**Colegiul Național ,,Roman-Vodă”**

**Jocul Spânzurătoarea**

**Lucrare de atestat la informatică**

Candidat, Coordonator,

Dumitriu Adelina prof. Florin Moldovanu

Theodora

2015

Cuprins

[Argument 3](#_Toc418125947)

[1.Introducere în .NET și în limbajul C#. 4](#_Toc418125948)

[1.1.Despre limbajul C# şi platforma .NET 4](#_Toc418125949)

[1.2.NET Framework 5](#_Toc418125950)

[1.3.Compilarea programelor pe platforma .NET 5](#_Toc418125951)

[1.4.Principalele caracteristici ale arhitecturii .NET 6](#_Toc418125952)

[1.5.Structura unui program C# 7](#_Toc418125953)

[1.6. Sintaxa limbajului 8](#_Toc418125954)

[1.7.Cuvinte cheie in C# 9](#_Toc418125955)

[1.8.Tipuri de date 9](#_Toc418125956)

[1.9.Mediul de dezvoltare Visual C# (prezentarea interfeţei) 10](#_Toc418125957)

[1.10.Construirea interfeţei utilizator 11](#_Toc418125958)

[1.10.1.Ferestre 11](#_Toc418125959)

[1.10.2.Controale 12](#_Toc418125960)

[2.Jocul Spânzurătoarea 14](#_Toc418125961)

[2.1. Prezentare 14](#_Toc418125962)

[2.2.Construirea HangPost-ului și a corpului 15](#_Toc418125972)

[2.3. Alegerea aleatorie a unui cuvânt 16](#_Toc418125977)

[2.4. Crearea spațiilor pentru litere 16](#_Toc418125980)

[2.5.Resetarea jocului 17](#_Toc418125983)

[2.6.Implementarea butoanelor 17](#_Toc418125985)

[Bibliografie 19](#_Toc418125986)

**Argument**

Am ales această temă – *Jocul Spânzurătoarea* – deoarece am încercat să creez o aplicație interactivă, fiind un joc simplu ce se bazează pe intuiție și strategie. Am făcut posibil acest lucru folosind *Microsoft Visual Sudio Ultimate 2010* .

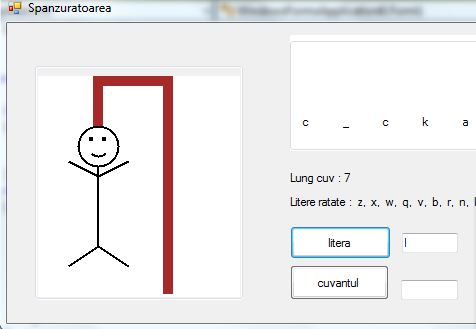
**Spânzurătoarea** este un [joc](http://ro.wikipedia.org/wiki/Joc) pentru doi jucători. Unul din jucători se gândește la un [cuvânt](http://ro.wikipedia.org/wiki/Cuv%C3%A2nt), iar celălalt încearcă să-l ghicească sugerând [litere](http://ro.wikipedia.org/wiki/Liter%C4%83).

Cuvântul ce trebuie ghicit este reprezentat de un șir de linii, fiecare linie reprezentând o literă a cuvântului. Dacă jucătorul care ghicește sugerează o literă ce se află în cuvânt, celălalt jucător o completează în toate pozițiile unde aceasta apare. Dacă litera nu se află în cuvânt, celălalt jucător desenează un element din diagrama „spânzurătoarea”.

Jocul se încheie când:

* Jucătorul care ghicește completează tot cuvântul, sau îl ghicește exact.
* Celălalt jucător completează diagrama.

Diagrama este gândită să semene cu cea a unui om aflat la spânzurătoare. Cu toate ca au apărut dezbateri privind bunul gust al imaginii, aceasta încă se folosește și astăzi.



**1.Introducere în .NET și în limbajul C#.**

**1.1.Despre limbajul C# şi platforma .NET**

Numele limbajului **C#** a fost inspirat din notaţia ♯ (*diez)* din muzică, care indică faptul că nota muzicală urmată de ♯ este mai înaltă cu un semiton. Este o similitudine cu numele limbajului C++, unde ++ reprezintă atât incrementarea unei variabile cu valoarea 1, dar şi faptul că C++ este mai mult decât limbajul C.

Limbajul C# a fost dezvoltat în cadrul Microsoft. Principalii creatori ai limbajului sunt Anders Hejlsberg, Scott Wiltamuth şi Peter Golde. Prima implementare C# larg distribuită a fost lansată de către Microsoft ca parte a iniţ iativei **.NET** în iulie 2000. Din acel moment, se poate vorbi despre o evoluţ ie spectaculoasă. Mii de programatori de C, C++ şi Java, au migrat cu uşurinţă spre C#, graţ ie asemănării acestor limbaje, dar mai ales calităţilor noului limbaj. La acest moment, C# şi-a câştigat şi atrage în continuare numeroşi adepţi, devenind unul dintre cele mai utilizate limbaje din lume.

Creatorii C# au intenţionat să înzestreze limbajul cu mai multe facilităţi. Succesul de care se bucură în prezent, confirmă calităţile sale:

* Este un limbaj de programare simplu, modern, de utilitate generală, cu productivitate mare în programare.
* Este un limbaj *orientat pe obiecte*.
* Permite dezvoltarea de aplicaţii industriale robuste, durabile.
* Oferă suport complet pentru dezvoltarea de componente software, foarte necesare de pildă în medii distribuite. De altfel, se poate caracteriza C# ca fiind nu numai *orientat obiect*, ci şi *orientat spre* *componente*.

La aceste caracteristici generale se adaugă şi alte trăsături, cum este de pildă suportul pentru internaţionalizare, adică posibilitatea de a scrie aplicaţii care pot fi adaptate cu uş urinţă pentru a fi utilizate în diferite regiuni ale lumii unde se vorbesc limbi diferite, fără să fie necesare pentru aceasta schimbări în arhitectura software.

