Adrian NIȚĂ Maria NIȚĂ

DEZVOLTAREA JOCURILOR VIDEO

SUPORT DE CURS
APLICAȚIE PRACTICĂ

2024

Dezvoltarea jocurilor video

Grupul de lucru, coordonat de **Livia Demetra Țoca**, consilier superior la **CNPEE** (Centrul Național de Politici și Evaluare în Educație), a fost format din:

- **Adrian Modrișan** — profesor de informatică la Colegiul Național „Andrei Șaguna”, Brașov;
- **Adrian Niță** — profesor de informatică la Colegiul Național „Emanuil Gojdu”, Oradea;
- **Maria Niță** — profesor de informatică la Colegiul Național „Emanuil Gojdu”, Oradea;
- **Roxana Gabriela Tîmplaru** — profesor de informatică la Colegiul „Ștefan Odobleja”, Craiova.

Document conceput cu sprijinul **RGDA** (Romanian Game Developers Association) — organizație nonprofit.
Grupul de colaboratori din partea RGDA, coordonat de
Andreea Medvedovici-Per, director executiv al asociației RGDA (Romanian Game Developers Association), a fost format din:

- **Alexandru Chica**, Studio Technical Director, Electronic Arts Romania;
- **Dan Teodorescu**, Game Director, Fortis Games Romania.

„Dezvoltarea jocurilor video”, aprobată prin Ordinul Ministerul Educației
Nr. 7233 din 10 octombrie 2024, publicat în Monitorul Oficial al României
Nr. 1083/30.X.2024

Cuprins

Dezvoltarea jocurilor video

Aplicație practică

În această parte a Suportului de curs, vom prezenta modalitatea prin care putem crea un joc, folosind mediul de dezvoltare integrat (**IDE**) **Visual Studio 2022 Community** (gratuit pentru învățământ), dezvoltat de **Microsoft** și limbajul **Visual C#**.

1. Visual Studio

1.1. De ce am ales Visual Studio?

Visual Studio este unul dintre cele mai complete **IDE**-uri pentru dezvoltarea jocurilor datorită integrării cu motoare de joc populare, a instrumentelor avansate de debugging și optimizare, a suportului pentru colaborare și dezvoltare multiplatformă.

1.2. De ce vom folosi limbajul Visual C#?

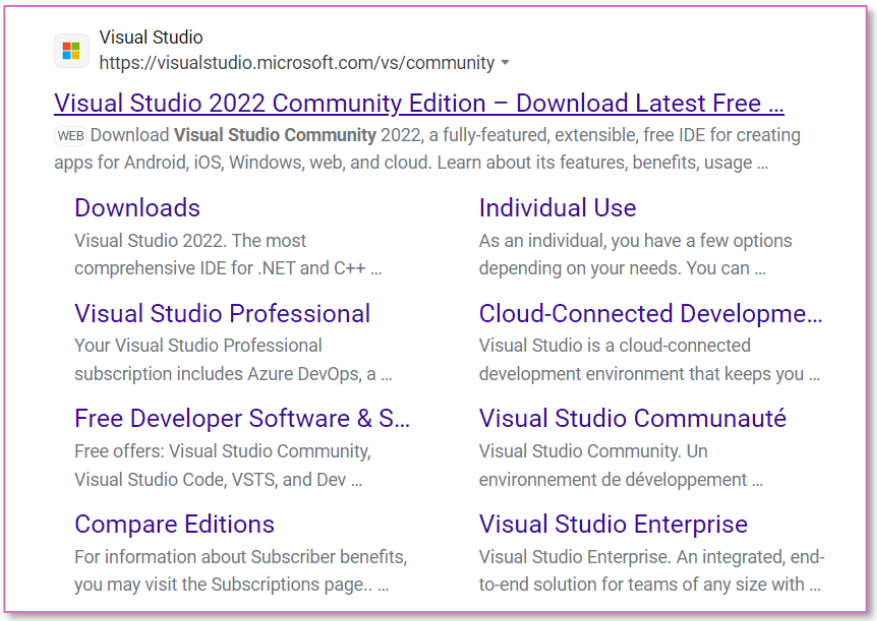
În primul rând folosirea limbajului **Visual C#** vine în mod firesc după experiența pe care elevii au avut-o cu limbajul **C++**. Folosirea limbajului **Visual C#** în **Visual Studio** pentru dezvoltarea jocurilor oferă un echilibru între productivitate, performanță și ușurința în utilizare. Chiar dacă, în ceea ce urmează, nu vom folosi toate avantajele folosirii acestui limbaj, vom aminti:

- combinația dintre **Visual C#** și **Unity** face dezvoltarea jocurilor mult mai ușoară și accesibilă
- echilibru bun între putere și ușurința de utilizare, fiind mai puțin complex în gestionarea memoriei comparativ cu **C++**
- funcționalități pentru dezvoltarea de jocuri dinamice, cu multiple procese paralele, precum în cazul sistemelor de inteligență artificială, fizică sau randare grafică.
- gestionarea automată a memoriei prin **Garbage Collector**

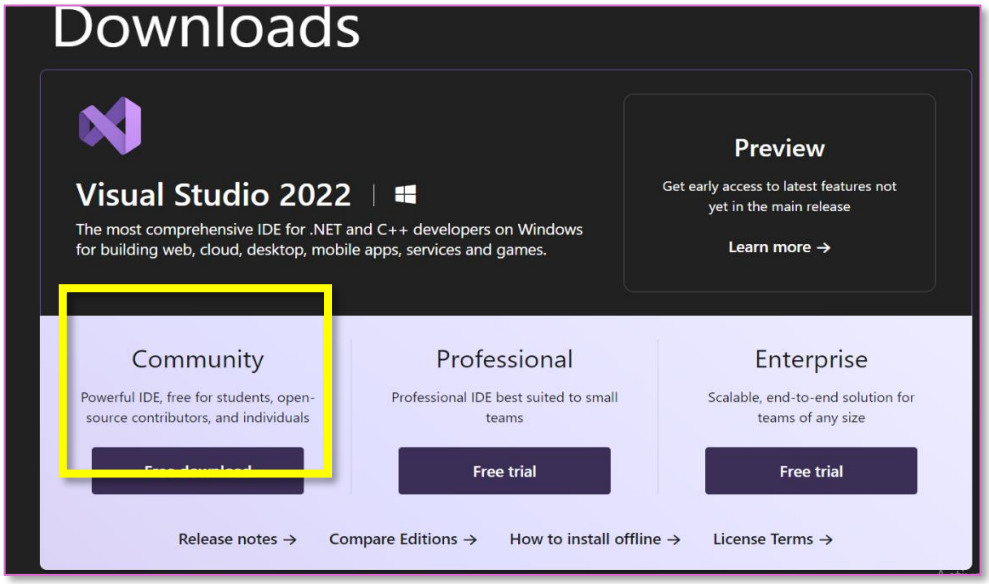
- Visual Studio, în combinație cu **C#**, oferă instrumente puternice de dezvoltare, cum ar fi **IntelliSense**, **debugging avansat**, **profiling** și **refactoring** de cod
- **Limbajul C#** permite dezvoltarea **multiplatformă**. În **Unity**, folosind **C#**, se pot dezvolta jocuri care rulează pe diverse platforme, inclusiv **Windows, macOS, Android, iOS, PlayStation, Xbox**

1.3. Cum instalăm Visual C# Community pe calculatorul nostru?

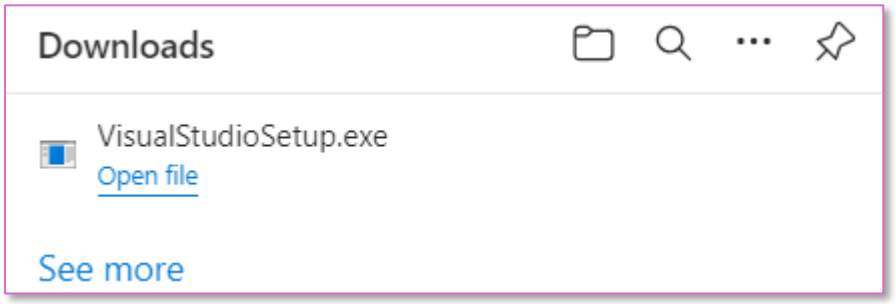
Căutăm pe Internet „Visual Studio Community”. Alegem Community (gratuit pentru învățământ). Obținem:



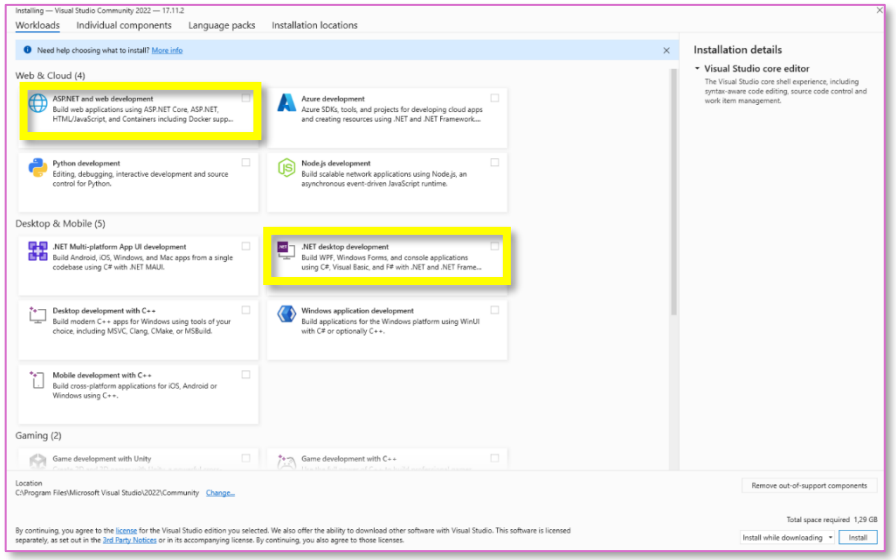
prin selectarea **Free download** de la **Community**



Se descarcă fișierul **VisualStudioSetup.exe** care va permite instalarea;

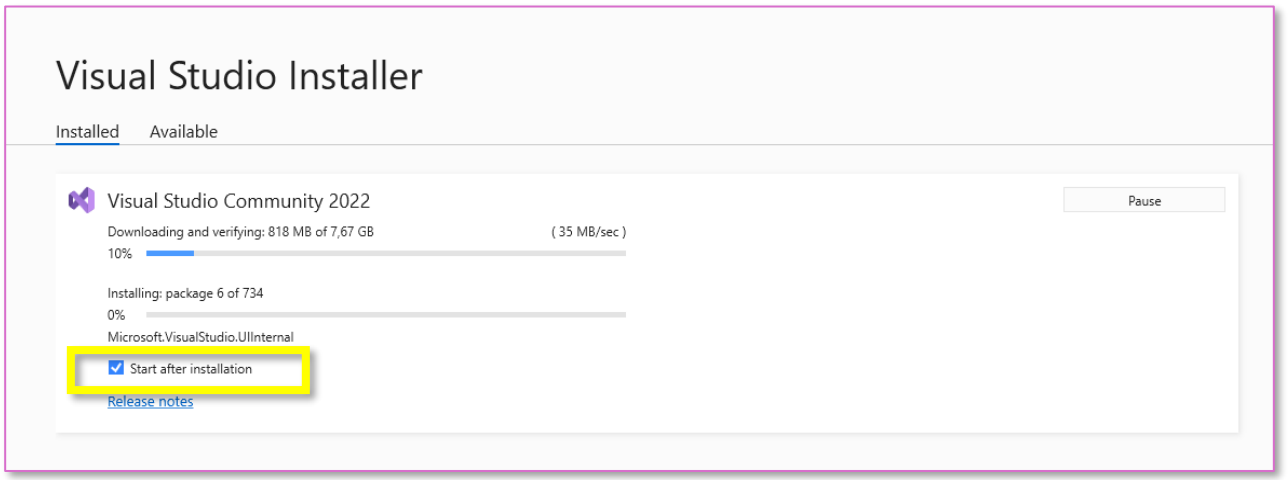


Installer-ul va cere utilizatorului să selecteze modulele pe care dorește să le instaleze. Alegem **ASP.NET and web developement**, **.NET desktop developement**, **Game developement with Unity**



În colțul din dreapta, jos, al ferestrei se menționează spațiul total necesar, pentru instalarea modulelor selectate.

