabet.IndexOf(tempChar) + tempShift) % alphabet.Count]);  findList = true;  if (++currentKey >= keyLength) currentKey = 0;  break;  }  }  if (!findList)  sb.Append(tempChar);  }  outputField.text = sb.ToString();  }  else outputField.text = "";  }  private List<int> ValidateKey(string key)  {  List<int> intKey = new List<int>();  for (int i = 0; i < key.Length; i++)  {  if (key[i] >= '0' && key[i] <= '9')  intKey.Add(key[i] - '0');  else return null;  }  return intKey;  }  } |

**Задание №14**

**«Шифр Плейфера»**

**Описание шифра**

Шифр Плейфера использует матрицу 5х5 для латинского алфавита, и матрицу 4х8 для кириллического, содержащую ключевое слово или фразу. Чтобы составить ключевую матрицу, в первую очередь нужно заполнить пустые ячейки матрицы буквами ключевого слова без повторения символов, потом заполнить оставшиеся ячейки матрицы символами алфавита, не встречающимися в ключевом слове. Ключевое слово может быть записано в верхней строке матрицы слева направо, либо по спирали из левого верхнего угла к центру. Ключевое слово, дополненное алфавитом, составляет матрицу и является ключом шифра.

Для того чтобы зашифровать сообщение, необходимо разбить его на биграммы (группы из двух символов), например «Hello World» становится «HE LL OW OR LD», и отыскать эти биграммы в таблице. Два символа биграммы соответствуют углам прямоугольника в ключевой матрице. Определяем положения углов этого прямоугольника относительно друг друга. Затем, руководствуясь следующими 4 правилами, зашифровываем пары символов исходного текста:

1. Если два символа биграммы совпадают (или если остался один символ), добавляем после первого символа «Х», зашифровываем новую пару символов и продолжаем. В некоторых вариантах шифра Плейфера вместо «Х» используется «Q».
2. Если символы биграммы исходного текста встречаются в одной строке, то эти символы замещаются на символы, расположенные в ближайших столбцах справа от соответствующих символов. Если символ является последним в строке, то он заменяется на первый символ этой же строки.
3. Если символы биграммы исходного текста встречаются в одном столбце, то они преобразуются в символы того же столбца, находящиеся непосредственно под ними. Если символ является нижним в столбце, то он заменяется на первый символ этого же столбца.
4. Если символы биграммы исходного текста находятся в разных столбцах и разных строках, то они заменяются на символы, находящиеся в тех же строках, но соответствующие другим углам прямоугольника.

Для расшифровки необходимо использовать инверсию этих четырёх правил, откидывая символы «Х» (или «Q»), если они не несут смысла в исходном сообщении.

**Реализация шифра**

Реализация шифра Плейфера показана на рисунке 15.