În strânsă legatură cu ***Arhitectura .NET*** (***.NET Framework***) pe care funcţionează, C# gestioneaz ă în mod automat memoria utilizată. Eliberarea memoriei ocupate (*garbage collection*) de către obiectele care nu mai sunt necesare aplicaţ iei, este o facilitate importantă a limbajului. Programatorii nu mai trebuie să decidă singuri, aşa cum o fac de pildă în C++, care este locul şi momentul în care obiectele trebuie distruse.

În C# se scriu de asemenea aplicaţii pentru sisteme complexe care func ţionează sub o mare varietate de sisteme de operare, cât ş i pentru sisteme dedicate (*embeded systems*). Acestea din urmă se întind pe o arie largă, de la dispozitive portabile cum ar fi ceasuri digitale, telefoane mobile, MP3 playere, până la dispozitive staţionare ca semafoare de trafic, sau controlere pentru automatizarea producţiei.

Din punct de vedere sintactic C# derivă din limbajul C++, dar include şi influenţe din alte limbaje, mai ales Java.

**1.2.NET Framework**

***Arhitectura .NET*** este o componentăsoftware care oferăun mediu de programareşi de execuţie aaplicaţiilor pentru sistemele de operare Microsoft.

***.NET Framework*** este un mediu care permite dezvoltareaşi rularea aplicaţiilorşi a serviciilor Web,independente de platformă.

Limbajul C# se află într-o strânsă legatură cu arhitectura .NET. Iniţial, C# a fost dezvoltat de către Microsoft pentru crearea codului platformei .Net, la fel cum destinaţia iniţială a limbajului C a fost aceea de a implementa sistemul de operare UNIX. .NET pune la dispoziţie o colecţie impresionantă de clase organizate în biblioteci, pe care C# le utilizează.

Este momentul să precizăm că C# funcţionează având .NET ca infrastructură, dar .NET suportă şi alte limbaje, cum este C++, Visual Basic sau Java. În oricare dintre aceste limbaje programaţi, aveţi la dispoziţie aceleaşi biblioteci de clase. .NET se realizează în acest fel **interoperabilitatea limbajelor**.

**.NET** este constituit din douăentităţi importante:

* ***Common Language Runtime*** (**CLR**)

Acesta este **mediul de execuţie** al programelor. Este modulul care se ocupă cu managementul şi execuţia codului scris în limbaje specifice **.NET**. **CLR** furnizează de asemenea servicii importante, cum sunt securitatea aplicaţiilor, portabilitatea acestora, managementul memoriei şi tratarea excepţiilor.

* ***Base Class Library***

Este vorba despre *Biblioteca de Clase .NET*. Această bibliotecă acoperă o arie largă a necesităţilor de programare, incluzând interfaţa cu utilizatorul, conectarea cu bazele de date şi accesarea datelor, dezvoltarea aplicaţiilor web, comunicarea în reţele şi altele. Codul bibliotecii este precompilat, fiind încapsulat de regulă în funcţii, numite metode, pe care programatorul le poate apela din propriul program. La rândul lor, metodele aparţin claselor, iar clasele sunt organizate şi separate între ele cu ajutorul spaţiilor de nume (*namespaces*). Despre toate aceste noţiuni vom vorbi pe larg în capitolele următoare. Ceea ce trebuie reţinut pentru moment, este că programatorii combină propriul cod cu codul Bibliotecii de Clase .NET pentru producerea de aplicaţii.

**1.3.Compilarea programelor pe platforma .NET**

**Limbaje interpretate**

Când programaţi într-un limbaj de programare interpretat, veţi scrie codul şi veţ i rula aplicaţia. Cu ajutorul unui interpretor specific limbajului, fiecare linie de cod este interpretată chiar în momentul rulării ş i este preschimbată imediat în cod maşină şi executată. Partea bună este că puteţi rula codul până la primul bug, puteţi înlătura eroarea în cod iar apoi să continuaţi execuţ ia codului. Limbajul nu necesită timp de compilare şi de legare. Partea neplăcută este că limbajele interpretate sunt lente. Limbajul Basic a fost iniţial un limbaj interpretat, dar mai apoi, începând cu anii 1980, au apărut şi versiuni compilate. Marea majoritate a limbajelor de scriptare Web sunt de asemenea limbaje interpretate.

**Limbaje compilate**

Codul scris într-un asemenea limbaj, numit *cod sursă*, este translatat de către compilator într-un cod apropiat de nivelul maşinii, numit cod executabil (de exemplu codul conţinut în fi şierele (\*.exe). Dacă în timpul compil ării apar erori, atunci este necesar să le înlăturaţi, după care veţi compila din nou. Dacă aplicaţia trece acum de compilare fără erori de sintax ă, atunci se va produce codul executabil şi veţi putea să rulaţi aplicaţia. Limbajele C şi C++ sunt exemple clasice de limbaje compilate.

Din această perspectivă C# este un limbaj compilat. Dar nu în sensul descris mai sus. Ca să înţelegeţi, este necesar să ştiţi că în urma compilării unui program C#, nu se crează un cod executabil. Se creeaz ă un fişier numit ***assembly*** care de regulăse identificăcu extensia**.exe**sau**.dll**. Un asemenea fişier nu poate fi executat pe unsistem pe care nu există infrastructura **.NET**. Fişierul conţine un tip special de cod, numit ***Limbaj Intermediar***, pe scurt **CIL** (*Common Intermediate Language*). Limbajul **CIL** defineşte un set de instrucţiuni portabile, independente de orice tip de procesor şi platformă. Figura ilustrează procesul de creare a codului executabil pe platforma .NET.

În momentul în care un program este executat, **CLR** activează un compilator special, numit **JIT** (*just in time*). Acesta preia codul **CIL** şi îl transformă în cod executabil. Transformarea se face “la cerere”, în sensul că o secvenţă de cod se compileaz ă doar în momentul în care este utilizată pentru prima oară. Un program compilat în format **CIL** poate rula pe orice sistem pe care s-a instalat ***Common Language Runtime***. Aceasta asigură portabilitatea aplicaţiilor **.NET**.

Fişierul .exe sau .dll produs la compilare conţine pe lângă codul **CIL**, aşa numitele ***metadate***. Metadatele descriu datele utilizate de către aplicaţie.

