**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ**

**ФАКУЛЬТЕТ АЕРОНАВІГАЦІЇ, ЕЛЕКТРОНІКИ ТА ТЕЛЕКОМУНІКАЦІЙ**

**Кафедра авіоніки**

**ДОПУСТИТИ ДО ЗАХИСТУ**

Завідувач кафедри

_____С.В. Павлова

“_____”_____2020 р.

# ДИПЛОМНА РОБОТА

**(ПОЯСНЮВАЛЬНА ЗАПИСКА)**

**ВИПУСКНИКА ОСВІТНЬО-КВАЛІФІКАЦІЙНОГО РІВНЯ “БАКАЛАВР”**

**Тема: "Андроїд програма - "Pocket avionics" для електронного планшету персоналу авіакомпанії”**

**Виконав:** _____Козаков М.О.

**Керівник:** _____Белінський В.Н.

**Нормоконтролер:** _____Левківський В.В.

КИЇВ 2020

**MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE**

**NATIONAL AVIATION UNIVERSITY**

**FACULTY OF AIR NAVIGATION, ELECTRONICS AND TELECOMMUNICATION**

**Department of avionics**

**APROVED FOR DEFENCE**

Head of the Department

_____ S.V. Pavlova

“ ” _____2019

# GRADUATE WORK

## (EXPLANATORY NOTE)

**GRADUATE OF AN EDUCATIONAL DEGREE "BACHELOR"**

**Theme: Android program - "Pocket avionics" for an electronic tablet of airline staff**

**Done by:** _____M.A.Kozakov

**Supervisor:** _____V.N.Belinskyi

**Standart controller:** _____V.V.Levkivskyi

Kyiv 2020

**NATIONAL AVIATION UNIVERSITY**

**Faculty of Aeronavigation, Electronics and Telecommunications**

**Department of Avionics**

**Training Direction 6.051103 «Avionics»**

APPROVED

Head of the Department

_____S.V. Pavlova

“ ” _____2020

**TASK**

**For execution graduate work**

Student’s name:_____Kozakov M.A._____

1. Theme of bachelor work: «**Android program - "Pocket avionics" for an electronic tablet of airline staff** », approved by the Rector’s order

2. Background to the work: Android programs about avionics

3. Contents of the explanatory notes:

4. What is Pocket Avionics? Education is in trend. All kinds of educational applications today occupy the first lines in the App Store and are in demand by hundreds of thousands of users. Educational apps are services that help users of all ages and backgrounds to study certain academic disciplines.

5. Timetable

| № | Task | Duration | Evaluation of the performance |
|---|---|---|---|
| 1. | Selection of literature for performing of bachelor's graduation diploma work | 01.10.2020-14.12.2020 | |
| 2. | Introduction | 01.10.2020-14.12.2020 | |
| 3. | Occupational Safety and Health | 01.10.2020-14.12.2020 | |
| 4. | Environmental protection | 01.10.2020-14.12.2020 | |
| 5. | Information noise problem. Weak motivation and indifference | 01.10.2020-14.12.2020 | |
| 6. | Application concept | 01.10.2020-14.12.2020 | |
| 7. | Graphic component | 01.10.2020-14.12.2020 | |
| 8. | Technical component | 01.10.2020-14.12.2020 | |
| 9. | Conclusions | | |

8. Date of assignment: "  "_____2020

9. Supervisor _____V.N. Belinsky

The task took to perform _____M.A. Kozakov

# ABSTRACT

Explanatory note to graduation work «**Android program - "Pocket avionics" for an electronic tablet of airline staff**»: 60 pages, 11 pics

UNITY 3D, PICKET VIONICS, C#

Learn Aircraft Avionics is an educational application Pocket Avionics. If you are looking for basic Aircraft Avionics app so you are in a right place. This application will provide you most important & informative lessons. This basic Aircraft Avionics app will give you definition and classification.

Learn Aircraft Avionics app is essential for beginners. It will give you most common & useful chapters. This app will provide you example and explanation. So now you can carry your basic Aircraft Avionics book collection anywhere by Pocket avionics app and can learn anytime.

Pocket avionics is app for:

-Electrical Engineering

-Automobile Engineering

-Pilots studies

-Airport Studies

-Avionics

-Aircraft Instruments

-Aerodynamics

-Airplane Structures

-Mechanical Engineering

## CONTENT

# INTRODUCTION

Pocket Avionics defines Aviation terms in a way that is easy for anybody to understand.

This Pocket avionics app isn’t a simple that you find in the stationary some aircrafts & in your Pocket avionics app. This Pocket avionics app is written and explained in such a way that anyone can learn Aviation language within a short time of duration. Each Aviation terms are given with the audio voice ability so you can recognize the basic word behind the jargon.

Pocket avionics app have a basic idea of an Aviation terms that assists people learn new vocabulary and architectural engineering speedily that helps them to recall info for the long time. Once you install the Pocket avionics app, enter the word in search box that you are looking for and get the detail explanation with the usage of it.

Aviation is the activities surrounding mechanical flight and the aircraft industry. Aircraft includes fixed-wing and rotary-wing types, morphable wings, wing-less lifting bodies, as well as lighter-than-air craft such as hot air balloons and airships.

Aviation began in the 18th century with the development of the hot air balloon, an apparatus capable of atmospheric displacement through buoyancy. Then a large step in significance came with the construction of the first powered airplane by the Wright brothers in the early 1900s. Since that time, aviation has been technologically revolutionized by the introduction of the jet which permitted a major form of transport throughout the world.

## CHAPTER 1: THE PROBLEM OF INFORMATION NOISE

Information noise is a cultural phenomenon that arose in the 20th century, which describes the presence in the text of elements that complicate its understanding, distort the meaning of what is stated or completely prevent an adequate understanding of its content. The concept of information noise refers to the modern media space and the methods of disseminating information in conditions of information "overload" of society. Information noise (or information overload) occurs in conditions of an excess of information ("over-awareness"), which, as a result, adversely affects the ability of an individual to adequately analyze the situation and "filter" the received information due to the congestion caused by the abundance of information messages.

It is worth noting that this concept, contrary to popular belief, arose before the Internet and was partially described in literary works (in particular, in the book "Shock of the Future" by Alvin Toffler in 1970).

Subsequently, the term became widespread: Bill Gates in his book "The Road to the Future", published in 1996, noted that this phenomenon, namely "information overload is quite common." It follows from this that the concept under study should not be associated only with the online sphere: long before the Internet became a network used all over the world, the problem of information overload took place (in traditional media, for example).

This phenomenon was most substantively described in the book "My Life After Death" by Robert Wilson. In one of the chapters ("Information Overload"), the author, using the example of a frame from "Seal of Evil" by Orson Welles, shows how seriously such a method as editing frames and correctly selected musical sequence can become a way of propaganda and deliberately provoke a distorted perception of information. analysis of this phenomenon, it is worth paying attention to the classification given by the Moldavian scientist Arkady Dmitrievich Ursul. So, the professor distinguishes two groups of information noise:

Noise arising from an overabundance of information unimportant to the individual;

Noise is a consequence of an overabundance of important and relevant, but at the same time, repetitive information.

Thus, Ursul considers advertising messages, any kind of propaganda information, spam and contextual advertising to the first group of "noises". The second group, in turn, consists of, in fact, information-filled and necessary for the recipient messages, repeating each other, for example, note-taking.

However, information noise is usually divided into intentional (intentional) and unintentional. Intentional noise is an informational background, often equated with the process of disinformation. In other words, the creation of deliberate “noise” can be characterized as the creation of an artificial situation in which informants rely on the abundance and even an overabundance of information messages, which leads to the complexity of the recipients' perception of information, as well as blurring the boundaries between truth and fiction.

## CHAPTER 2: GENERAL CONCEPT OF THE POCKET AVIONICS APP

### 2.1 Cards and their variety in the Pocket Avionics APP

There will be 3 types of cards in the game: planes, parts, personalities. Airplane maps are displayed with an airplane symbol. Parts Cards with a Gear symbol. Personality cards with the "Personality" symbol. Examples of maps **(Pics 1.1)**

**Pics 2.1**

Each card has a quality level:

- Rare

- Mythical

- Legendary

The quality of a card determines its rarity and the number of free points that it gives in case of repeated drops. The quality of the card also determines the historicity of the aircraft / personality. The more individual the plane is in the world, the better it is in the application. Fragment of the map responsible for the rarity **(Pics 2.1)**

## 2.2 Player card collection in the Pocket Avionics APP

From the start, the user does not have any cards. The player receives cards for special tasks: passing a test, solving a puzzle, and so on. When a player receives a new card, it appears in the collection. **(Pics 2.2)**


**Pics 2.2**

By clicking on the received card, the card opens in a new window. Different types of cards have different descriptions. For example, if this map is an airplane, then the map has a description of the aircraft parameters. If this is done, then the methods of using this unit on the aircraft or a description of the part. **(Pics 2.3)**


**Pics 2.3**

### 2.3 Method of receiving cards in the Pocket Avionics APP

To receive a card, the user must fill in a special scale. The scale fills up as you get repeated cards or as you progress through the levels / test. The general idea is to encourage the player to take the test and memorize facts from the aviation field. For this, the player will receive a random reward - 5 random cards (Among which there may be repeated ones) **(Pics 2.4)**


**Pics 2.4**

# CHAPTER 3: TECHNICAL IMPLEMENTATION OF THE APPLICATION FUNCTIONAL: “POCKET AVIONICS”

## 3.1 Using Unit3D engine to write an application

Unity3d is a modern cross-platform engine for creating games and applications using Unity Technologies. With this engine, you can develop not only applications for computers, but also for mobile devices (for example, based on Android), game consoles and other devices.