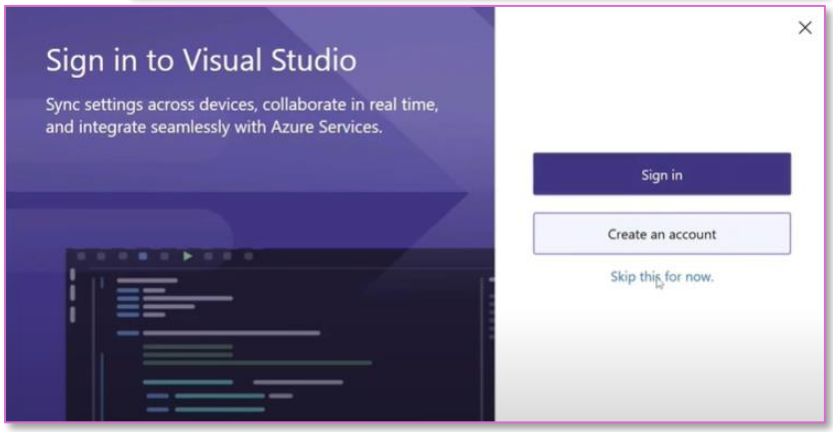
*Prin acționarea butonului **Install**, **Installer**-ul începe descărcarea și instalarea **Visual Studio Community 2022**.*



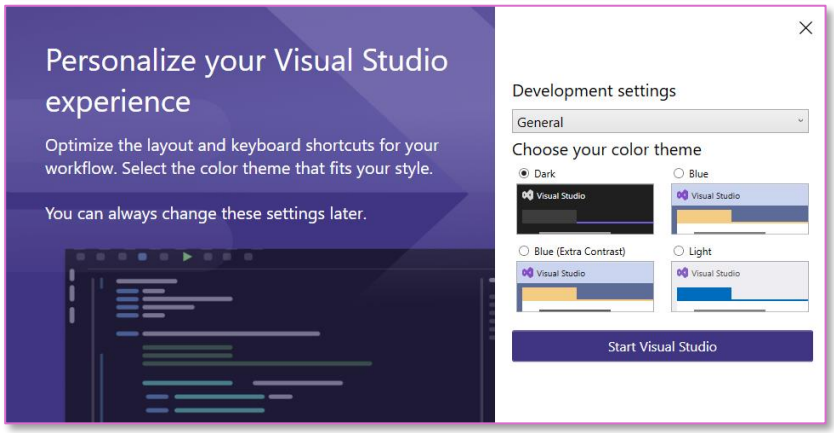
În cazul în care dorim ca **Visual Studio**-ul să pornească imediat după instalare, putem bifa (implicit) **Start after instalation**.

După instalare, lansăm/se lansează automat **Visual Studio Community 2022**:

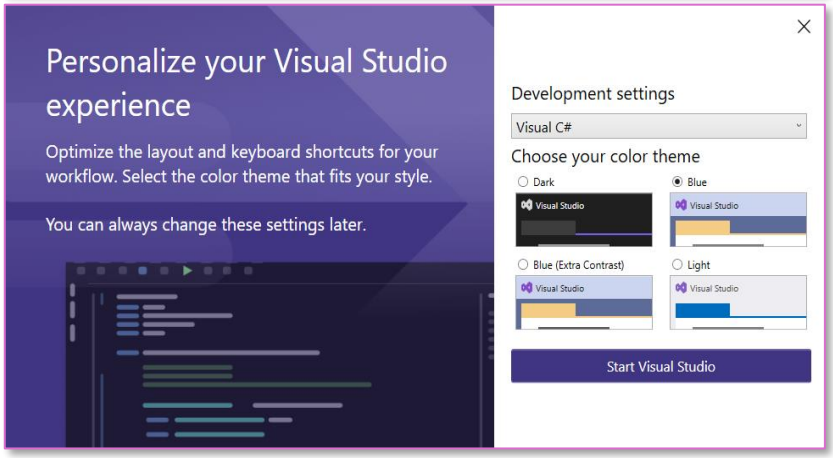
Avem posibilitatea de a ne conecta prin termediul unui cont Microsoft (dacă îl avem deja), fie a ne crea un cont sau a trece peste conectarea prin intermediul unui cont.



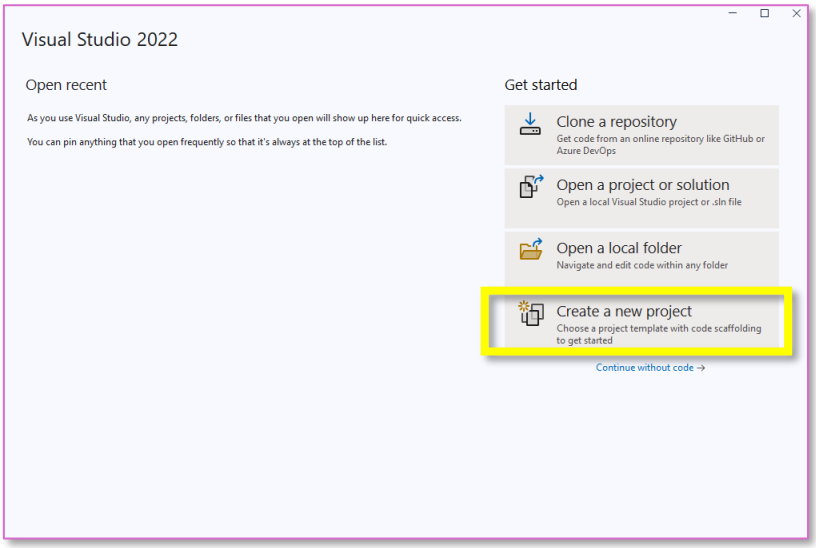
Pasul următor constă în personalizarea **Visual Studio**-ul pe care l-am instalat



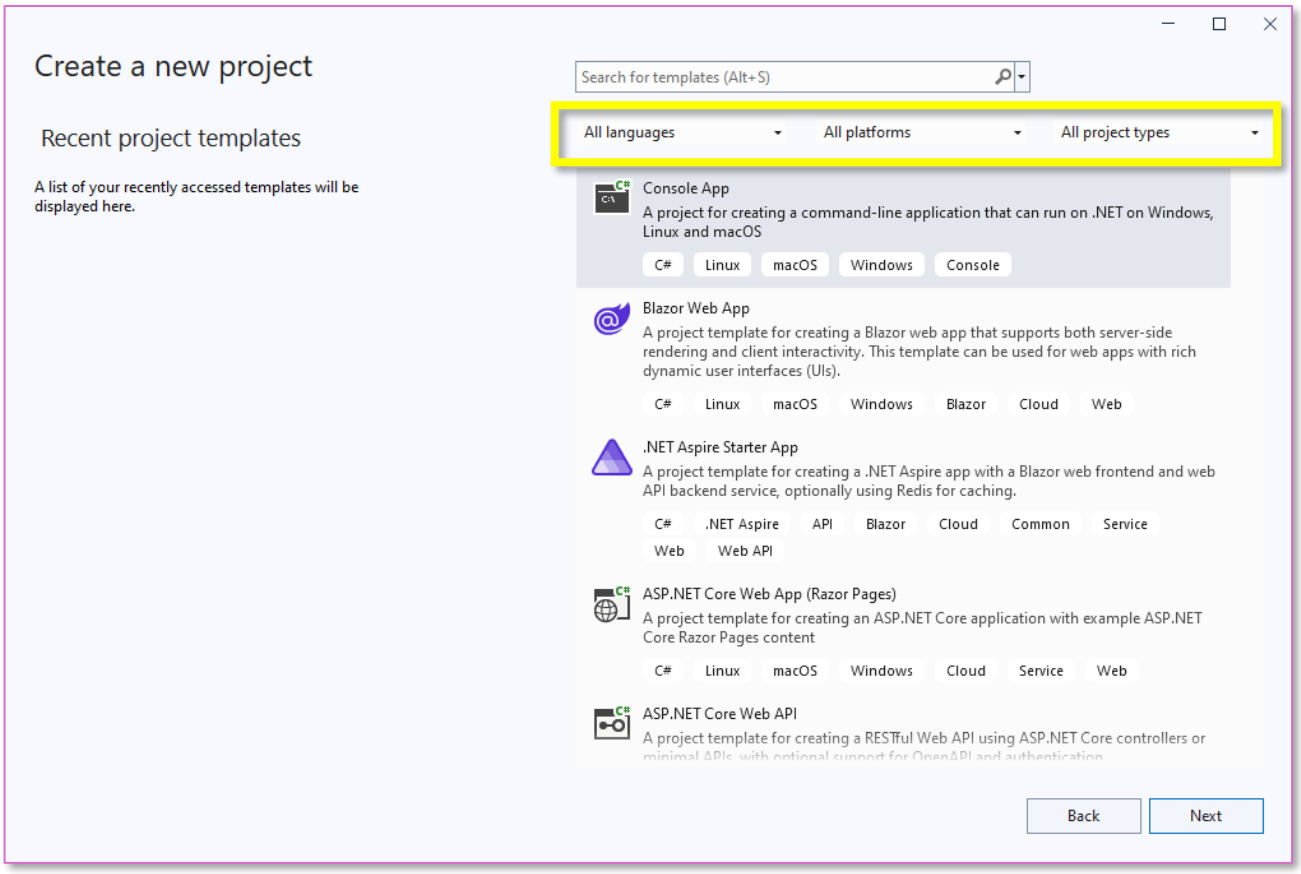
Putem să ne alegem limbajul pe care îl vom folosi în proiectele noastre, precum și schema de culori a interfeței lui **Visual Studio**. După aceste setări, apăsăm pe butonul **Start Visual Studio**.



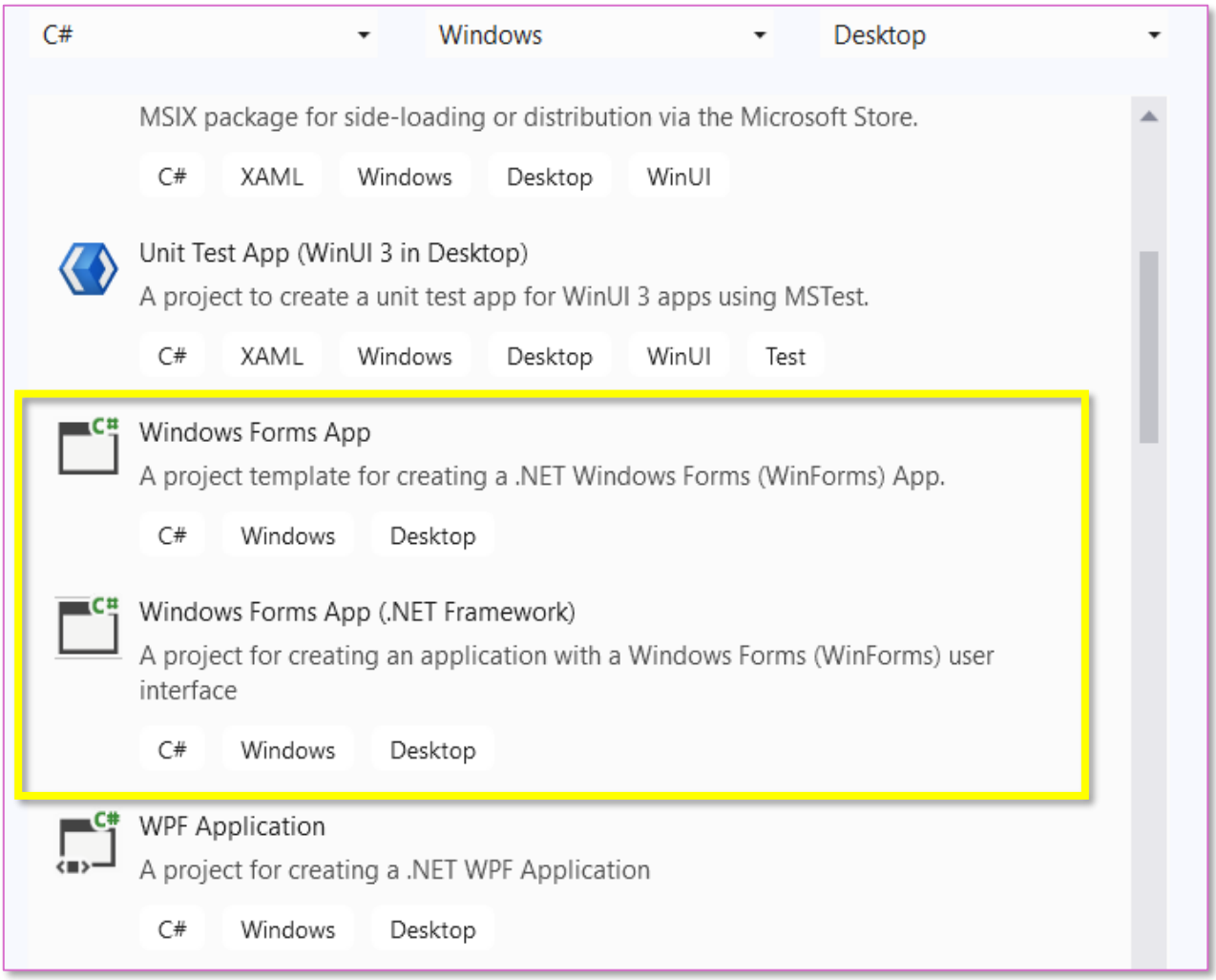
Se deschide fereastra a cărei imagine o vedem alăturat și selectăm butonul **Create a new project**



Din fereastra care se deschide selectăm tipul de proiect pe care dorim să îl realizăm:



Selectăm limbajul pe care îl vom folosi – **C#**
Platforma pentru care realizăm aplicația – **Windows**
Ținta proiectului - **Desktop**



Se pune problema selectării tipului de aplicație pe care dorim să o dezvoltăm:

- **Windows Forms App**
- **Windows Forms App (.Net Framework)**

1.4. Ce este .NET Framework?

.NET Framework este o platformă de dezvoltare software creată de **Microsoft**, lansată în 2002 Aceasta permite dezvoltarea și rularea de aplicații pentru **Windows**. **.NET Framework** este o colecție de tehnologii și biblioteci menită să ajute programatorii să creeze aplicații desktop, web și servicii, folosind diferite limbaje de programare precum **C#, VB.NET**, și **F#**.

Utilizatorii trebuie să știe că **.NET Framework** este **limitat la Windows** și nu mai primește actualizări majore sau noi funcționalități. Dezvoltarea activă a platformei s-a mutat către **.NET Core** și, ulterior, la **.NET 5/6+**, care sunt cross-platform și oferă mai multe funcționalități moderne.