**1.4.Principalele caracteristici ale arhitecturii .NET**

**Independenţa de procesor şi de platformă**

Codul **CIL** este independent de sistemul de operare şi de procesor. De aceea, în scrierea aplicaţiilor nu trebuie să fiţi preocupaţi de caracteristicile hardware sau software ale sistemului. Spre surpriza multor programatori, aplicaţiile

.NET pot fi dezvoltate şi executate pe sisteme de operare non-Microsoft, (Mac OS X, numeroase distribuţii Linux şi Solaris, ca să numim numai câteva).

**Managementul automat al memoriei**

Alocara şi eliberarea memoriei nu mai este o problemă care trebuie să-i preocupe pe programatori, datorită mecanismului automat de ***garbage collection***.

**Interoperabilitatea limbajelor**

Este un fapt comun ca diversele componente ale unei aplicaţii să fie scrise în limbaje diferite, suportate de către platforma .NET.

**Securitate**

.NET furnizează un model comun de securitate, valabil pentru toate aplicaţiile, care include un mecanism unificat de *tratare a excepţiilor* . O excepţie este un eveniment neprevăzut, care întrerupe execuţia unui program, atunci cândde pildă, se execută o intrucţiune ilegală.

**Portabilitate**

Un program scris pentru platforma .NET poate rula fără nici o modificare pe oricare sistem pe care platforma este instalată.

Caracteristicilor de mai sus li se adaugă şi altele, care ies însă din cadrul acestei lucrări.

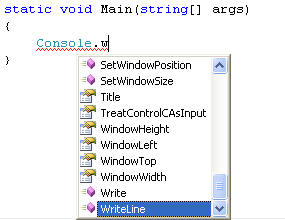
**1.5.Structura unui program C#**

Majoritatea cărţilor care tratează limbaje de programare încep cu un exemplu, devenit celebru, apărut pentru prima dată în ediţia din 1978 a cărţii „The C Programming Language” a lui Brian W. Kernighan şi Dennis M. Ritchie, „părinţii” limbajului C. Vom prezenta şi noi acest exemplu adaptat la limbajul C#:

*Codul sursă generat este :*

|  |
| --- |
| using System;    namespace HelloWorld  {  class Program  {  static void Main(string[] args)  {  Console.WriteLine("HelloWorld!");  Console.ReadLine();  }  }  } |

La scrierea codului în program sunteţi asistaţi de ***IntelliSense*** ,ajutorul contextual.



Pentru compilarea programului,selectaţi Build din meniul principal sau tasta F5(compilare rapidă cu asistenţă de depanare).

Aplicaţia de mai sus conţine :

* Linia 1 – *using System* ; este o directivă care specifică faptul că se vor folosi clase incluse în spaţiul de nume *System*.
* Linia 3 – spaţiul nostu de nume.
* Linia 5 – clasa( orice program C# este alcătuit din una sau mai multe clase).
* Linia 7 – metoda *Main*,”punctul de intrare” în program.
* Linia 9 – clasa **Console**, este folosită pentru operaţiile de intrare/ieşire.Aici se apelează metoda **WriteLine** din această clasă,pentru afişarea mesajului dorit pe ecran.

O aplicaţie C# este formată din una sau mai multe ***clase***,grupate in ***spatii de nume (namespaces)***.Este obligatoriu ca doar una din aceste clase sa conţină un „punct de intrare” (entry point), şi anume metoda (funcţia) ***Main***.

* + ***Clasa (class)*** : în termeni simplificaţi, reprezintă principalul element structural si de organizare în limbajele orientate spre obiecte,grupând date cât şi funcţii care prelucrează datele respective.
  + ***Spaţiul de nume (Namespaces)*** : din raţiuni practice, programele mari,sunt divizate in module,dezvoltate separat , de mai multe persoane. Din acest motiv,există posibilitatea de a apărea identificatori cu acelaşi nume.Pentru a evita erori furnizate din acest motiv,in 1955 limbajul C++ introduce noţiunea şi cuvântul cheie namespace.Fiecare mulţime de definiţii dintr-o librărie sau program este grupată într-un spaţiu de nume,existând astfel posibilitatea de a avea într-un program definiţii cu nume identice, dar situate in alte spaţii de nume.

**1.6.** **Sintaxa limbajului**

Ca si limbajul C++ cu care se înrudeşte, limbajul C# are un alfabet format din litere mari si mici ale alfabetului englez, cifre si alte semne. Vocabularul limbajului este format din acele „simboluri” cu semnificaţii lexicale în scrierea programelor: cuvinte (nume), expresii, separatori, delimitatori şi comentarii.

**1.7****.Cuvinte cheie in C#**

Cuvintele cheie sunt identificatori predefiniţi cu semnificaţie specială pentru compilator.

Definim in C# urmatoarele cuvinte cheie:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Abstract | As | base | bool | break |
| Byte | **case** | **catch** | **char** | **checked** |
| Class | **const** | **continue** | **decimal** | **default** |
| Delegate | **do** | **double** | **else** | **enum** |
| Event | **explicit** | **extern** | **falsee** | **finally** |
| Fixed | **float** | **for** | **foreach** | **goto** |
| If | **implicit** | **in** | **int** | **interface** |
| Internal | **is** | **lock** | **long** | **namespace** |
| New | **null** | **object** | **operator** | **out** |
| Override | **params** | **private** | **protected** | **public** |
| Readonly | **ref** | **return** | **sbyte** | **sealed** |
| Short | **sizeof** | **stackalloc** | **static** | **string** |
| Struct | **switch** | **this** | **throw** | **truee** |
| Try | **typeof** | **uint** | **ulong** | **unchecked** |
| Unsafe | **ushort** | **using** | **virtual** | **void** |
| Volatile | **while** |  |  |  |

Pentru a da semnificaţii specifice codului, în C# avem şi cuvinte cheie contextuale:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ascending | by | descending | equals | from |
| get | **group** | **into** | **join** | **Let** |
| value | **where** | **yield** |  |  |

**1.8.Tipuri de date**

În C# există două categorii de tipuri de date:

* tipuri valoare
  + tipul simplu predefinit: byte, char, int, float etc.
  + tipul enumerare – enum
  + tipul structură - struct
* tipuri referinţă
  + tipul clasă – class
  + tipul interfaţă – interface
  + tipul delegat – delegate
  + tipul tablou - array

**1.9.****Mediul de dezvoltare Visual C# (prezentarea interfeţei)**

Mediul de dezvoltare Microsoft Visual C# dispune de instrumente specializate de proiectare, ceea ce permite crearea aplicaţiilor în mod interactiv, rapid şi uşor.