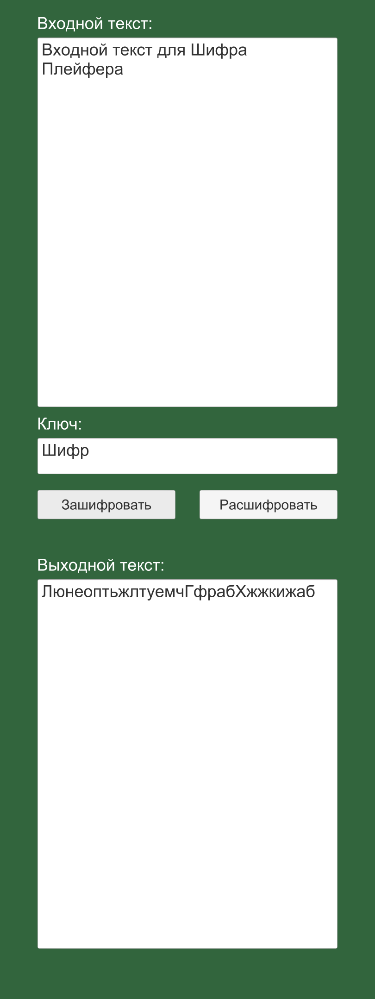


Рисунок 15. Реализация шифра Плейфера

*Код программы:*

|  |
| --- |
| using System;  using System.Text;  using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using TMPro;  class PlayfairCypher : MonoBehaviour  {  [SerializeField] TMP\_InputField inputField;  [SerializeField] TMP\_InputField keyField;  [SerializeField] TMP\_InputField outputField;  public void Encript()  {  DoPlayfair(true);  }  public void Decript()  {  DoPlayfair(false);  }  private void DoPlayfair(bool adding)  {  string key = keyField.text;  string textToEncript = string.Join("", inputField.text.Replace('j', 'i').Replace('J', 'I').Replace('ё', 'е').Replace('Ё', 'Е').Split(new char[] { ' ' }));  if (key != null && key.Length != 0)  {  key = key.Replace('j', 'i').Replace('J', 'I').Replace('ё', 'е').Replace('Ё', 'Е');  List<char> currentAlphabet;  if (!ValidateKey(key, out currentAlphabet))  outputField.text = "Неправильный ключ";  else if (textToEncript.Length > 0)  {  CharEnumerator inCE = textToEncript.GetEnumerator();  StringBuilder sb = new StringBuilder();  List<char> transKey = new List<char>();  foreach (char letter in key)  if (!transKey.Contains(char.ToLower(letter)) && letter != ' ')  transKey.Add(char.ToLower(letter));  foreach (char letter in currentAlphabet)  if (!transKey.Contains(letter))  transKey.Add(letter);  char tempChar;  int symbolsInRow;  int rowsInmatrix;  if (transKey.Count == 25)  {  symbolsInRow = 5;  rowsInmatrix = 5;  }  else  {  symbolsInRow = 8;  rowsInmatrix = 4;  }  char? firstChar = null;  bool moved = inCE.MoveNext();  while (moved || firstChar.HasValue)  {  if (moved && !transKey.Contains(char.ToLower(inCE.Current)))  {  outputField.text = "Неправильный входной текст";  return;  }  if (!firstChar.HasValue)  firstChar = moved ? inCE.Current : transKey.Count == 25 ? 'x' : 'х';  else  {  tempChar = moved ? inCE.Current : transKey.Count == 25 ? 'x' : 'х';  char[] bigram = new char[2];  if (char.ToLower(tempChar) == char.ToLower(firstChar.Value))  {  bigram[0] = tempChar;  bigram[1] = transKey.Count == 25 ? 'x' : 'х';  }  else  {  bigram[0] = firstChar.Value;  bigram[1] = tempChar;  firstChar = null;  }  int X0 = transKey.IndexOf(char.ToLower(bigram[0]));  int Y0 = X0 / symbolsInRow;  X0 = X0 % symbolsInRow;  int X1 = transKey.IndexOf(char.ToLower(bigram[1]));  int Y1 = X1 / symbolsInRow;  X1 = X1 % symbolsInRow;  if (Y0 == Y1)  {  X0 = adding ? ++X0 % symbolsInRow : (--X0 + symbolsInRow) % symbolsInRow;  X1 = adding ? ++X1 % symbolsInRow : (--X1 + symbolsInRow) % symbolsInRow;  }  else if (X0 == X1)  {  Y0 = adding ? ++Y0 % rowsInmatrix : (--Y0 + symbolsInRow) % rowsInmatrix;  Y1 = adding ? ++Y1 % rowsInmatrix : (--Y1 + symbolsInRow) % rowsInmatrix;  }  else  {  int tempX = X0;  X0 = X1;  X1 = tempX;  }  sb.Append(char.IsLower(bigram[0]) ? transKey[Y0 \* symbolsInRow + X0] : char.ToUpper(transKey[Y0 \* symbolsInRow + X0]));  sb.Append(char.IsLower(bigram[1]) ? transKey[Y1 \* symbolsInRow + X1] : char.ToUpper(transKey[Y1 \* symbolsInRow + X1]));  }  moved = inCE.MoveNext();  }  outputField.text = sb.ToString();  }  else outputField.text = "";  }  else outputField.text = "Введите ключ";  }  private bool ValidateKey(string key, out List<char> alphabet)  {  string loverKey = string.Join("", key.ToLower().Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries));  List<char> russian = new List<char>(Alphabets.russianPlayfair);  List<char> english = new List<char>(Alphabets.englishPlayfair);  List<char> currentAlphabet;  if (russian.Contains(loverKey[0]))  {  currentAlphabet = russian;  alphabet = russian;  }  else if (english.Contains(loverKey[0]))  {  currentAlphabet = english;  alphabet = english;  }  else  {  alphabet = null;  return false;  }  for (int i = 0; i < loverKey.Length; i++)  {  if (!currentAlphabet.Contains(loverKey[i]))  return false;  }  return true;  }  } |