Let's talk a little about the characteristics of the engine. First, it is worth noting that the engine is integrated in the Unity development environment, in other words, you can test the application without leaving the editor. Secondly, Unity supports a huge number of different formats, which allows you to develop games, construct the models themselves in a more convenient application, and use Unity for its intended purpose - product development. Thirdly, scripting (scripting) is performed in the most popular programming languages - C # and JavaScript.

Thus, Unity3d is a relevant platform with which you can create your own applications and export them to various devices, mobile phone or Nintendo Wii console. To create your application, you only need 1 programming language: C # / JavaScript.

Unity is more than an engine, it is an environment for developing computer games that combines various software tools used to create software - a text editor, compiler, debugger, and so on. At the same time, thanks to its ease of use, Unity makes the creation of games as simple and comfortable as possible, and the multiplatform nature of the engine allows game developers to cover as many gaming platforms and operating systems as possible.

First of all, as we have already mentioned, the Unity3D engine makes it possible to develop games without requiring any special knowledge. It uses a component-based approach, in which the developer creates objects (for example, the main character) and adds various components to them (for example, the visual display of the character and how to control it). Thanks to a convenient Drag & Drop interface and a functional graphic editor, the engine allows you to draw maps and place objects in real time and immediately test the result.

The second advantage of the engine is the presence of a huge library of assets and plugins, with which you can significantly speed up the game development process. They can be imported and exported, whole blanks can be added to the game - levels, enemies, AI behavior patterns, and so on. No programming hassle. Many assets are available for free, others are offered for a small price, and if you want, you can create your own content, publish it to the Unity Asset Store and profit from it.

The third strength of Unity 3D is support for a huge number of platforms, technologies, APIs. Games created on the engine can be easily ported between Windows, Linux, OS X, Android, iOS, on the PlayStation, Xbox, Nintendo, VR and AR devices. Unity supports DirectX and OpenGL, works with all modern rendering effects, including the latest real-time ray tracing technology.

### 3.2. Using the C # language to write an application

In June 2000, it became known about a new programming language that was born in the bowels of Microsoft. It became part of a new Microsoft technology called .NET (read "Dot Net"). This technology provides a single runtime environment for programs (Common Language Runtime, CLR) written in different programming languages. One of these languages, the main one in this environment, is C # (C #, reads "C sharp", "C sharp"). By the name of the language, of course, they wanted to emphasize its relationship with C ++, because # are two crossed pluses. But most of all, the new language is similar to Java. And there is no doubt that one of the reasons for its appearance was Microsoft's desire to meet the challenge of Sun.

Although the authors of C # are not officially named, Anders Hejlsberg, the creator of Turbo Pascal and Delphi, who moved to Microsoft in 1996, and Scott Wiltamuth are named on the title page of one of the preliminary editions of the language reference.

The unified program execution environment is based on the use of the intermediate language IL (Intermediate Language), which plays almost the same role as the bytecode of the Java virtual machine. Compilers used in the framework of .NET technology from various languages translate programs into IL-code. Like Java bytecode, IL code is the instructions for a hypothetical stacked computer. But there is also a difference in the design and use of IL.

First, unlike the JVM, IL is not tied to a single programming language. The preliminary versions of Microsoft.NET include compilers from C ++, C #, Visual Basic. Third-party developers can add other languages by building compilers from those languages to IL code.

Secondly, IL is not intended for program interpretation, but for subsequent compilation into machine code. This allows you to achieve significantly higher performance of programs. The files containing IL-code contain enough information for the optimizing compiler to work.

### 3.2 Basic use of C# features for writing a Pocket Avionics APP

“C # is a simple, modern, object-oriented language with a safe type system, derived from C and C ++. C # will be convenient and understandable for programmers who know C and C ++. C # combines the productivity of Visual Basic with the power of C ++. " These words begin the description of C #. We will consider the technical features of the language.

1. A compilation unit is a file (as in C, C ++, Java). The file can contain one or several type descriptions: classes (class), interfaces (interface), structures (struct), enumerations (enum), types-delegates (delegate) with indication (or without indication) of their distribution over namespaces.

2. Namespaces regulate the visibility of program objects (as in C ++). Namespaces can be nested. The use of program objects is allowed without explicitly specifying the namespace to which this object belongs. Just a general mention of the use of this namespace in the using directive (as in Turbo Pascal) is enough. There are aliases for namespace names in the using directive (as in Oberon).

3. Elementary data types: 8-bit (sbyte, byte), 16-bit (short, ushort), 32-bit (int, uint) and 64-bit (long, ulong) signed and unsigned integers, single real (float) and double (double) precision, Unicode characters (char), boolean type (bool, not compatible with integers), decimal type providing 28 significant digits precision (decimal).

4. Structured types: classes and interfaces (as in Java), one-dimensional and multidimensional (as opposed to Java) arrays, strings (strings), structures (almost the same as classes, but allocated not on the heap and without inheritance), enumerations incompatible with integers (as in Pascal).

5. Types-delegates or simply "delegates" (similar to procedural types in Module 2 and Oberon, pointers to functions in C and C ++).

6. Types are subdivided into reference (classes, interfaces, arrays, delegates) and value types (elementary types, enumerations, structures). Objects of reference types

are allocated in dynamic memory (heap), and variables of reference types are, in fact, pointers to these objects. In the case of value types, variables are not pointers, but the values themselves. Implicit type conversions are allowed only in cases where they do not violate the type safety system and do not lead to loss of information. All types, including elementary ones, are compatible with the object type, which is the base class of all other types. There is an implicit conversion of value types to the type object, called boxing, and an explicit inverse conversion, unboxing.

7. Automatic garbage collection (as in Oberon and Java).

8. Extensive set of operations with 14 priority levels. Overriding operations (as in Algol-68, Ada, C ++). The checked and unchecked operators control overflow control when performing integer operations.

9. Methods with parameter values, reference parameters (ref) and output parameters (out). The ref and out words must be written before the parameter, not only in the method description, but also when calling. The presence of output parameters allows you to control the execution of defining assignments. According to the rules of the language, any variable must be guaranteed to receive a value before attempting to use it.

10. Control statements: if, switch, while, do, for, break, continue (as in C, C ++ and Java). A foreach statement that loops through each item in a "collection", several flavors of the goto statement.

11. Handling of exceptions (as in Java).

12. Properties - elements of classes (objects), access to which is carried out in the same way as to fields (you can assign or get a value), but implemented by implicitly called get and set subroutines (as in Object Pascal - the input language of the Delphi system).

13. Indexers are elements of classes (objects) that allow accessing objects in the same way as arrays (by specifying the index in square brackets). Implemented by the implicitly called get and set routines. For example, read access to the characters in a

string can be performed as to elements of an array because an indexer is implemented for the standard string class.

14. Events are class members (fields or properties) of a procedural type (delegates), to which outside the class where they are defined, only the + = and - = operations are applicable, allowing you to add or remove event handler methods for objects of this class.

15. Unsafe (unsafe) code using pointers and address arithmetic is localized in the parts of the program marked with the unsafe modifier.

16. A preprocessor that provides, unlike C and C ++, only conditional compilation facilities.

Let's consider first the simplest complete program, the process of its compilation and execution.

```
class Hello {
static void Main() {
System.Console.WriteLine("Hello, World!");}}
```

To compile the program, you can use the csc compiler, which is included in the Microsoft .NET Framework SDK - a development kit for the Microsoft .NET environment and is run from the command line:

```
csc Hello.cs
```

After compilation, the executable file Hello.exe will be obtained. However, it can only be run on a computer running Windows if Microsoft .NET support is installed on that computer. The fact is that the file obtained after compilation (despite its name) does not contain ordinary machine instructions, but IL-code that will be converted into processor code when loading and running the program.

However, if the .NET Framework SDK is installed, it means that the corresponding support is available. By running Hello.exe, we get:

Hello.exe

Hello, World!

Now let's turn to the text of the program. It defines a single class Hello, which contains the description of the static method (class method) Main. A static method called Main (uppercase and lowercase letters are different in C #) is the entry point to a program written in C #. Execution of this method starts the work of the program. Unlike Java, the Main method in C # can be without parameters or with parameters, it does not matter whether it returns a value (being a function) or not. In our example, Main has no parameters and no return value (void).

The only statement in the Main method is a call to the static WriteLine method. It is a method of the Console class that provides access to the standard output and input streams. The Console class belongs to the (predefined) System.

Console is referred to as its full name System.Console (qualified identifier), including the System namespace. Using the using directive, you can abbreviate the notation by using non-namespace-qualified designations

There is only one difference. The word "length" is written with a capital letter: Length (in Java - length). Length is a property of the standard System.Array class, which is the ancestor of arrays in C #.

At the time of this writing, there is only a preliminary description of the C # language and a preliminary version of the development tools for programs in this language. Therefore, it is too early to draw any general conclusions. But some judgments can be made.