Pentru a construi o aplicaţie Windows ( File 🡪 New Project) se selectează ca template ***Windows Forms Application***.

O aplicaţie Windows conţine cel puţin o fereastra (***Form***) în care se poate crea o interfaţă cu utilizatorul aplicaţiei.

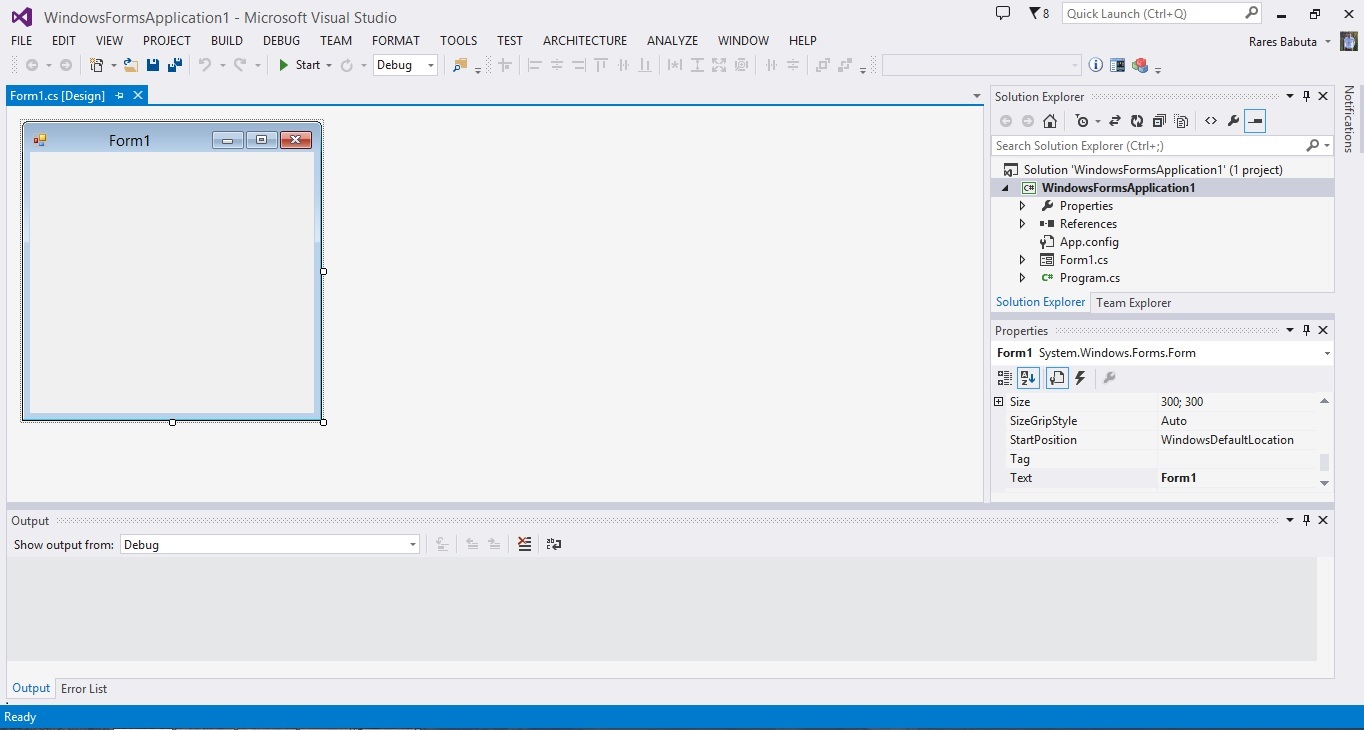
Componentele vizuale ale aplicaţiei pot fi prelucrate în modul ***Designer*** pentru a plasa noi obiecte, a le stabili proprietăţile etc. Codul „din spatele” unei componente vizuale este accesibil în modul ***Code*** (F7) .

În fereastra ***Solution Explorer*** sunt afişate toate fişierele pe care Microsoft Visual C# 2013 le-a inclus în proiect. ***Form1.cs*** este formularul creat implicit ca parte a proiectului.

Fişierul Form1.cs conţine un formular (fereastra Form1 derivata din clasa Form) care este reprezentată în cadrul din dreapta în formatul Design (Form1.cs [Design], adică într-un format în care se poate excuta proiectare vizuală,prin inserarea controalelor necesare selectate din fereastra ***ToolBox*** care se activeaza atunci cand este „atinsă” cu mouse-ul.

Fişierul Form1.cs poate fi văzut ca fişier text sursă prin selectarea lui în fereastra Solution Explorer, click dreapta cu mouse-ul şi selecţia opţiunii *View Code.* ***Fereastra Properties*** (Ctrl+W,P) este utilizată pentru a schimba proprietăţile obiectelor*.* ***ToolBox*** (Ctrl+W,X) conţine controale standard *drag-and-drop* şi componente utilizate în crearea aplicaţiei Windows. Controalele sunt grupate în categoriile logice din imaginea alăturată.Ferestrele care sunt afişate în fereastra principală se pot stabili prin selecţie din meniul View.

La creare unei noi aplicaţii vizuale, Microsoft Visual C# 2013 genereaza un spaţiu de nume care conţine clasa statică ***Program.cs*** ,cu metoda statică ce constituie punctul de intrare (de lansare) a aplicaţiei.



static void Main()

{

...