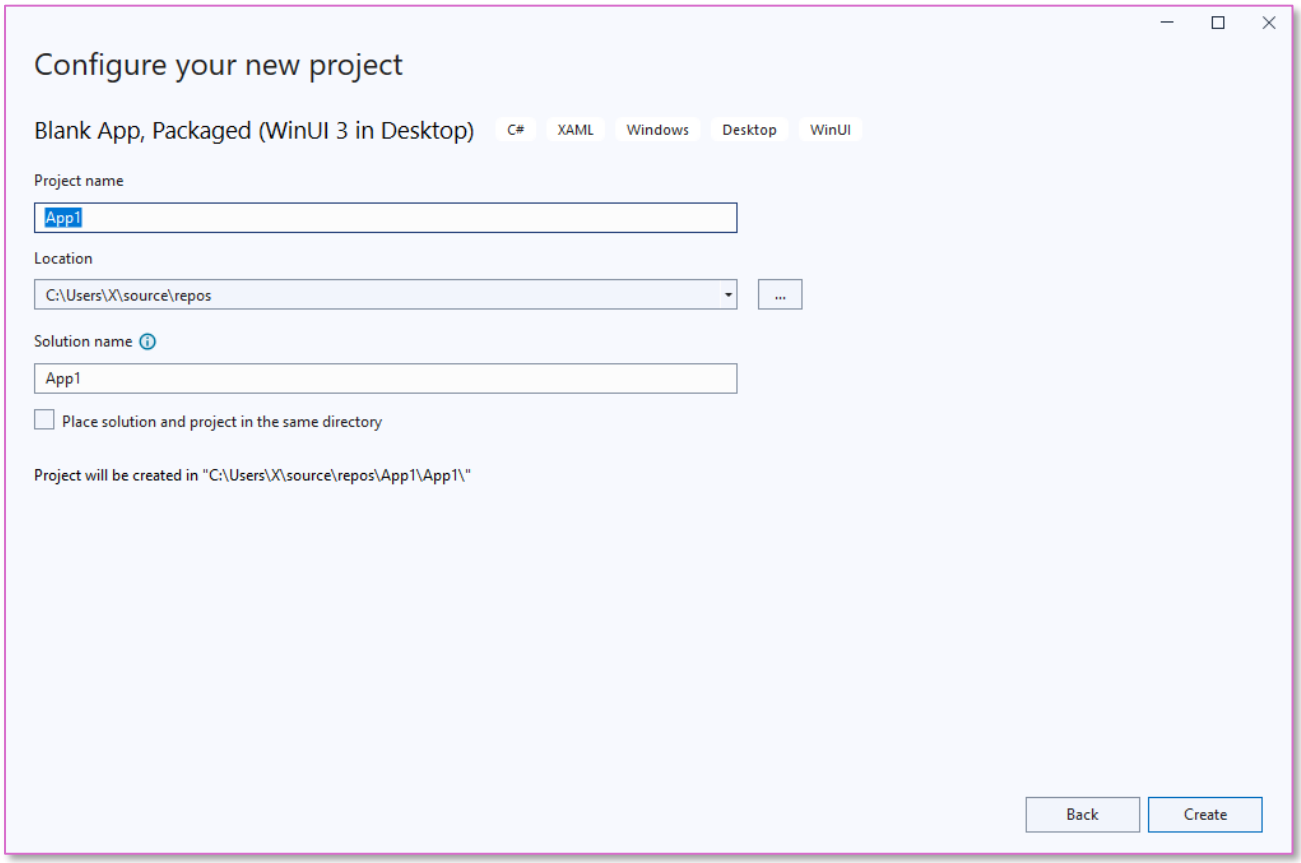
1.5. Concluzie

Dacă dorim să realizăm un proiect nou și modern, **.NET Core** sau **.NET 5+** vom selecta **Windows Forms App**. Dacă trebuie să întreținem o aplicație existentă sau să folosim tehnologii mai vechi, atunci selectăm **Windows Forms App (.NET Framework)**.

2. Dezvoltarea aplicației

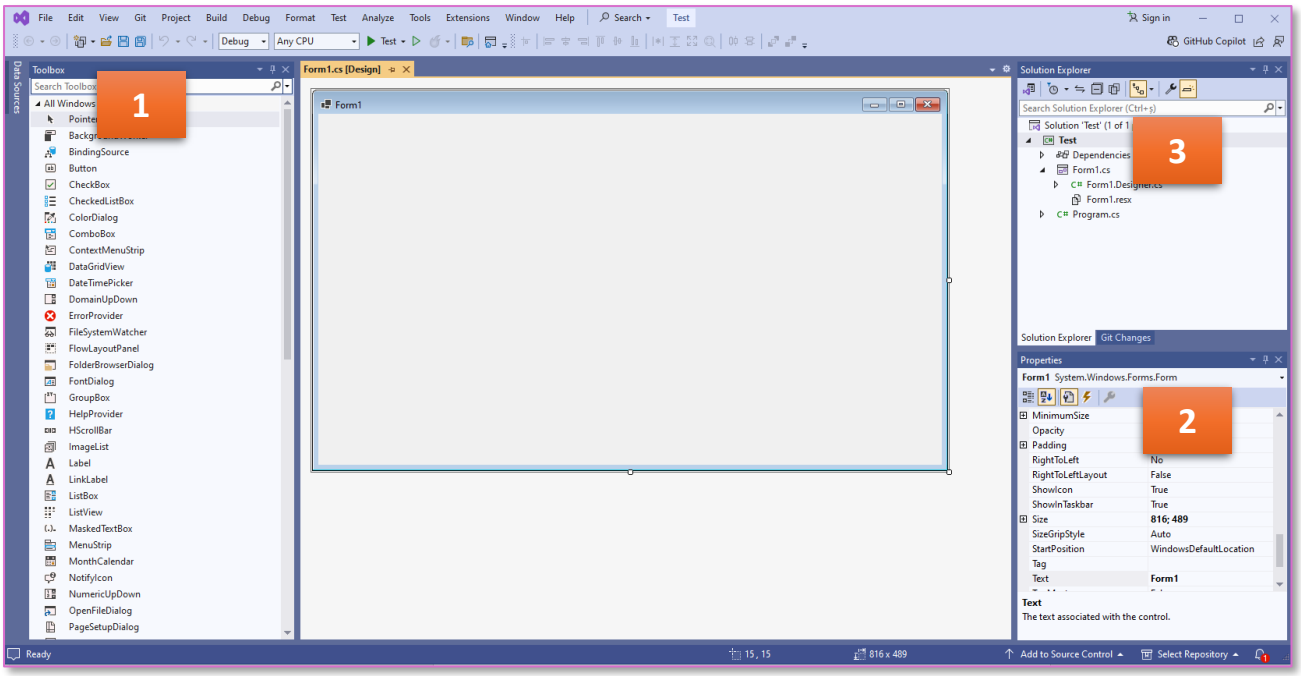
2.1. Configurarea proiectului

Selectăm **Windows Forms App**
În fereastra care se deschide trebuie să facem câteva setări:



Project name – numele proiectului nostru (**Albastru_01**)
Location – selectăm locația în care se va crea/salva proiectul

După completarea respectivelor cerințe, executăm clic pe butonul **Create**



Pentru a putea lucra la proiectul nostru, trebuie să avem deschise panourile:
1. Toolbox,
2. Properties
3. Solution Explorer.
În cazul în care unul dintre panouri nu este afișat, îl vom activa din fila **View**.

2.2. Scenariul

Într-un univers albastru, în care nu există decât obiecte de formă dreptunghiulară / pătrată, un extraterestru, evident de forma unui pătrat albastru, încearcă să culeagă bănuți pentru a-și îmbunătăți nivelul de trai. În acest univers există o excepție de la forma dreptunghiulară / pătrată: ochii extraterestrului care sunt rotunzi. Bănuții se află plasați pe mai multe platforme, aflate la înățimi diferite, iar ca să ajungă la ei, extraterestrul trebuie să execute salturi pe aceste platforme. Ajutați extraterestrul să culeagă toți bănuții de care are nevoie!

2.3. Mecanica jocului

Toate elementele care intervin sunt de tip **PictureBox**, pentru a lăsa posibilitatea programatorului să-și particularezeze proiectul prin folosirea de imagini pentru jucător, platforme, bănuți și fundal.

Personajele jocului:

- **Player** – personajul care va colecta bănuții. Caracteristici:
    - Pentru a simula întoarcerea jucătorului în direcția în care se mișcă, îl vom reprezenta din profil
    - Mișcările pe care le poate executa:
        - Deplasare la stânga
        - Deplasare la dreapta
        - Salt pe platforme
        - Oprirea pe platforme
        - Ricoșarea în cazul în care încearcă să sară, de jos în sus, prin platformă
        - Interzicerea ieșirii personajului prin marginea din stânga sau din dreapta a **Form**-ului
- **Platforme** – dreptunghiuri pe care **player**-ul are posibilitatea să sară. Pe aceste platforme sunt așezați bănuți.
- **Bănuții** – dreptunghiuri albastre așezate pe platforme. În cazul în care **player**-ul atinge un bănuț, pentru a simula culegerea lui, acest bănuț dispare de pe ecran.
- **Un contor** – indică numărul de **bănuți** pe care **player**-ul îl mai are de cules
- **fundalul jocului**, avem mai multe variante, în funcție de tipul de joc pe care îl dezvoltăm:
    - Fundal ca **BackgroundImage** al ferestrei jocului, în cazul în care aplicația pe care o dezvoltăm se desfășoară doar în **Form**-ul inițial. Putem, în acest tip de animație, să folosim și varianta ca fundalul jocului să fie **PictureBox**.
    - Fundal ca **PictureBox**, în cazul în care dorim să simulăm deplasarea personajului, stânga – dreapta mai mult decât în **Form**-ul inițial. Alegerea fundalului ca **PictureBox** va permite realizarea unei animații, în care acesta, în același timp cu deplasarea **player**-ului, se va mișca în direcția opusă mișcării **jucătorului**.

2.4. Mașina de stări

În continuare este util să introducem **mașina de stări** pentru a transforma povestea și descrierea jocului într-un **sistem funcțional și organizat**, pe care programul să-l poată gestiona logic. **Mașina de stări** este pasul tehnic ce face

legătura între ideea generală a jocului și **implementarea codului care gestionează comportamentul jocului**.

Mașina de stări joacă un rol important în:

1. **Organizarea logicii**: ne permite să definim clar fiecare stare a **Player**-ului și tranzițiile dintre acestea, asigurând o structură clară și ușor de gestionat.
2. **Controlul comportamentului în funcție de evenimente**: gestionează modul în care **Player**-ul răspunde la evenimente precum apăsarea tastelor, coliziuni cu platforme sau atingerea marginilor ferestrei, făcând jocul previzibil și stabil.
3. **Modularizarea codului**: se poate izola logica fiecărei stări, facilitând dezvoltarea, testarea și întreținerea codului. De exemplu, dacă vrem să ajustăm comportamentul **Player**-ului în timpul săriturii, putem face acest lucru direct în secțiunea dedicată stării **Săritură**.
4. **Flexibilitate și extindere**: Dacă dorim să adăugăm noi funcționalități sau stări, acestea pot fi integrate mai ușor într-o structură bine definită de stări și tranziții.
5. **Experiența de joc**: **Mașina de stări** asigură că **Player**-ul se comportă în mod coerent și previzibil pentru utilizator, îmbunătățind astfel experiența generală de joc.

În concluzie, rolul **mașinii de stări** în proiectul nostru este de a **orchestra toate acțiunile și tranzițiile posibile ale Player-ului**, făcând jocul să funcționeze fluid și intuitiv.

Mașina de stări poate fi exprimată atât **prin cod**, cât și **grafic**. Cele două abordări se completează și servesc scopuri diferite:

1. **Mașina de stări exprimată prin cod** reprezintă cea mai comună și practică formă de implementare a unei **mașini de stări** în cadrul unui joc sau aplicație. Aici, **stările** și **tranzițiile** dintre ele sunt reprezentate direct în logica programului.

2. **Mașina de stări exprimată în mod grafic** (**diagrama de stare** sau **State Diagram**) este o modalitate vizuală de a descrie toate stările și tranzițiile posibile într-un sistem. Aceasta este folosită de obicei pentru planificare, documentare și înțelegerea clară a comportamentului sistemului. Ea este alcătuită din:
   a. **Stările**: Fiecare **stare** este reprezentată ca un nod.

b. **Tranzițiile**: săgețile dintre stări indică trecerile dintr-o stare în alta, incluzând, de obicei, condițiile necesare pentru ca tranziția să aibă loc.
c. **Starea de start și starea finală**: De obicei, diagrama începe cu o stare inițială (marcată cu un punct sau etichetată ca **START**) și poate include o stare finală (care semnifică sfârșitul jocului sau oprirea activităților).

Ne ocupăm în continuare de **mașina de stări** exprimată în mod grafic.

Identificăm **stările jocului**:

- **Stare de inactivitate (Idle)**: **Player**-ul este pe o platformă și nu execută nicio acțiune (nu sunt apăsate tastele de deplasare sau săritura).
- **Deplasare la stânga**: **Player**-ul se deplasează la stânga și se schimbă profilul pentru a reflecta această direcție.
- **Deplasare la dreapta**: **Player**-ul se deplasează la dreapta și profilul se schimbă în direcția corectă.
- **Săritură**: **Player**-ul sare și trebuie să verifice dacă aterizează pe o platformă sau cade.
- **Cădere**: **Player**-ul cade în cazul în care nu aterizează pe o platformă.
- **Ricoșare**: **Player**-ul încearcă să sară printr-o platformă de jos în sus și este împins înapoi.
- **Culegerea bănuților**: Dacă **Player**-ul atinge un bănuț, acesta dispare și contorul este actualizat.
- **Limita ecranului**: **Player**-ul ajunge la marginea din stânga sau din dreapta a ecranului și este blocat.
- **Finalizarea jocului**: Toți bănuții sunt colectați și jocul se termină.