In C #, some traditional constructions are preserved and even put in order: enumerations, structures, multidimensional arrays. Java in this regard shows a more extremist approach: objects and nothing but objects. The obvious absurdities of Java, such as the lack of passing parameters by reference in the absence of pointers, have been eliminated. The mechanism for passing parameters in C # is well thought out, providing for passing both by value and by reference.

At the same time, there are many constructs in C # that, at the very least, can raise questions. The language is redundant and, as a result, difficult to use and implement. Some C # tools can cause errors. Doubts about C # become especially strong when it is compared with the languages created by N. Wirth, first of all with Oberon. In C #, as in other languages derived from C, the simple and clear concept of the module has not been embodied. Instead, namespaces are used - a tool that appeared in the late stages of C ++ standardization. Namespaces are a very general mechanism, absorbing, in particular, the capabilities provided by modules. But here there is an over-generalization that is not due to urgent needs, which provides the programmer with redundant tools, and with them opportunities for abuse. The nesting of namespaces, their long compound designations serve as an obstacle to requiring the explicit (qualified) use of names taken from these spaces, as is done in Oberon for identifiers imported by the module. Implicit imports allowed by the using directive are a source of name collision errors. Here's an example.

Consider a program in which the namespace Hello is defined, and within that namespace are nested classes A and B. Class B contains a single static

```
namespace Hello {
public class A {
public class B {
    public static string C = "HELLO"; }
```

The content of the Hello.cs file is not an independent program, but it can be a separate compilation unit, which can be translated into a dynamically linked library.

(file with dll extension). To do this, when starting the csc compiler, use the / target parameter:

csc / target: library Hello.cs

As a result of compilation, the Hello.dll library will be obtained.

Now let's write a main program that can use the resources of our library. And the resource, in fact, is one - a line containing "Hello!". We will print it:*// Эта*

Now in the statement Console.WriteLine (A.B.C); In the Print program, the identifier A is understood to denote the namespace A, not the class A of the namespace Hello! The C # language has failed us. Moreover, twice. The first time there was a collision between the name of the class A of the namespace Hello and the name of the namespace A. This collision was somehow resolved in favor of the namespace name, while the using Hello directive created within its scope a local scope in which local names should take precedence. Second, the resulting inconsistency was not found even though the two different fields named C were of different types. If the Write method were not so liberal with the type of its parameters, there was a chance to find an error.

It may seem that the demonstrated situation was created artificially, but in practice such coincidences are unlikely. But we can talk about programs in the hundreds of thousands and millions of lines, and then fatal coincidences are by no means excluded. Moreover, given the rules regarding namespaces that apply in C #, it turns out that the programmer must know the list of all namespaces available to his program, otherwise he risks distorting the work of already written and correctly working parts of his program. The danger arises even if your own program is small, but uses something

from libraries written by others. The situation with the names of C # namespaces is similar to the situation in the most primitive versions of BASIC, when all variables are global, and you need to remember them all so as not to get confused.

What is the cause of such gaps and how could they be eliminated? The fact is that in relation to the names of namespaces in C #, the general rule, recognized since the time of Algol-60, does not apply, according to which any identifier in a program cannot be used without a (preliminary) description. To avoid these collisions, the using directives must be required, along with the mandatory qualification of identifiers by the namespace name. That is, it would be necessary to require that the Print program can only be written in this form:

*using System;*

*using Hello;*

*class Print {*

*static void Main() {*

*System.Console.WriteLine(Hello.A.B.C);}}*

C # allows such writing, but, alas, does not require it. But in the Oberon language, the import is organized in this way, and the occurrence of the problems considered is excluded.

In C #, as in Java, neither field descriptions nor method descriptions can be placed outside the definitions of classes (as well as interfaces and structures). This is a rather strange rule, especially for a language like C #, where the boundaries of namespaces are enclosed in explicit brackets.

Static (with the static descriptor) fields and methods - class members - are objects that are completely different in their essence than non-static fields and methods - elements of class instances. Static fields and methods are ordinary

procedures and variables that have nothing to do with objects of the class within which they are defined. Just to mention them, you need to indicate the name of the class to which they belong. It turns out that the description of a class plays two different roles - it is a description of the type of objects and at the same time a container containing definitions of static fields and methods. In its second role, a class acts, in fact, as a module or имен namespace, which could replace classes in this capacity. There are no obstacles in sight to allowing the definitions of static objects to be removed from classes and immersed in namespaces that encompass class descriptions. In this case, you can get several advantages at once.

First, there is no need for a static descriptor, which clutters up the code. Fields described outside of classes will be considered static. It is also very natural for the following reason. In the current situation in C # (and Java), the very space of the program text is used ineffectively. The ownership of a field or method is determined not by its location in the text, but by the presence or absence of the static specifier.

Secondly, the fully qualified identifier of a static field or method could in this case have no longer triple, but only a double designation. This creates a prerequisite for the mandatory use of qualified names and the using directive. The creators of C #, for obvious reasons, did not require mandatory qualifications for at least three levels of naming (namespace, class, field).

Finally, the class ceases to unnaturally combine two different roles - describing the type and the space for static fields and methods. This combination, by the way, makes it difficult to understand and learn the Java and C # languages.

Let's consider the listed possibilities using the already discussed example. Namespace A containing a (static) field C could be defined like this

*namespace A {*

*public double C = 2.71828182845904523}*

Properties mask a call to a procedure (method) as a call to a variable (field). This is widely used in visual programming systems when changing the value of a field that determines the appearance of a screen element must be accompanied by a redrawing of this element. When using properties, you can do both of these things by assigning a value to the property.

Let the string (of string type) property Title of display objects of the Element class specify the title caption of such objects. This property declaration might look like this:

```
public class Element {
string title;
public string Title { get { return title;}
    set {
    title = value;
    Repaint(); }
```

The constructions are a little more cumbersome, but they do not mislead the reader of the program. It does not hide the fact that the title change is performed by a subroutine, which, in addition to assigning a value, can perform other actions. In the first case, it is impossible to distinguish the use of properties from the use of ordinary fields in the text of the program (without referring to the description of the Element class).

Programming in the environment of visual systems like Delphi, from where properties migrated to the C # language, can be divided into several levels. The first is application development. In this case, the programmer, creating a program with a graphical user interface, as a rule, uses only ready-made components with already programmed properties. In this case, the external indistinguishability of properties and fields does not interfere too much, since we are talking about using already debugged libraries, and the programmer has a built-in help system at his disposal. Moreover,

many of the properties of visual components do not even appear in the part of the program that the programmer writes manually. The values of the properties of windows, buttons and other interface elements are simply set in the dialog mode with the visual system. An application programmer in a visual environment, in fact, does not deal with the entire programming language, but only with that part of it, which includes the possibility of using ready-made properties of ready-made classes, but does not include their definition.

The second level is the development of visual components. In this case, the programmer is fully responsible for the correctness of the created class system.

Finally, you should consider using a property-equipped language outside of the context of the visual environment. In this situation, that is, in the general case, when properties are applied not only for visual elements, the indistinguishability of calling subroutines (get and set) from calling a field can do a bad job, impairing the ability to understand the program. It is no longer possible to understand from the text of the program whether the action is reduced only to changing or getting the value of the field, or whether it is associated with the execution of other operations.

The C # construction called an indexer can be very ambiguous. Indexers are elements of classes (as well as interfaces and structures) that allow objects to be treated as arrays (by specifying the index in square brackets). This access is implemented by implicitly called get and set routines.

For example, accessing individual bits of a 32-bit integer value (bits) can be disguised as a logical array access. For this, the following description is created (in this case, the structure):

```
public struct BitSet {
int bits;
public bool this[int i] {
get{return 0<=i && i<32? (bits & 1<<i) != 0: false;}
   set {
   if(i<0 || i>31) return;
   if(value) bits |= 1<<i; else bits &= ~(1<<i);}
```

When using properties and indexers, the programmer hides the costs that occur during the work of get and set accessor programs, which provokes the use of inadequate and ineffective techniques. For example, using an indexer to access the elements of a linear list by their number with a significant length of the list is much less effective than accessing an array element, although it looks the same. Using such an indexer to step through the list is completely absurd, which, nevertheless, you are nudged into.

The ability to understand the program is deteriorating. From its text, it becomes impossible to know whether we are dealing with a field or a property, an array or an indexer. This is despite the fact that accessing a field is always associated only with getting or setting its value, and accessing a property can involve performing any actions. The situation is similar for arrays and indexers.

I will give an example in C #, when the simplest fragment, whose operation in other circumstances would be understood absolutely unambiguously, when using C # is completely unpredictable.

```
int i, s=0;
for (i=1; i<=100; i++ ) a[i]=i;
for (i=1; i<=100; i++ ) s += a[i];
System.Console.WriteLine(s);
```

Well, what's wrong with that? First, the elements a are alternately assigned the values of the first hundred numbers of the natural series 1, 2, 3, □, 99, 100. Then the sum of these numbers is calculated and displayed, which must be equal to 5050. Nothing like that! The value printed by this program can be anything. For example, equal to 338350, if a is an indexable object of this type:*class Array {*

*public int this[int i]{*

*get{ return i*i; }*

*set {}*

A side effect is that the values of variables change when the expression is evaluated. Many languages (including Pascal and Oberon) can define functions that have mutable parameters or can change the values of global variables. These features have side effects. Their use is considered bad practice as they pose a number of problems. The meaning of the side-effect expression can depend on the order in which the operands are evaluated.