Application.Run(new Form1());

}

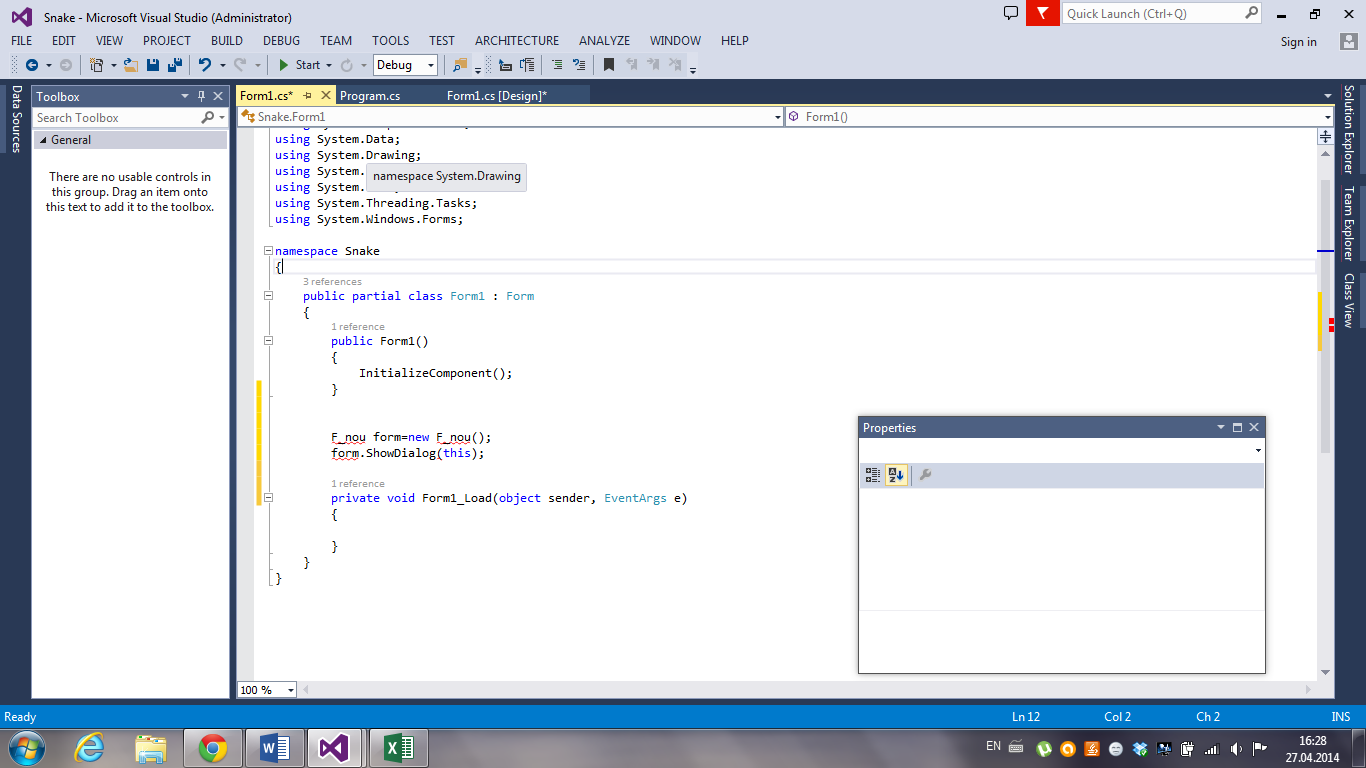
Clasa Application este responsabilă cu administrarea unei aplicaţii Windows, punând la dispoziţie proprietăţi pentru a obţine informaţii despre aplicaţii, metode de lucru cu aplicaţia şi altele.Toate metodele şi proprietăţile clasei Application sunt statice.

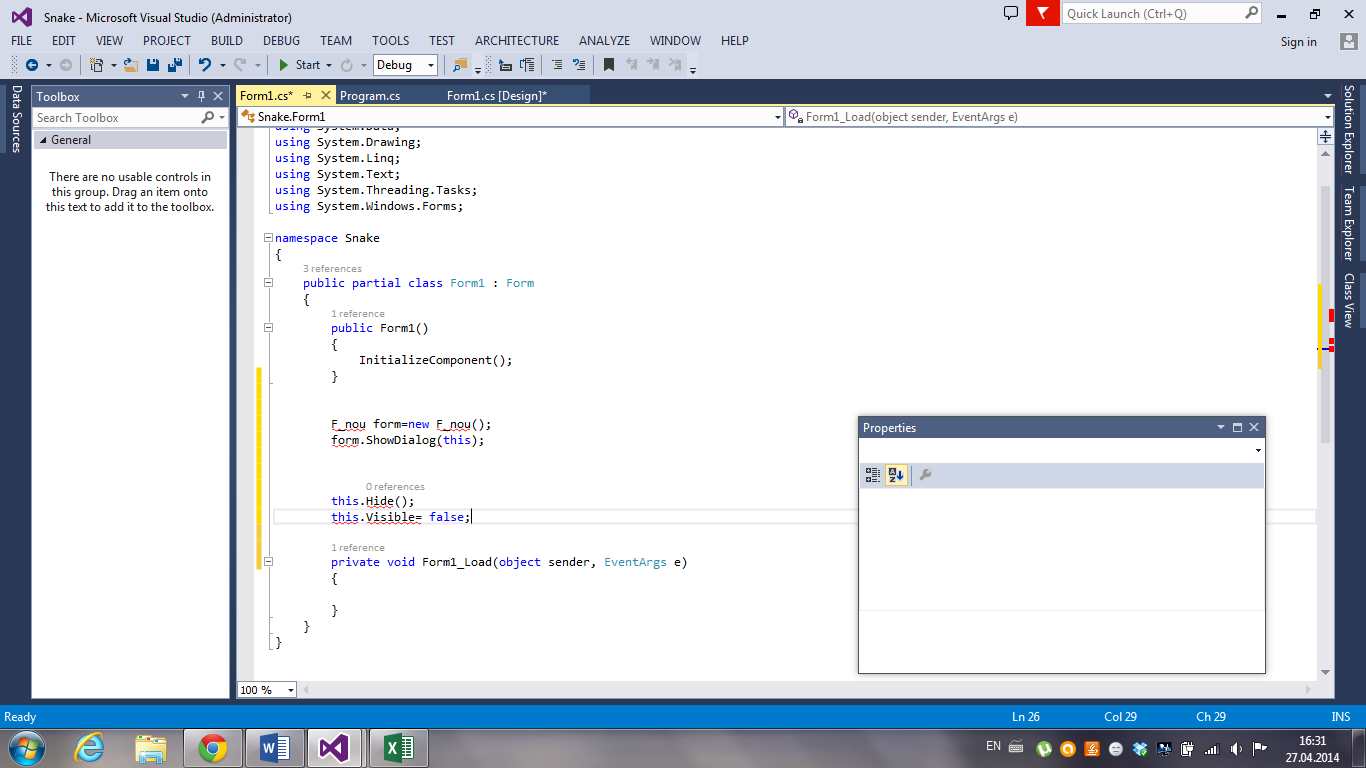
**1.10.Construirea interfeţei utilizator**

