 «Քվանտ» վարժարան

Հետազոտական աշխատանք

Թեմա՝ Շաշկի խաղի ստեղծումը c# ծրագրավորման լեզվով

Դասարան` 101

Առարկա՝ Ինֆորմատիկա

Աշակերտ` Մանուչարյան Արգիշտի

Ղեկավար՝ Կարապետյան Մ․

Երևան-2023

# Բովանդակություն

[ԳԼՈՒԽ 1․ Աշխատանքի նպատակ, խնդրի սահմանում և առանձնահատկություն 3](#_Toc133418696)

[Աշխատանքի նպատակը 3](#_Toc133418697)

[Առանձնահատկությունը 5](#_Toc133418698)

[ԳԼՈՒԽ 2․ C# պատմություն, կլասեր 10](#_Toc133418699)

[Պատմություն 10](#_Toc133418700)

[Կլասեր 10](#_Toc133418701)

[ԳԼՈՒԽ 3․ Խաղի ինտերֆեյսի ձևավորում 12](#_Toc133418702)

[ԳԼՈՒԽ 4․ Ծրագրային կոդի ստեղծում 18](#_Toc133418703)

[Հայտարարություններ 18](#_Toc133418704)

[Ծրագրի գործարկում 20](#_Toc133418705)

[Խաղի սկիզբ 22](#_Toc133418706)

[DrawCell() ֆունկցիա 24](#_Toc133418707)

[ShowPossibleSteps1() ֆունկցիա 26](#_Toc133418708)

[OnFigurePress պրոցեդուրա 27](#_Toc133418709)

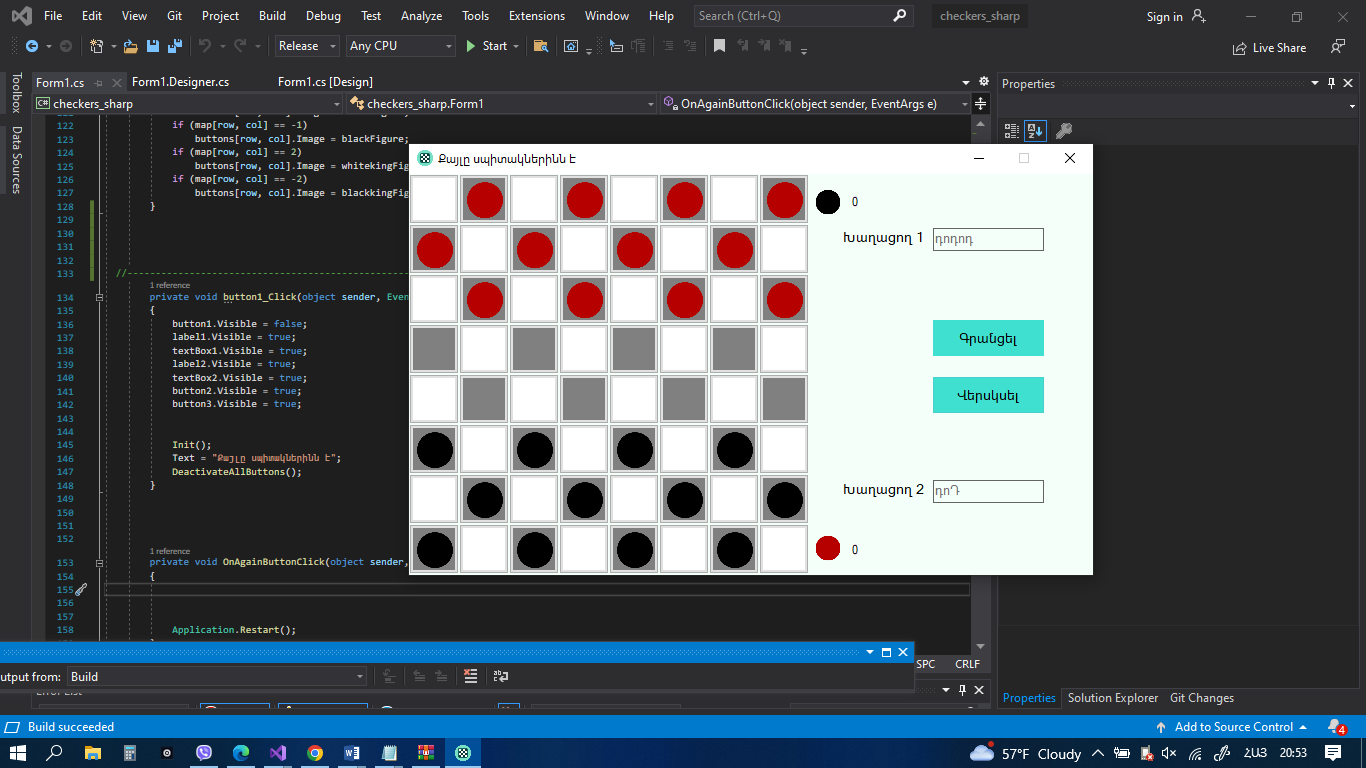
[Եզրակացություն 29](#_Toc133418710)

[Օգտագործված գրականության ցանկ 30](#_Toc133418711)

# ԳԼՈՒԽ 1․ Աշխատանքի նպատակ, խնդրի սահմանում և առանձնահատկություն

## Աշխատանքի նպատակը

Հետազոտական աշխատանքի նպատակն է ուսումնասիրել ծրագրավորման հնարավորությունների միջոցով շաշկի խաղի ստեղծման հիմունքները ու դրա առանձնահատկությունները, ինչպես նաև ձեռք բերել որոշակի հմտություններ GUI ծրագրեր ստեղծելու գործում: C# ծրագրավորման լեզուն բավականին շատ ընդհանրություններ ունի C++ ծրագրավորման լեզվի հետ, այդ իսկ պատճառով ես ընտրեցի այն GUI ծրագիր ստեղծելու համար։

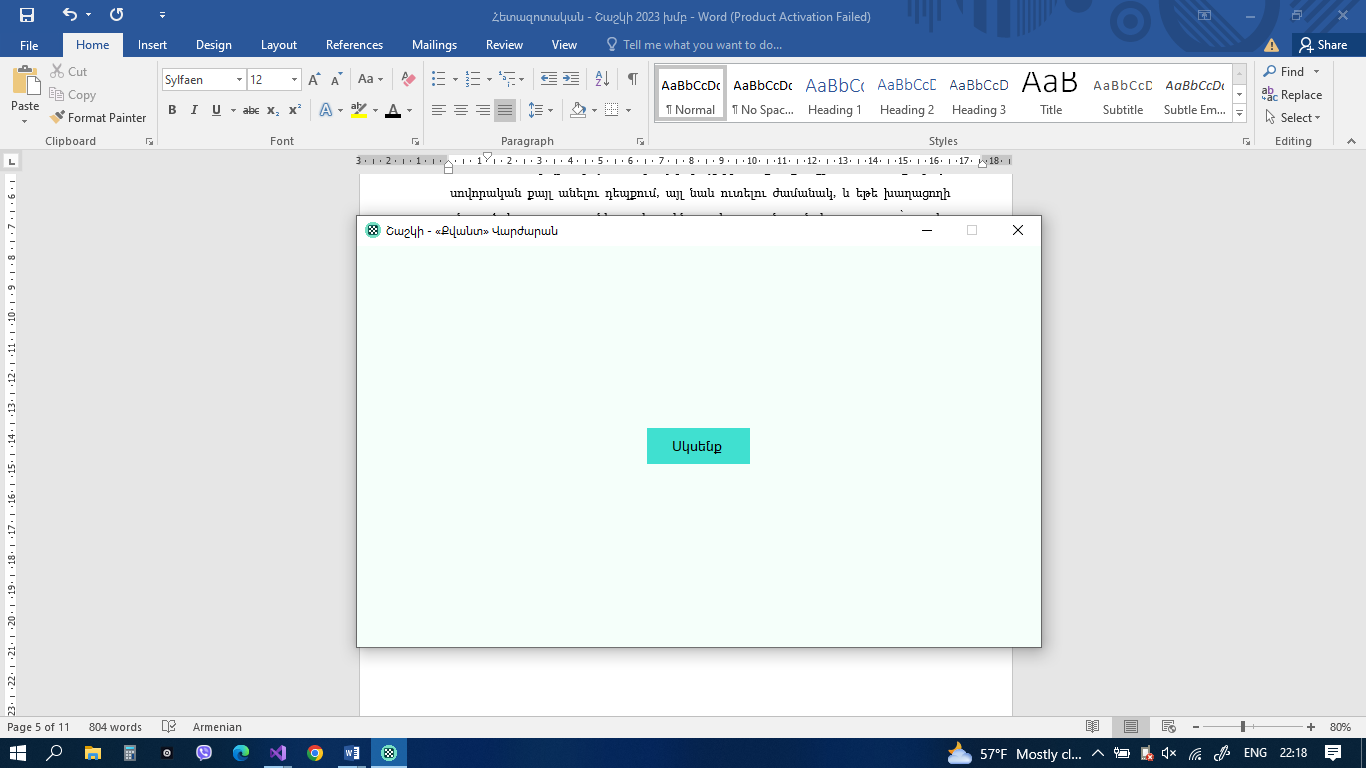


**Շաշկի**, խաղ 8x8 տախտակի վրա երկու խաղացողի միջև։ Շաշկին խաղացվում է հատուկ խաղաքարերով։ Խաղում են երկուսով՝ երկգույն վանդակներով քառակուսի խաղատախտակի վրա, սև ու սպիտակ(կարմիր և սև) խաղաքարերով։ Նպատակն է ոչնչացնել մրցակցի խաղաքարերը կամ խաղատախտակի վրա ստեղծել այնպիսի դրություն, որ դրանք զրկվեն քայլ կատարելու հնարավորությունից։