Let's look at an example. Interpreters often use the stack when evaluating expressions. Let Pop (S) be the function that returns the value popped from the S stack, and Push (S, V) the procedure that pushes the value V onto the S stack. When Pop (S) is called, the stack changes, this function has a side effect. To replace the top two values in the stack with their difference (the value at the top must be subtracted from the value below the top), you can try to write Push (S, –Pop (S) + Pop (S)). The programmer expects that the first of the two recorded calls Pop (S) will be executed first. In this case, the value taken from the top of the stack will participate in the calculation with a minus sign. In fact, if the language does not set the order of evaluation of the operands (this is the case, for example, in Pascal and C), the compiler can swap the terms and program this action as Push (S, Pop (S) –Pop (S), which will lead to the wrong result.

The presence of side-effect expressions makes the program difficult to understand and is a potential source of errors. The task of optimizing compilers also becomes more complicated.

However, in languages such as Algol-68 and C, side-effect exploitation is one of the main techniques for obtaining a compact program record. In C and its derived languages, increment and decrement operations (++, —) and the use of operators (in particular, assignments) as expressions create side effects. Having these capabilities is one of the main prerequisites for getting confusing and obscure programs.

C # has even more room for side effects. The properties and indexers we've looked at make a side effect normal since it's based entirely on it. A completely harmless notation, similar to accessing a field or array element (which does not create a side effect), in fact, can mean performing any action. And changing the value of a property almost necessarily has a side effect.

Let's take an example. The loop below is fairly typical for C programs.

```
while (b = a [n ++]) {};
```

An expression in parentheses has a double side effect in C. Firstly, each of its calculations assigns to the variable b the value of the nth element of the array a, and secondly, increases the value of the variable n. In the absence of a habit of the C language style, it is not easy to understand such a construction, but it is possible. The same notation is allowed in a C # program. But looking at her, you can no longer tell what is happening. After all, a can be an indexed object, and b - a property, and "inside" both one and the other can be anything.

Such constructions of the C # language as namespaces, properties, indexers lead to situations when it is impossible to understand the nature of the objects used in the program from the text of the program (program unit). Class names can be mistaken for

namespace names and vice versa, properties are indistinguishable from fields, indexed objects from arrays. When using the using directive, there is an ambiguity in determining the belonging of identifiers to one or another namespace.

The noted problems with the unambiguous identification of program objects can be partially solved using the hints system built into the programming environment. The C # language is intended primarily for use in the powerful Microsoft Visual Studio programming environment. It is equipped with a developed help system and tools that allow, in the course of a dialogue with the system, to determine the characteristics and belonging of the program objects.

The foregoing means that the C # language assumes a "heavyweight" implementation, when the programming system should include complex auxiliary tools, without which the development of C # programs is complicated. Significant costs for the creation of programming systems for the C # language, in addition to the rather high complexity of the language itself, are also due to the fact that an extensive system library (the System namespace) is an integral part of it. At one time, when creating the Ada language, its authors put forward an important principle that guided them in developing the language. The language should facilitate the production of readable, clear and understandable programs. The ease of writing the program is not the primary factor. However, the importance of improving "readability", albeit at the expense of brevity and ease of writing, was recognized several years earlier, at the time of the adoption of structured programming.

It's overloaded with adjectives. Of course, the description of the verbosity class is deliberately made so verbose (verbosity). But the language allows it. The syntax of C # is designed in such a way that the use of multiple modifiers and descriptors is the norm. Their following one after another without any separators makes it difficult to perceive the program.

Let me explain the notation used in the example. All defined elements of a class are of type verbosity. The protected internal modifiers mean that access to the members of the class is restricted to the limits of the given project (internal) or classes derived from verbosity. readonly means read-only access; virtual - the ability to override a method in derived classes. The @ symbol in a constant name allows you to use the const reserved word as an identifier. The word this, denoting this instance of the class, is a mandatory element of the indexer description.

At the same time, the absence in the C # language of special words denoting a method and a property (like the words procedure, function in Pascal-like languages) makes us distinguish their descriptions from each other and from the description of fields and indexers by indirect signs. There are parentheses after the method name in the method description; in the property description - curly; the indexer has square; there are no brackets in the field description, but an equal sign may be present. Just a test for attentiveness is obtained.

The verbosity of C # (as well as Java) looks unattractive and stylistically flawed. The rules borrowed from C allow you to write expressions and operators very compactly using a variety of special characters. At the same time, object innovations are cumbersome and, conversely, ignore the possibilities of punctuation marks. As a result, it turns out that writing is difficult and not easy to read.

As mentioned, with judicious use of program space, the number of different descriptors could be fewer. The static specifier would not be needed. The number of words regulating access could have been less. An example of a simple, convenient and visual design of access is given, again, by the languages Oberon and Oberon-2.

Of course, the discussed shortcomings of C # do not at all deprive the language of prospects. It is preferable to C ++ in many ways. General dissatisfaction with the C ++ language, which is recognized by the very emergence of the new language, is one of the main prerequisites for the success of C #.

Comparing C # to Java, you can see many similarities. True, if Java systems are multi-platform, then the C # implementation exists so far only for the Windows operating system and only one. But despite the cumbersomeness, the language can be expected to be implemented for other systems as well. In addition, the Microsoft .NET platform itself, with a unified program execution environment, can be promoted to alternative architectures, primarily UNIX systems.

C # seems to be a more realistic language than Java. Unlike Java, it is self-sufficient. That is, you can write any program in C # without resorting to other languages. This is possible due to the presence of "unsafe" code blocks that open access directly to the hardware. In Java, native methods must be used to access low-level tools, and must be programmed in other languages.

# CHAPTER 4: LOGIC OF POCKET AVIONICS APP

## 4.1 Container with parameters for the card

A ScriptableObject is a data container that you can use to save large amounts of data, independent of class instances. One of the main use cases for ScriptableObjects is to reduce your Project's memory usage by avoiding copies of values. This is useful if your Project has a Prefab

that stores unchanging data in attached MonoBehaviour scripts

Every time you instantiate that Prefab, it will get its own copy of that data. Instead of using the method, and storing duplicated data, you can use a ScriptableObject to store the data and then access it by reference from all of the Prefabs. This means that there is one copy of the data in memory.

Just like MonoBehaviours, ScriptableObjects derive from the base Unity object but, unlike MonoBehaviours, you can not attach a ScriptableObject to a GameObject

. Instead, you need to save them as Assets in your Project.

When you use the Editor, you can save data to ScriptableObjects while editing and at run time because ScriptableObjects use the Editor namespace and Editor scripting. In a deployed build, however, you can't use ScriptableObjects to save data, but you can use the saved data from the ScriptableObject Assets that you set up during development.

Data that you save from Editor Tools to ScriptableObjects as an asset is written to disk and is therefore persistent between sessions.

Using a ScriptableObject

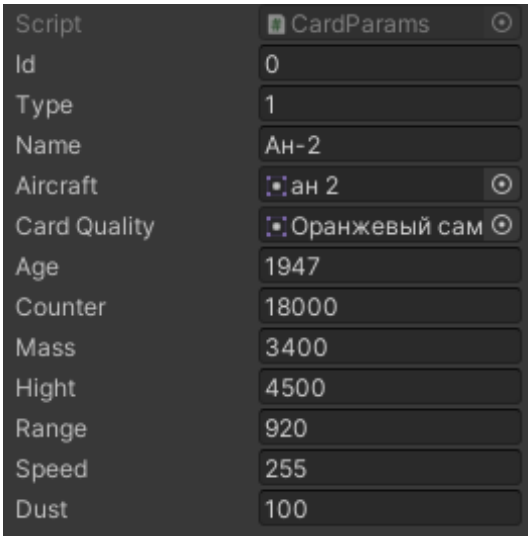
The main use cases for ScriptableObjects are:

Saving and storing data during an Editor session

Saving data as an Asset in your Project to use at run time

To use a ScriptableObject, create a script in your application's Assets folder and make it inherit from the ScriptableObject class. You can use the CreateAssetMenu attribute to make it easy to create custom assets using your class. For example **Pics 4.1**

```
using UnityEngine;

[CreateAssetMenu(fileName = "New Card", menuName = "Card")]
Скрипт Unity | Ссылок: 3
public class CardParams : ScriptableObject
{
    public int id;
    public int type;
    public string name;
    public Sprite aircraft;
    public Sprite cardQuality;
    public int Age;
    public int Counter;
    public int Mass;
    public int Hight;
    public int Range;
    public int Speed;
    public int Dust;

}
```




**Pics 4.2**

### 4.2 Card availability check

Creates a new game object, named name.