**1.10.1.Ferestre**

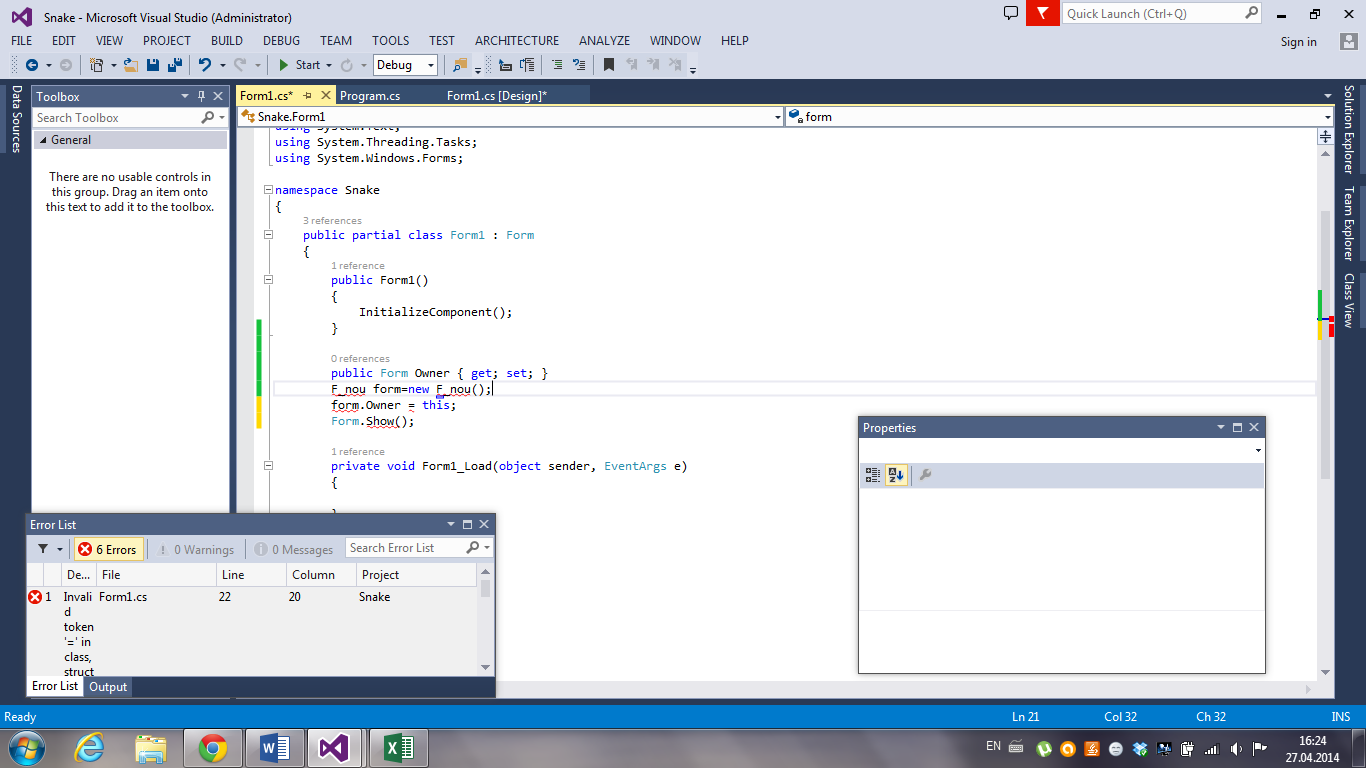
Spaţiul Forms ne oferă clase specializate pentru : creare de ferestre sau formulare (***System.Windows.Forms.Form***), elemente specifice (controale) cum ar fi butoane (***System.Windows.Forms.Button***) casete de text ( ***System.Windows.Forms.TextBox***) etc.

Proiectarea unei ferestre are la bază un cod complex, generat automat pe masură ce noi desemnăm componentele şi comportamentul acesteia. În fapt,acest cod realizează : derivarea unei clase proprii din ***System.Windows.Forms.Form***, clasă care este înzestrată cu o colecţie de controale (iniţial vidă). Constructorul ferestrei realizează instanţieri ale claselor ***Button, MenuStrip, Timer*** etc. (orice plasăm noi în fereastră) şi adaugă referinţele acestor obiecte la colecţia de controale ale ferestrei.

Daca modelul de fereastră reprezintă fereastra principala a aplicaţiei ,atunci ea este instanţiată automat în programul principal ( metoda ***Main***). Dacă nu,trebuie să scriem noi codul care realizează instanţierea.

Clasele derivate din Form ***moştenesc*** (inheritance) o serie de proprietăţi care determină atributele vizuale ale ferestrei ( stilul marginilor,culoare de fundal etc), metode care implementeză anumite comportamente (***Show,Hide,Focus***,etc) şi o serie de metode specifice (handlere) de tratare a evenimentelor (***Load,Click*** etc).

O fereastra poate fi activată cu ***form.Show()*** sau ***cu form.ShowDialog()***, metoda a doua permiţând ca revenirea în fereastra din care a fost activat noul formular să se faca numai după ce noul formular a fost închis (spunem că formularul nou este deschis modal).