**Задание №15**

**«Шифр Трисемуса»**

**Описание шифра**

Для шифрования строится таблица, в которую сначала вписывают буквы ключевого слова, а после – оставшиеся буквы алфавита. На следующем рисунке изображена таблица с ключевым словом «дядина».



Каждая буква открытого сообщения заменяется буквой, расположенной под ней в том же столбце. Если буква находится в последней строке таблицы, то для ее шифрования берут самую верхнюю букву столбца. Например, исходное сообщение «абрамов», зашифрованное – «жзцжуфй».

**Реализация шифра**

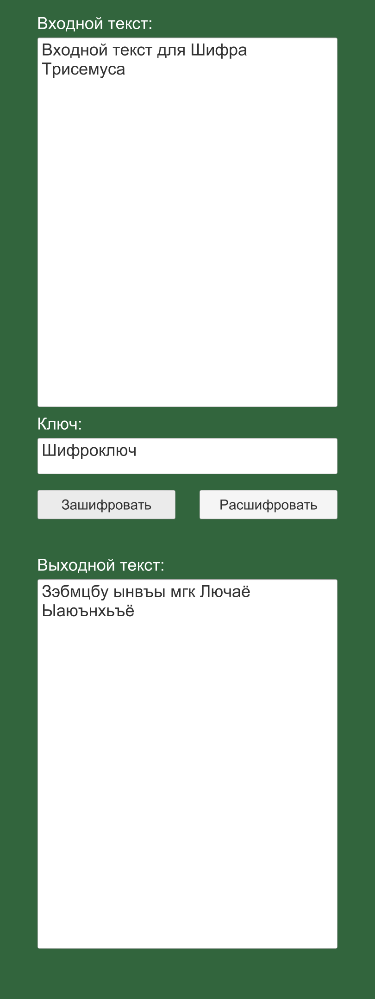


Рисунок 16. Реализация шифра Трисемуса

Реализация шифра Трисемуса показана на рисунке 16.