Transform is always added to the GameObject that is being created. The creation of a GameObject with no script arguments will add the Transform but nothing else. Similarly, the version with just a single string argument just adds this and the Transform. Finally, the third version allows the name to be specified but also components to be passed in as an array **Pics 4.3**

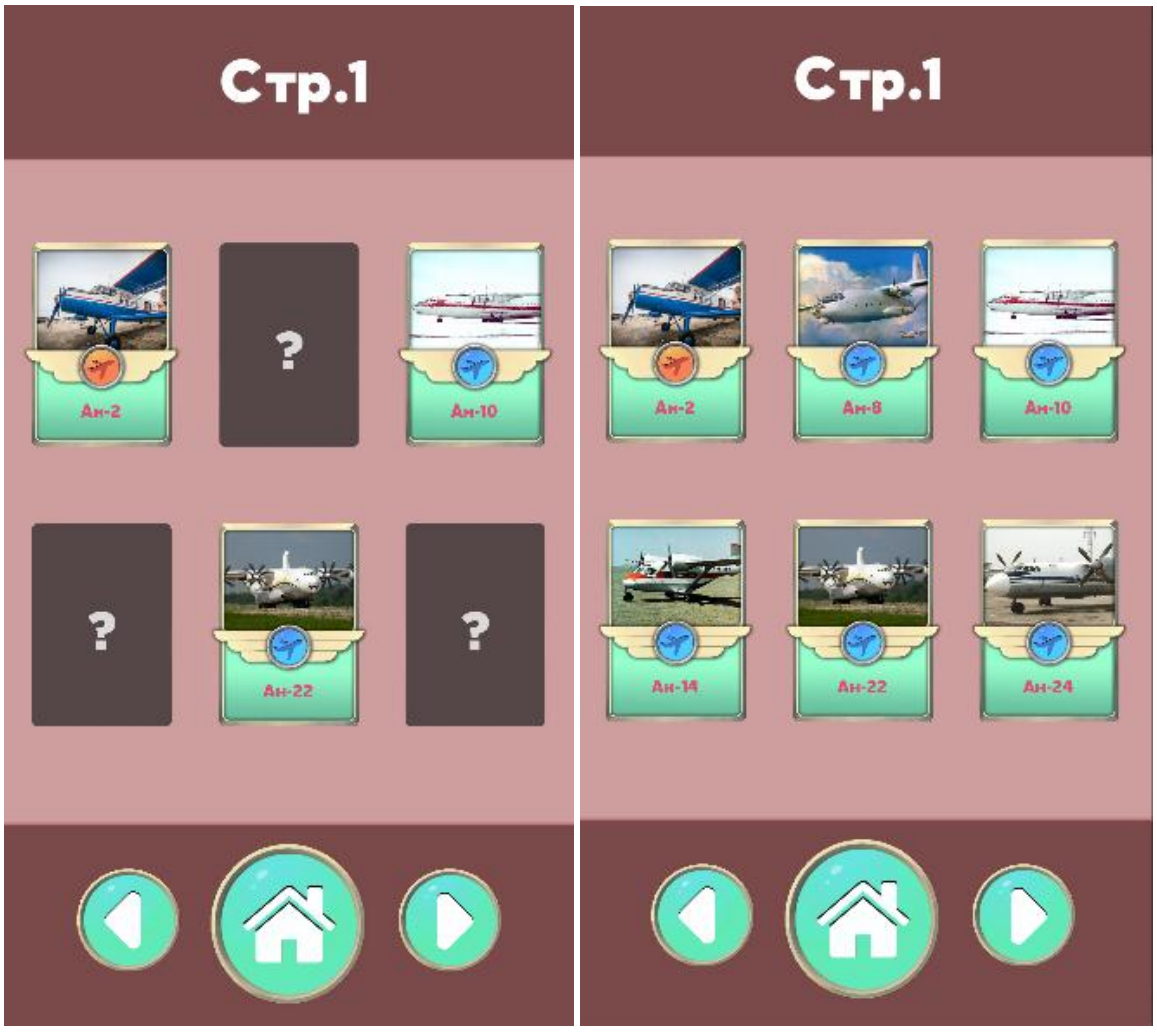
```
public GameObject[] Pages;
public GameObject[] AllCards;
public bool[] MyCollections;
public Text PageNumberText;
int PageNumber = 1;

Сообщение Unity | Ссылок: 0
void Start()
{

    for(int i = 0; i < MyCollections.Length; i++ )
    {
        if (MyCollections[i] == true)
        {
            AllCards[i].SetActive(true);
        }
    }
}
```

```
public void PageChanger(int direction)
{
    PageNumber += direction;
    if (PageNumber > Pages.Length)
    {
        PageNumber = 1;
    }
    else if(PageNumber <= 0)
    {
        PageNumber = Pages.Length;
    }

    foreach (GameObject Page in Pages)
    {
        Page.SetActive(false);
    }
    Pages[PageNumber - 1].SetActive(true);
    PageNumberText.text = "Стр." + PageNumber.ToString();
}
```

**Pics 4.3**

Every time a player enters the game, the program checks the player's cards and activates those cards that he has. Missing cards are shown blank.One canvas for all UI elements is sufficient, but multiple canvases in a scene are acceptable. It is also possible to use multiple canvases, with one exposed as a child of the other, for optimization. The nested canvas uses the same Render Mode as its parent.

Traditionally, user interfaces are displayed directly on the screen as simple elements. This means they have no idea of the 3D space displayed by the camera. Unity supports this display-space rendering method, but also allows interfaces to be drawn as objects in the scene, depending on the Render Mode. The available modes are Screen Space - Overlay, Screen Space - Camera and World Space **Pics 4.4**.




**Pics 4.4**

### 4.3. Implementation of a collection of maps in the application

The Hierarchy window contains a list of every GameObject in the current Scene. Some of these are direct instances of Asset files (like 3D models), and others are instances of Prefabs, which are custom GameObjects that make up most of your game. When you add or remove GameObjects the Scene (or when your gameplay mechanic adds and removes them), they appear and disappear from the Hierarchy as well **Pics 4.5**


**Pics 4.5**

By default, the Hierarchy window lists GameObjects by order of creation, with the most recently created GameObjects at the the bottom. You can re-order the GameObjects by dragging them up or down, or by making them “child” or “parent” GameObjects (see below).

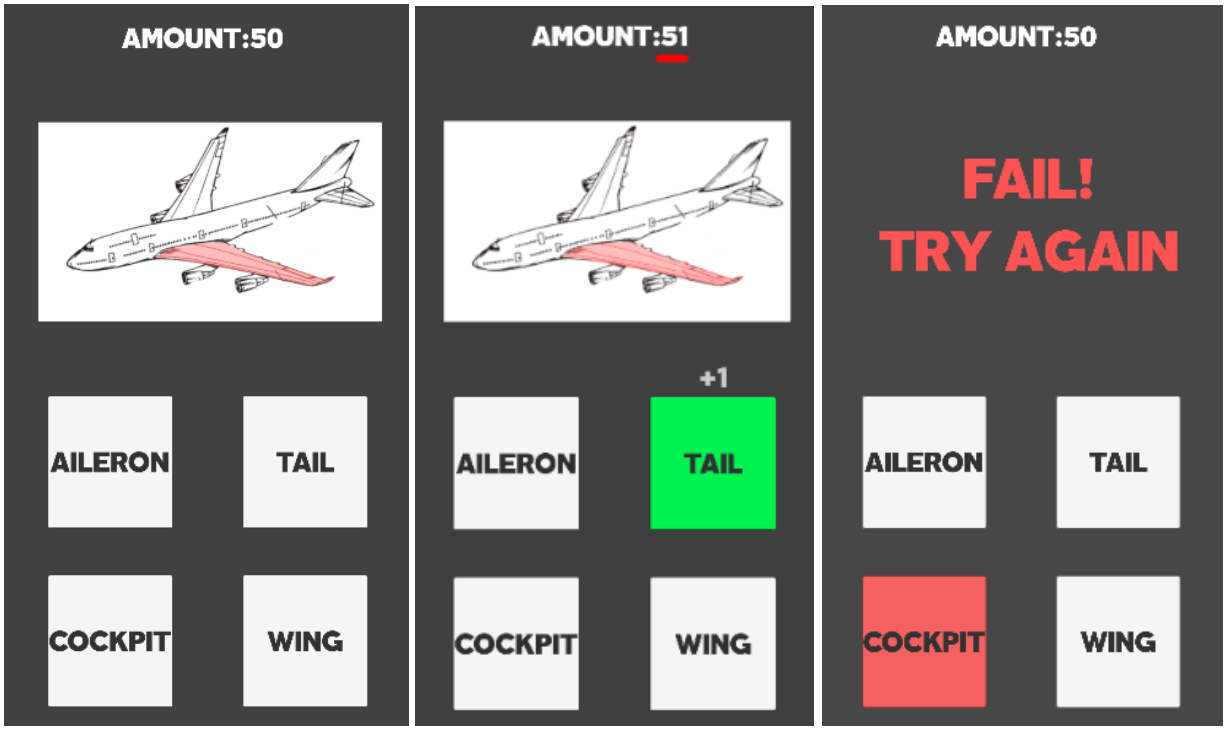
Unity uses a concept called Parenting. When you create a group of GameObjects, the topmost GameObject or Scene is called the “parent GameObject”, and all GameObjects grouped underneath it are called “child GameObjects” or “children”. You can also create nested parent-child GameObjects (called “descendants” of the top-level parent GameObject) **Pics 4.6**

```
Ссылок: 0
public void BigCardClose()
{
    BigCardMenu.SetActive(false);
}
Ссылок: 0
public void BigCardParams(int CardID) //Вытаскивание параметров SO
{
    BigCardMenu.SetActive(true);
    ScriptableObject = SOCards[CardID];
    AircraftImage.sprite = ScriptableObject.aircraft;
    AircraftParamsImage.sprite = ScriptableObject.cardQuality;
    CardName.text = ScriptableObject.name;

    Age.text = "Год выпуска: " + SOCards[CardID].Age;
    Count.text = "Тираж: " + SOCards[CardID].Counter;
    Mass.text = "Масса: " + SOCards[CardID].Mass + " кг";
    Hight.text = "Потолок: " + SOCards[CardID].Hight + " м";
    Range.text = "Дальность полета: " + SOCards[CardID].Range + " км";
    Speed.text = "Максимальная скорость : " + SOCards[CardID].Speed + " км/ч";

}
```

