UNIVERSITATEA DIN ORADEA
FACULTATEA DE INFORMATICĂ ȘI ȘTIINȚE
PROGRAMA DE STUDIU INFORMATICĂ
FORMA DE ÎNVĂȚĂMÂNT: CU FRECVENȚĂ

LUCRARE DE LICENTA

COORDONATOR ȘTIINȚIFIC
Lector univ. dr. LASLO EUGEN

ABSOLVENT
Dőmőtőr Zsolt - Béla

ORADEA
2020

UNIVERSITATEA DIN ORADEA
FACULTATEA DE INFORMATICĂ ȘI ȘTIINȚE
PROGRAMA DE STUDIU INFORMATICĂ
FORMA DE ÎNVĂȚĂMÂNT: CU FRECVENȚĂ

BillAI

COORDONATOR ȘTIINȚIFIC
Lector univ. dr. LASLO EUGEN

ABSOLVENT
Dőmőtőr Zsolt - Béla

ORADEA
2020

Contents

1 Consideratii generale

1.1 Structura lucrarii

Lucrarea este formată din patru capitole acestea fiind următoarele:

Capitolul 1 care conține informații privind scopul aplicației și contextul necesare pentru înțelegerea temei propuse.

Capitolul 2 prezintă clase folosite și explică modul de procesarea informației. Sunt prezentate principalele variabile și metode care asigură funcționalitatea aplicației.

Capitolul 3 prezintă proiectul din punct de vedere al programării vizuale. Sunt trecute în revistă formularele și controlale definite de utilizator care stau la baza construcției aplicației.

Capitolul 4 este construit ca și un manual de prezentare ce vine în ajutorul utilizatorilor pentru familiarizarea acestora cu modul de joc și accesarea diverselor setări sau facilități puse la dispoziție de aplicație.

1.2 Aplicația propusă pentru simulare

Această aplicație simulează jocul Bila 8 sau cunoscut si sub numele de Biliard[1].

Scopul acestui joc este că cei doi jucători să reușească să-si bage bilele în gaura în cât mai puține lovituri posibile, lovind cu un tac, bila albă, care va lovi mai departe celălalte bile.

La începutul jocului bilele colorate se găsesc în formație de triunghi într-un capăt al mesei, iar cea albă în celălalt. Un jucător din prima lovitură trebuie să spargă formația alcătuită. Dacă reușește ca o bilă să se nimerească într-o gaură, acesta jucător va trebui să continue să joace cu același grup de bile.

În caz contrar va fi rândul celuilalt jucător care va putea juca cu oricare grupă.

Există două tipuri de bile:

- cele *pline* fiind de la 1 până la 7.
- cele *goale* fiind de la 9 până la 15.

[1] Este un joc jucate pe o masă, cu un băț, folosit pentru a lovi bilele, ca acestea să se miște pe suprafața mesei.

- iar cea cu numărul 8 nu se include în nici una dintre ele, chiar dacă având modelul bilelor pline.

Odată ce un jucător a reușit să bage o bilă, celălalt jucător va avea grupa opusă. Spre exemplu dacă primul jucător bagă o bilă plină, celălalt va avea grupa celor goale.

Figură 1 Formatie initiala a jocului
https://unsplash.com/photos/wa2kYVQaOdc - 2020

Jocul se joacă pe ture. O tură se termina atunci când jucătorul nu a reușit să bage nici o bilă din grupul lui.

Ca un jucător să câștige trebuie să bage toate bilele din grupul lui iar apoi bila cu numărul 8.

Un jucător pierde dacă bagă bila numărul 8 mai repede decât ar trebui.

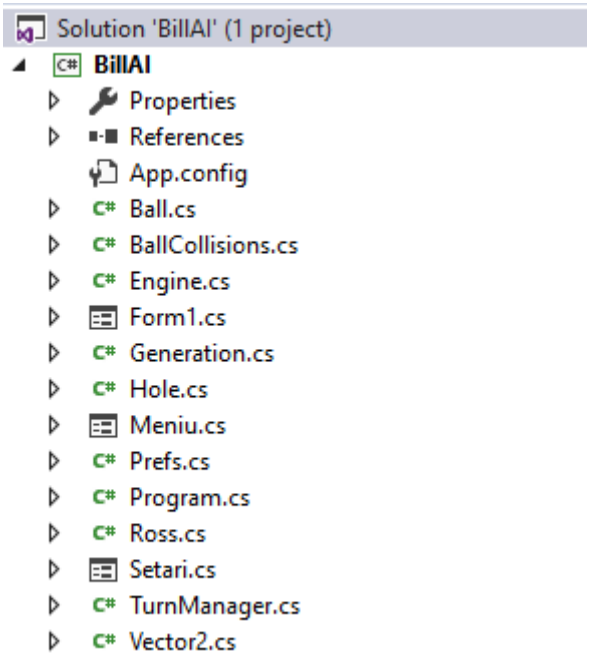
În acest program este vorba despre o versiune mai simplificată a jocului descris anterior.

Constând doar dintr-un un singur jucător care încearcă să bage toate bilele într-un interval de lovituri, definit de noi la rularea programului. Bilele aparținând doar de o grupă, cele pline, acestea au fost ales arbitrar și nu există nici o bilă cu proprietățile bile cu numărul 8 din jocul clasic. În acest mod de joc există un punctaj care este reprezentat de numărul de bile băgate. Jucătorul pierde instantaneu în cazul în care bagă bilă albă în gaură .