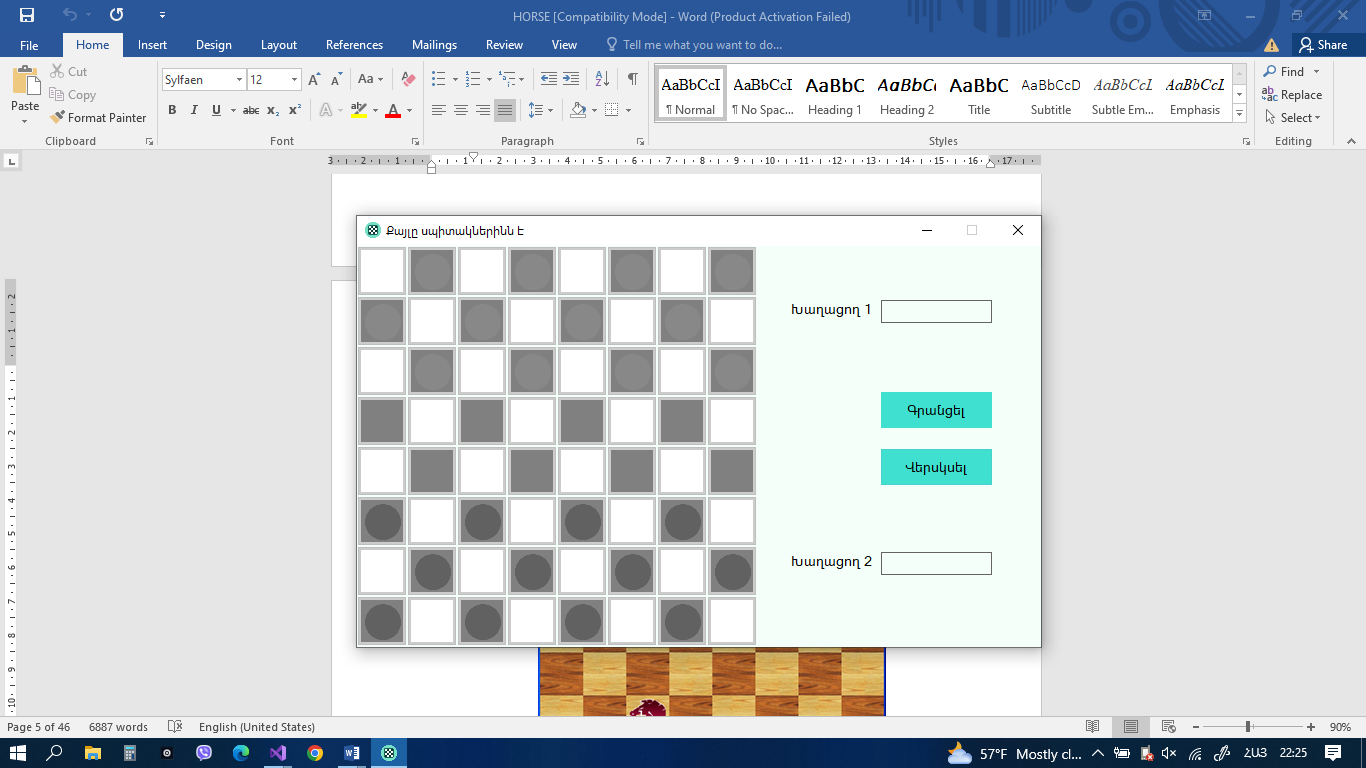
Գոյություն ունեն շաշկու բազմաթիվ տարատեսակներ։ Կան շաշկու ռուսական, անգլիական, ամերիկյան, իտալական, կանադական և այլ տեսակներ։ Տվյալ հետազոտական աշխատանքում ես կստեղծեմ շաշկու ամերիկյան տարատեսակի համակարգչային տարբերակը։

Ամերիկյան շաշկին խաղում են 64 սև ու սպիտակ դաշտերի բաժանված խաղատախտակի վրա՝ 12-ական սև ու սպիտակ խաղաքարերով, որոնք շարվում են առաջին երեք հորիզոնական շարքերի բոլոր սև դաշտերի վրա։ Քայլ է անվանվում խաղաքարի տեղաշարժը դեպի հարևան ազատ դաշտը (հանդարտ քայլ)։ Կան նաև հարվածող քայլեր, երբ քայլ կատարելու իրավունք ստացած խաղացողը «ուտում» է, այսինքն՝ խաղատախտակից հանում է հակառակորդի՝ սեփական խաղաքարի ճանապարհին հարվածի տակ հանդիպող մեկ և ավելի խաղաքարեր։ Սրանք խաղատախտակից հեռացվում են քայլի ավարտից հետո միայն։ Չի կարելի նույն խաղաքարի վրայով անցնել երկու անգամ։ Լինում են հասարակ խաղաքարեր և «դամաներ». «դամա» է դառնում վերջին հորիզոնականին հասած հասարակ խաղաքարը, որը կարող է անկյունագծով ազատ շարժվել 1 դաշտ։

## Առանձնահատկությունը

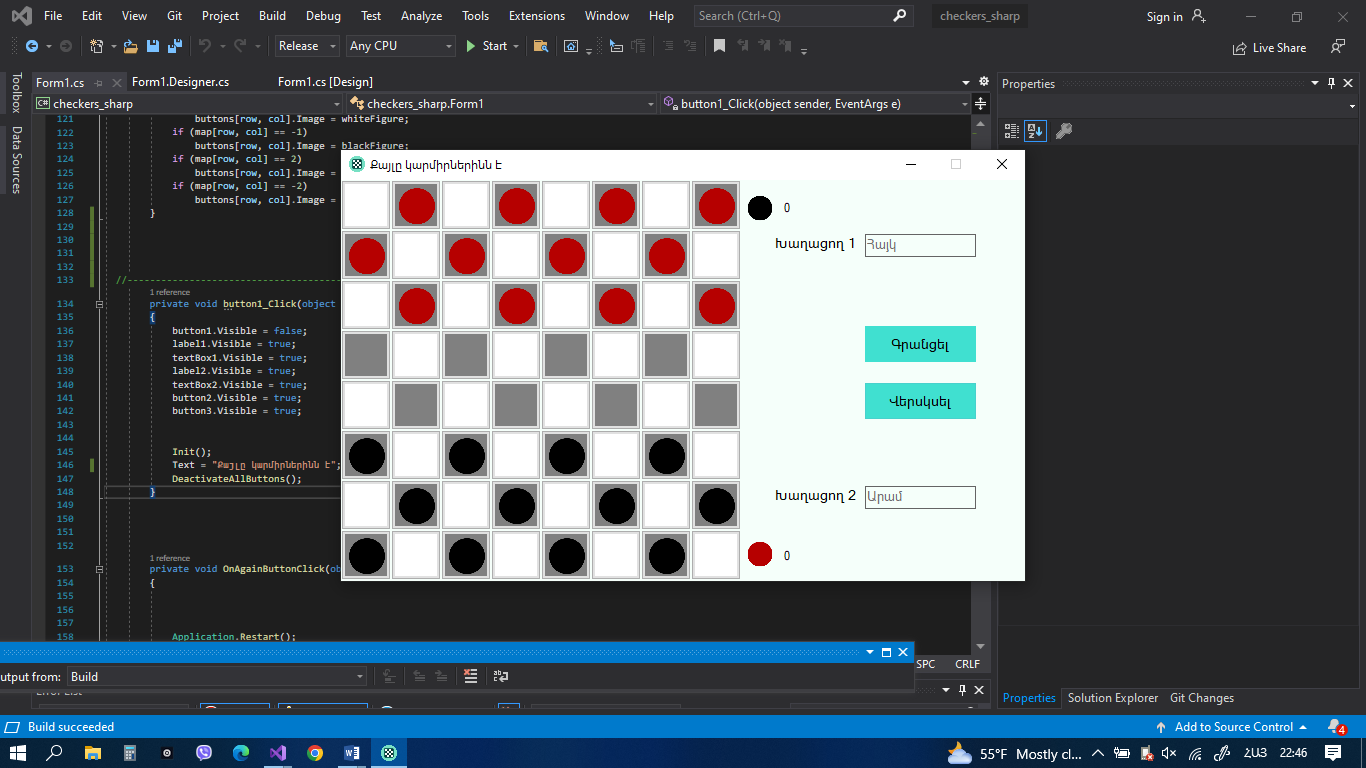
Խաղի առանձնահատկությունն այն է, որ այն ցույց է տալիս փորձարկողների (խաղացողների) ընտրած խաղաքարի բոլոր հնարավոր քայլերը։ Ըստ խաղի ալգորիթմի ամեն անգամ, երբ խաղացողները ընտրում են որևէ խաղաքար, այն խաղադաշտերը՝ որտեղ խաղացողների ընտրած խաղաքարերը կարող են քայլ անել ստանում են դեղին գույն։ Հնարավոր քայլերի հուշումը աշխատում է ոչ միայն սովորական քայլ անելու դեպքում, այլ նաև ուտելու ժամանակ, և եթե խաղացողի ընտրած խաղաքարը ունի ուտելու հնարավորություն, այն խաղադաշտը՝ որտեղ կհայտնվի խաղաքարը ուտելուց հետո ստանում է դեղին գույն, հետո ալգորիթմը ստուգում է, եթե ընտրած խաղաքարը ունի էլի ուտելու հնարավորություն, ապա այն խաղադաշտը՝ որտեղ կհայտնվի խաղաքարը ուտելուց հետո ավտոմատ ստանում է դեղին գույն։ Էկրանին պետք է երևա պատուհան, պատուհանը պետք է ունենա <Շաշկի - «Քվանտ» Վարժարան> անվանումը, պատուհանի կետնրոնում տեղադրված է կոճակ [Սկսենք]։ 

Փորձարկողներից (խաղացողներից) մեկը սեղմելով [Սկեսենք] կոճակը, կերևա նոր պատուհան, որտեղ տեղադրված է շաշկու տախտակ, 2 կոճակ՝ [Գրանցել] և [Վերսկսել], խաղացողների անունների համար նախատեսված արկղեր(textBox)։