**Pics 4.6**

### 4.4 Implementation of the "Test" block in the application

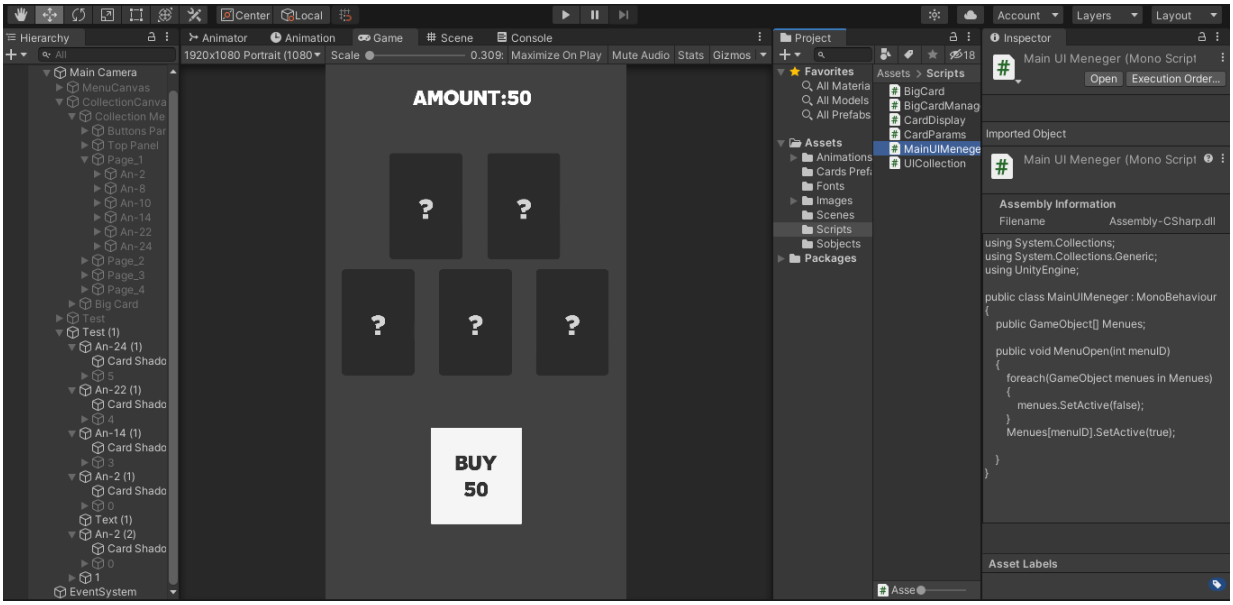
Every day the player gets the opportunity to take a test. For each correct answer in the test, the player receives points. If the answer is incorrect, the player drops out of the test, all points obtained during the test remain with the player. 4-Choice Tests **Pics 4.7**



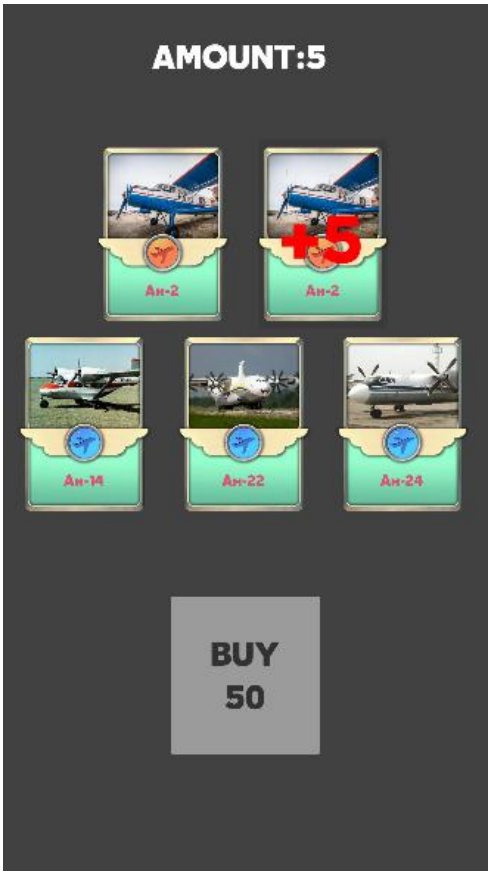

**Pics 4.7**

## 4.5 Implementing drops from cards and writing them to the collection

Having accumulated 50 points, the player can purchase 5 random cards that will be added to the collection. Each repeated card instantly turns into points **Pics 4.8**




**Pics 4.8**




**Pics 4.9**

## CHAPTER 5: OCCUPATIONAL HEALTH AND SAFETY

Occupational safety is a system of legal, socio-economic, organizational and technical, sanitary and hygienic, medical and preventive measures aimed at preserving life, health and ability to work in the process of work.

The main provisions on labor protection are enshrined in the Law of Ukraine "On Labor Protection" (Resolution of the Verkhovna Rada of Ukraine of October 14, 1992 № 2695-XII), the Labor Code of Ukraine and in the regulatory and technical documentation on labor protection, which provides for responsibilities on labor protection, the content and procedure for approval of instructions on labor protection, requirements for production facilities and equipment, free issuance of overalls and personal protective equipment for work with harmful working conditions, the issuance of milk, preventive nutrition, etc.

List of dangerous and harmful production factors operating in the work area

The developed system of external television video surveillance is designed to work around the aircraft, in the operating room, where a centralized surveillance panel is installed.

Classification ”on employees servicing such equipment may be subject to the following dangerous and harmful production factors:

1. Increased or underestimated relative temperatures humidity, speed and dustiness of the air flow.

2. Intense neuro-psychological state associated with mental tension and emotional overload.

3. Increased voltage in the electrical circuit, the short circuit of which can pass through the human body when touching and touching damaged parts of electrical equipment on board the aircraft, erroneous supply of voltage during maintenance and repair, as well as inspection of devices and equipment around the aircraft, in operating room, as a result of insulation damage, etc.

4. Fire and explosion hazard.

Organizational and technical measures to eliminate or reduce the level of hazardous and harmful factors

Normalization of the microclimate

"Sanitary and hygienic requirements for the air of the working area", means a combination of temperature, relative humidity, speed and dust. These parameters affect the functional activity of man, his health and well-being and the reliability of technology.

Under the optimal microclimatic parameters are understood those that with long-term and systematic exposure to humans ensure the preservation of normal functional and thermal state of the body without enhanced thermoregulatory reactions, create a feeling of thermal comfort and are a prerequisite for a high level of efficiency.

A place where an employee is more than 50% of his working time or more than 2 hours continuously is called a permanent place of work.

When performing operator-type work related to nervous and emotional stress, the optimal values of air temperature 22-24oC, its relative humidity 60-40% and speed (not more than 0.1 m / s) must be observed.

To maintain the optimal microclimate in the operating room requires the use of central or autonomous heating, humidifier, ventilation devices, air conditioning.

Ensuring a stable neuro-psychological state

To normalize the neuro-psychological state associated with mental tension and emotional overload, the following measures are taken:

1) adheres to the established mode of work and rest;

2) complex automation and mechanization of processes is carried out;

3) employees are provided with the necessary, serviceable personal protective equipment;

4) sanitary and technical propaganda and training in safe work practices are carried out;

5) rational arrangement of equipment is applied;

6) architectural, planning and technological solutions aimed at isolating noise sources are used.

Ensuring electrical safety

Calculation of protective earthing of the remote equipment of the surveillance system.

Grounding is made of tubular vertical grounding and is connected by a metal strip.