1.3 Resurse folosite

Această aplicație nu folosește resurse text sau resurse de imagine. Toate imaginile sunt desenate cu ajutorul librăriei grafice. Imaginile pe care le veți vedea în acest document sunt gratis și se pot folosi cât pentru uz comercial cât și non-comercial fără permisiunea autorului.

2 Dezvoltarea aplicației la nivelul programării orientate pe obiect



Figură 2 Structura de fișier al proiectului

- *Ball.cs* Clasa care reprezintă o bilă și conține metodele principale ale ei.
- *BallCollisions.cs* Conține cea de a doua parte a clasei Ball și se ocupă strict de coliziunea cu celelalte bile și găuri.
- *Engine.cs* Se ocupă de inițializarea și actualizarea datelor globale cât și conține funcții auxiliare pentru depanare.
- *Form1.cs* Acesta este formularul care conține controalele și se va fișa utilizatorului.
- *Generation.cs* Instanțele acestei clase conține toate bilele și toate loviturile aferente, cât și verificările și mutațiile necesare pentru a ajunge la o generație nouă, mai bună.
- *Hole.cs* Este o clasă auxiliară pentru a reprezenta gaura unei mese.
- *Meniu.cs* Formularul care conține meniul aplicației. Acesta se va rula la pornirea programului.
- *Prefs.cs* Această clasă are specificatori de acces public și este una statică pentru că, conține datele ajustabile de către utilizator, pentru a personaliza aspectul jocului și de unde se poate schimba evoluția algoritmului.

- *Ross.cs* Este o clasă auxiliară care ajută la afișarea (desenarea) obiectelor pe ecran. Și acesta fiind una publică și statică, putând fi folosit de către orice altă clasă. Numele este inspirat de către pictorul Bob Ross[2].
- *Setari.cs* Din acest formular va putea utilizatorul să personalizeze configurările jocului.
- *TurnManager.cs* Conține lista de generații și se ocupă de crearea, îmbunătățirea, ștergerea, recrearea și afișarea lor. Acesta mai conține și cea mai bună generație de până acum, doar acesta se va afișa.
- *Vector2.cs* este un tip de date definit de mine pentru a ajuta în calculele 2D. este un tip de date definit de mine pentru a ajuta în calculele 2D. Este inspirat din mediul și game engine-ul[3] Unity[4].

2.1 Clasa Engine

Această clasă de bază are ca scop de a inițializa fluxul aplicației și oferă posibilitatea de a afișat text utilizatorului similar cu modelul unei aplicații în consolă. Metodele și variabilele se vor folosi și de către celelalte clase de aceea cât și clasa în sine cât și variabilele și funcțiile au specificatori de acces public și static.

```
public static class Engine
{
        public static ListBox console;

        public static List<Hole> holes;

        public static Random rnd = new Random();

        public static void Init(ListBox _console){...}
        private static void InitHoles(){...}


        public static void Update(){...}

        public static void CW(object v){...}
        public static void ClearConsole(){...}
}
```

- `public static ListBox console;`
  Variabila care conține referință către controlul ListBox[5] în care se vor afișa mesajele pentru utilizator.

[2] Robert Norman Ross a fost un pictor american, instructor de arte, și gazdă de televiziune.
[3] Tip de software menit pentru dezvoltarea jocurilor video.
[4] Program de dezvoltarea jocurilor video.
[5] Un control din mediul Windows pentru a afișa o listă de elemente.

- `public static List<Hole> holes;`
  Va fi lista de găuri al mesei, acesta se va defini la începutul aplicației și va rămâne nemodificat.

- `public static Random rnd = new Random();`
  Variabila aleatoare care ajută la obținerea unghiurilor și a puterilor de lovit diferite, pentru fiecare lovitură din fiecare generație.

- `public static void Init(ListBox _console){...}`
  Această metodă se va apela la începutul aplicației și va primi ca parametru ListBox-ul din formular, inițializa găurile de pe masă și va porni rundele/generațiile jocului.

```
public static void Init(ListBox _console) {
	console = _console;
	InitHoles();
	TurnManager.Start();
}
```

- `private static void InitHoles(){...}`
  În această metodă se vor defini cele șase cazuri, 3 sus 3 jos, care vor rămâne tot aceleași pe parcursul jocului.

```
private static void InitHoles()
{
	holes = new List<Hole>();
	 holes.Add(new Hole(7, 5));
	 holes.Add(new Hole(Ross.Width / 2f, 5));
	 holes.Add(new Hole(Ross.Width - 7, 5));

	holes.Add(new Hole(7, Ross.Height - 5));
	 holes.Add(new Hole(Ross.Width / 2f, Ross.Height - 5));
	 holes.Add(new Hole(Ross.Width - 7, Ross.Height - 5));
}
```

- `public static void Update(){...}`
  Se va apela la fiecare actualizare a cadrului și se ocupă de declanșarea actualizari generațiilor și desenarea elementelor pe ecran.

```
public static void Update()
{
	TurnManager.Update();

	//Drawing the balls
	Ross.BeginDraw();
	TurnManager.Draw();

	foreach (Hole hole in holes)
		hole.Draw();

	Ross.EndDraw();
}
```

- `public static void CW(object v){...}`
  Primindu-ș numele de la prescurtarea funcției din aplicațiile cu consola (Console.WriteLine[6]), această funcție va afișa text pentru utilizator.

```
public static void CW(object v)
{
    console.Items.Add(v.ToString());
}
```

- `public static void ClearConsole(){...}`
  Pentru că afișarea mesajelor către utilizator în această aplicație funcționează similar cu o aplicație în consolă, pentru a afișa date noi fără a arăta istoricul, lista mea veche de mesaje trebuie să se golească

```
public static void ClearConsole()
{
        console.Items.Clear();
}
```

2.2 Clasa Ball

Această clasă este despărțită în două fișiere, pentru a putea categoriza funcționalitățile si pentru a avea un cod mai curat si mai citibil. Despărțirea s-a realizat cu ajutorul cuvântului de cheie partial găsit în limbajul C#.