Să identificăm **tranzițiile**:

- Din **Idle**:
  - Dacă se apasă tasta stânga, se trece la **Deplasare la stânga**.
  - Dacă se apasă tasta dreapta, se trece la **Deplasare la dreapta**.
  - Dacă se apasă tasta săriturii, se trece la **Săritură**.
- Din **Deplasare la stânga**:
  - Dacă se eliberează tasta, se trece la **Idle**.
  - Dacă se apasă săritura, se trece la **Săritură**.
- Din **Deplasare la dreapta**:
  - Dacă se eliberează tasta, se trece la **Idle**.
  - Dacă se apasă săritură, se trece la **Săritură**.
- Din **Săritură**:

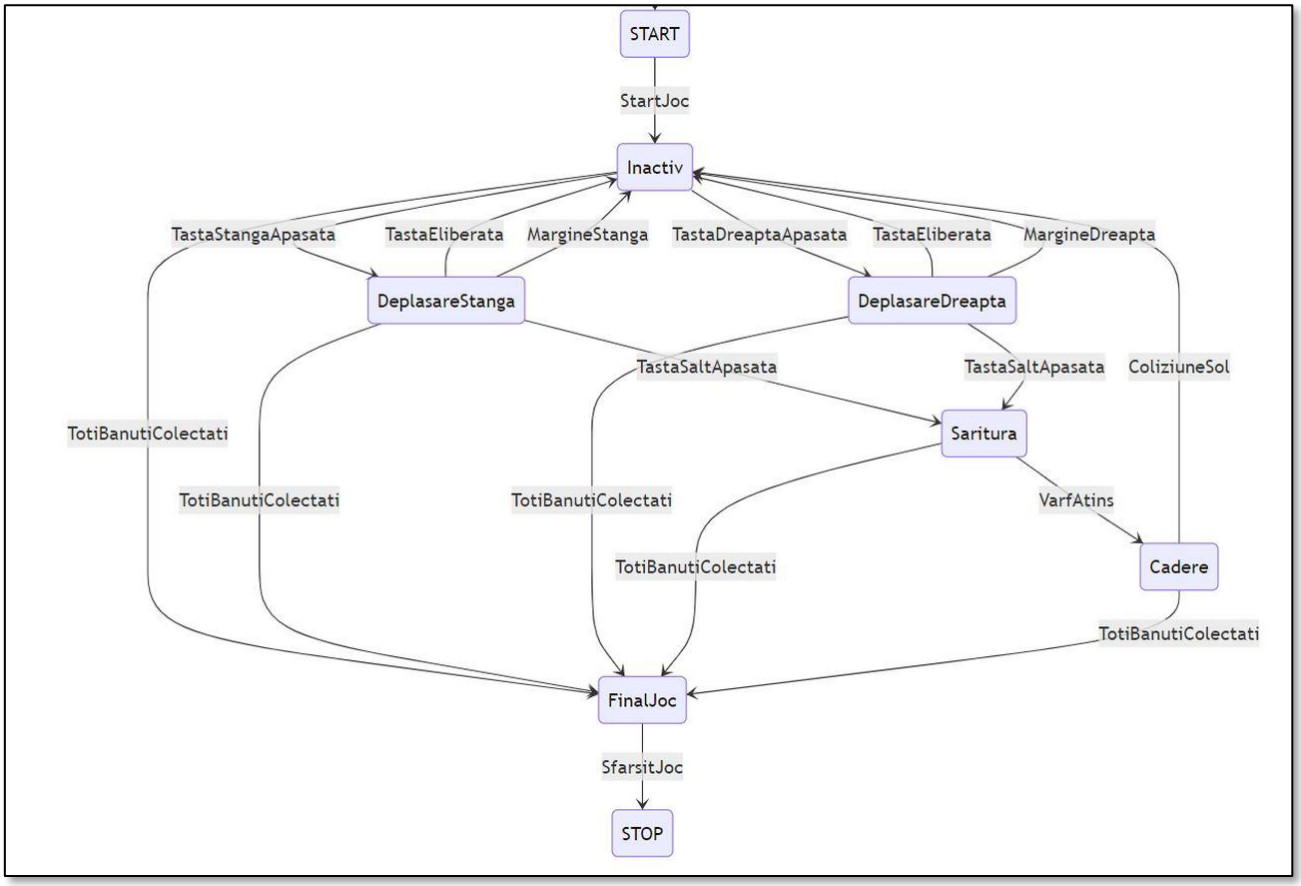
- Dacă player-ul aterizează pe o platformă, trece în **Idle**.
- Dacă nu, continuă în **Cădere**.
- Dacă sare în platformă de jos, trece la **Ricoșare**.

- Din **Culegerea bănuților**: Revenire la starea anterioară după dispariția bănuțului.

Putem folosi **Mermaid,** o bibliotecă JavaScript care permite crearea de diagrame dintr-o sintaxă textuală simplă. Folosind sintaxa specifică pentru diagrame de stare, librăria va crea diagrama. Accesați **https://mermaid.live** și introduceți codul:

```
stateDiagram-v2

    START --> Inactiv: StartJoc

    Inactiv --> DeplasareStanga: TastaStangaApasata
    Inactiv --> DeplasareDreapta: TastaDreaptaApasata
    DeplasareStanga --> Inactiv: TastaEliberata
    DeplasareDreapta --> Inactiv: TastaEliberata
    DeplasareStanga --> Saritura: TastaSaltApasata
    DeplasareDreapta --> Saritura: TastaSaltApasata
    Saritura --> Cadere: VarfAtins
    Cadere --> Inactiv: ColiziuneSol

    DeplasareStanga --> Inactiv: MargineStanga
    DeplasareDreapta --> Inactiv: MargineDreapta

    Inactiv --> FinalJoc: TotiBanutiColectati
    DeplasareStanga --> FinalJoc: TotiBanutiColectati
    DeplasareDreapta --> FinalJoc: TotiBanutiColectati
    Saritura --> FinalJoc: TotiBanutiColectati
    Cadere --> FinalJoc: TotiBanutiColectati

    FinalJoc --> STOP: SfarsitJoc
```

Vom obține:

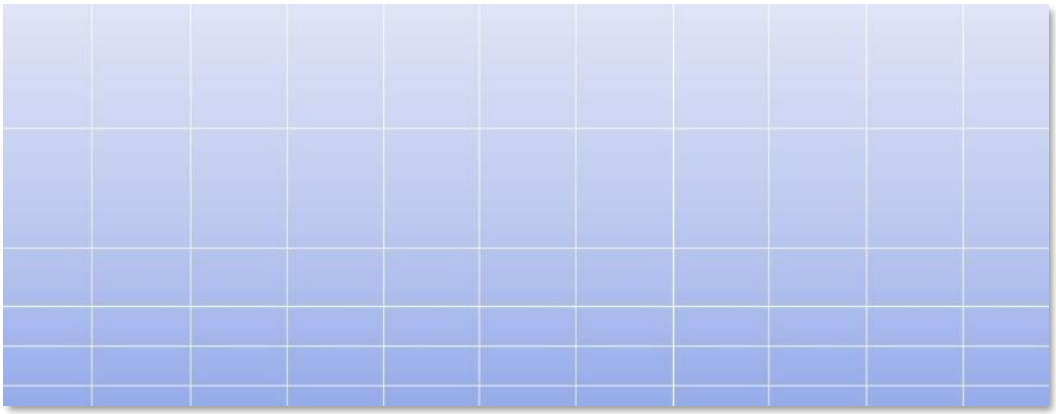


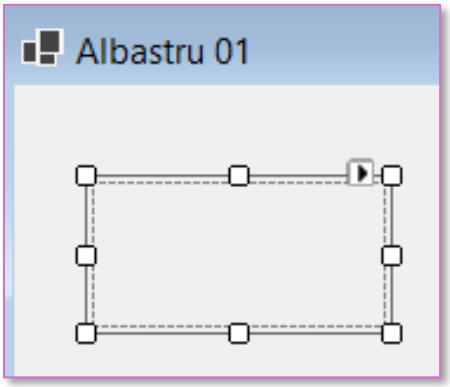
2.5. Dezvoltarea proiectului

2.5.1. Fundal, player și platforme

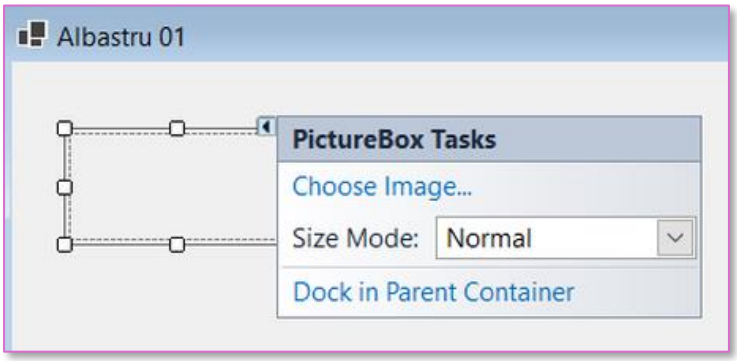
Pentru exemplul care urmează, am preferat să ne desenăm propriul **fundal**, **player**-ul și **platformele**. Toate fiind **PictureBox**-uri, pot fi modificate, ulterior, de către cititor, folosind imagini de pe Internet.

Vom folosi **MS Word**, pentru a desena **fundalul** și **personajul** jocului.

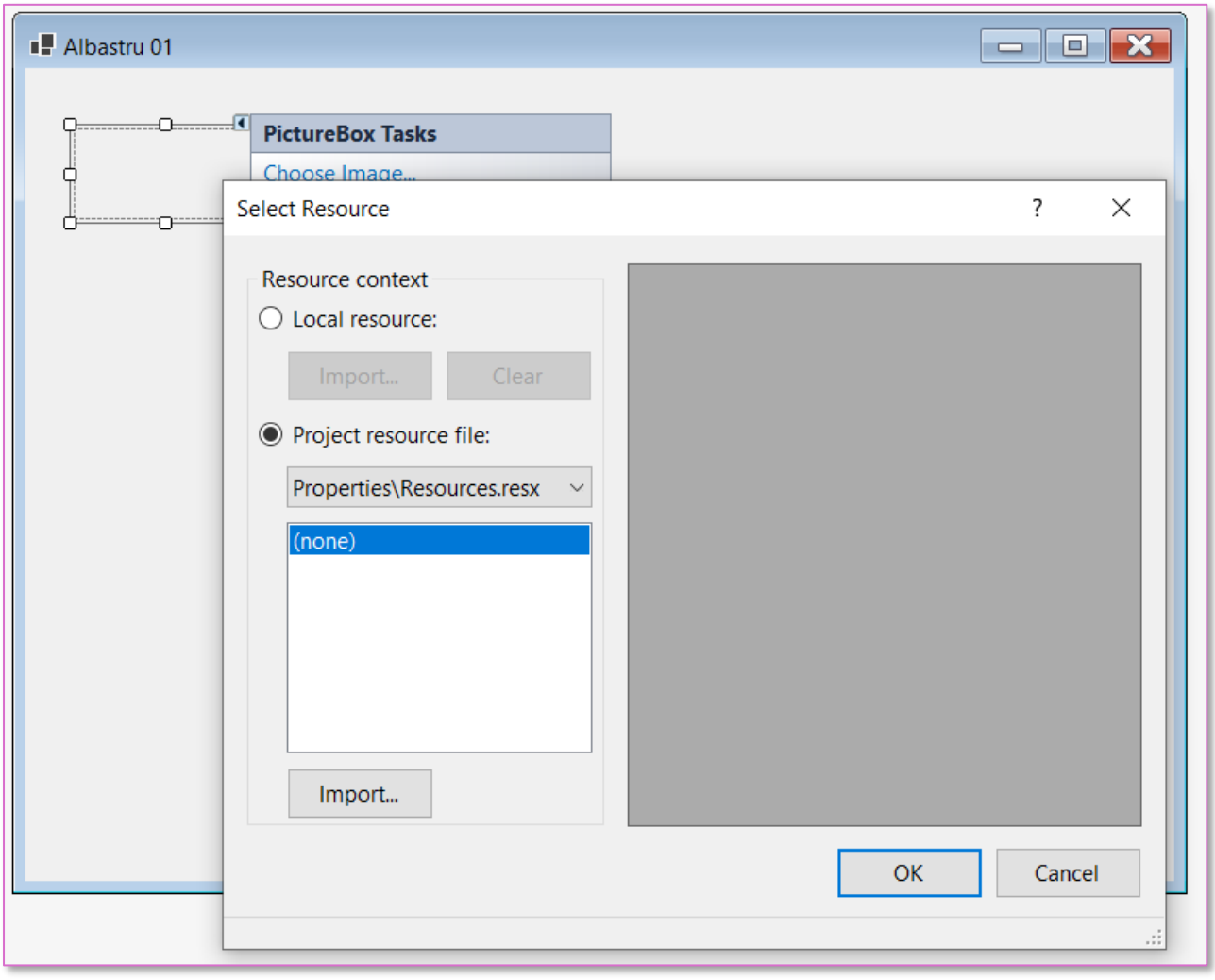




Începem cu fundalul. Aducem din **Toolbox** un **PictureBox**. Executăm clic pe triunghiul din dreapta sus a **PictureBox**-ului, pentru a selecta imaginea dorită.