1. Determine the calculated value of the resistivity of the soil for vertical grounding $(\rho'_{розр.})$ and $(\rho''_{розр.})$:

$$\rho'_{розр.} = \rho \cdot k'_n$$

$$\rho''_{розр.} = \rho \cdot k''_n$$

$k'_n, k''_n$ - coefficients of specific soil resistance, equal to 1.6 and 2.5, respectively, for the third climatic zone.

$$\rho'_{розр.} = 10^2 \cdot 1{,}6 = 1{,}6 \cdot 10^2 Ом \cdot м$$

$$\rho''_{розр.} = 10^2 \cdot 2{,}5 = 2{,}5 \cdot 10^2 Ом \cdot м$$

2. Determine the current resistance of one vertical grounding by the formula:

$$R_{тр} = \frac{1}{2\pi} \cdot \frac{\rho'_{розр.}}{l} (\lg \frac{2l}{d} + \frac{1}{2} \lg \frac{4H + l}{4H - l})$$

where **l** is the length of the grounding pipe, 3 m;

**d** is the diameter of the pipe, 0.05 m;

**H** is the depth of the tubular ground, which is equal to the distance from the ground to the middle of the rod, 2.3 m.

$$R_{тр} = \frac{1}{2\pi} \cdot \frac{1{,}6 \cdot 10^2}{3} (\lg \frac{2 \cdot 3}{0{,}05} + \frac{1}{2} \lg \frac{4 \cdot 2{,}3 + 3}{4 \cdot 2{,}3 - 3}) = 4{,}395 Ом$$

3. Determine the conditional number of vertical grounding:

$$n' = \frac{R_{тр.}}{R_{доп.}}$$

$$n' = \frac{4{,}395}{4} \approx 1{,}1$$

The coefficient of use of a single ground, which takes into account the mutual shielding of pipes:

$$\eta_{тр.} = 0{,}5$$

Determine the actual number of vertical grounding:

$$n = \frac{n'}{\eta} = \frac{1{,}1}{0{,}5} = 2{,}2$$

Based on the conditional and refined calculations of vertical grounding conductors, we take their number equal to 2.

4. The spreading current of the connecting strip:

$$R_c = \frac{0{,}16}{L} \cdot \rho''_{розр.} \cdot \lg \frac{2L^2}{b \cdot h}$$

where **L** is the length of the line connecting the vertical grounding, 6.9 m;

**h** - depth of laying the strip, 0.8 m;

**b** - width of the strip, 0.04 m.

$$R_c = \frac{0,16}{6,9} \cdot 2,5 \cdot 10^2 \cdot \lg \frac{2 \cdot 6,9^2}{0,04 \cdot 0,8} = 4,62 Ом$$

Coefficient taking into account the mutual shielding of the strip and vertical grounding:

$$\eta_c = 0,24$$

5. Determine the current resistance of the entire grounding device:

$$R_з = \frac{R_{mp.} \cdot R_c}{R_{mp.} \cdot \eta + n \cdot R_c \cdot \eta_{mp.}}$$

$$R_з = \frac{4,395 \cdot 4,62}{4,395 \cdot 0,24 + 2,2 \cdot 4,62 \cdot 0,5} = 3,6 Ом$$

The organization of work on fire safety at the enterprise is entrusted to its head, and in shops, services, departments and sections by the order of the head of the enterprise - to the corresponding heads.

Combustion is a complex of physicochemical transformations that is accompanied by the release of heat and, in most cases, the emission of light.

Fire is a combustion that develops in time and space and ceases to be controlled.

Fire safety of the designed external video surveillance system is provided in accordance with “Fire safety. General requirements ”, explosive - according to “Explosion safety. General requirements ”.

According to the designed devices are dangerous in the fire relation because the electric current is a source of fire and in installation plastic, paint and varnish coverings, vinyl isolation of wires, printed circuit boards are used.

Fire safety during operation of installation is provided:

- system of fire prevention measures;
- system of fire protection measures;
- organizational and technical measures.

Fire prevention is provided by the following:

- in the design of the installation there are no flammable materials;

- the cross section of the mounting wires is selected according to the allowable current density

$j=2\div3\ А/мм^2$

- all resistors used are selected according to the allowable scattering power;

- capacitors are selected taking into account the allowable voltage;

- in the projected installation, the possibility of an explosion or the formation of a spark is excluded, as in its design there are no open switching contacts, all elements of the circuit are closed and hermetically sealed. The case and the front panel are made of duralumin of the D16 brand which at blow on other metals does not form sparks;

- installation details and printed circuit boards are covered with an electrically insulating varnish that excludes a possibility of hit on circuit elements of moisture, and, accordingly, and short circuit, corrosion and ignition;

- Fuses are provided in the power supply unit to protect the network from overload and short circuit.

Fire protection is provided by:

- application of the automatic fire alarm system, at the panel of the centralized supervision;

- the use of fire extinguishers.

Organizational and technical measures include:

- training of engineering and technical personnel in fire safety rules;

- organization of fire protection;

- development of instructions on the procedure for working with fire-hazardous substances and materials.

Admission to work with electrical installations of persons who have been instructed and trained in safe methods and techniques of work, who have passed the test according to the rules of safety of work with the assignment of a qualification group for safety and not
have medical contraindications established by the Ministry of Health.

2. Safety requirements before starting work

- Before starting work, it is necessary to free the workplace from unnecessary flammable objects and materials.
- Make sure that the fuses correspond to the ratings indicated on the front panel of the device.
- Check that the device is grounded.

3. Safety requirements during work

- Connect and disconnect plug connectors, device cables with the power off.
- Repair work associated with the replacement of elements, modules of the alarm system to carry out in a de-energized state.
- Repair and maintenance of the device to do only the serviceable tool having external isolation and the corresponding marking.
- Carry out electrical measurements only with approved certified devices.
- It is allowed to install fuses only with the nominal data indicated in the diagram.

4. Safety requirements after work

- At the end of the work, check the quality of the power supply connection of the alarm system.
- You need to clean your workplace.
- Sanitary norms and rules of personal hygiene must be observed.
- If there are any defects in the operation of the device, you must notify the supervisor.

5. Safety requirements in emergency situations

- In case of fire, use only carbon dioxide fire extinguisher.

There must be a carbon dioxide fire extinguisher (OU-2 or OU-5) in the room where the centralized monitoring panel is located.

Conclusion:

We calculated the protective grounding for the remote equipment of the external video surveillance system, the resistance of which satisfies the requirements, Rs <4 Ohms.

## CHAPTER 6: ENVIRONMENTAL PROTECTION

Environmental protection is becoming a complex problem, which is determined by the complexity of the system that combines nature, society and production. Along with environmental problems, it also solves socio-economic problems - improving human living conditions, maintaining their health. Obviously, a written, science-based approach is needed to protect the environment. Any technical process to some extent affects the protection of the environment, polluting it.

Digital cameras, when operated during aircraft protection, are not capable of causing direct damage to the environment because they do not form any carcinogenic, toxic or other harmful substances that affect the air, water bodies, humans, animals, vegetation or soil. .

However, in the manufacture of the device is indirect damage to the environment:

- high level of electricity consumption;
- electromagnetic radiation;
- acoustic pollution;
- soft X-rays;

The growth of use in various sectors of the economy requires the most serious attitude to issues related to the impact of computers on the living environment.

The complexity of this problem can be analyzed on the example of the manufacture of integrated circuits, which reveals a number of factors that

have an impact on the environment. The elements of the circuit are mounted on printed circuit boards made of foil fiberglass. When processing its inevitable waste: pieces of boards, powder dust, which, getting into the soil, is stored for a long time. And the fumes that are formed during digestion have a detrimental effect on the workers employed in this production and are released into the environment.

During operation, the elements of its design emit heat (heated chips, transistors, resistors), resulting in heated protective varnishes, paints, creating toxic substances in the atmosphere in the form of volatile fractions. This is only a superficial analysis of one of the components

Electromagnetic energy is used in radio communication, radar, radio navigation, television, metallurgy and metalworking industry for induction melting, welding, metal spraying.

In production facilities, sources of electromagnetic radiation are unshielded operating elements of high-frequency installations (inductors, capacitors, high-frequency transformers, feeder lines, capacitor banks, coils of oscillating circuits, etc.). When operating RF, VHF, UHF transmitters on radio and TV centers, the sources of electromagnetic radiation are high-frequency generators, antenna switches, devices for assembling the power of the electromagnetic field, communications (from generator to antenna device), antennas.

The degree of exposure of workers depends on the number of transmitters placed in the room (in some areas, radio and television centers can be up to 20), their power, degree of shielding, placement of individual units inside and outside the room.

In terms of production, electromagnetic radiation is characterized by a variety of modes of generation and options for workers (radiation in the near zone, induction zone, general and local, which often acts together with other adverse environmental factors). Radiation can be an isolated combination of mixed and combined (when another adverse factor is acting at the same time). It can be constant or intermittent. The latter, in turn, can be periodic and aperiodic. The electromagnetic field has a negative effect on the human body. There are two forms of electromagnetic radiation in the radio frequency range - acute and chronic, which, in turn, is divided into three stages: mild, moderate and severe. The chronic form is characterized by functional disorders of the nervous, cardiovascular and other systems of the body, manifested by asthenic syndrome, and autonomic disorders, mainly of the cardiovascular system.

People who are exposed to chronic radiation are more likely (1.9 times men and 1.5 times women) than those who are not exposed to radiation, complain of poor health, including headaches (1 , 5 times men and 1.3 times women), heart pain (1.8 times men and 1.5 times women), palpitations, general weakness, drowsiness, tinnitus, paresthesia, etc.

Electromagnetic radiation is a powerful physical stimulus. Different organisms have different sensitivities to natural and anthropogenic (artificial): the nature and presence of the biological effect depend on the parameters and level of organization of the biosystem. Millimeter waves affect mainly the receptor apparatus, longer waves - on the central nervous system.