2.2.1 Generalități

În prima parte a clasei sunt definite variabilele necesare, proprii pentru fiecare bila și metodele pentru mișcare și desenarea ei.

```
public partial class Ball
    {
        public Vector2 position;
        public Vector2 direction;
        public Vector2 force;

        public Color color;
        public bool isMoving;

        public Ball(float x, float y, Color color){...}

        public void SetForce(Vector2 v){...}
        public void RemoveForce() {...}
        public void SetForce(float angle, float force) {...}

        public void Move() {...}

        public void Draw() {...}
    }
```

- `public Vector2 position;`
  Arată poziția curentă pe masă.

[6] Funcție din limbajul C# care va afișa un număr sau un text în consola utilizatorului.

- `public Vector2 direction;`
  Reprezintă direcția în care se îndreaptă, dacă este în mișcare. Va avea valori unitare, Spre exemplu (-1,0), (1,1) etc.
  Prima valoare va indica direcția pe axa X (-1 = stânga, 0 = nu te deplasezi pe axa aceasta, 1 = dreapta) , iar cea de a doua pe axa Y (-1 = sus, 0 = nu te deplasezi pe axa aceasta, 1 = jos)
- `public Vector2 force;`
  Este viteza cu care se deplasează, dacă este în mișcare. Valorile ei vor fi mereu pozitive sau 0, pe amândouă axele.
- `public Color color;`
  Variabila care reprezintă culoarea ei.
- `public bool isMoving;`
  Variabilă booleană care arată dacă este în mișcare sau nu.
- `public Ball(float x, float y, Color color){...}`
  Singurul constructor al clasei, conținând numai parametrii necesari și creând un obiect cu variabilele setate implicit pe un stadiu de repaus.

```
public Ball(float x, float y, Color color)
{
	position = new Vector2(x, y);
	direction = new Vector2(0, 0);
	force = new Vector2(0, 0);

	isMoving = false;
	this.color = color;
}
```

- `public void SetForce(Vector2 v){...}`
  Această metodă reprezintă o lovitură din lumea reală.
  Primind ca parametru instanța clasei Vector2, se vor calcula forța și direcția în care va trebui să se îndrepte.

```
public void SetForce(Vector2 v)
{

		isMoving = true;
		direction = v.ToUnit();
		force = v.Abs();
}
```

- `public void SetForce(float angle, float force) {...}`

  Este doar o funcție auxiliară care va converti unghiul și forța, primită ca parametru, Într-o lovitură cu ajutorul celelalte funcții SetForce.

```
public void SetForce(float angle, float force)
{
	Vector2 v = Vector2.FromAngle(angle) * force;
	this.SetForce(v);
}
```

- `public void RemoveForce() {...}`

Cu ajutorul ei voi putea opri o bilă manual, ne trebuind să aștept să se oprească singură. Utilă în cazul în care trebuie să opresc o generație mai devreme.

```
public void RemoveForce()
{
	direction = new Vector2(0, 0);
	force = new Vector2(0, 0);
	isMoving = false;

}
```

- `public void Draw(){...}`

Funcția care va afișa bila, în afara cazului în care a fost deja băgată în gaură.

```
public void Draw()
{
	if (isInHole == false)
		Ross.Draw(this);
}
```

- `public void Move() {...}`

Acesta este metoda care va face bila să se miște.

Se va apelat de mai multe ori pe secundă, așa că pentru optimizare am pus că dacă este în repaus, nu are direcție în care să meargă sau viteza cu care să se deplaseze, se vor sări peste viitoarele calcule ieșind imediat din funcție.

Această linie de cod este responsabilă pentru deplasarea bilei. Având variabila direction una unitară iar force una mereu pozitivă, cu simpla înmulțirea lor pot obține cursul mișcării. Iar la final adaug rezultatul obținut la poziția curentă pentru a primi noua poziția bilei pe masă.

```
position += direction * force;
```

În continuare am două verificări în caz de lovirea pereților, pentru a face bilele să sară înapoi. Important e să menționez că cele două nu sunt legate cu else if, o axă nefiind influențată de cealaltă.