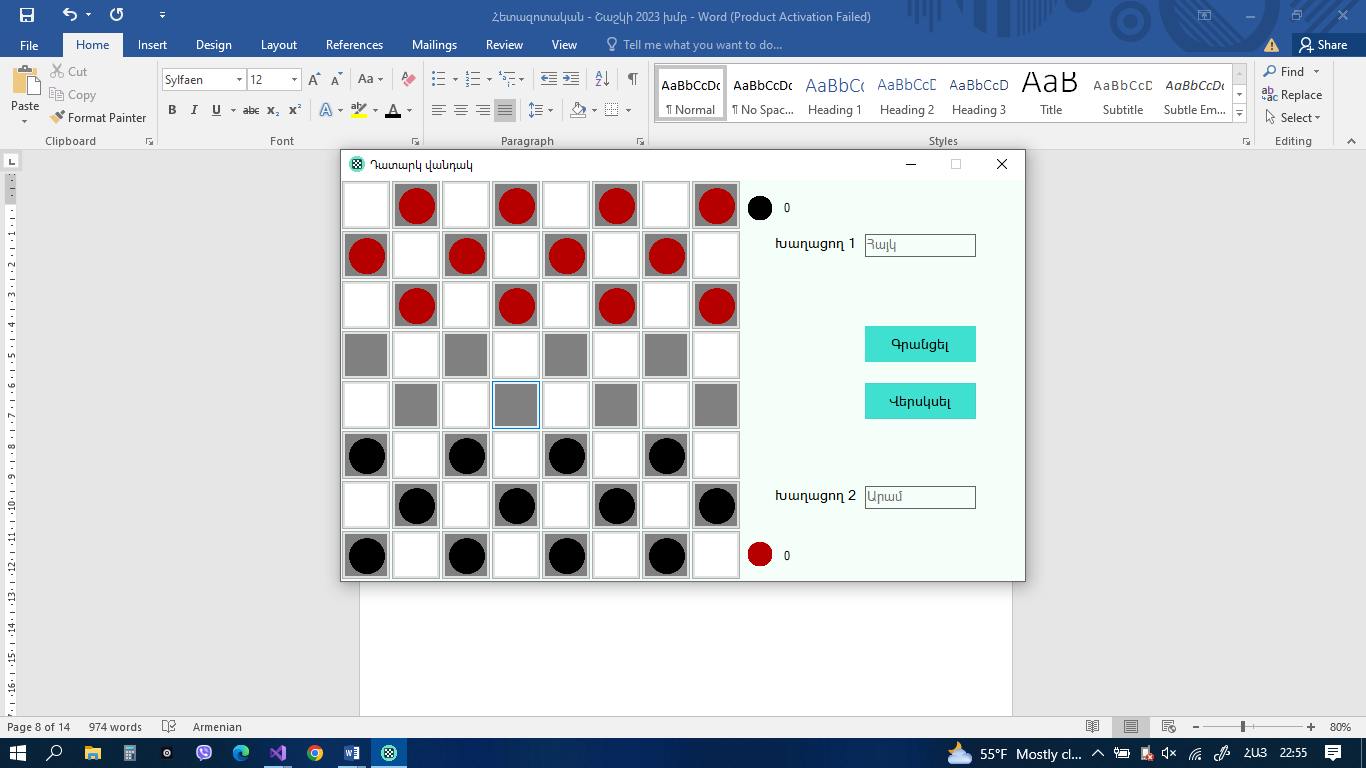


Փորձարկողները (խաղացողները) պետք է որոշեն թե ով է խաղում սպիտակների(կարմիրների) խաղաքարերով, ով սևերի։ Այն խաղացողը՝ ով պետք է խաղա սպիտակների(կարմիրների) խաղաքարերով, պետք է իր անունը ներմուծի Խաղացող 1-ի կողքի արկղում, իսկ մյուս խաղացողը Խաղացող 2-ի կողքի արկղում։ Այն բանից հետո երբ խաղացողները ներմուծեցին իրենց անունները, պետք է սեղմեն [Գրանցել] կոճակը, և խաղատախտակը կակտիվանա, մինչև խաղացողները չներմուծեն իրենց անունները և չսեղմեն [Գրանցել] կոճակը՝ խաղատախտակը կմնա ապաակտիվացված, և խաղացողները չեն կարողանա որևէ քայլ անել՝ խաղադաշտերը չեն արձագանքի մկնին։

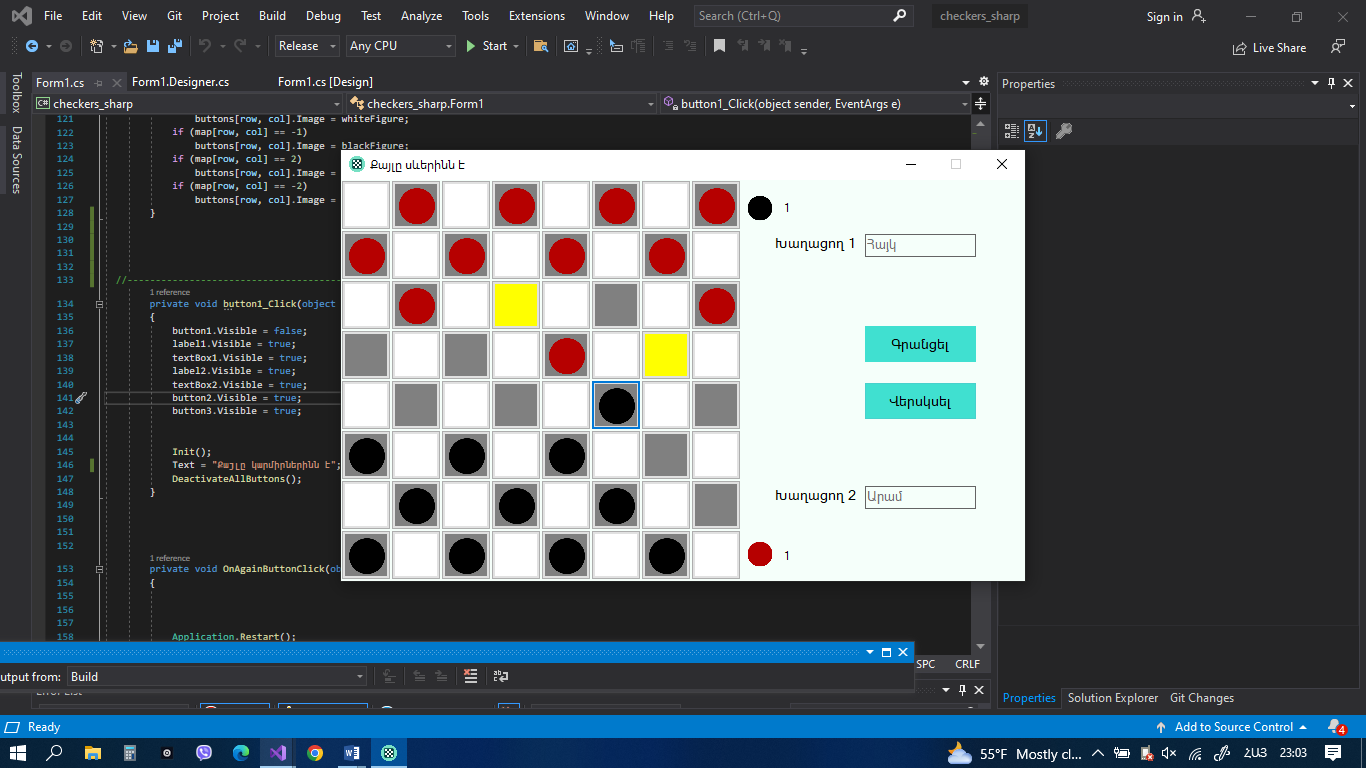
Այն բանից հետո, երբ խաղացողները ներմուծեցին իրենց անունները և սեղմեցին [Գրանցել] կոճակը, խաղատախտակը կակտիվանա, և խաղադաշտերը կսկեն արձագանքել մկնիկին, կհայտնվի հաշիվ՝ ցույց կտա, թե ամեն խաղացող հակառակորդի քանի հատ խաղաքար է կերել, textBox-երը կապաակտիվացվեն և հնարավորույթուն չի լինի խաղացողների անունները փոփոխել, պատուհանի անվանման հատվածում կերևա թե ում հերթն է խաղալու։



Քանի որ խաղատախտակի յուրաքանչյուր խաղադաշտ կոճակ է(button), ուստի երբ խաղացողները մկնիկը պահում են որևէ խաղադաշտի վրա, այդ խաղադաշտը արձագանքում է։ Եթե խաղադաշտի վրա, որևէ խաղաքար չկա, ապա պատուհանի անվանման հատվածում, ասվում է որ վանդակը դատարկ է։



Եթե խաղի ընթացքում խաղացողները որոշեն նորից սկսել խաղը, ապա նրանք պետք է սեղմեն [վերսկսել] կոճակը, երբ [վերսկսել] կոճակը սեղմվի՝ խաղը նորից կսկվի, այնսինք ամենասկզբի պատուհանը կբացվի՝ կենտրոնում [Սկսենք] կոճակը։



# ԳԼՈՒԽ 2․ C# պատմություն, կլասեր

## Պատմություն

C# ծրագրավորման լեզուն ստեղծվել է 2000 թվականի հունիսին՝ Անդերս Հեյլսբերգի գլխավորած Microsoft-ի ծրագրավորողների մեծ թիմի քրտնաջան աշխատանքի արդյունքում։ Այս մարդը հայտնի է որպես IBM ​​համակարգիչների համար կազմված առաջին ծրագրավորման լեզուներից մեկի՝ Turbo Pascal-ի հեղինակ: Հավանաբար նախկին Խորհրդային Միության տարածքում շատ փորձառու ծրագրավորողներ և պարզապես մարդիկ, ովքեր այս կամ այն ​​ձևով ծրագրավորում են սովորել համալսարաններում, զգացել են այս արտադրանքի հմայքն ու օգտագործման հեշտությունը:

## Կլասեր

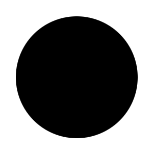
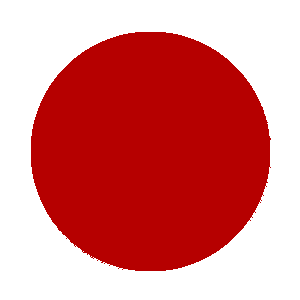
**Class**-ը (դասը) աբստրակտ տվյալների տիպ է (abstract data type): Class ի միջոցով սահմանվում են որոշակի էություններ (entities) (հատկանիշները և հնարավոր գործողությունները): Դասը հանդիսանում է ելակետային կոդի տերմինալոգիայով նկարագրված, դեռ գոյություն չունեցող օբյեկտի մոդել։ Փաստացի այն նկարագրում է օբյեկտի կառուցվածքը, հանդիսանալով օբյեկտի ‹‹գծագիր››։ Օրինակ՝ class-ը կարող է նկարագրել մրգերը, ավտոմեքենաները և այսպես շարունակ:

**Object**: Սահմանելով class-ը՝ կարող ենք սահմանել նրա որևէ օրինակ-օբյեկտ (instace-object): Object-ը արդեն Class-ի կոնկրետ ներկայացումն է:

* Կլասի private անդամներին կարող են դիմել միայն այդ կլասի ուրիշ անդամները:
* protected անդամներին կարող են դիմել միայն այդ կլասի ուրիշ անդամները, ժառանգող կլասի անդամները :
* public անդամներին կարող եք դիմել այն բոլոր տեղերից, որտեղ որ այդ կլասը տեսանելի է:

# ԳԼՈՒԽ 3․ Խաղի ինտերֆեյսի ձևավորում

Ծրագրին գեղեցիկ տեսք տալու համար նախապես Photoshop ծրագրով ստեղծվում են խաղաքարերը։

Խաղի պատուհանը ունի 700 լայնություն և 440 բարձրությունը։ Յուրաքանչյուր խաղադաշտ կստեղծվի 50 պիքսել լայնությամբ և նույնքան բարձրությամբ։ Իսկ խաղաքարերը կստեղծվեն 45 պիքսել լայնությամբ և նույքան բարձրությամբ՝ որպեսզի ամբողջ խաղադաշտը խաղաքարը չզբաղեցնի։

Յուրաքանչյուր խաղադաշտ իրենից ներկայացնում button։ Button-ի ընտրությունը պայմանավորված է նրանով, որ ի տարբերություն PicturBox-ի Button-ը արձագանքում է մկնիկի հպումներին։ Իսկ խաղաքարերը ներկայցվում են Image ղեկավարման էլեմենտի միջոցով։

Image whiteFigure;

Image blackFigure;

Image whitekingFigure;

Image blackkingFigure;

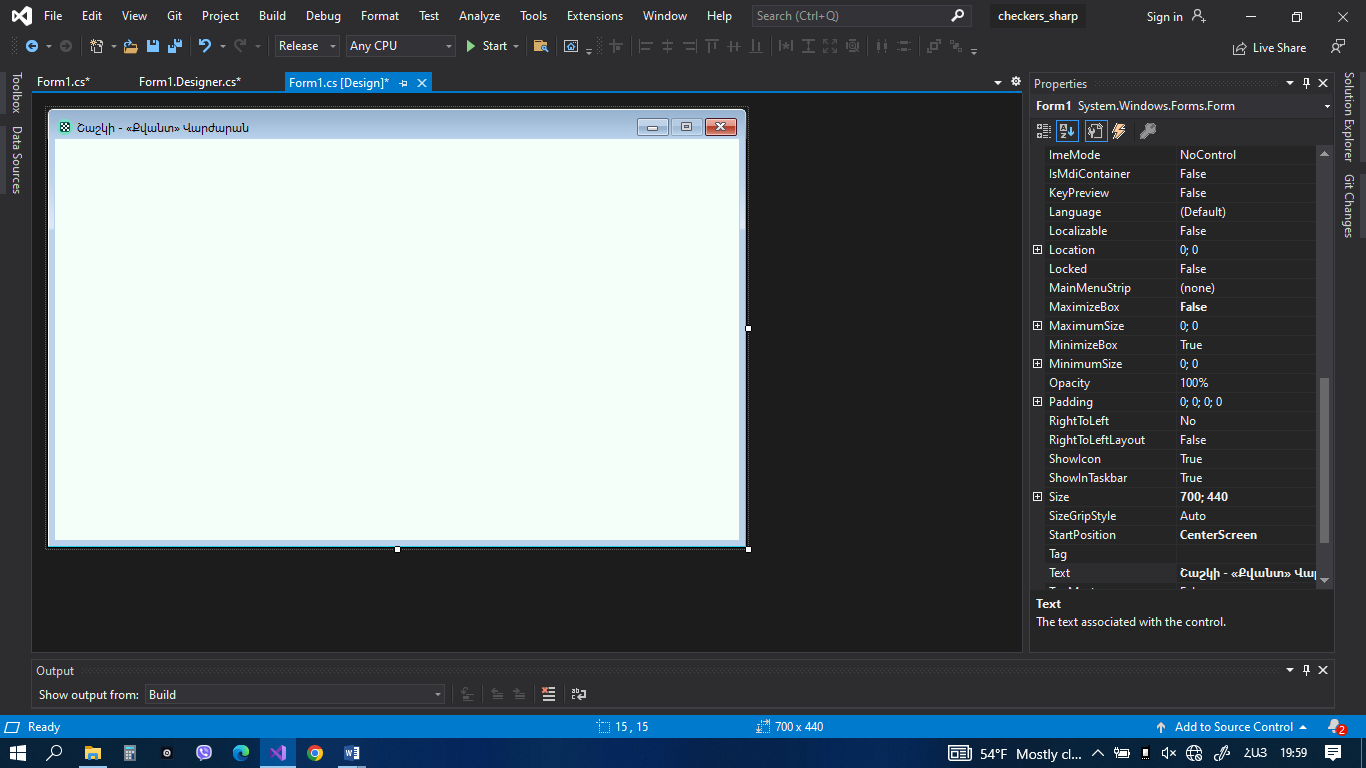
whiteFigure = new Bitmap(new Bitmap(@"w.png"), new Size(45, 45));

blackFigure = new Bitmap(new Bitmap(@"b.png"), new Size(45, 45));

whitekingFigure= new Bitmap(new Bitmap(@"wk.png"), new Size(45, 45));

blackkingFigure = new Bitmap(new Bitmap(@"bk.png"), new Size(45, 45));

Կոդի մեջ ստեղծում ենք Bitmap օբյեկտներ՝ խաղաքարերի .png նկարներով, յուրաքանչյուր bitmap օբյեկտ 45 պիքսել լայնությամբ և բարձրությամբ։



Մենք կարող ենք պատուհանի կարգավորումները տալ properties պատուհանում, բայց կարող ենք նաև հենց կոդի մեջ փոխել պատուհանի կարգավորումները՝ չափսերը, անվանումը և այլն։ Մեր դեպքում մենք պատուհանի անվանումը սահմանել ենք properties-ում, իսկ պատուհանի չափսերը հենց կոդի մեջ։ Եթե մենք պատուհանի չափսերը սահմանենք properties-ում, բայց հետո նաև կոդի մեջ սահմանենք, ապա պատուհանը կստանա այն չափսերը, որոնք սահմնաված են կոդի ներսում։ Գեղեցկության համար մենք նախապես ստեղծում ենք ծրագրի լոգոն(icon) և properties-ում սահմանում ծրագրի լոգոն(icon):

Properties-ում մենք սահմանում ենք, որ ծրագրի պատուհանի սկզբնական դիրքը՝ StartPosition-ը լինի մեր համակարգչի էկրանի կենտրոնը՝ CenterScreen: Properties-ում արգելում ենք մեծացնել ծրագրի պատուհանը MaximizesBox-ը դարձնում ենք false: Եթե սա անեինք կոդի մեջ, ապա կգրեինք MaximizeBox = false: Մենք արգելում ենք մեծացնել ծրագրի պատուհանը, քանի որ երբ պատուհանը մեծացվի, խաղատախտակը, կոճակները չեն մեծանում, և ծրագրի պատուհանը մնում է դատարկ։

Այսպիսով այն ամենը ինչ մենք կարող ենք սահմանել properties-ում, կարող ենք սահմանել հենց կոդի մեջ։ Ինչպես օրինակ ծրագրի պատուհանի ետնագույնը(BackColor), կոդի մեջ՝ BackColor = Color.MintCream : Սա խաղի ինտերֆեյսի ձևավորման սկզբնական հատվածն էր։

Մինչև խաղատախտակի ինտերֆեյսի ձևավորմանն անցնելը, եկենք խոսենք կոճակների, խաղացողների անունների ներմուծման համար նախատեսված TextBox-երի, Խաղացող 1 և Խաղացող 2 գրությունների , ինչպես նաև մնացած ամենի ինչի մասին, իսկ հետո արդեն կխոսենք խաղատախկտակի ձևավորման մասին։

[Սկսենք], [Գրանցել], [Վերսկսել] կոճակները սովորական button-եր են։ Կոճակների գույները, չափսերը, տեղադրությունը սահմնված են Properties-ում, բայց ինչպես արդեն նշել եմ, դրանք կարող ենք սահմանել նաև կոդի մեջ։ Այս կոճակների կատարող գործողությունների(Event) մասին կխոսենք հաջորդ գլխում։ Ամենասկզբում երբ բացի [Սկսենք] կոճակից, ուրիշ ոչինչ չի երևում, դա այն պատճառով, որ մենք մնացած ամեն ինչը՝ կոճակները, TextBox-երը, PictureBox-երը, label-երը դարձրել ենք անտեսանելի, այսինքն ոչ միայն չենք տեսնում դրանք, այլ նաև չենք կարող որևէ կերպ օգտագործել դրանք։ Խաղատախտակը «նկարելու» գործընթացը սկսելու համար կա Init() ֆունկցիա, որը սկսում է աշխատել միայն այն ժամանակ, երբ մենք կտացնենք [Սկսենք] կոճակին։ Այն ժամանակ երբ մենք կտացնում ենք [Սկսենք] կոճակին, այն դառնում է անտեսանելի՝ չենք կարող որևէ կերպ օգտագործել, իսկ մնացած ամեն բան դանում է տեսանելի, ինչպես նաև սկսում է Init() ֆունկցիան։

Խաղացողների անունները ներմուծելու համար վանդակները TextBox-եր են, մենք հենց properties-ում սահմանել ենք դրանց չափսերը, եզրագծերի ձևերը, TextBox-երի մեջ ներմուծվող տեքստի ձը, չափսը, TextBox-երի ետնագույները(BackColor):

Խաղացող 1 և Խաղացող 2 գրությունները label-ներ են, նախապես properties-ում սահմանել ենք դրանց չափսերը, և ձևերը։