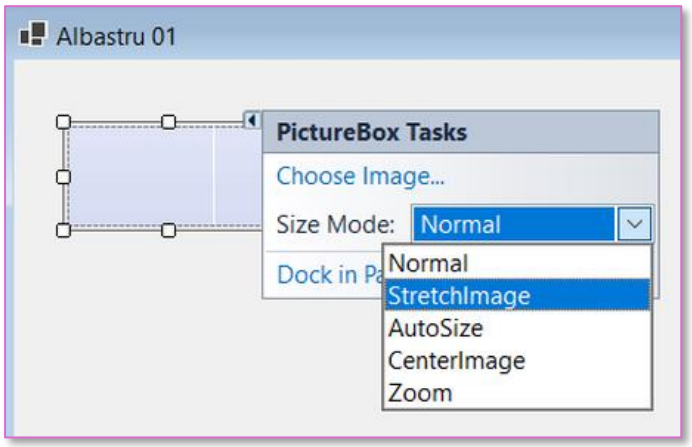


Selectăm opțiunea **Choose Image** din ferestra care se deschide.

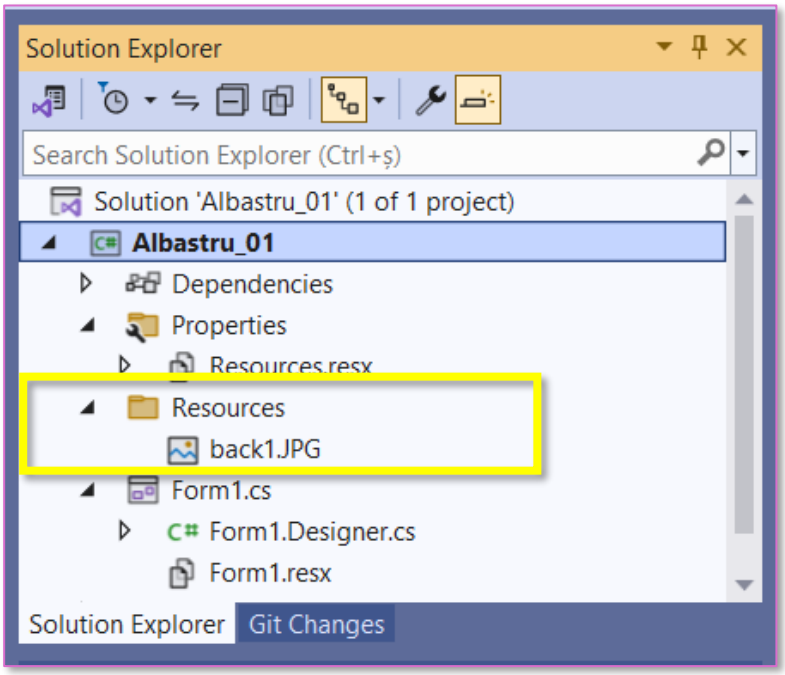


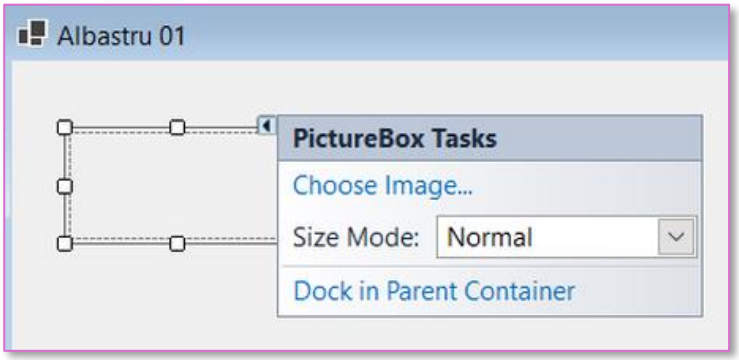
Pentru alegerea imaginii dorite folosim opțiunea, implicită, **Project resource file** și apoi **Import** pentru a o căuta.

După selectarea imaginii pe care dorim să o utilizăm, vom seta modul în care aceasta va fi afișată în **PictureBox**. Selectăm opțiunea **StretchImage** pentru ca imaginea să se redimensioneze automat, în funcție de redimensionarea **PictureBox-ului**.



După selectarea imaginii dorite, observăm că în panoul **Solution Explorer**, se generează automat un folder, **Resources**, care va conține toate imaginile pe care le folosim în proiect.





După ce am ales imaginea pentru fundal și modul de afișare al acesteia în **PictureBox**, apelăm la opțiunea **Dock in Parent Container** pentru ca imaginea să ocupe întregul **Form**.

Programul nostru debutează cu **public Form1**() care este constructorul implicit al clasei **Form**. Această metodă este apelată automat atunci când o instanță a clasei **Form** este creată într-un program **C#**.

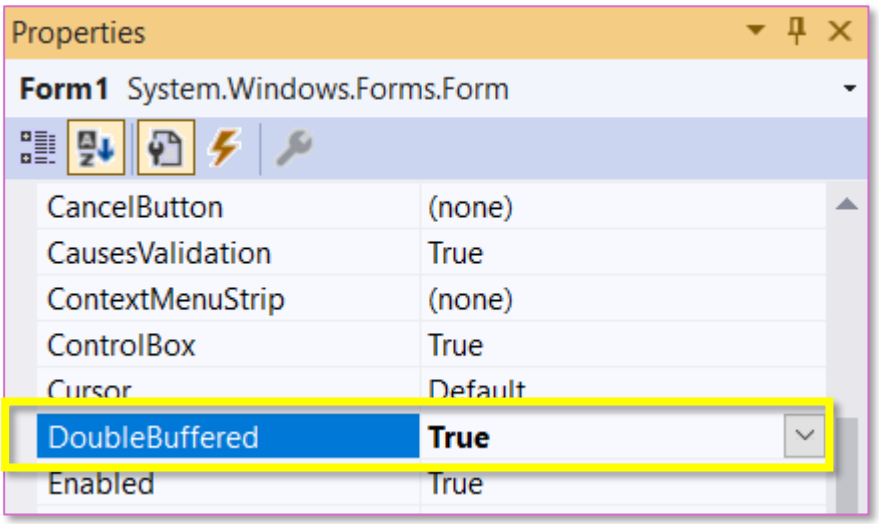
Constructorul **Form1** poate fi utilizat pentru a inițializa starea obiectului **Form1**. De exemplu, acesta este locul unde se pot inițializa componentele vizuale ale formularului, cum ar fi controalele, culorile și atributele formularului. Tot aici se pot stabili o serie de proprietăți care sunt specifice formularului, cum ar fi dimensiunea formularului, culoarea de fundal, imaginea de fundal, etc.

```
public Form1()
{
    InitializeComponent();
}
```

Deoarece aplicația noastră va conține și mișcarea unor obiecte, pentru a preveni efectul de pâlpâire (**flickering**) în momentul redării animațiilor, vom seta proprietatea **DoubleBuffered** a formularului la **true**. Mai multe explicații despre **DoubleBuffered** vor fi date în paragraful legat de varianta **a2.** de finalizare a aplicatiei (pag.31)

```
public Form1()
{
    InitializeComponent();
    this.DoubleBuffered = true;
}
```

Același efect îl putem obține prin setarea pe care o putem realiza din panoul **Properties** corespunzătoare formularului.



2.5.2. Mișcarea player-ului

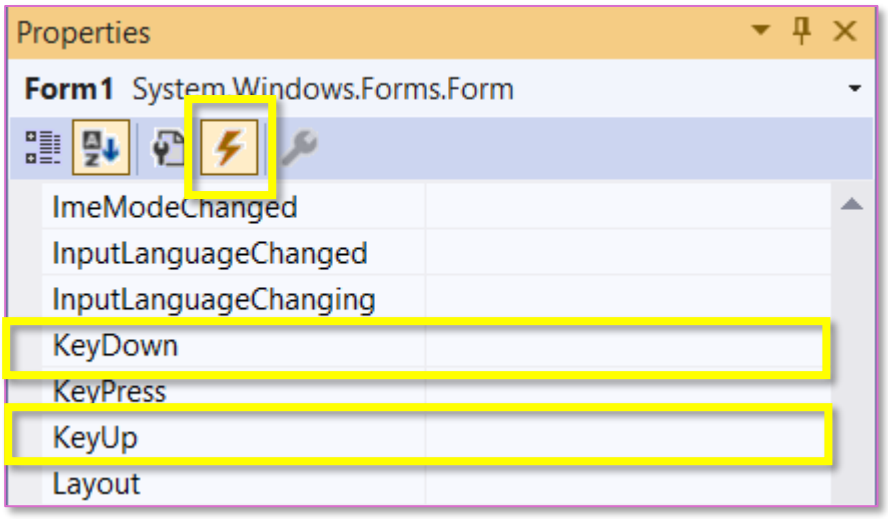
În continuare vom defini două funcții:

```
private void Form1_KeyDown(object sender, KeyEventArgs e)
```

```
private void Form1_KeyUp(object sender, KeyEventArgs e)
```

folosite pentru a detecta și a răspunde la evenimente legate de apăsarea sau nu a tastelor în cadrul ferestrei corespunzătoare lui **Form1**.

Pentru a putea utiliza aceste funcții, trebuie să asociem evenimentele **KeyDown**, respectiv **KeyUp** din **Form1** cu funcțiile **Form1_KeyDown**, respectiv **Form1_KeyUp**. O modalitate de a realiza acest lucru se poate realiza din designerul vizual.



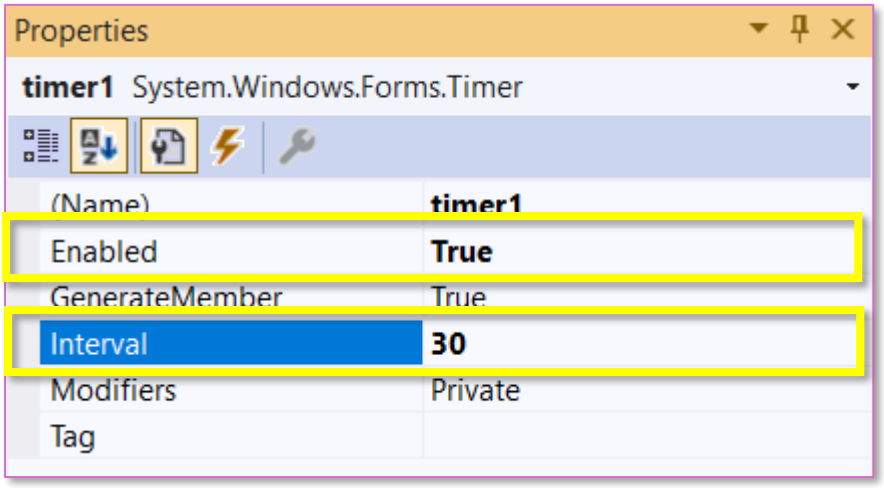
În exemplul nostru vom defini mișcările **player**-ului (stânga, dreapta și săritura) cu ajutorul săgeților direcționale. Vom folosi, de asemenea, tasta **Esc** pentru ieșirea din aplicație.

Definim variabilele boolene **st**, **dr**, **sus**.

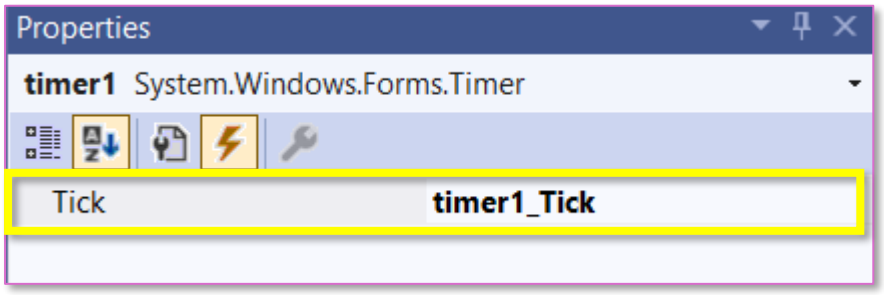
```
private void Form1_KeyDown(object sender, KeyEventArgs e)
{
    if (e.KeyCode == Keys.Up && !sus) sus = true;
    if (e.KeyCode == Keys.Left) st = true;
    if (e.KeyCode == Keys.Right) dr = true;
    if (e.KeyCode == Keys.Escape) Application.Exit();
}
```

```
private void Form1_KeyUp(object sender, KeyEventArgs e)
{
    if (e.KeyCode == Keys.Up) sus = false;
    if (e.KeyCode == Keys.Left) st = false;
    if (e.KeyCode == Keys.Right) dr = false;
}
```

Pentru aplicațiile la care intervine mișcarea unui obiect, vom folosi obiectul **timer** de tipul **Timer**, utilizat pentru declanșarea unor evenimente, în mod repetat, la intervale regulate de timp (**tick interval**). **Interval**-ul de timp este reprezentat în milisecunde.