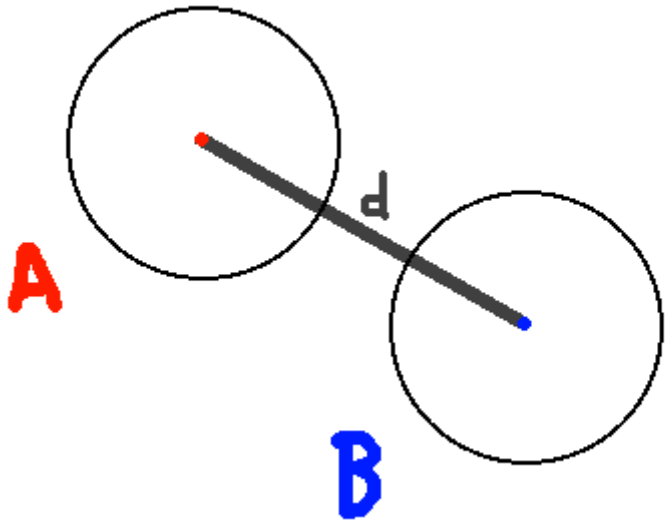
Iar apoi aplic forța de frecare, pentru a încetini cu timpul bilele. Dacă viteza va ajunge sub marjă, se va seta automat 0.

```
public void Move()
{
	if (direction.isZero() || force.isZero())
	{
		isMoving = false;
		return;
	}

	isMoving = true;
	position += direction * force;

	if (position.x - Prefs.ballSize / 2 <= 0 || position.x + Prefs.ballSize / 2 >=
Ross.Width)
	{
		this.force.x -= Prefs.friction * 2;
```

```
                direction.x *= -1;
        }

        if (position.y - Prefs.ballSize / 2 <= 0 || position.y + Prefs.ballSize / 2 >=
Ross.Height)
        {
                this.force.y -= Prefs.friction * 2;
                direction.y *= -1;
        }

        if (this.force.x > 0.005f)
                this.force.x -= Prefs.friction;
        else
                this.force.x = 0;

        if (this.force.y > 0.005f)
                  this.force.y -= Prefs.friction;
        else
                  this.force.y = 0;
}
```

2.2.2 Coliziune

Aici putem găsi cea de a doua parte a clasei Ball, mai exact partea care se ocupă de verificările pentru coliziunea cu alte entități.
Despre coliziunea cu pereții am discutat în subcapitolul 2.2.1 , așa că a mai rămas cea între bile și găuri. Acestea sunt reprezentate ca și cercuri, deci problema coliziunii se reduce la coliziunea între două cercuri.



Figură 3 Distanța între centrul a doua cercuri

Iar două cercuri se intersectează dacă distanța dintre centrele lor este mai mică decât suma razelor. Vezi Figură 3 Distanța între centrul a doua cercuri

```
public partial class Ball
{
        public bool isInHole = false;

        public bool IsColliding(Ball other){...}
        public void Oncollide(Ball other){...}
```

```
        public bool IsColliding(Hole other){...}
        public void Oncollide(Hole hole){...}

}
```

- `public bool isInHole = false;`
  Variabilă booleană care arată dacă această bilă a fost sau nu băgată în gaură. Odată ce o bilă a ajuns deja într-o gaură, nu se mai aplică verificările pentru alte intersecții.

- `public bool IsColliding(Ball other){...}`
  Această funcție va arăta dacă bila curentă atinge o altă bilă. Verificarea se face pe baza Figură 3 Distanța între centrul a doua cercuri , dar înainte se vor asigura dacă este cazul de verificări de coliziune.

```
public bool IsColliding(Ball other)
{
        if (isInHole || other.isInHole)
                return false;

        if (this.isMoving == false && other.isMoving == false)
                return false;

        return ((other.position.x - this.position.x) * (other.position.x -
this.position.x) +
                  (other.position.y - this.position.y) * (other.position.y -
this.position.y))
                  <= Prefs.ballSize * Prefs.ballSize;
}
```

- `public void Oncollide(Ball other){...}`
  Această funcție se apelează în momentul când două bile se lovesc una de cealaltă. Aici se vor face calculele pentru a obține noile traiectorii ale bilelor.

```
public void Oncollide(Ball other)
{
        Vector2 collision = this.position - other.position;
        float distance = collision.GetLength();

        collision /= distance;
        float aci = Vector2.Dot(this.direction * this.force, collision);
        float bci = Vector2.Dot(other.direction * other.force, collision);

        this.SetForce(this.direction * this.force + collision * (bci - aci));
        other.SetForce(other.direction * other.force + collision * (aci - bci));
}
```

- `public bool IsColliding(Hole other){...}`
  Funcționează similar cu metoda care verifică dacă o bilă o atinge pe cealaltă, doar că în acest caz trebuie să verificăm cu o gaură. Abordarea fiind aceeași, căci o gaură este reprezentată tot ca un cerc, singura diferență va fi că este plasată pe margine, rezultând astfel că doar o parte fiind pe masă , așa cum e în jocul original.

```
public bool IsColliding(Hole other)
{
        if (isInHole)
                return true;
```

```
        if (other.position == this.position)
                return false;

        return ((other.position.x - this.position.x) * (other.position.x -
this.position.x) +
                (other.position.y - this.position.y) * (other.position.y -
this.position.y))
                <= Prefs.ballSize * Prefs.ballSize / 2;

}
```

- `public void Oncollide(Hole hole){...}`
  Se va apela la coliziune cu o gaură marcând-o și oprind orice mișcare pe care a avut o până acum.

```
public void Oncollide(Hole hole)
{
        isInHole = true;
        this.RemoveForce();
}
```

2.3 Clasa Generation

Această clasă se ocupă cât de reprezentare a cât și de îmbunătățirea jocurilor de biliard din acest program. Mai precis fiind responsabilă de generarea și executarea loviturilor și de stării actuale ale sale. Fiecare joc poate avea una din două stări:

- **In rulare**: Când mai sunt lovituri care trebuiesc date și/sau când se mai află vreo bilă în mișcare.
- **Moartă/Terminată**: Un joc se consideră mort sau terminat dacă, bilă albă a fost băgată sau a fost realizată fiecare lovitură din lista de lovituri