*Код программы:*

|  |
| --- |
| using System;  using System.Text;  using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using TMPro;  class TrycemusCypher : MonoBehaviour  {  [SerializeField] TMP\_InputField inputField;  [SerializeField] TMP\_InputField keyField;  [SerializeField] TMP\_InputField outputField;  private const int rowSize = 6;  public void Encript()  {  string key = keyField.text;  string textToEncript = inputField.text;  if (key != null && key.Length != 0)  {  List<char> currentAlphabet;  if (!ValidateKey(key, out currentAlphabet))  outputField.text = "Неправильный ключ";  else if (textToEncript.Length > 0)  {  CharEnumerator inCE = textToEncript.GetEnumerator();  StringBuilder sb = new StringBuilder();  List<char> transKey = new List<char>();  foreach (char letter in key)  if (!transKey.Contains(char.ToLower(letter)) && letter != ' ')  transKey.Add(char.ToLower(letter));  foreach (char letter in currentAlphabet)  if (!transKey.Contains(letter))  transKey.Add(letter);  char tempChar;  while (inCE.MoveNext())  {  tempChar = inCE.Current;  if (transKey.Contains(char.ToLower(tempChar)))  sb.Append(char.IsLower(tempChar) ? transKey[(transKey.IndexOf(char.ToLower(tempChar)) + rowSize) % transKey.Count] : char.ToUpper(transKey[(transKey.IndexOf(char.ToLower(tempChar)) + rowSize) % transKey.Count]));  else sb.Append(tempChar);  }  outputField.text = sb.ToString();  }  else outputField.text = "";  }  else outputField.text = "Введите ключ";  }  public void Decript()  {  string key = keyField.text;  string textToEncript = inputField.text;  if (key != null && key.Length != 0)  {  List<char> currentAlphabet;  if (!ValidateKey(key, out currentAlphabet))  outputField.text = "Неправильный ключ";  else if (textToEncript.Length > 0)  {  CharEnumerator inCE = textToEncript.GetEnumerator();  StringBuilder sb = new StringBuilder();  List<char> transKey = new List<char>();  foreach (char letter in key)  if (!transKey.Contains(char.ToLower(letter)) && letter != ' ')  transKey.Add(char.ToLower(letter));  foreach (char letter in currentAlphabet)  if (!transKey.Contains(letter))  transKey.Add(letter);  char tempChar;  while (inCE.MoveNext())  {  tempChar = inCE.Current;  if (transKey.Contains(char.ToLower(tempChar)))  sb.Append(char.IsLower(tempChar) ? transKey[(transKey.IndexOf(char.ToLower(tempChar)) - rowSize + transKey.Count) % transKey.Count] : char.ToUpper(transKey[(transKey.IndexOf(char.ToLower(tempChar)) - rowSize + transKey.Count) % transKey.Count]));  else sb.Append(tempChar);  }  outputField.text = sb.ToString();  }  else outputField.text = "";  }  else outputField.text = "Введите ключ";  }  private bool ValidateKey(string key, out List<char> alphabet)  {  string loverKey = string.Join("", key.ToLower().Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries));  List<char> russian = new List<char>(Alphabets.russian);  List<char> english = new List<char>(Alphabets.english);  List<char> currentAlphabet;  if (russian.Contains(loverKey[0]))  {  currentAlphabet = russian;  alphabet = russian;  }  else if (english.Contains(loverKey[0]))  {  currentAlphabet = english;  alphabet = english;  }  else  {  alphabet = null;  return false;  }  for (int i = 0; i < loverKey.Length; i++)  {  if (!currentAlphabet.Contains(loverKey[i]))  return false;  }  return true;  }  } |

**Задание №16**

**«Шифр Транслита Виженера»**

**Описание шифра**

Данный шифр основан на шифре Виженера и также является полиалфавитным шифром замены. Сначала, русские буквы переводятся в английский транслитерацией и использованием незадействованных английских букв для похожих русских букв.

Набор (*"а", "б", "в", "г", "д", "е", "ё", "ж", "з", "и", "й", "к", "л", "м", "н", "о", "п", "р", "с", "т", "у", "ф", "х", "ц", "ч", "ш", "щ", "ъ", "ы", "ь", "э", "ю", "я"*) заменяется на набор (*"a", "b", "v", "g", "d", "ye", "yo", "zh", "z", "i", "y", "k", "l", "m", "n", "o", "p", "r", "s", "t", "u", "f", "h", "c", "ch", "sh", "w", "j", "ji", "q", "e", "yu", "ya"*) соответственно.

Затем, происходит замена английских букв шифром Виженера с помощью заданного ключа. После чего, английские буквы заменяются обратно русскими. Финальный текст не только оказывается устойчивым к частотному анализу, отличающееся количество букв в зашифрованном тексте усиливает устойчивость шифра ко взлому.

**Реализация шифра**

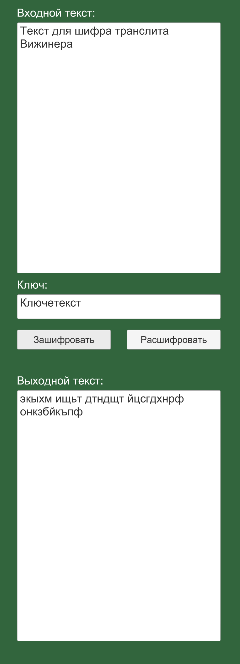


Рисунок 17. Реализация шифра Транслита Виженера

Реализация шифра Транслита Виженера показана на рисунке 17.