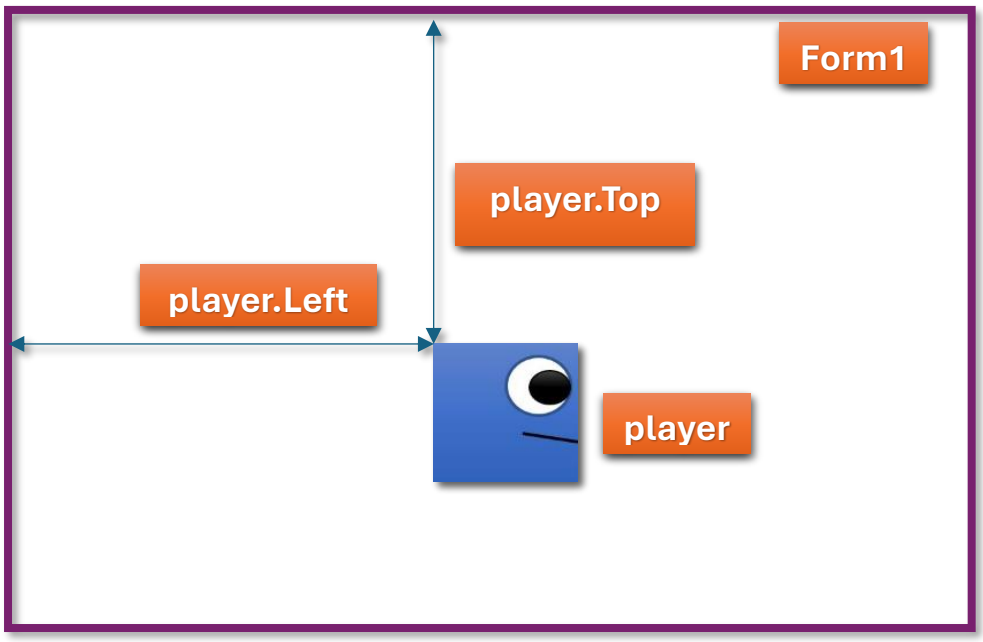


Pentru gestionarea evenimentului de **tick**, trebuie adăugat un **handler de eveniment**, numit **timer1_Tick**, declanșat automat de fiecare dată când timerul **timer1** generează un **eveniment de tick**.




```
private void timer1_Tick(object sender, EventArgs e)
{

}
```

Reamintim că mișcarea unui obiect se face, printre alte metode, prin referirea la coordonatele din colțul stânga sus al **player**-ului. Coordonatele se referă la distanța până la marginea din stânga, respectiv la marginea de sus a lui **Form1**



Să introducem condițiile de mișcare ale **player**-ului:
- mișcarea la stânga:

```
//player-ul se deplaseaza spre stanga, cu viteza "viteza"
//pana intalneste marginea din stanga a ferestrei

if (st && player.Left>0)
    player.Left -= viteza;
```

- mișcarea la dreapta:

```
//player-ul se deplaseaza spre dreapta, cu viteza "viteza"
//pana intalneste marginea din dreapta a a ferestrei

if (dr && player.Left<this.Width-player.Width-20)
    player.Left += viteza;
```

Nu întâmplător am desenat **player**-ul cu ochiul și gura îndreptate într-o direcție (dreapta în cazul nostru). La mișcarea **player**-ului stânga – drepta, acesta ar trebui să se îndrepte spre direcția de deplasare.

În **Visual C#**, putem roti și oglindi o imagine folosind metoda **RotateFlip** a obiectului **Image**. Această metodă poate fi utilizată pentru a efectua mai multe tipuri de transformări asupra imaginii, inclusiv rotația și oglindirea.
Dorim să realizăm o rotire a imaginii **player**-ului cu un unghi de 0 grade și o oglindire pe axa orizontală. Pentru aceasta vom introduce codul de mai jos în ambele instrucțiuni **if** care verifică deplasarea stânga – dreapta a **player**-ului

Image.RotateFlip(RotateFlipType.RotateNoneFlipX)

Pentru ca **player**-ul să nu se oglindească decât la schimbarea direcției, vom folosi o variabilă globală booleană, pe care o numim **directie**, și care își schimbă valoarea la schimbarea direcției.

De asemenea vom pune condiția ca **player**-ul să se redeseneze pentru a rezolva probleme legate de actualizarea interfeței utilizatorului: **player.Refresh()**

```
//player-ul se deplaseaza spre stanga, cu viteza "viteza"
//pana intalneste marginea din stanga a ferestrei

if (st && player.Left > 0)
{
    player.Left -= viteza;
    if (!directie)
    {
         player.Refresh();
         player.Image.RotateFlip(RotateFlipType.RotateNoneFlipX);
         directie = true;
    }
}
```

```
//player-ul se deplaseaza spre dreapta, cu viteza "viteza"
//pana intalneste marginea din dreapta a ferestrei

if (dr && player.Left < this.Width - player.Width - 20)
{
    player.Left += viteza;
    if (directie)
    {
        player.Refresh();
        player.Image.RotateFlip(RotateFlipType.RotateNoneFlipX);
        directie = false;
}
```

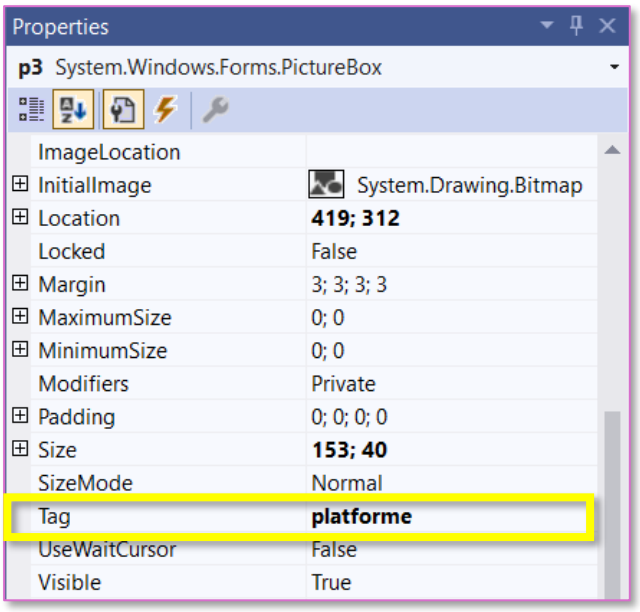
2.5.3. Player-ul și platformele

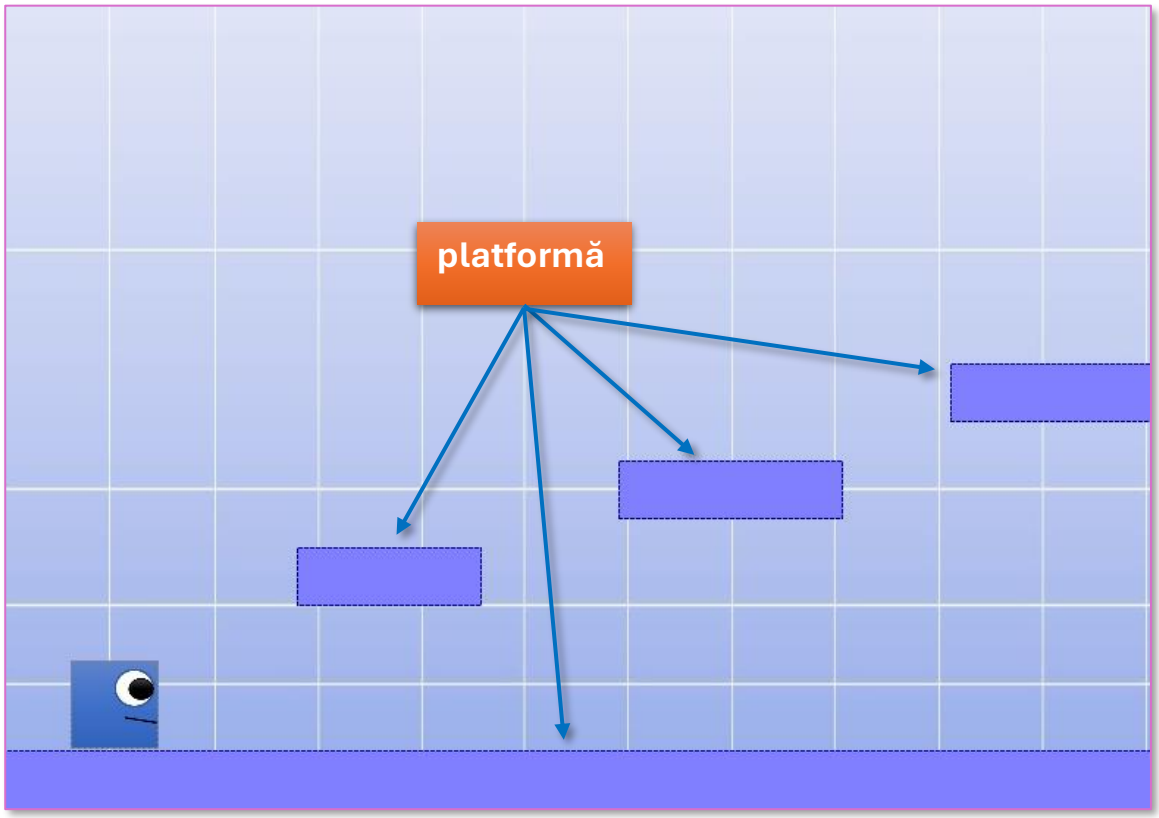
Adăugăm pe **Form** mai multe controale (vorbim de platformele pe care va sări **player**-ul) de tip **PictureBox**.

De ce am ales controale de tip **PictureBox**? Pentru a permite celor care vor lucra la dezvoltarea aplicației, să particularizeze respectivele platforme folosind

imagini corespunzătoare dorințelor lor. În aplicația pe care o dezvoltăm în continuare, vom pune un **BackColor** albastru platformelor.

Vom pune condiția ca **player**-ul, dacă întâlnește o **platformă**, să se oprească pe aceasta. Deoarece există mai multe platforme, nu vom scrie cod pentru fiecare dintre ele, ci ne vom referi la mulțimea, **Tag** , din care fac acestea parte. Setăm proprietatea **Tag** a elementelor cu valoarea **platforme**.

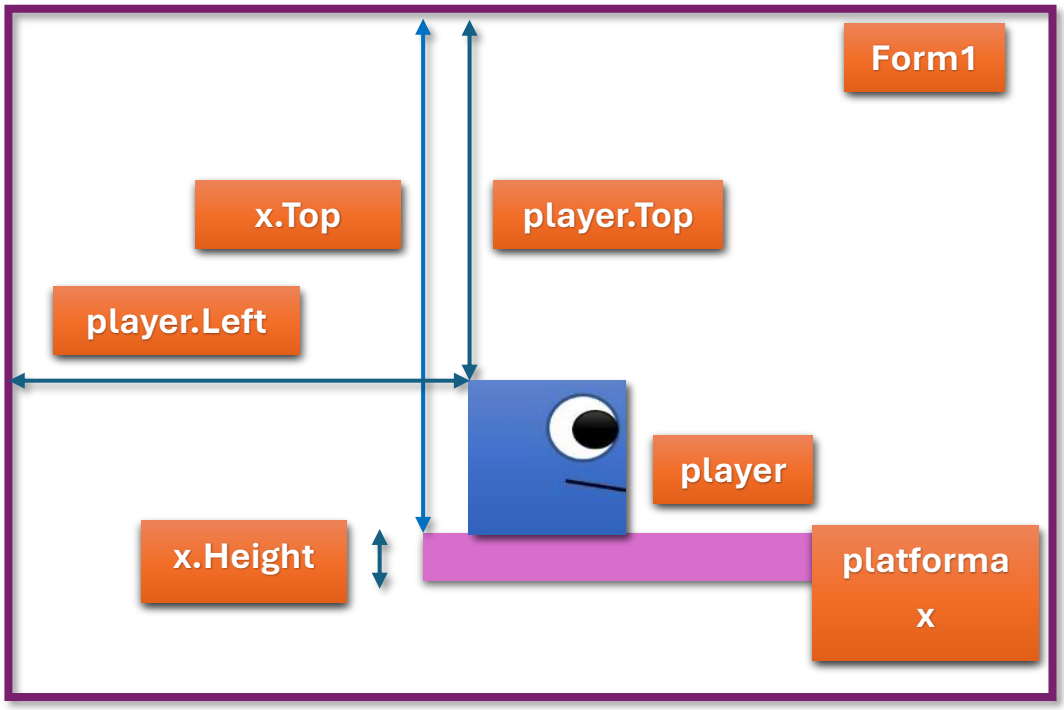




Să ne ocupăm, în continuare, de celelalte mișcări pe care le atribuim **player**-ului.