La baza acestei clase se află așa numita listă de lovituri, care de fapt în cod constă din două liste de lungime egală, lista de unghiuri și lista de puteri de lovit, acestea vor fi parcurse împreună și modificate sau duse mai departe la următoarele generații.
Loviturile se aplică strict asupra bilei albe, conform regulilor din jocul clasic.

```
public class Generation
{
        public int shootIndex;
        public List<float> angles;
        public List<float> forces;

        public bool isAlive = true;
        public bool isDone = false;

        public Ball whiteBall = null;
        public List<Ball> otherBalls;
        public List<Ball> allBalls;

        public Generation(){...}
```

```
        public Generation(Generation old) {...}

        #region Ai
        public void Mutate() {...}

        public float GetFitness() {...}
        #endregion

        public void Shoot() {...}

        public void Update() {...}

        private void Dead() {...}

        private void CheckIfIsAllStopped() {...}

        public void Draw() {...}

        #region Initializare
        private void InitWhiteBall() {...}

        private void CreateOtherBall(Vector2 position, Color color) {...}
        private void InitOtherBall() {...}
        #endregion
}
```

- `public int shootIndex;`
  Variabila care ne spune la a câta lovitură suntem din joc. Acesta crește de la 0, după fiecare lovitură dată, până la lungimea listei angles și/sau forces. În cazul în care bila albă a fost băgată în gaură, acest contor nu va mai crește.

- `public List<float> angles;`
  Reprezintă o parte din așa numita lista de lovituri. Conține unghiurile de la fiecare lovitură.
  Valorile ei sunt măsurate în grade.

- `public List<float> forces;`
  Reprezintă cealaltă parte din lista de lovituri. Conține puterea de lovit de la fiecare lovitură.

- `public bool isAlive = true;`
  Variabila booleana care arată dacă această generație este în rulare sau dacă este una moartă.

- `public bool isDone = false;`
  Variabila booleana care arată dacă bilele s-au oprit de pe urma ultimei lovituri.

- `public Ball whiteBall = null;`
  Referința către bila albă din joc.

- `public List<Ball> otherBalls;`
  Lista de referințe către celelalte bile din joc, cele colorate care trebuiesc băgate.

- `public List<Ball> allBalls;`
  Lista de referințe către toate bilele din joc, adică cele colorate și cea albă.
- `#region Ai ... #endregion` și `#region Initializare ... #endregion`
  Folosind această sintaxă pot defini zone de cod și să le grupezi pe baza funcționalităților.

- `public Generation(){...}`
  Acesta este constructorul unei generații oarecare, creând o instanță cu lovituri aleatoare. Odată creat un joc ea va începe automat dâdând prima lovitură.

```
public Generation()
{
    allBalls = new List<Ball>();
    InitWhiteBall();
    InitOtherBall();

    isAlive = true;
    shootIndex = 0;

    angles = new List<float>();
    forces = new List<float>();

    int numberOfShoots = Engine.rnd.Next(Prefs.minShootsCount, Prefs.maxShootsCount);
    for (int i = 0; i < numberOfShoots; i++)
    {
        angles.Add(Prefs.GetRandomAngle());
        forces.Add(Prefs.GetRandomForce());
    }

    Shoot();
}
```

- `public Generation(Generation old) {...}`
  Este folosită pentru a crea o copie a unei generații. Copiile vor fi modificate ulterior sau nu, în cazul în care este vorba de cea mai bună generație.

```
public Generation(Generation old)
{
    allBalls = new List<Ball>();
    InitWhiteBall();
    InitOtherBall();

    isAlive = true;
    shootIndex = 0;

    angles = new List<float>();
    forces = new List<float>();

    int numberOfShoots = old.angles.Count;
    for (int i = 0; i < numberOfShoots; i++)
    {
        angles.Add(old.angles[i]);
        forces.Add(old.forces[i]);
    }
```

```
    Shoot();
}
```

- `public void Mutate() {...}`
  În acest program procesul de mutație înseamnă că o parte sau chiar toată lista de lovituri se va schimba. Numărul de elemente schimbate se va face pe baza unei șanse ales de către utilizatorul programului. Trecând prin fiecare element verificăm dacă sunt șanse de modificare.

```
public void Mutate()
{
    for (int i = 0; i < angles.Count; i++)
    {
        if (Engine.rnd.Next(100) < Prefs.mutationChance)
        {
            this.angles[i] = Prefs.GetRandomAngle();
            this.forces[i] = Prefs.GetRandomForce();
        }
    }
}
```

- `public float GetFitness() {...}`
  Fitnessul[7] unei generații înseamnă numărul de bile băgate, în cazul unui joc finalizat cu succes. Contrar în cazul în care nu a fost finalizat cu succes, adică bilă albă a fost băgată, generația va primi un „punctaj de pedeapsă” de -1. Este considerat ca un „punctaj de pedeapsă” pentru că este sub minimul oricărei joc finalizat cu succes. Valoarea obținută se va folosi ulterior pentru generarea noilor generații.
  Cu cât mai mare este punctajul cu atât mai bună este generația.
  Menționez că, valoarea pe care o are returnează funcția aș putea să o amplific introducând-o într-o funcție exponențială simplă pentru a crește mai mult diferența dintre două generații, favorizând mai multe generațiile care au mai multe bile băgate.

```
public float GetFitness()
{
    if (whiteBall.isInHole)
        return -1;

    int numberBallsInHoles = 0;

    for (int i = 0; i < allBalls.Count; i++)
        if (allBalls[i].isInHole)
            numberBallsInHoles++;

    return numberBallsInHoles;//(float)Math.Pow(Math.E,numberBallsInHoles);

}
```

- `public void Shoot() {...}`
  Această funcție se va apela la lovirea bilei albe. Lovirea efectivă este realizată de funcția SetForce, dar acesta se va apela doar în cazul în care jocul este în rulare și mai sunt lovituri disponibile.

```
public void Shoot()
{
```