Un proprietat este o fereastra care contribuie la comportarea formularului deţinut. Activarea proprietarului unui formular deschis modal va determina activarea formularului deschis modal. Când un nou formular este activat folosind ***form.Show()*** nu va avea nici un deţinător, acesta stabilindu-se direct:

Formularul deschis modal va avea un proprietar setat pe NULL.Deţinătorul se poate stabili setând proprietarul înainte să apelăm Form.Show.Dialog() sau apelând Form.Show.Dialog() cu proprietarul ca argument.

Vizibilitatea unui formular poate fi setată folosind metodele ***Hide*** sau ***Show***. Pentru a ascunde un formular putem folosi :

**1.10.2.Controale**

Unitatea de bază a unei interfeţe Windows o reprezintă un control ,care poate fi „găzduit” de un container ce poate fi un formular sau un alt control.

Un control este o instanţă a unei clase derivate din ***System.Windows.Forms*** şi este responsabil cu desenarea unei părţi din container.Visual Studio.NET vine cu o serie de controale standard,disponibile în ToolBox.

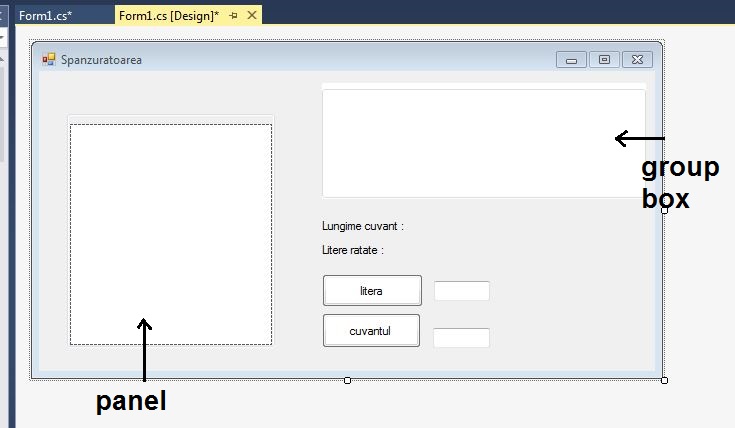
În tabelul de mai jos veţi găsi o listă cu controalele cel mai des folosite şi descrierea lor.Exemple de folosire a acestor controale vor urma după explicarea proprietăţilor comune ale controalelor şi formularelor.

|  |  |
| --- | --- |
| Button | Sunt folosite pentru a executa o secvenţă de instrucţiuni în momentul activării lor de către utilizator |
| MonthCalendar | **Afişează implicit un mic calendar al lunii curente.Acesta poate fi derulat şi înainte şi înapoi** |
| CheckBox | **Oferă utilizatorului opţiunile :da/nu sau include/exclude** |
| Label | **Sunt folosite pentru afişarea etichetelor de text,şi pentru a eticheta controalele** |
| Listbox | **Afişeaya o listă de articole din care utilizatorul poate alege** |
| PictureBox | **Este folosit pentru adăugarea imaginilor** |
| Pointer | **Este utilizat pentru selectarea ,mutarea sau redimensionarea unui control** |
| RadioButton | **Este folosit pentru ca utilizatorul să selecteze un singur element dintr-un grup de selecţii** |
| TextBox | **Este utilizat pentru afişarea textului generat de o aplicaţie sau pentru a primi datele introduse de la tastatura de catre un utilizator** |

**2.Jocul Spânzurătoarea**

**2.1. Prezentare**

Aplicația are la bază un singur form :



În acesta vom crea :

* un panel, în care vom desene HangPostul ( schema de care va atârna caricatura noastră)
* un group box, în care vor apărea spațiile care trebuie completate cu litere
* 2 butoane : unul pentru a insera in spațiile din group box o litera și unul pentru a insera un cuvânt
* 2 text box-uri : unul pentru a introduce o literă si unul pentru a introduce un cuvânt
* 2 labeluri, în care vom monitoriza literele introduse de noi de la tastatură, dar care nu se găsesc în cuvânt și lungimea cuvântului.

Pentru acest joc avem nevoie de variabila word(în care vom memora un cuvânt ales aleatoriu) și nr (în care vom număra câte părți ale corpului au fost desenate) si de lista litere (cu ajutorul căreia vom crea liniuțele care maschează cuvântul, dar care totodată ne permite să înlocuim o liniuță cu litera ghicită).

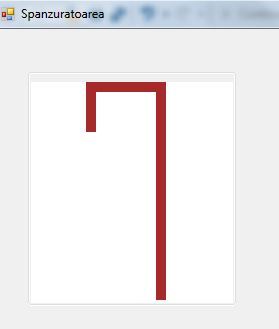
private string word = string.Empty;

List<Label> litere = new List<Label>();

private int nr;

**2.2.Construirea HangPost-ului și a corpului**

Pentru a desena HangPost-ul, creăm funcția DrawHangPost( ) în care vom crea în panelul nostru 3 linii, care semnifica schema de care va atârna „persoana” spânzurată în joc.



void DrawHangPost()

{

Graphics g = panel1.CreateGraphics();

Pen p = new Pen(Color.Brown, 10);

g.DrawLine(p, new Point(130, 218), new Point(130, 5));

g.DrawLine(p, new Point(135, 5), new Point(65, 5));

g.DrawLine(p, new Point(60, 0), new Point(60, 50));

}

Vom declara o enumerare a părților corpului, pe care o vom folosi într-o funcție DeseneazaCorp( ), care pe măsură ce înaintăm în joc și greșim litere, va desena pe rând câte o parte din corpul caricaturii noastre.

enum BodyParts

{

head,

left\_eye,

right\_eye,

mouth,

body,

right\_arm,

left\_arm,

right\_leg,

left\_leg

}

void DeseneazaCorp(BodyParts bp)

{

Graphics g = panel1.CreateGraphics();

Pen p = new Pen(Color.Black, 2);

if(bp==BodyParts.head)

{

g.DrawEllipse(p, 40, 50, 40, 40);

}

else if(bp==BodyParts.left\_eye)

{

SolidBrush s = new SolidBrush(Color.Black);

g.FillEllipse(s, 50, 60, 5, 5);