Căderea **player**-ului:

```
//caderea player-ului
player.Top += viteza;
```




```
foreach (Control x in this.Controls)
{
      if(x is PictureBox && (string)x.Tag == "platforme")
      {
              if(player.Bounds.IntersectsWith(x.Bounds))
              {
                     if (player.Top > x.Top)
                     player.Top += x.Height - 5;
              }
}
```

În sfârșit, avem de rezolvat problemele legate de săritură (apăsarea pe tasta direcțională, săgeată sus):

- prin apăsarea tastei săgeată sus, să nu permitem **player**-ului să execute sărituri multiple
- în cazul în care **player**-ul se află sub o platformă și se încearcă o săritură în sus, **player**-ul să ricoșeze în jos

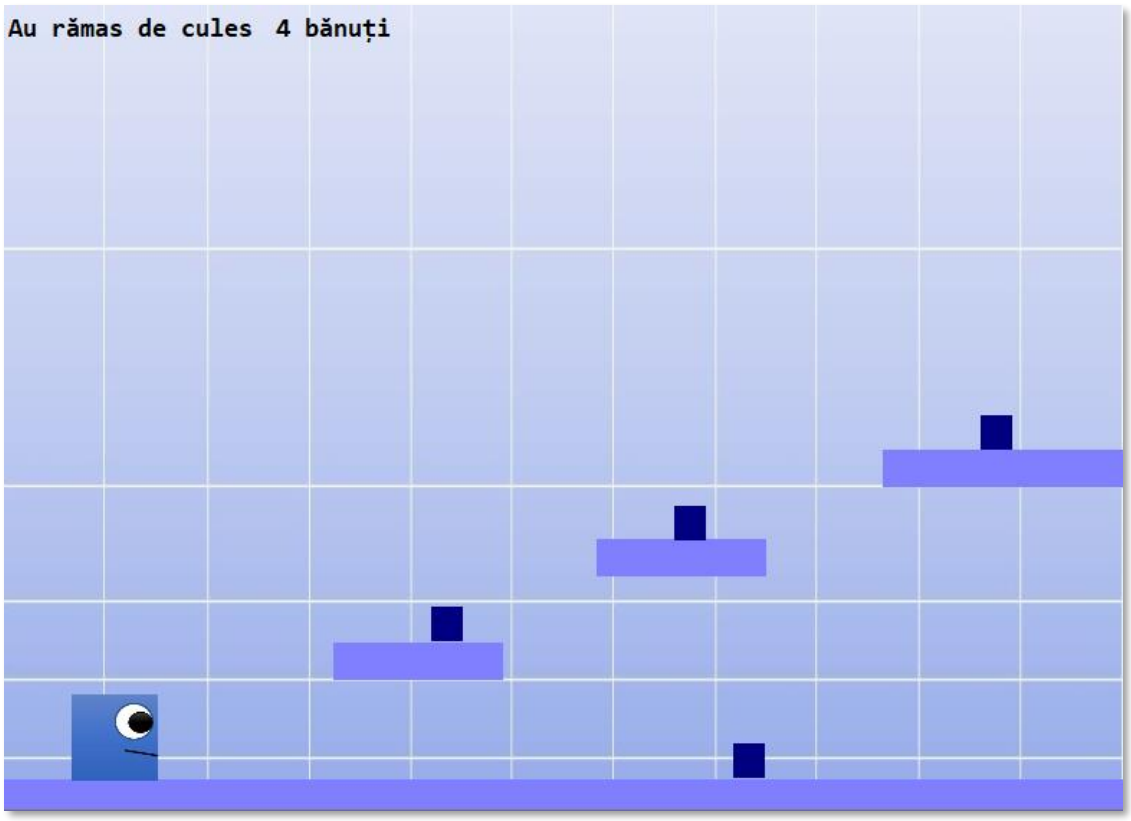
```
//caderea player-ului
player.Top += viteza;

if (sus && saritura < 0)
       sus = false;

if (sus)
{
       viteza = -10; //amplitudinea sariturii
       saritura -= 1;
}
else
{
        viteza = 10;
}

foreach (Control x in this.Controls)
{
      if(x is PictureBox && (string)x.Tag == "platforme")
      {
              if(player.Bounds.IntersectsWith(x.Bounds))
              {
                      if (player.Top > x.Top)
                            player.Top += x.Height - 5;
                      if (!sus && player.Top < x.Top + x.Height - 10)
                      {
                            saritura = 16;
                            viteza = 0;
                            player.Top = x.Top – player.Height + 1;
                      }
                }
       }
}
```

Am definit ca variabile globale:

```
bool sus = false, st = false, dr = false;
int viteza;
int saritura = 20;  //durata sariturii
int scor = 0;
bool directie = true;
```

2.5.4. Bănuții

Adăugăm pe **Form** mai multe controale (vorbim de bănuții pe care **player**-ul va trebui să îi culeagă) de tip **PictureBox**. Setăm proprietatea **Tag** a bănuților cu valoarea **banuti**.



Reamintim că scopul jocului este de a culege, prin intermediul **player**-ului, toate obiectele din **Tag**-ul **banuti**.

Ne propunem ca pe ecran să apară să apară și numărul de bănuți pe care îi avem de cules. O variantă pentru a realiza acest lucru este să folosim:

- un **Label** care conține textul **Au rămas de cules**
- un **Label**, numit **lbl_banuti** care conține textul **4**. Acest text (**4**) va fi actualizat, în funcție de bănuții culeși
- un **Label** care conține textul **bănuți**

După culegerea unui bănuț, pentru a simula această acțiune, îl vom face invizibil. Pentru aceasta vom seta proprietatea **x.Visible**, a bănuțului, la **false**, adică **x.Visible=false**

```
foreach (Control x in this.Controls)
{
        . . . . .

        if(player.Bounds.IntersectsWith(x.Bounds) && x.Visible==true)
        {
                if ((string)x.Tag=="banuti")
                {
```

```
                        x.Visible = false;
                        scor--;
                        nr_banuti.Text =   Convert.ToString(scor);
                    }
            }
}
```

În loc de **nr_banuti.Text = Convert.ToString(scor)** se mai poate folosi **lbl_banuti.Text = scor.ToString()**

2.5.5. Finalizarea aplicației/jocului

A sosit momentul în care trebuie să ne hotărâm cum să continue aplicația noastră. Avem de ales între mai multe variante:

a. dacă culegem toți bănuții, aplicația să se încheie, fie printr-un mesaj, fie se deschide o „**poartă**” prin care părăsim aplicația. Putem folosi:
   a1. un **MessageBox.Show**
   a2. un **Label**
   a3. un nou **Form**
   a4. O „**poartă**” prin care putem părăsi aplicația

b. dacă culegem toți bănuții din primul ecran (**Nivelul 1**) să trecem în următorul ecran (**Nivelul 2**) unde trebuie să îndeplinim aceleași sarcini ca la **Nivelul 1**. După culegerea tuturor bănuților aplicația se încheie fie printr-un mesaj, fie se deschide o „**poartă**” prin care părăsim aplicația. Această „**poartă**” o putem plasa în **Nivel 1** (ceea ce presupune o întoarcere a **player**-ului la acest nivel, fie apare „**poarta**” respectivă la **Nivelul 2**).

a1.
Pentru a putea determina finalizarea aplicației, folosim variabila **scor**, care numără bănuții pe care îi avem de cules.
Dacă am cules toți bănuții, se afișează un mesaj corespunzător și ieșim din aplicație.

```
if(player.Bounds.IntersectsWith(x.Bounds) && x.Visible==true)
{
        if ((string)x.Tag=="banuti")
        {
```

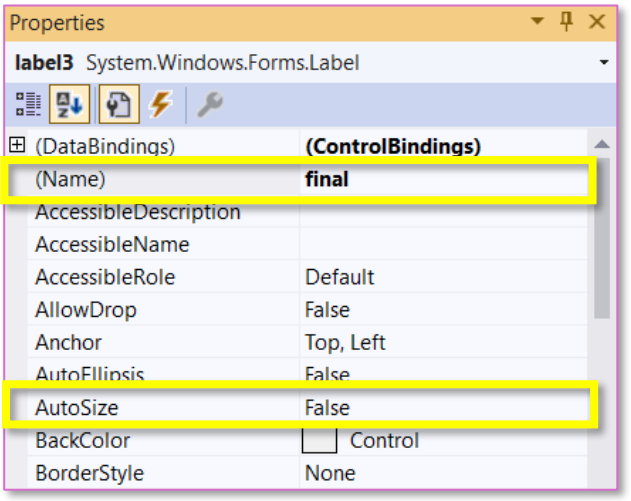
```
                x.Visible = false;
                scor--;
                nr_banuti.Text = Convert.ToString(scor);
                if (scor == 0)
                {
                        timer1.Stop();
                        MessageBox.Show("Game Over!");
                        Application.Exit();
                 }
        }
}
```

În cazul în care **timer1.Stop()** lipsește din codul nostru, fereastra cu mesajul **Game Over!** se va multiplica, pe ecran, la infinit.

Observație: încheierea aplicației folosind un **MessageBox.Show** nu este prea elegantă. Putem să apelăm la alte două variante:

a2.
În cazul finalizării aplicației, am cules toți bănuții, pe ecran apare un **Label** care anunță încheierea aplicației. Inițial, **Label**-ul este invizibil, el devenind vizibil doar la finalizarea sarcinii (culegerea tuturor bănuților).

Inserăm un **Label** pe care îl redenumim, de exemplu, **final**. Pentru a îl putea redimensiona așa cum dorim, vom folosi **False** pentru proprietatea **AutoSize** a acestuia.



Un exemplu de redimensionare, colorare a fundalului și a mesajului corespunzător, îl vedeți în imaginea de mai jos.

În **Visual C# (Windows Forms)**, un **Label** nu suportă direct două linii de text cu fonturi diferite prin fereastra **Properties** sau prin setările implicite. **Label**-urile din **Windows Forms** au un singur stil de text pentru tot conținutul, nefiind posibilă setarea unor fonturi diferite pentru fiecare linie sau cuvânt prin fereastra **Properties**.

Folosim două **Label**-uri **final** și **final_mesaj** pentru afișarea celor două mesaje, **Game Over!**, respectiv **Pentru a ieși din joc, apăsați tasta Escape**!

În funcția constructor a formei vom scrie:

```
public Form1()
{
    InitializeComponent();
    this.DoubleBuffered = true;
    final.Visible = false;
    final_mesaj.Visible = false;
}
```

unde:

InitializeComponent(); **-** este metoda generată automat care inițializează toate controalele definite pe formă (de exemplu, butoane, etichete, etc.).

this.DoubleBuffered = true; - activează **buffering**-ul dublu pentru formă. Este folosită pentru a îmbunătăți performanța grafică a formei, mai ales atunci când sunt actualizări vizuale frecvente, reducând **flickering**-ul.

Double buffering este o tehnică folosită pentru a reduce sau elimina efectul de **flickering** (clipire) atunci când un control sau o formă este actualizată grafic frecvent, cum ar fi atunci când desenezi, miști sau redimensionezi elemente vizuale. Această tehnică funcționează prin desenarea mai întâi a conținutului într-un **buffer** (o zonă de memorie temporară) și apoi afișarea acestuia dintr-o dată pe ecran. În acest mod, toate actualizările grafice sunt făcute simultan, prevenind afișarea parțială a modificărilor.

final.Visible = false; și final_mesaj.Visible = false; - ascund controalele, **label**-urile respective, la lansarea aplicației.

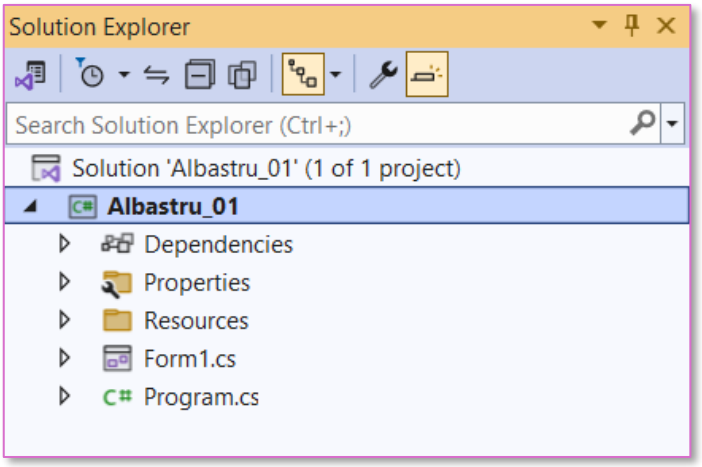
În secvența de program anterioară, vom înlocui
MessageBox.Show("Game Over!");

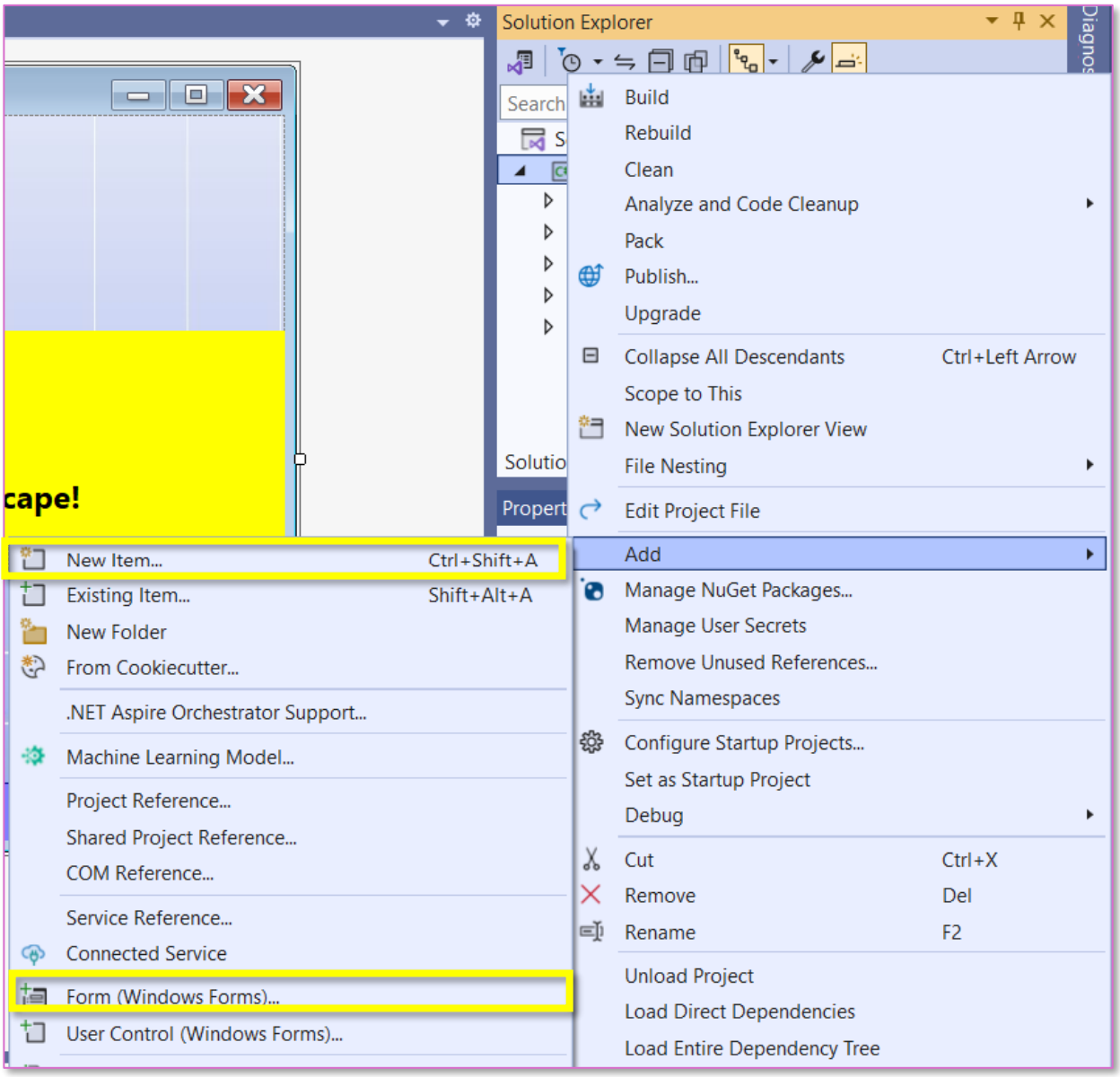
```
if(player.Bounds.IntersectsWith(x.Bounds) && x.Visible==true)
{
        if ((string)x.Tag=="banuti")
        {
                x.Visible = false;
                scor--;
                nr_banuti.Text = Convert.ToString(scor);
                if (scor == 0)
                {
                        timer1.Stop();
                        final.Visible = true;
                        final_mesaj.Visible = true;
                        //MessageBox.Show("Game Over!");
                        //Application.Exit();
                }
         }
}
```

a3.