[7]Fitness este o valoare care reprezintă cât de bună este o generație.

```
    if (isAlive == false)
        return;

    if (shootIndex >= angles.Count || shootIndex >= forces.Count)
    {
        isAlive = false;
        return;
    }

    isDone = false;
    whiteBall.SetForce(angles[shootIndex], forces[shootIndex]);
    shootIndex++;
}
```

- `public void Update() {...}`
  Ea sa va apelat de mai multe ori pe parcursul aplicației. Fiind responsabilă de actualizarea stării jocului, mișcarea bilelor și verificarea coliziunilor.
  Odată ce jocul este mort, acesta nu va mai necesita actualizare.
  În această funcție se vor verifica dacă oricare bilă atinge oricare altă bilă sau oricare gaură și apelează funcții de stabilite lor.

```
public void Update()
{
    if (!isAlive)
        return;

    CheckIfIsAllStopped();

    //Updateing the balls
    whiteBall.Move();
    foreach (Ball ball in otherBalls)
        ball.Move();

    //Collisions
    for (int i = 0; i < allBalls.Count; i++)
    {

        for (int j = i + 1; j < allBalls.Count; j++)
        {
            if (allBalls[i].IsColliding(allBalls[j]))
                allBalls[i].Oncollide(allBalls[j]);
        }

        foreach (Hole hole in Engine.holes)
        {
            if (allBalls[i].IsColliding(hole))
                if (allBalls[i] == whiteBall)
                {
                    whiteBall.isInHole = true;
                    Dead();
                    allBalls[i].RemoveForce();
                }
                else
                    allBalls[i].Oncollide(hole);

        }

    }
}
```

- `private void Dead() {...}`
  Acesta se va apela când bila albă intră în gaură. Va seta variabilele care se sunt responsabile pentru calcularea stării jocului și va opri automat toate bilele în poziția în care erau, nefiind nevoie ca ele să se mai miște, având un joc greșit[8].

```
private void Dead()
{
    isDone = true;
    isAlive = false;
    for (int i = 0; i < allBalls.Count; i++)
        allBalls[i].RemoveForce();
}
```

- `private void CheckIfIsAllStopped() {...}`
  După executarea acestei funcție Putem afla dacă starea jocului este una în care bilele s-au oprit din rostogolit sau sunt încă în mișcare.

```
private void CheckIfIsAllStopped()
{
    for (int i = 0; i < allBalls.Count; i++)
        if (allBalls[i].isMoving)
            return;
    isDone = true;
}
```

- `public void Draw() {...}`
  Responsabil pentru apelarea funcției de afișare a bilelor. Doar cel al bilelor căci, numai acestea sunt unice per generație. Spre exemplu găurile sau culoarea mesei rămân aceleași indiferent de lovituri sau de generații.

```
public void Draw()
{
    //Drawing the balls
    foreach (Ball ball in otherBalls)
        ball.Draw();
    whiteBall.Draw();
}
```

- `private void InitWhiteBall() {...}`
  Această funcție creează instanța bile albe. Bila este poziționată pe prima treime al mesei, regula al acestei versiuni de joc.

```
private void InitWhiteBall()
{
       whiteBall = new Ball(Ross.Width / 3f, Ross.Height / 2f, Color.White);
       allBalls.Add(whiteBall);
}
```

- `private void CreateOtherBall(Vector2 position, Color color) {...}`
  Funcție auxiliară care creează o bilă colorată într-o poziție dorită. Există doar cu scopul de a îmbunătăți structura codului.

```
private void CreateOtherBall(Vector2 position, Color color)
{
       Ball ball = new Ball(position.x, position.y, color);
       otherBalls.Add(ball);
```

[8] Adică un joc în care bila albă a fost băgată.

```
        allBalls.Add(ball);
}
```

- `private void InitOtherBall() {...}`
  Creează instanțele bilelor colorate cu ajutorul funcției descrise anterior. Acesta sunt create în formație de triunghi și fiecare colorat diferit. Culorile și pozițiile sunt specificate de mână, dar calculate în funcție de dimensiunea mesei.

```
private void InitOtherBall()
{
        float xSpace = 1f, ySpace = 1f;
        otherBalls = new List<Ball>();
        Vector2[] positions =
        {
                //Col 1
                new Vector2(Ross.Width/4f*3f-2*Prefs.ballSize,Ross.Height/2),

                //Col 2
                new Vector2(Ross.Width/4f*3f-Prefs.ballSize+xSpace,Ross.Height/2-
Prefs.ballSize/2-ySpace),
                  ...

         };
         Color[] colors = {
                Color.Yellow,
                Color.Red, Color.Blue,
                ...
        };

        for (int i = 0; i < positions.Length; i++)
                CreateOtherBall(positions[i], colors[i]);

}
```

2.4 Clasa TurnManager

Acesta este una dintre cele mai esențiale clase, având rolul de a administra toate jocurile din fiecare generație.

Numele clasei se referă la o „tură” ca fiind o generație de generații, dar pentru a nu crea confuzii am denumit-o tură.

Iar în acest document voi face la fel, referindu-mă la o *tur*ă ca fiind o mulțime de jocuri; prin *urmatoarea tură* mă refer la urmatoarea mulțime de jocuri care vine după această tură, ele fiind deja dupa mutație; iar prin *tura anterioare* la mulțimea de jocuri care s-au jucat inaintea turei actuale.

```
public static class TurnManager
{
    public static int generationCount = 0;
    public static int deadCount = 0;

    public static Generation bestGeneration;
    public static List<Generation> generations;

    public static void Start(){...}

    public static void Update() {...}
```

```
    private static void ProcessOldGeneration() {...}

    private static void NextGeneration() {...}

    public static void Draw() {...}
}
```

- `public static int generationCount = 0;`
  Variabila care ne arată la a câta generație/tura suntem.