Խաղացողների անունները ներմուծելուց, [Գրանցել] կոճակը սեղմելուց հետո, ցուցադրվում է թե յուրաքանչյուր խաղացող հակառակորդի քանի խաղաքար է կերել, ինչպես նաև մինչև այս ապաակտիվացած խաղատախտակը ակտիվանում է, խաղատախտակի ապաակտիվացման և ակտիվացման համար կան առանձին ֆունկցիաներ, որոնց կանդրադառնանք հաջորդ Գլխում։ ԻՆչպես արդեն նախկինում նշել էի ծրագրի պատուհանի անվանման հատվածում նշվում է թե, որ խաղացողի հերթն է, դա արվում է հենց կոդի մեջ Text = "Քայլը սևերինն է"; կամ Text = "Քայլը կարմիրներինն է"։

Խաղատախտակի կողքին ինչպես արդեն նշել էի՝ ցուցադրվում է թե յուրաքանչյուր խաղացող հակառակորդի քանի խաղաքար է կերել, հաշվի թվերը label-ներ են, իսկ խաղաքարերի պատկերները PictureBox-եր։ PictureBox-երի չափսերը, տեղադրությունը սահմանել ենք properties-ում։

Խաղատախտակի ինտերֆեյսի ձևավորումը տեղի է ունենում, կոդի մեջից, մենք օգտագործում ենք առաձին ֆունցկիաներ, որոնց կանդրադառնանք հաջորդ Գլխում։

Իսկ ղեկավարման էլեմենտների տեսանելիության(visible) համար կարող ենք properties-ում visible-ը դարձնենք false կամ true, կամ էլ կարող ենք կոդի մեջ գրել՝

button1.Visible = false;

label1.Visible = true;

textBox1.Visible = true;

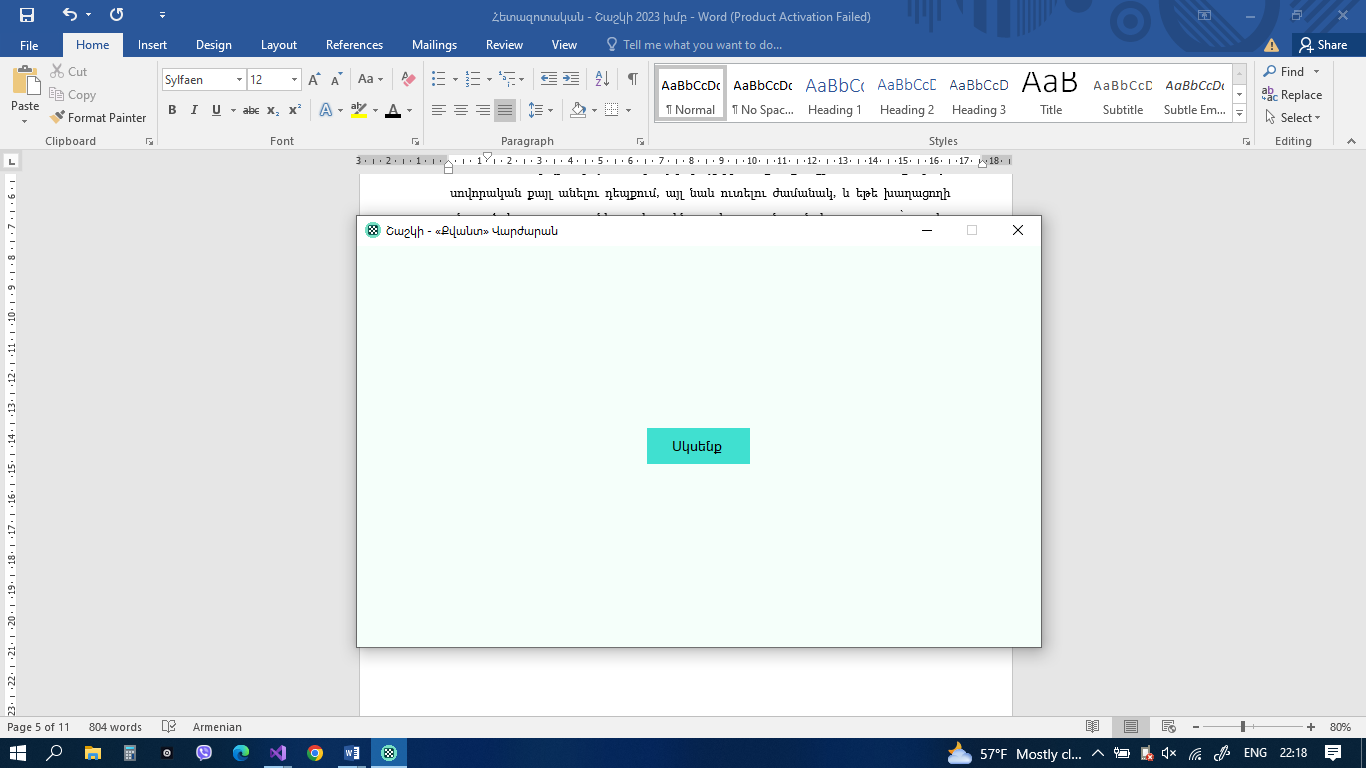
label2.Visible = true;

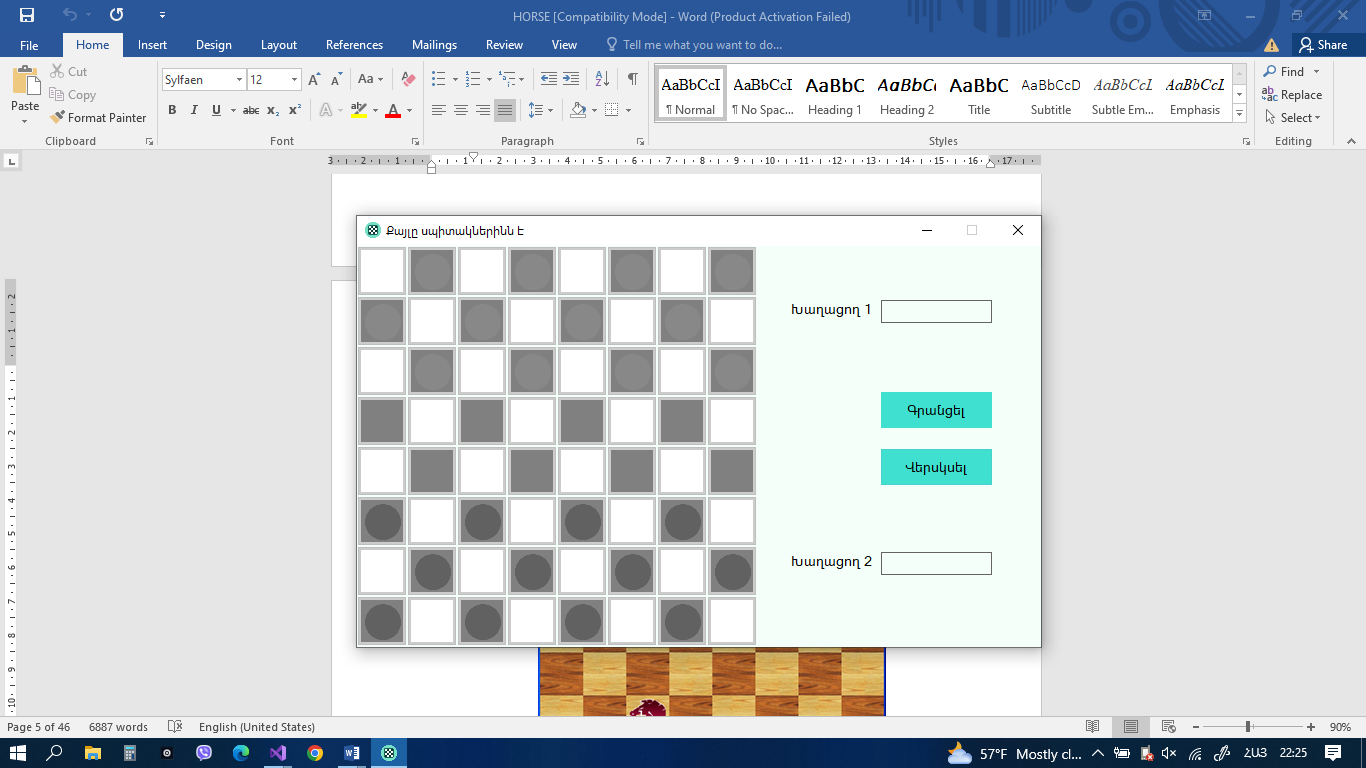
textBox2.Visible = true;

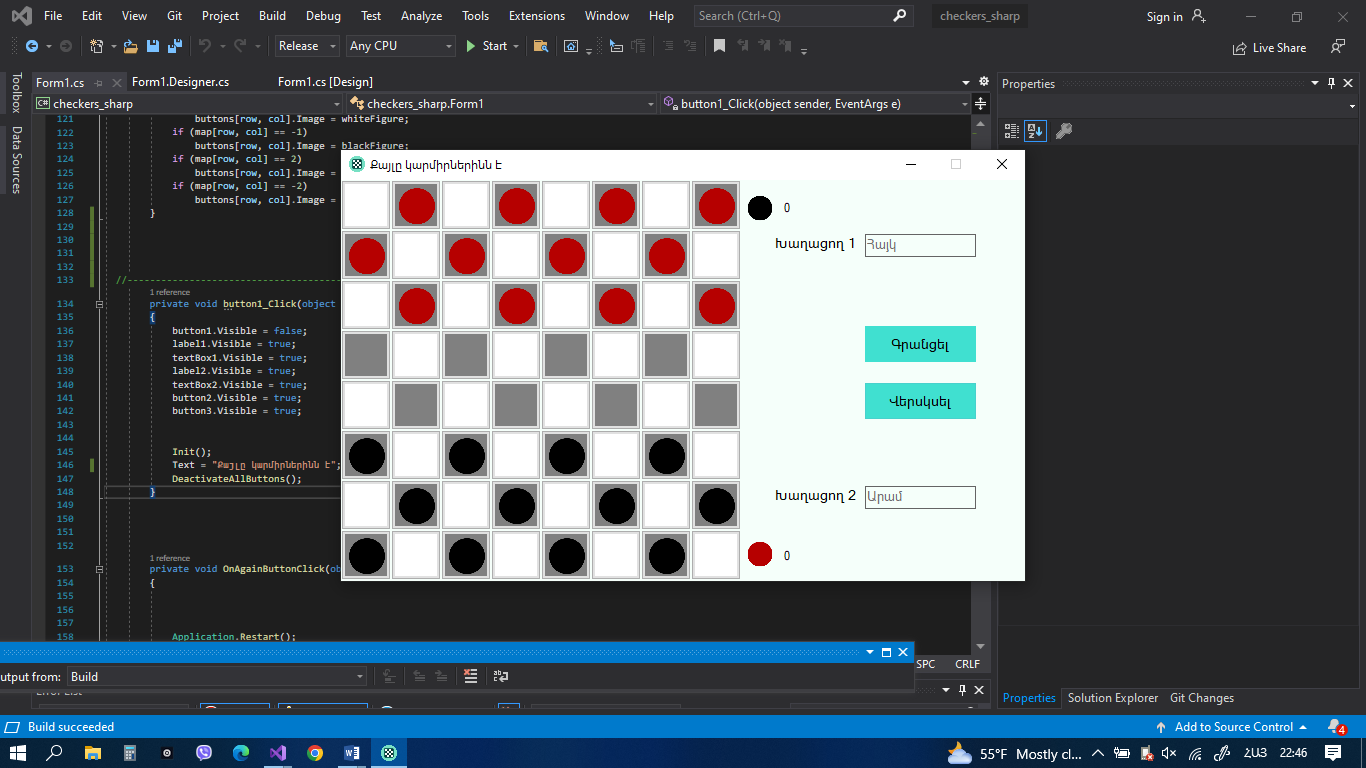
button2.Visible = true;