Radiofrequency radiation is absorbed differently by different organs and systems of the body: their shape and linear dimensions, orientation relative to the source are essential. Primary changes in the functions of the central nervous system and related disorders cause biological effects at the level of organs and systems. Prolonged exposure to high levels of electromagnetic radiation leads to overstrain of adaptive-compensatory mechanisms, significant deviations in the functions of

organs and systems, metabolic disorders and enzymatic activity, hypoxia, organic changes. Because electromagnetic radiation acts in the production environment, usually in combination with other factors, its effect on the human body is enhanced. Protective and adaptive reactions that occur in humans under the influence of electromagnetic radiation are non-specific. The most common adaptive reactions are excitation of the central nervous system and increased metabolism.

The effects of exposure to human biological tissues of electromagnetic radiation in the low-power radio frequency range are divided into thermal and non-thermal. The thermal effect can be shown at the person or increase in body temperature, or selective (selective) heating of its separate bodies which thermoregulation is complicated, gall bladders, stomach, intestines, testicles, lenses, vitreous, etc.). The effect of electromagnetic radiation on a biological object is detected when the intensity of radiation is lower than its thermal threshold values, ie there are non-thermal effects or specific action of radio waves, which is determined by the information aspect of electromagnetic radiation perceived by the body and depends on the source and communication channel . Obviously, information processes also play a role in the thermal action of the electromagnetic field on the body. In addition, the action of low-intensity electromagnetic radiation leads to local heating - micro-heating.

Conditionally distinguish the following mechanisms of biological action:

- direct effect on tissues and organs, when the function of the central nervous system changes and the associated neurohumoral regulation;

- reflex changes of neurohumoral regulation;

- a combination of the main mechanisms of pathogenesis, actions with a predominant metabolic disorder, enzyme activity. The proportion of each of these mechanisms is determined by physical and biological changes in the human body.

Therefore, the effect of electromagnetic radiation is systemic in nature and requires appropriate systemic measures to protect against it.

The biological effect of the electrostatic field on a person depends on its duration, the shape of the conductive parts of the equipment, the location of the workplace relative to the radiation source, climatic conditions and so on. Experimental animal studies have been shown to affect the nervous, cardiovascular, endocrine and other body systems. In particular, changes in the electrical activity of the cerebral cortex and conditioned reflex activity were registered. The electrostatic field causes changes in blood pressure, which are unstable and phase in nature, the rate of blood clotting, the content of sulfhydryl groups in the blood.

Exposure to workers leads to irritability, headache, sleep disturbances, loss of appetite, impaired central nervous system function, changes in heart rate (usually bradycardia) and carbohydrate, lipid, protein and mineral metabolism, as well as a decrease enzyme activity.

Measures to protect against static electricity are aimed at reducing the generation of electric charges or their removal from the electrified material by increasing its electrical conductivity. These measures include grounding of metal and electrically conductive elements of equipment, installation of static neutralizers, increasing the surface and bulk conductivity of dielectrics. The elements of the equipment in which electric charges are formed and the isolated electrically conductive sections of technological installations are subject to grounding. Devices for protection against static electricity are almost always combined with protective earthing devices.

The most effective of these measures to combat static electricity is to increase the surface and bulk conductivity of dielectrics. Increasing the relative humidity to 60-75% significantly increases the surface conductivity of dielectric hydrophilic materials (adsorb a thin film of moisture on its surface). The use of antistatic

substances is based on this principle. Surfactants are applied to the surface or introduced into the mass of the material (the latter is more rational, as it promotes long-term storage of polymers with antistatic properties).

It is also possible to neutralize electric charges by means of air ionization. To do this, use static electricity neutralizers, the principle of which is to create near the electrified materials of positive and negative ions. For antistatic protection, you can also use the principle of shielding with metal sheets. The field formed on the walls of the screen neutralizes the external field. In order for electric charges from the human body to be diverted to the ground faster, electrically conductive floors are used. Individual means of protecting the human body from static electricity include antistatic robes, grounding wristbands, antistatic shoes, and others. When choosing such means, it is necessary to take into account the peculiarities of the technological process, physical and chemical properties of the processed material, the microclimate of the premises, etc.

The television system of external video surveillance developed in this diploma project is a necessary element and use accelerates process of calculations and allows to receive results on development and improvement of system. Incoming data:

Розрахунок видатку електроенергії визначають в залежності від потужності базового обладнання, кількість годин роботи з урахуванням коефіцієнта корисної дії:

$$\sum_{i=1}^{n} W_{об} = \frac{\sum_{i=1}^{n} M_{уі} * Ф_{оі} * К_{зі} * К_0}{\eta * К_{вм}} \qquad (6.1)$$

where: Myi - total power of the i-th equipment, kW; Fdi - the actual time fund of the i-th equipment before the introduction of the complex: FdEOM = (365-144) * 8 = 2652 h - before the introduction of the complex; FdEOM '= (365-144) * 6

= 1768; Fd '= 365 * 24 = 8760 h - after the introduction of the complex; Kzi - load factor of the i-th equipment; Kz = 0.8 - before the introduction of the complex; Kz '= 0.5 - after the introduction of the complex; Kz '= 0,2 (as in parallel the Server serves 4 more); K0 - renewal rate: K0 = 0.8; η - efficiency: η = 0.85; Kvs - coefficient of losses in networks:

Kvs = 0.95; Wob = 3 * WEOM = 1261.13 KW * year;

We = (0.2 * 1768 * 0.5 * 0.8) / (0.95 * 0.85) = 175.1579 kW * year;

Wp = (0.23 * 8760 * 0.2 * 0.8) / (0.95 * 0.85) = 399.22 kW * year;

Determine the cost of electricity for lighting the production room by the formula:

$$\sum_{i=1}^{n} W_{oy} = \frac{P_y * F_{ui} * \Phi_{ri} * K}{1000} \qquad (6.2)$$

The specific consumption of electricity per 1 m2 of area, depending on the type of lamp, in this case is equal to 60 * 2/20 = 6 W;

Ftsi - area of the i-th section, 20 m2;

Fgi - the number of operating hours of lighting fixtures:

Fgi = 5 * 221 = 1105 h,

Fg = 4 * Fgi = 4420 h;

F'gi = 4 * 221 = 884 h,

Fg = 4 * Fg = 3536 h.

K - loss factor:

Total electricity consumption

$$W = W_{об} + W_{оу} \quad (5.3)$$

before the introduction of the complex:

$$W = W_{об} + W_{оу}$$

$$W = 1261.13 + 556.92 = 1818.057$$

after the introduction of the complex:

$$W' = W'_{об} + W'_{оу}$$

$$W = 1377.43 + 891.07 = 1195.069 \quad (5.4)$$

Averted economic damage to the environment:

Inflicted environmental damage

$$У_{ел} = W * У_{е} \quad (5.5)$$

$$У_{ел} = W * У_{е} = 1818{,}057 * 0{,}15 = 272{,}71$$

$$У'_{ел} = W' * У_{е} = 1195{,}069 * 0{,}15 = 179{,}26$$

As a result of using a new type of human-machine system and the transition to a network structure, the program can improve working conditions, save electricity and reduce not only the damage to the environment, but also the time spent on data processing.

Conclusions

This section discusses the impact of harmful factors in the production of an external video surveillance system on the human body that occur during the process, as well as the impact of other related equipment on the environment.

The systems, during operation, are not capable of causing direct damage to the environment because they do not form any substances of carcinogenic, toxic or other

harmful nature that affect the air, water bodies, humans, animals, vegetation or soil. However, in the manufacture of the device is indirect damage to the environment, namely:

- high level of electricity consumption;
- electromagnetic radiation;
- acoustic pollution.

# CONCLUSIONS

Unity3d is a modern cross-platform engine for creating games and applications using Unity Technologies. With this engine, you can develop not only applications for computers, but also for mobile devices (for example, based on Android), game consoles and other devices.

Let's talk a little about the characteristics of the engine. First, it is worth noting that the engine is integrated in the Unity development environment, in other words, you can test the application without leaving the editor. Secondly, Unity supports a huge number of different formats, which allows you to develop games, construct the models themselves in a more convenient application, and use Unity for its intended purpose - product development. Thirdly, scripting (scripting) is performed in the most popular programming languages - C #.

## REFERENCES

[1] Skripets A.V. Fundamentals of Aviation Engineering Psychology. Kyiv,2002.

[2] https://www.skybrary.aero/index.php/Pilot_Mental_Health

[3] https://www.coloradofirecamp.com/swiss-cheese/introduction.htm

[4] 1. Beregovoy G. T., Lomov B. F., Ponomarenko V. A. Experimental psychological studies in aviation and astronautics. M., 1978.

[5] Hryschchenko Y. V., Kravets I.V. Signaling methods about deteriorating quality of aircraft flight // Науково-технічна конференція «Проблеми розвитку глобальної системи зв'язку, навігації, спостереження та організації повітряного руху CNS/ATM», 21-23 листопада, 2018. – с. 66.

[6] Hryshchenko Y.V. Reliability problem of ergatic control systems in aviation // Methods and Systems of Navigation and Motion Control // Y.V. Hryshchenko / IEEE 4th International Conference (October 18-20, 2016) – Kyiv, Ukraine, pp. 126-129.

[7] Gryshchenko Yu.V. Preparation of pilots for flights in special situations, taking into account the phenomenon of strengthening the dynamic stereotype // Cybernetics and computer technology. K.: NAS of Ukraine, 2003. - Vip. 139. P. 81-85; UDC 629.735: 614.8