- `public static int deadCount = 0;`
  Numărul de jocuri moarte, fie prin introducerea bile albe, fie prin jucarea fiecărei lovituri.

- `public static Generation bestGeneration;`
  Referința la cea mai bună generație de până acum. Acesta se va recalcula și copia la începutul fiecărei ture. Acesta este și jocul care este afișat utilizatorului.

- `public static List<Generation> generations;`
  Lista de generații din tura curentă.

- `public static void Start(){...}`
  Este funcția care inițializează variabilele și va crea o listă de generație cu lovituri aleatoare. Acesta se va rula o dată la începutul aplicației.

```
public static void Start()
{
    deadCount = 0;
    bestGeneration = null;
    generations = new List<Generation>();

    for (int i = 0; i < Prefs.countInGeneration; i++)
    {
        Generation generation = new Generation();
        generations.Add(generation);
    }
    generationCount++;
}
```

- `public static void Update() {...}`
  Actualizează fiecare joc încă în rulare, chiar și de mai multe ori dacă utilizatorul așa a ales din Formularul de Setări. Apelează funcțiile care sunt stabilite pentru actualizarea bilelor și în cazul în care se poate, efectuează următoarea lovitură.

  În momentul când un joc s-a terminat, din oricare motiv, informațiile pentru utilizator se vor împroaspăta, afișând datele noi, corecte.

  Menționez că am lăsat codul responsabil pentru unele informații extra comentat, pentru că

am considerat că este o informație de mai puțină valoare, comparativă cu cele care se afișează până acum. Dar la nevoie acestea pot fi din nou implementate cu ușurință

```
public static void Update()
{
    for (int time = 0; time < Prefs.timeScale; time++)
    {
        for (int i = 0; i < generations.Count; i++)
        {
            if (generations[i].isAlive)
            {
                generations[i].Update();

                if (generations[i].isDone)
                    generations[i].Shoot();

                if (generations[i].isAlive == false)
                {
                    deadCount++;

                    Engine.ClearConsole();
                    Engine.CW("Generation #" + generationCount);
                    Engine.CW(generations[0].GetFitness() + "/6");
                    Engine.CW("Loading: " + (deadCount * 100) / generations.Count +
"%");
                    //Engine.CW("a murit generatia " + i + " cu fitnessul: " +
generations[i].GetFitness());

                    if (deadCount == generations.Count)
                    {
                        //Engine.CW("\n");
                        ProcessOldGeneration();
                        //Engine.CW("S-a terminat generatia " + generationCount);
                        //Engine.CW("\n");
                        NextGeneration();
                    }
                }
            }
        }
    }
}
```

- `private static void ProcessOldGeneration() {...}`
  Odată ce o tură sa- terminat singurul lucru care mai rămâne de făcut cu el este ca generațiile din acestea să se ordoneze crescător în funcție de cât de bune au fost.

```
private static void ProcessOldGeneration()
{
    generations.Sort((a, b) => (int)b.GetFitness() - (int)a.GetFitness());
}
```

- `private static void NextGeneration() {...}`
  Aceasta este responsabilă pentru a crea noua tură. Când ajunge aici generațiile din tura anterioară sunt ordonate deja, pentru a face selectarea și pentru a crea noile generații mai ușor. Distribuția noii generații sunt în următoarele proporții:
  - 1 element care este cel mai bun joc jucat de până acum, pentru a nu-l pierde.

- Copie fidelă a ¼ cele mai bune din tura actuală.
- ¼ cele mai bune dar aplicând mutație pe ele.
- ½ aleator, astfel ajutând la diversificarea generațiilor, pentru a nu rămâne blocat.

```
private static void NextGeneration()
{
    deadCount = 0;

    //Salvez best of all time
    if (bestGeneration == null || bestGeneration.GetFitness() <
generations[0].GetFitness())
        bestGeneration = new Generation(generations[0]);

    List<Generation> newGenerations = new List<Generation>();
    newGenerations.Add(new Generation(bestGeneration));

    //Cel mai bune
    for (int i = 1; i < Prefs.countInGeneration / 4; i++)
        newGenerations.Add(new Generation(generations[i]));

    //Cele mai bune Mutated
    for (int i = Prefs.countInGeneration / 4; i < Prefs.countInGeneration / 2; i++)
    {
        Generation generation = new Generation(generations[i]);
        generation.Mutate();
        newGenerations.Add(generation);
    }

    //Random
    for (int i = Prefs.countInGeneration / 2; i < Prefs.countInGeneration; i++)
    {
        Generation generation = new Generation();
        newGenerations.Add(generation);
    }

    generations = newGenerations;
    generationCount++;
}
```

- `public static void Draw() {...}`
  Afișarea celei mai bune generații. Pe baza construirii generațiilor mereu primul element din listă va fi cel mai bun.

```
public static void Draw()
{
    generations[0].Draw();
}
```

2.5 Clasa Vector2

Stând la baza programului și tot ceea ce înseamnă calcul, această clasă își are originea din programul Unity[9]. Conține două proprietăți importante fiecare dintre acestea reprezentând o valoare pe o axă.

```
public class Vector2
{
    public float x, y;

    public Vector2(float x, float y){...}

    public Vector2() {...}

    public bool isZero() {...}

    public static Vector2 FromAngle(float angle) {...}

    public Vector2 Abs() {...}

    public Vector2 ToUnit() {...}

    public override string ToString() {...}

    #region Math

    public static float Dot(Vector2 A, Vector2 B) {...}

    public float GetLength() {...}

    public void Normalize() {...}
    #endregion


    #region Operators

    public static bool operator ==(Vector2 A, Vector2 B) {...}

    public static Vector2 operator +(Vector2 A, Vector2 B) {...}

    public static Vector2 operator -(Vector2 original, float scale) {...}

    #endregion
}
```

- `public float x, y;`
  Cele 2 proprietăți care stau la paza clasei, fiecare reprezintă o valoare pe axa sa.