button3.Visible = true; ։

Պատուհանի վերջնական արտաքին տեսքը՝ օգտագործման տարբեր դեպքերում, կլինի։







# ԳԼՈՒԽ 4․ Ծրագրային կոդի ստեղծում

## Հայտարարություններ

* Նախ հայտարարում ենք intenger տիպի երկչափ զանգված՝ 8 տող, 8 սյուն։

int[,] map = new int[8, 8];

Integer տիպի ընտրությունը պայմանավորված է նարանով, որ մենք օգտագործելու ենք փոքր թվեր` 0; 1; -1; 2; -2 և Long տիպը կիրառելու անհրաժեստություն չկա:

* Հայտարարում ենք intenger տիպի row, col փոփոխականներ։

int row, col;

Integer տիպի ընտրությունը պայմանավորված է նարանով, որ մենք օգտագործելու ենք փոքր թվեր` 0-ից մինչև 8 և Long տիպը կիրառելու անհրաժեստություն չկա: Այս փոփոխականներին Այս փոփոխականների մեջ կսահմանվեն ընտրված խաղադաշտի տողը և սյունը։

* Հայտարարում ենք intenger տիպի curRow, curCol փոփոխականներ։

int curRow, curCol;

Integer տիպի ընտրությունը պայմանավորված է նարանով, որ մենք օգտագործելու ենք փոքր թվեր` 0-ից մինչև 8 և Long տիպը կիրառելու անհրաժեստություն չկա: Այս փոփոխականներին կվերագրվեն row և col փոփոխականների արժեքները համապատասխանաբար։

* Հայտարարում ենք Image ղեկավարման էլեմենտներ, յուրաքանչյուր խաղաքարի համար։

Image whiteFigure;

Image blackFigure;

Image whitekingFigure;

Image blackkingFigure;

* Հայտարարում ենք Button տիպի երկչափ զանգված՝ 8 տող, 8 սյուն։ Այս զանգվածը իր մեջ կարող է ներառել 64 հատ Button տիպի օբյեկտ։

Button[,] buttons = new Button[8, 8];

* Հայտարարում ենք integer տիպի currentPlayer փոփոխական։

int currentPlayer;

Integer տիպի ընտրությունը պայմանավորված է նրանով, որ մենք օգտագործելու ենք փոքր թվեր` 1; -1 և Long տիպը կիրառելու անհրաժեստություն չկա: Տվյալ փոփոխականը կօգտագործենք որոշելու համար թե ում հերթն է խաղալու։

* Հայտարարում ենք bool տիպի currentPlayerKing փոփոխական, և տալիս ենք նրան false արժեքը։

bool currentPlayerKing=false;

Bool տիպի ընտրությունը պայմանավորված է նրանով, որ մենք ընացքում պետք է ստուգենք, ընտրված խաղաքարը դամա է թե ոչ։ Բայց մենք կարողեինք նաև վերցնել int տիպի փոփոխական և փոխել նրա արժեքը եթե ընտրված խաղաքարը դամա է։

* Հայտարարում ենք integer տիպի black և white փոփոխականներ, և տալիս նրանց 0 արժեք։

int white = 0, black = 0;

Integer տիպի ընտրությունը պայմանավորված է նրանով, որ ամենամեծ արժեքը, որը կարող են ստանալ տվյալ փոփոխականները, 12-ն է և այդ իսկ պատճառով Long տիպը կիրառելու անհրաժեշտություն չկա: Տվյալ փոփոխականներում պահվելու են խաղացողների կերած խաղաքարերի թվաքանակները։

* Հայտարարում ենք integer տիպի yellow փոփոխական և տալիս նրան 0 արժեք։

int yellow = 0;

Integer տիպի ընտրությունը պայմանավորված է նրանով, որ ամենամեծ արժեքը, որը կարող է ստանալ տվյալ փոփոխականը, 3-ն է և այդ իսկ պատճառով Long տիպը կիրառելու անհրաժեշտություն չկա:

Այսքանով հայտարարությունների բաժինն ավարտվում է։

## Ծրագրի գործարկում

Ծրագիրը գործարկելիս կանչվում է private void Form1\_Load(object sender, EventArgs e) պրոցեդուրան: Այս պրոցեդուրան պատասխանատու է համակարգչի հիշողությունում պատուհանը բեռնավորելու համար, ինչպես նաև կարող է օգտագործվել նախնական արժեքավորումներ կատարելու համար:

private void Form1\_Load(object sender, EventArgs e) պրոցեդուրայում կատարվում է հետևյալը՝

* Սահմանվում է ծրագրի պատուհանի լայնությունն ու բարձրությունը
* Ստեղծվում են bitmap օբյեկտներ, յուրաքանչյուր խաղաքարի նկարից
* Ստեղծվում է Size օբյեկտ՝ յուրաքանչյուրը 45 լայնությամb և նույնքան բարձրությամբ
* Size օբյեկտը որպես պարամետր փոխանցվում է Bitmap կոնստրուկտորին
* Կոնստրուկտորը վերցնում է նախորդ ստեղծած bitmap օբյեկտը և Size օբյեկտը, ստեղծում մի նոր bitmap օբյեկտ, որը նախորդ bitmap օբյեկտի չափափոխած տարբերակն է
* Label3.Text-ին տրվում է white փոփոխականի արժեքը, քանի որ Label3.Text կարող է ստանալ միայն string, այդ իսկ պատճառով white փոփոխականի արժեքին գումարում ենք “”, այդպես white-ի արժեքին գումարելով դատարկ տող, մենք ստանում ենք տող, և դա կարողենք վերագրել Label3.Text – ին։
* Label4.Text-ին տրվում է black փոփոխականի արժեքը, քանի որ Label4.Text կարող է ստանալ միայն string, այդ իսկ պատճառով black փոփոխականի արժեքին գումարում ենք “”, այդպես black-ի արժեքին գումարելով դատարկ տող, մենք ստանում ենք տող, և դա կարողենք վերագրել Label4.Text – ին։

private void Form1\_Load(object sender, EventArgs e) պրոցեդուրայի ամբողջական տեսքը՝

private void Form1\_Load(object sender, EventArgs e)

{

this.Width = 700;

this.Height = 440;

whiteFigure = new Bitmap(new Bitmap(@"w.png"), new Size(45, 45));

blackFigure = new Bitmap(new Bitmap(@"b.png"), new Size(45, 45));

whitekingFigure= new Bitmap(new Bitmap(@"wk.png"), new Size(45, 45));

blackkingFigure = new Bitmap(new Bitmap(@"bk.png"), new Size(45, 45));

label3.Text = white + "";

label4.Text = black + "";

}

## Խաղի սկիզբ

Արդեն անդրադարձել ենք նրան թե ինչ է լինում երբ մենք սեղմում ենք [Սկսենք] կոճակը։ Ինչպես գիտենք քանի դեռ անունները ներմուծված չեն, և [Գրանցել] կոճակը սեղմված չէ, խաղատախտակը մնում է ապաակտիվացած։ Խաղատախտակի ակտիվացումը և ապաակտիվացումը կատարվում է կոնկրետ ֆունցկիաների օգնությամբ։ Բայց մինչև այդ ֆունկցիաներին անդրադառնալը եկեք հասկանանք թե ինչպես է խաղատախտակը «նկարվում»։

Մենք Հայտարարություններ բաժնում ունեինք map[8, 8] երկչափ դատարկ զանգված և նրա բոլոր արժեքները 0 են։ Մենք այս երկչափ զանգվածը արժեքավորում ենք հետևյալ կերպ՝

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | **1** | **0** | **1** | **0** | **1** | **0** | **1** |
| **1** | **0** | **1** | **0** | **1** | **0** | **1** | **0** |
| **0** | **1** | **0** | **1** | **0** | **1** | **0** | **1** |
| **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **-1** | **0** | **-1** | **0** | **-1** | **0** | **-1** | **0** |
| **0** | **-1** | **0** | **-1** | **0** | **-1** | **0** | **-1** |
| **-1** | **0** | **-1** | **0** | **-1** | **0** | **-1** | **0** |

CreateMap() ֆունցկիայում մենք հերթով ստեղծում ենք կոճակներ, տալիս ենք այդ կոճակների տեղադրությունը, սահմանում ենք նրանց չափսերը՝ 50 լայնություն և նույնքան բարձրություն, Tag-եր ենք օգտագործում, յուրաքանչյուր կոճակի համար ստեղծվում է երկչափ զանգված, որն իր մեջ պահում է i-ի և j-ի արժեքները, սահմանում ենք այն գործողությունը, որը կկատարի կոճակը, եթե սեղմեն նրա վրա, Հայտարարությունների մեջ հայտարարած buttons զանգվածի [i, j] էլեմենտին վերագրում ենք տվյալ կոճակները։ Controls.Add(button)-ի միջոցով, յուրաքանչյուր ստեղծված կոճակ ավելացվում է կարգավորիչների(controls) խմբին։