}

else if (bp == BodyParts.right\_eye)

{

SolidBrush s = new SolidBrush(Color.Black);

g.FillEllipse(s, 63, 60, 5, 5);

}

else if (bp == BodyParts.mouth) g.DrawArc(p, 50, 60, 20, 20, 45, 90);

else if (bp == BodyParts.body) g.DrawLine(p, new Point(60, 90), new Point(60, 170));

else if (bp == BodyParts.left\_arm)

{g.DrawLine(p, new Point(60, 100), new Point(30, 85));}

else if (bp == BodyParts.right\_arm)

{g.DrawLine(p, new Point(60, 100), new Point(90, 85));}

else if (bp == BodyParts.left\_leg)

{g.DrawLine(p, new Point(60, 170), new Point(30, 190));}

else if (bp == BodyParts.right\_leg)

{g.DrawLine(p, new Point(60, 170), new Point(90, 190));}

}

**2.3. Alegerea aleatorie a unui cuvânt**

Aceast pas va fi realizat cu ajutorul funcției GetRandomWord( ) care descarcă de pe Internet o listă cu aproximativ 80 000 cuvinte în engleză și returnează un cuvânt aleatoriu utilizând funcția Random .

string GetRandomWord()

{

WebClient wc = new WebClient();

string wordList = wc.DownloadString("http://www.puzzlers.org/pub/wordlists/ospd.txt");

string[] words = wordList.Split('\n');

Random ran = new Random();

return words[ran.Next(0, words.Length - 1)];

}

**2.4. Crearea spațiilor pentru litere**

Aici, vom crea funcția MakeLabels( ), în care cu ajutorul listei precizate anterior vom reuși să creăm câte un underscore care va fi înlocuit de litera ghicită. Și tot aici, calculăm lungimea cuvântului, ca să ne dăm seama cum împărțim spațiul din group box astfel încât underscorurile să nu aibă o distanță inegală între ele.

void MakeLabels()

{

word = GetRandomWord();

char[] chars = word.ToCharArray();

int between = 330 / chars.Length - 1;

for(int i=0;i<chars.Length-1;i++)

{

litere.Add(new Label());

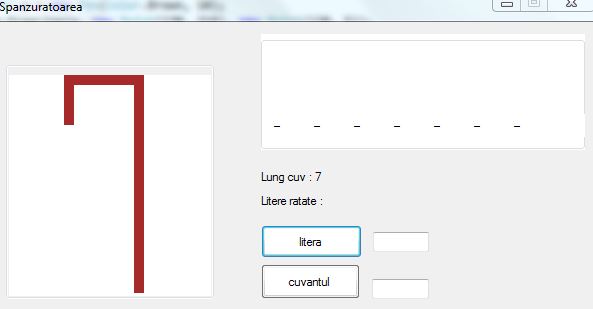
litere[i].Location=new Point((i\*between)+10,80);

litere[i].Text = "\_";

litere[i].Parent = groupBox1;

litere[i].BringToFront();

litere[i].CreateControl();

 }

label1.Text = "Lung cuv : " + (chars.Length - 1).ToString();

}

**2.5.Resetarea jocului**

Resetarea jocului constă in ștergerea elementelor desenate în panel, alegerea unui nou cuvânt, desenarea underscorurilor si a hangpostului.

void ResetGame()

{

Graphics g = panel1.CreateGraphics();

g.Clear(panel1.BackColor);

GetRandomWord();

MakeLabels();

DrawHangPost();

label2.Text = "Litere ratate : ";

textBox1.Text = "";

}

**2.6.Implementarea butoanelor**

Am configurat două butoane, unul pentru a introduce câte o literă și celălalt pentru a introduce un cuvânt. Așadar, prin intermediul butonului 1 putem introduce o literă de la tastatură. Dacă aceasta se regăsește în cuvânt, toate underscorurile care îi corespund, vor fi înlocuite de litera respectivă. În caz contrat, contorul nr va crește cu 1 și se va desena o parte din corp. Când contorul ajunge la 9, jocul este declarat necâștigător, iar cuvântul ne este afișat într-un mesaj. Prin intermediul butonului 2, putem ghici întreg cuvântul, caz în care jocul este declar ca fiind câștigător, iar în caz contrat, ni se va desena încă o parte din corp.

private void button1\_Click(object sender, EventArgs e)

{

char letter = textBox1.Text.ToLower().ToCharArray()[0];

if(!char.IsLetter(letter))

{

MessageBox.Show("Introduceți doar litere!!");

return;

}

if(word.Contains(letter))

{

char[] letters = word.ToCharArray();

for(int i=0;i<letters.Length;i++)

{

if (letters[i] == letter) litere[i].Text = letter.ToString();

textBox1.Text = "";

}

foreach(Label l in litere)

if (l.Text == "\_") return;

MessageBox.Show("Bravo! Ai câștigat!");

ResetGame();

}

else

{

label2.Text += " " + letter.ToString() + ", ";

DeseneazaCorp((BodyParts)nr);

nr++;

if(nr==9)

{

MessageBox.Show("Ai pierdut. Cuvântul era : " + word);

ResetGame();

}

}

textBox1.Text = "";

}

private void button2\_Click(object sender, EventArgs e)

{

if(textBox2.Text==word)

{

MessageBox.Show("Bravo! Ai câștigat!");

ResetGame();

}

else

{

DeseneazaCorp((BodyParts)nr);

nr++;

if(nr==9)

{

MessageBox.Show("Ai pierdut. Cuvântul era : " + word);

ResetGame();

}

}

textBox2.Text = "";

}

**Bibliografie**

* [**http://www.cs.ubbcluj.ro/~vcioban/Bistrita/Manuale/Manual\_POO\_si\_Vizuala.pdf**](http://www.cs.ubbcluj.ro/~vcioban/Bistrita/Manuale/Manual_POO_si_Vizuala.pdf)
* [**http://www.math.uaic.ro/~cgales/csharp/Curs1.pdf**](http://www.math.uaic.ro/~cgales/csharp/Curs1.pdf)
* https://www.youtube.com/watch?v=HP-88eu-QQw