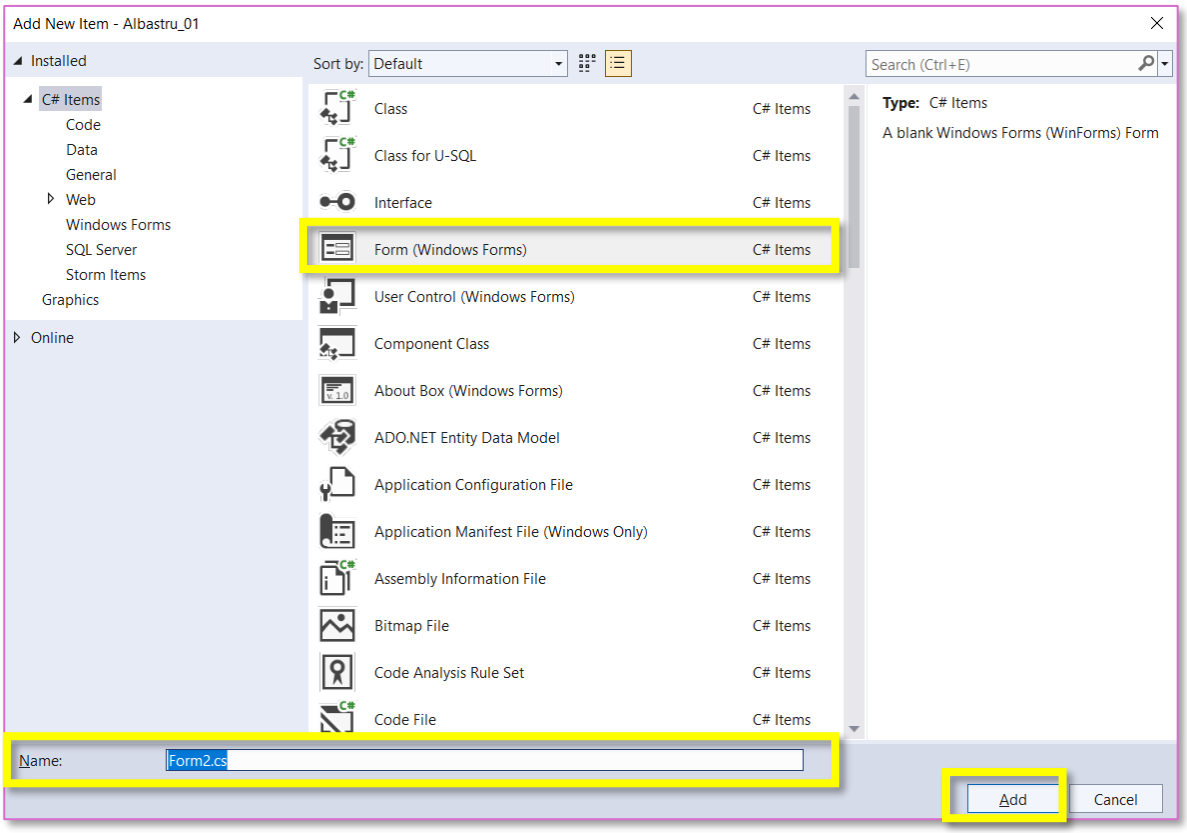
Să vedem cum putem adăuga un nou **Form**, pe care îl vom folosi pentru finalizarea aplicației noastre.

În panoul **Solution Explorer**, executăm clic dreapta pe **Albastru_01** (numele aplicației noastre). Accesăm opțiunea **Add** și **New Item** sau **Form (Windows Forms)**





Dacă accesăm **Form (Windows Forms)**, din fereastra care se deschide, selectăm **Form (Windows Forms)**, opțional, schimbăm numele formului, în caseta text **Name** și executăm clic pe butonul **Add**.



Propunem ca design pentru acest **Form2**, cu observația că dimensiunea acestei ferestre, din rațiuni de estetică, o setăm la dimensiunea lui **Form1** (panoul **Properties**, **Size**).

În acest **Form**, mesajul **Game Over!** este afișat prin intermediul unui **Label**, iar **Reluare** și **Ieșire** sunt **butoane,** redenumite **btn_Reluare** respectiv **btn_Iesire**.

Reamintim, scopul aplicației noastre este, ca prin intermediul **player**-ului, să culegem toți bănuții. În acel moment, fereastra **Form1** va deveni invizibilă, iar pe ecran afișăm fereastra **Form2** cu cele două opțiuni. Dacă selectăm opțiunea de **Reluare** a jocului, fereastra **Form2** se va închide, iar fereastra **Form1** redevine vizibilă. Aici, în **Form1**, trebuie să inserăm o nouă funcție care să aducă jocul la starea inițială: **player**-ul să pornească din același loc și să se miște ca la inceputul aplicației, iar bănuții, care culeși fiind erau invizibili, să redevină vizibili.

```
if(player.Bounds.IntersectsWith(x.Bounds) && x.Visible==true)
{
        if ((string)x.Tag=="banuti")
        {
             x.Visible = false;
             scor--;
             nr_banuti.Text = Convert.ToString(scor);
             if (scor == 0)
             {
                   timer1.Stop();
                   this.Hide();            // Ascunde Form1
                   Form2 f = new Form2();  //Creează o instanță a lui Form2
                   f.ShowDialog();  //Deschide Form2 în mod modal
                   this.Show();  // Afișează Form1 când Form2 este închis
                   ResetGame();  // Apelează metoda de resetare

                   //final.Visible = true;
                   //final_mesaj.Visible = true;
                   //MessageBox.Show("Game Over!");
                   //Application.Exit();
                }
         }
}
```

Metoda de resetare a stării inițiale în **Form1**

```
private void ResetGame()
{
    // Setează poziția inițială a player-ului
    player.Location = new Point(66, 245);

    // Resetează variabilele legate de gravitație și mișcare
    viteza = 0;  // Resetează viteza la zero
    saritura = 20;  // Resetează valoarea de săritură
    sus = false;  // Player-ul nu este în stare de săritură la început
    st = false;   // Resetează direcția stângă
    dr = false;   // Resetează direcția dreaptă
    directie = true; // Asigură că player-ul este orientat spre dreapta
```

```
    // Resetează scorul, dacă este cazul
    scor = 4;
    lbl_banuti.Text = scor.ToString();

    // Fă toți banii vizibili din nou
    foreach (Control x in this.Controls)
    {
        if (x is PictureBox && (string)x.Tag == "banuti")
        {
            x.Visible = true; // Reface banii vizibili
        }
    }
    // Resetarea orientării player-ului
    player.Image.RotateFlip(RotateFlipType.RotateNoneFlipX); // Imaginea
                                                // este orientată spre dreapta

    timer1.Start();
}
```

2.5.6. Ce înseamnă deschiderea unei ferestre în mod modal?

A deschide o fereastră **în mod modal** înseamnă că fereastra deschisă devine **prioritară** și blochează interacțiunea cu alte ferestre ale aplicației până când aceasta este închisă. Într-un context de interfață grafică, o fereastră modală solicită ca utilizatorul să interacționeze mai întâi cu ea înainte de a putea reveni la alte ferestre.

Putem aminti câteva caracteristici ale unei ferestre modale

- **Blocarea interacțiunii cu ferestrele de bază** - când deschidem o fereastră în mod modal, nu putem interacționa cu alte ferestre din aplicație până când nu închidem fereastra modală.
- **Așteptarea închiderii** - aplicația va aștepta să se finalizeze interacțiunea cu această fereastră înainte de a continua cu alte operațiuni.

Pentru a deschide o formă în mod modal, se folosește metoda **ShowDialog()**. Aceasta face ca fereastra să fie afișată și să aștepte până când utilizatorul o închide, iar abia după aceea execuția codului continuă în forma principală.

2.5.7. Ce este o fereastră non-modală (modeless)?

Opusul unei ferestre modale este o fereastră **non-modală**. Ea se deschide folosind metoda **Show()**. În acest caz, utilizatorul poate interacționa liber atât cu fereastra principală, cât și cu cea nou deschisă.

Revenim la aplicația noastră.

În **Form2** vom scrie codul corespunzător celor două butoane:

```
private void btn_Reluare_Click(object sender, EventArgs e)
 {
     this.Close();
 }

 private void btn_Iesire_Click(object sender, EventArgs e)
 {
     Application.Exit();
 }
```

a4.
O altă posibilitate de finalizare a aplicației noastre constă în ieșirea printr-o **poartă**. Avem mai multe variante referitoare la această **poartă** referitoare la vizibilitatea ei sau la locul unde este plasată

a41. Inițial, **poarta** este invizibilă

a42. Poarta este vizibilă de la începutul aplicației dar nu poate fi accesată decât după culegerea tuturor bănuților

Vom trata împreună cele două cazuri. Dacă **poarta** este inițial, invizibilă, punem condiția ca în cazul în care am cules toți **bănuții**, ea să devină vizibilă, iar programul poate să sesizeze intersecția dintre **player** și **poartă**, iar aplicația să se încheie.

```
if(player.Bounds.IntersectsWith(x.Bounds) && x.Visible==true)
{
        if ((string)x.Tag=="banuti")
        {
                x.Visible = false;
                scor--;
                nr_banuti.Text = Convert.ToString(scor);
                if (scor == 0)
                       poarta.Visible = true;
         }
```

```
}

if (player.Bounds.IntersectsWith(poarta.Bounds))
        Application.Exit();
```

2.5.8. Genericul aplicației

După cum orice carte are o copertă, orice film are un generic și aplicația noastră va avea genericul ei, generic cu care va debuta jocul. **Genericul** ar trebui să conțină:

- Numele aplicației/jocului
- Numele autorului/autorilor
- Data, anul când a fost creată aplicația
- Imagine relevantă

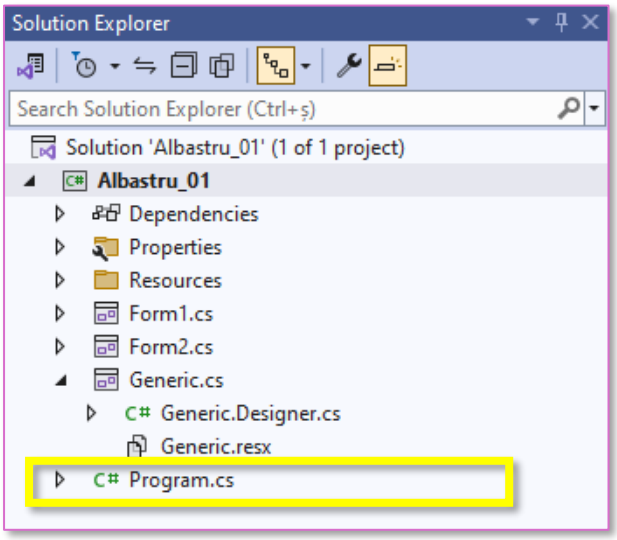
Vom adăuga un nou **Form** pe care îl vom redenumi **Generic** (redenumirea nu este obligatorie), așa cum am arătat mai sus:

- În fereastra **Solution Explorer**, executăm clic dreapta pe numele aplicației
- În fereastra care se deschide selectăm **Add** și apoi **Form (Windows Forms)**
- Schimbăm numele Formului, în caseta text din dreptul lui **Name**. Propunem **Generic.cs**, iar apoi executăm clic pe butonul **Add**

Setăm mărimea ferestrei la aceeași dimensiune cu celelate două ferestre definite anterior.

O variantă a **Genericului** este:

Pentru ca aplicația noastră să înceapă cu fereastra **Generic**, în fereastra **Solution Explorer**, executăm clic pe **Program.cs**



Înlocuim **Form1** cu **Generic**

Din acest moment aplicația noastră va începe cu fereastra **Generic**.

```
namespace Albastru_01
{
    0 references
    internal static class Program
    {
        /// <summary>
        ///  The main entry point for the application.
        /// </summary>
        [STAThread]
        0 references
        static void Main()
        {
            // To customize application configuration such as
            // see https://aka.ms/applicationconfiguration.
            ApplicationConfiguration.Initialize();
            Application.Run(new Form1());
        }
    }
}
```

Butonul **START**, redenumit **btn_START** va porni aplicația/jocul nostru. Codul corespunzător este:

```
private void btn_START_Click(object sender, EventArgs e)
{
    this.Hide();
    Form1 a = new Form1();
    a.ShowDialog();
}
```