Խաղատախտակի սկզբնական տեսքի համար՝

public void Init()

{

currentPlayer = 1;

map = new int[8, 8] {

{ 0,1,0,1,0,1,0,1 },

{ 1,0,1,0,1,0,1,0 },

{ 0,1,0,1,0,1,0,1 },

{ 0,0,0,0,0,0,0,0 },

{ 0,0,0,0,0,0,0,0 },

{ -1,0,-1,0,-1,0,-1,0 },

{ 0,-1,0,-1,0,-1,0,-1 },

{ -1,0,-1,0,-1,0,-1,0 }

};

CreateMap();

}

public void CreateMap()

{

for (int i = 0; i < 8; i++)

for (int j = 0; j < 8; j++)

{

Button button = new Button();

button.Location = new Point(j \* 50, i \* 50);

button.Size = new Size(50, 50);

button.Tag = new int[] { i, j };

button.Click += new EventHandler(OnFigurePress);

buttons[i, j] = button;

Controls.Add(button);

}

DrawAllCells();

}

public void DrawAllCells()

{

for (int i = 0; i < 8; i++)

for (int j = 0; j < 8; j++)

DrawCell(i, j);

}

## DrawCell() ֆունկցիա

Քանի որ մեզ մոտ խաղաքարերը նշանակվում են 1; -1; 2; -2 թվերով , իսկ դատարկ վանդակները 0-ներով, ուստի մենք կարող ենք ստեղծել ֆունցկիա, որը կստուգի թե map երկչափ զանգվածի map[i, j] էլեմենտը ինչ արժեք ունի և դրանից ելնելով կշարի խաղաքարերը, դատարկ վանդակները, ինչպես նաև հնարավոր քայլերի վանդակները կդարձնի դեղին գույն(հնարավոր քայլերի վանդակները ստանում են 3 արժեքը)։ Ֆունկցիան նախ ստուգում է եթե խաղադաշտի ինդեքսների գումարը զույգ չէ ապա խաղադաշտը սև է և նրա վրա դեռ խաղաքար չենք դնում, հակառակ դեպքում խաղադաշտը սպիտակ է և նրա վրա խաղաքար երբեք չի դրվում։ Այս ամենից հետո ստուգում է խաղադաշտի արժեքը, եթե այն 3 է(հնարավոր քայլ), ապա խաղադաշտը դառնում է դեղին։ Այսպես ֆունկցիան հերթով ստուգում է բոլոր հնարավոր արժեքները՝ 1; -1; 2; -2; 3։

DrawCell ֆունկցիայի ամբողջական տեսքը՝

public void DrawCell(int row, int col)

{

if ((row + col) % 2 != 0)

{

buttons[row, col].BackColor = Color.Gray;

buttons[row, col].Image = null;}

else

{

buttons[row, col].BackColor = Color.White;

buttons[row, col].Image = null;

}

if (map[row, col] == 3)

buttons[row, col].BackColor = Color.Yellow;

if (map[row, col] == 1)

buttons[row, col].Image = whiteFigure;

if (map[row, col] == -1)

buttons[row, col].Image = blackFigure;

if (map[row, col] == 2)

buttons[row, col].Image = whitekingFigure;

if (map[row, col] == -2)

buttons[row, col].Image = blackkingFigure;

}

Քանի որ մեզ մոտ խաղատախտակի խաղադաշտերը բաղկացած են առանձին կոճակներից ուստի մենք կարող ենք այդ կոճակները միացնել և անջատել, այդ իսկ պատճառով մենք շատ հեշտ կարող ենք գրել ֆունցկիաներ, որոնք միացնում կամ անջատում են խաղատախտակը։

Դա կարող ենք անել հետևյալ կերպ՝

public void DeactivateAllButtons()

{

for (int i = 0; i < 8; i++)

{

for (int j = 0; j < 8; j++)

{

buttons[i, j].Enabled = false;

}

}

}

public void ActivateAllButtons()

{

for (int i = 0; i < 8; i++)

{

for (int j = 0; j < 8; j++)

{

buttons[i, j].Enabled = true;

}

}

}

## ShowPossibleSteps1() ֆունկցիա

ShowPossibleSteps1() ֆունցկիային փոխանցվում են ընթացիկ փոփոխականներ row-ի և col-ի արժեքները, այսինքն այն խաղադաշտի ինդեքսները, որի վրա սեղմել են փորձարկողները (խաղացողները) ։

Նախ և առաջ curRow և curCol փոփոխականներին համապատասխանաբար վերագրում ենք row և col փոփոխականների ինդեքսները, որպեսզի վերջում ստուգենք եթե սկզբնական արժեքների և վերջնական արժեքների տարբերությունները զույգ են, ապա դա նշանակում է որ խաղացողը կերել է հակառակորդի խաղաքարը, հետևյալ կերպով մենք կգտնենք թե որտեղ է հակառակորդի խաղաքարը և կանհետացնենք դա խաղատախտակից։

if (Math.Abs(curRow - row) % 2 == 0 && Math.Abs(curCol - col) % 2 == 0)

{

map[(curRow + row) / 2, (curCol + col) / 2] = 0;

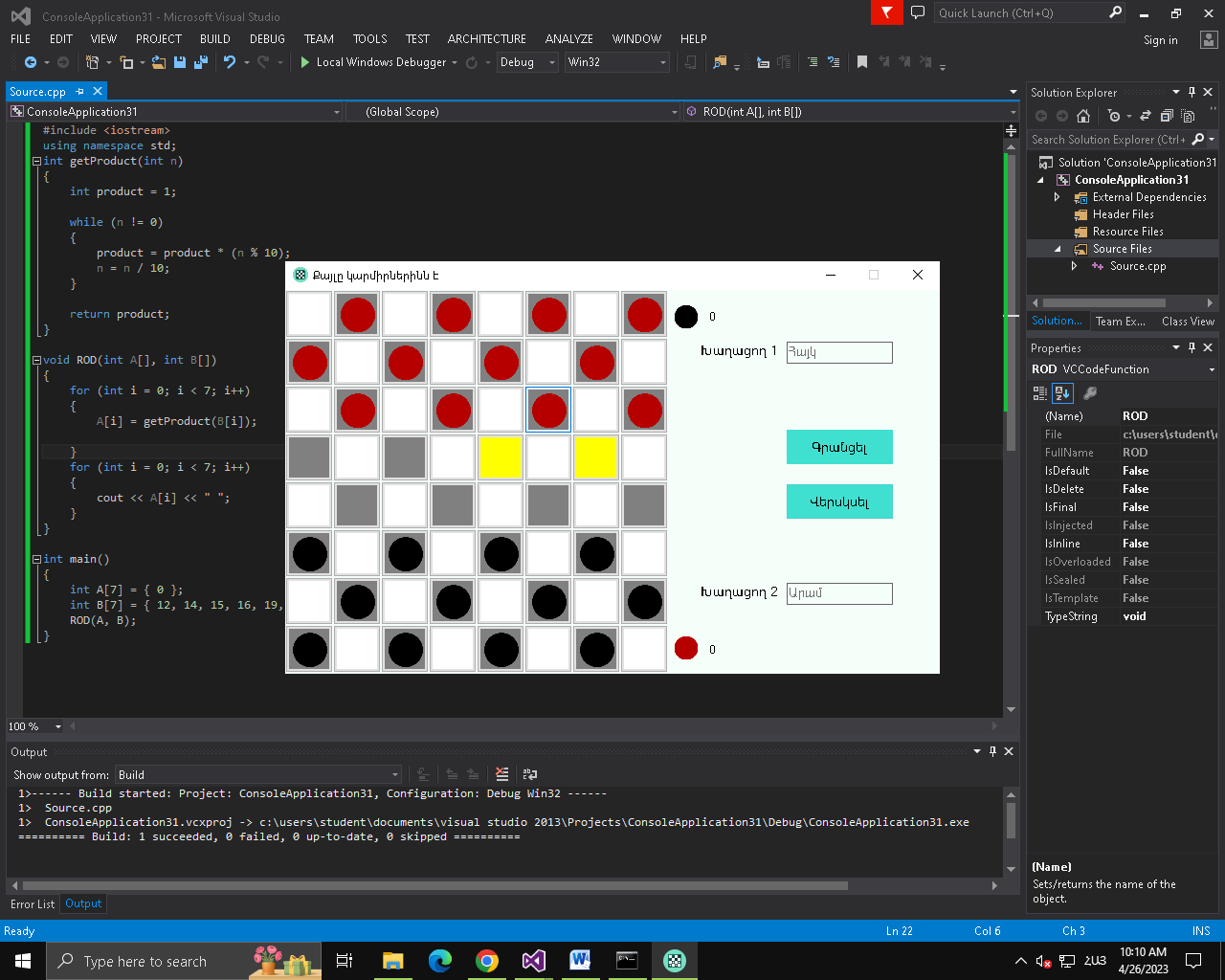
DrawCell((curRow + row) / 2, (curCol + col) / 2);