*Код программы:*

|  |
| --- |
| using System;  using System.Text;  using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using TMPro;  public class VigenereTranslitCypher : MonoBehaviour  {  [SerializeField] TMP\_InputField inputField;  [SerializeField] TMP\_InputField keyField;  [SerializeField] TMP\_InputField outputField;  public void Encript()  {  if (keyField.text.Length != 0)  {  List<int> key = ValidateKey(keyField.text.ToLower());  if (key != null)  DoCesar(key, true);  else outputField.text = "Неправильный ключ";  }  else outputField.text = inputField.text;  }  public void Decript()  {  if (keyField.text.Length != 0)  {  List<int> key = ValidateKey(keyField.text.ToLower());  if (key != null)  DoCesar(key, false);  else outputField.text = "Неправильный ключ";  }  else outputField.text = inputField.text;  }  private void DoCesar(List<int> key, bool encript)  {  string textToEncript = inputField.text.ToLower();  if (textToEncript.Length > 0)  {  StringBuilder sbTranslit = new StringBuilder();  StringBuilder newString = new StringBuilder();  List<char> english = new List<char>(Alphabets.english);  List<string> translitKeys = new List<string>(new string[] { "а", "б", "в", "г", "д", "е", "ё", "ж", "з", "и", "й", "к", "л", "м", "н", "о", "п", "р", "с", "т", "у", "ф", "х", "ц", "ч", "ш", "щ", "ъ", "ы", "ь", "э", "ю", "я" });  List<string> translitValues = new List<string>(new string[] { "a", "b", "v", "g", "d", "ye", "yo", "zh", "z", "i", "y", "k", "l", "m", "n", "o", "p", "r", "s", "t", "u", "f", "h", "c", "ch", "sh", "w", "j", "ji", "q", "e", "yu", "ya" });  int keyLength = key.Count;  for (int i = 0; i < keyLength; i++)  Debug.Log("key[" + i + "] = " + key[i]);  int currentKey = 0;  char tempChar;  int textLength = textToEncript.Length;  for (int i = 0; i < textLength; i++)  {  string nextChar = textToEncript[i].ToString();  if (translitKeys.Contains(nextChar))  newString.Append(translitValues[translitKeys.IndexOf(nextChar)]);  else newString.Append(nextChar);  }  string translitString = newString.ToString();  Debug.Log(translitString);  CharEnumerator inCE = translitString.GetEnumerator();  newString.Clear();  while (inCE.MoveNext())  {  tempChar = inCE.Current;  int tempShift;  if (english.Contains(tempChar))  {  tempShift = encript ? key[currentKey] : -key[currentKey];  while (tempShift < 0)  tempShift += english.Count;  tempShift = tempShift % english.Count;  sbTranslit.Append(english[(english.IndexOf(tempChar) + tempShift) % english.Count]);  if (++currentKey >= keyLength)  currentKey = 0;  }  else sbTranslit.Append(tempChar);  }  string sbTranslitString = sbTranslit.ToString();  Debug.Log(sbTranslitString);  for (int i = 0; i < sbTranslitString.Length; i++)  {  if (i < sbTranslitString.Length - 1)  {  string twoLetters = string.Concat(sbTranslitString[i], sbTranslitString[i + 1]);  if (translitValues.Contains(twoLetters))  {  newString.Append(translitKeys[translitValues.IndexOf(twoLetters)]);  i++;  }  else  {  string nextChar = sbTranslitString[i].ToString();  if (translitValues.Contains(nextChar))  {  newString.Append(translitKeys[translitValues.IndexOf(nextChar)]);  }  else newString.Append(nextChar);  }  }  else  {  string nextChar = sbTranslitString[i].ToString();  if (translitValues.Contains(nextChar))  {  newString.Append(translitKeys[translitValues.IndexOf(nextChar)]);  }  else newString.Append(nextChar);  }  }  Debug.Log(newString.ToString());  outputField.text = newString.ToString();  }  else outputField.text = "";  }  private List<int> ValidateKey(string key)  {  key = string.Join("", key.Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries));  List<int> intKey = new List<int>();  List<char> alphabet = new List<char>(Alphabets.russian);  for (int i = 0; i < key.Length; i++)  {  bool find = false;  if (alphabet.Contains(key[i]))  {  intKey.Add(alphabet.IndexOf(key[i]));  find = true;  }  if (!find)  return null;  }  return intKey;  }  } |