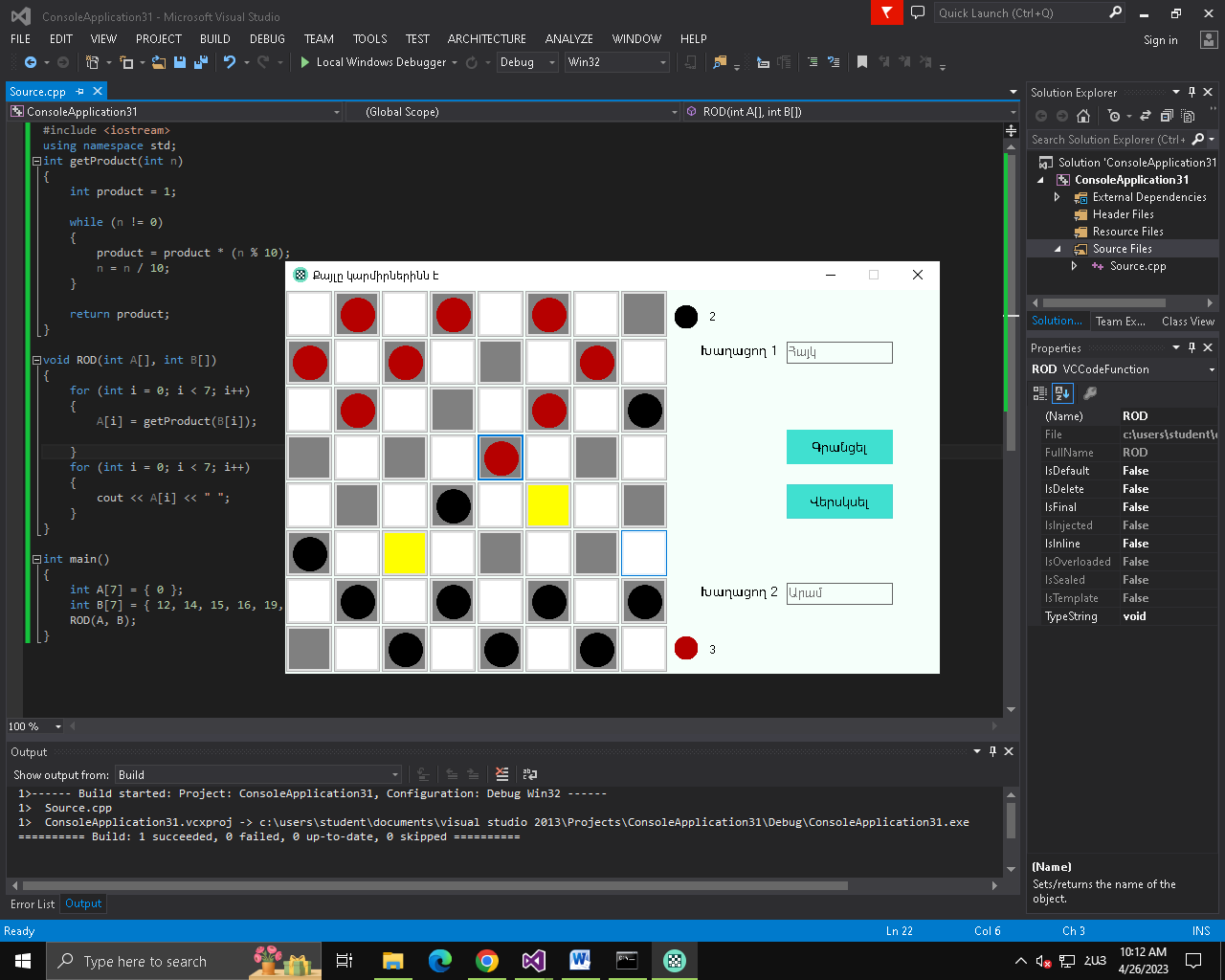
}

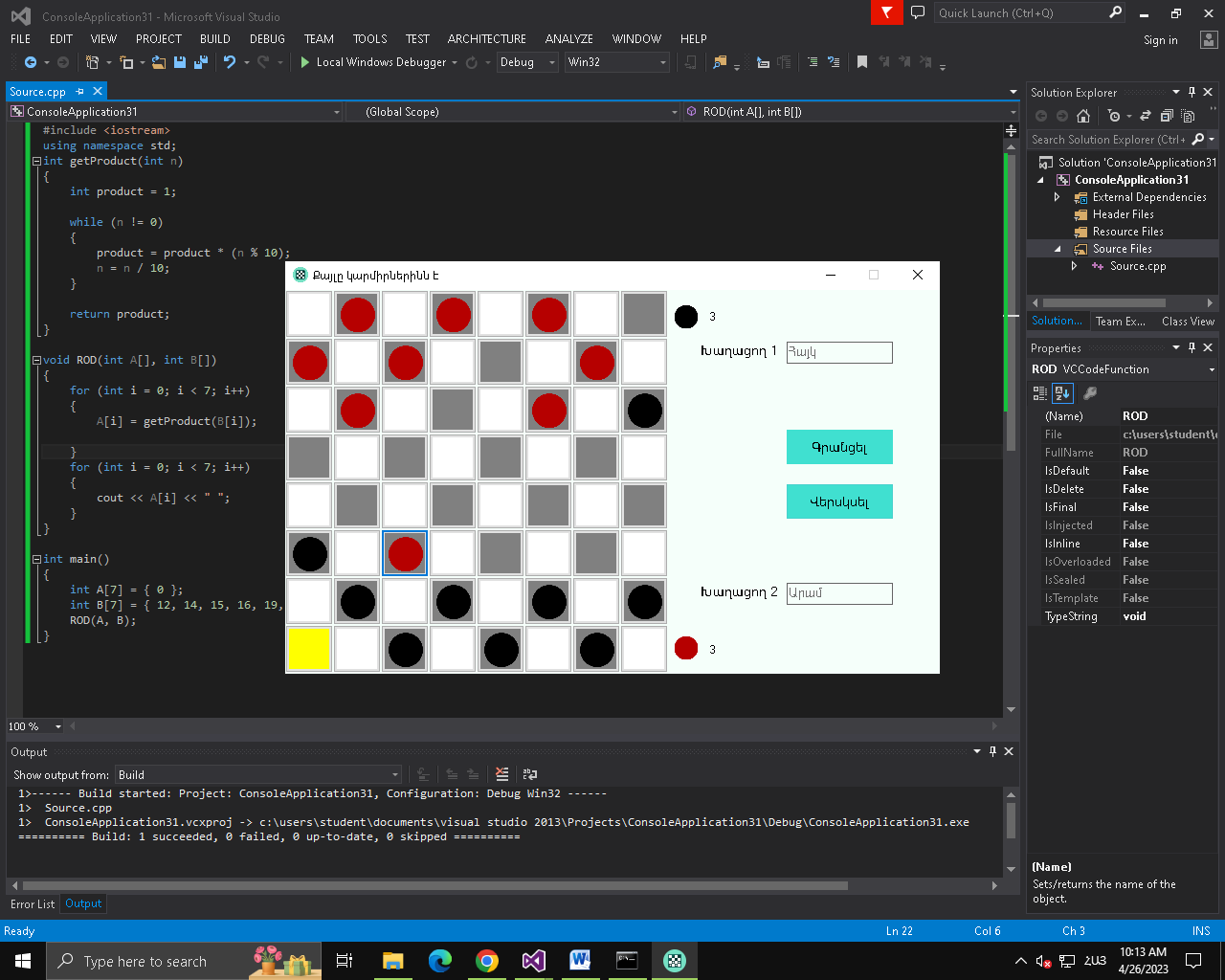
Ֆունկցիան ամենասկզբում ստուգում է արդյոք հնարավոր քայլը անելու դեպքում մենք կմնանք զանգվածի սահմաններում, իսկ հետո արդեն ստուգում է դիմացի անկյունագծերը, այստեղ մեզ օգնության է գալիս այն, որ մենք սև խաղացողներին նշանակել ենք -1-ով, քանի որ ցանկացած խաղացողի դեպքում, դիմացի աջ անկյունագիծը կլինի row+currentPlayer, col+1, իսկ ձախ անկյունագիծը կլինի row+currentPlayer, col-1: Ֆունկցիան ստուգելով, գտնելով հնարավոր քայլերի խաղադաշտերը, դրանց տալիս է 3 արժեքը։ Իսկ հետո օգտագործելով DrawCell() ֆունցկիան, խաղաշտը դարձնում է դեղին։

## OnFigurePress պրոցեդուրա

OnFigurePress պրոցեդուրան սկսում է աշխատել այն ժամանակ, երբ մենք սեղմում ենք խաղադաշտի վրա, օգտագործելով յուրաքանչյուր խաղադաշտի կոճակի Tag-ում պահված i-ն և j-ն, դրանք վերագրում է row և col փոփոխականներին համապատասխանաբար։ OnFigurePress պրոցեդուրան մեր ծրագրի հիմնական պրոցեդուրաներից մեկն է, քանի որ հենց նրա մեջ ենք օգտագործում նախապես գրած գրեթե բոլոր ֆունկցիաները։ OnFigurePress պրոցեդուրան որպես գործողություն կցված է buttons զանգվածի բոլոր կոճակներին:







# Եզրակացություն

Ամփոփելով այս հետազոտական աշխատանքը կարող ենք եզրակացնել, որ այն հաջողված է, բայց ոչ ավարտված, քանի որ դեռ կան շատ տարբեր մտքեր խաղն ավելի լավը դարձնելու համար, այդպիսի մտքերից է արհեստական բանականության ներընդգրկումը խաղի մեջ, առաջիկայում մեզ կթույլատրվի կատարել ընտրություն խաղալ մարդը մարդու դեմ, թե մարդը համակարգչի դեմ։ Առաջիկայում կուղղվեն նաև կոդի բոլոր հնարավոր սխալները։

Այս հետազոտական աշխատանքը ինձ հնարավորություն ընձեռեց ուսումնասիրելու C#-ի և Windows Forms-ի հնարավորությունները լիարժեք ֆունկցիոնալ խաղի ստեղծման համար: Այն ինձ օգնեց հասկանալ, թե որքան կարևոր է աշխատանքի պլանավորումը, թե որքան կարևոր է ծրագրի փորձարկումը՝ թեստավորումը։

Այսպիսով ես ստացա գիտելիքներ որոնք կարող եմ կիրառել ապագա նախագծերում։ Սա դժվար, բայց միաժամանակ լավ փորձառություն էր, որը մեզ օգնեց որոշակի չափով զարգանալ որպես ծրագրավորող։

# Օգտագործված գրականության ցանկ

* Վարդանյան Ռ․, Կարապետյան Ս․ - C++ի հիմունքները
* [Программирование на C# для начинающих](https://www.litres.ru/aleksey-nikolaevich-/programmirovanie-na-c-dlya-nachinauschih-osn-34338191/?lfrom=973672683)
* <https://ru.wikipedia.org/wiki/C_Sharp>
* <https://learn.microsoft.com/ru-ru/dotnet/api/system.drawing.bitmap?view=dotnet-plat-ext-7.0>
* <https://metanit.com/sharp/windowsforms/1.1.php>
* <https://www.codeguru.com/csharp/windows-forms-controls/>
* <https://www.mbmproject.com/blog/tutorials/windows-forms-c-loops-and-arrays>
* <https://www.zebra0.com/csharp/arrays/>