- `public Vector2(float x, float y){...}`
  Constructorul simplu al clasei.

```
public Vector2(float x, float y)
{
    this.x = x;
    this.y = y;
}
```

[9] Program de dezvoltarea jocurilor video

- `public Vector2() {...}`
  Supra scriu constructorul implicit al funcției folosindu-mă de constructorul simplu, pentru a seta manual o valoare implicită.

```
public Vector2() : this(0, 0) { }
```

- `public bool isZero() {...}`
  Metoda booleană care ne va arăta dacă acest tip de date este considerat ca fiind zero. Este considerat zero când ambele proprietăți sunt zero. Am folosit sintaxa de săgeată[10] pentru a face codul mai citibil.

```
public bool isZero() => x == 0 && y == 0;
```

- `public static Vector2 FromAngle(float angle) {...}`
  Cu ajutorul acestei funcții voi putea transforma un unghi definit în grade, într-o variabilă de tip Vector2 folosit ulterior pentru direcția de mișcarea a bilelor. Procesul de transformare se întâmpla în doi pași: primul fiind transformarea unghiului din grade in radian, iar al doilea este obținerea coordonatelor pe baza unghiului.

```
public static Vector2 FromAngle(float angle)
{
    double angleInRadian = (Math.PI / 180d) * -angle;
    return new Vector2((float)Math.Cos(angleInRadian),(float)Math.Sin(angleInRadian));
}
```

- `public Vector2 Abs() {...}`
  Cu acestea voi putea obține valoarea absolută al variabilei mele. Similar cu numerele reale, valoarea absolută și în cazul unui Vector2 este că se iau valorile și se fac pozitive.

```
public Vector2 Abs() => new Vector2(Math.Abs(this.x), Math.Abs(this.y));
```

- `public Vector2 ToUnit() {...}`
  Cu această funcție voi putea transforma variabilă mea întruna care să conțină valori unitare, conținând doar -1,0 sau 1 în ambele proprietăți. Acesta se face cu ajutorul unui prag. În intervalul [-prag, prag] va fi 0, iar in rest -1 sau 1 in funcție de pozitvitatea valorii.

```
public Vector2 ToUnit()
{
    float threshold = 0.005f;
    int newX = 0, newY = 0;

    if (this.x >= -threshold && this.x <= threshold)
        newX = 0;
    if (this.x < -threshold)
        newX = -1;
    if (this.x > threshold)
        newX = 1;

    if (this.y >= -threshold && this.y <= threshold)
        newY = 0;
```

[10] Este o alternativă compactă al unei funcții obișnuite. Singura diferență fiind sintactică.

```
    if (this.y < -threshold)
        newY = -1;
    if (this.y > threshold)
        newY = 1;

    return new Vector2(newX, newY);
}
```

- `public override string ToString() {...}`
  Funcție auxiliară care îmi returnează ca și text citibil un obiect de tip Vector2, mai exact valorile ei.

```
public override string ToString()
{
    return String.Format("({0}, {1})", this.x, this.y);
}
```

- `public static float Dot(Vector2 A, Vector2 B) {...}`
  Este produsul scalar a doi vectori, pe baza formulei magnitudinilor euclidiene sau cosinusul dintre 2 unghiuri.

```
public static float Dot(Vector2 A, Vector2 B)
{
    return A.x * B.x + A.y * B.y;
}
```

- `public float GetLength() {...}`
  Lungimea unei Vector2 nu este altceva decât distanța euclidiană dintre 0 și valorile mele, acestea pot fi considerate și ca și puncte.

```
public float GetLength()
{
    return (float)Math.Sqrt(this.x * this.x + this.y * this.y);
}
```

- `public void Normalize() {...}`
  Apelând această funcție voi putea să îl normalizez[11].

```
public void Normalize()
{
    this.x /= this.GetLength();
    this.y /= this.GetLength();
}
```

- `public static bool operator ==(Vector2 A, Vector2 B) {...}`
  Definim acest operator pentru asuprascrie pe cel implicit, pentru că în cazul nostru atunci sunt egali doua obiect de tip Vector2 când proprietățile lor sunt egale.

```
public static bool operator ==(Vector2 A, Vector2 B)
{
    return A.x == B.x && A.y == B.y;
}
```

- `public static Vector2 operator +(Vector2 A, Vector2 B) {...}`
  Definim acest operator pentru că în cazul nostru adunarea se face prin adunarea proprietăților corespunzătoare.

[11] Este procesul în care luăm un vector de orice lungime și îi schimbăm lungimea în 1, transformându-l în ceea ce se numește vector unitar, păstrându-l îndreptat în aceeași direcție.

```
public static Vector2 operator +(Vector2 A, Vector2 B)
{
    return new Vector2(A.x + B.x, A.y + B.y);
}
```

- `public static Vector2 operator -(Vector2 original, float scale) {...}`
  Definim acest operator pentru că în cazul nostru scăderea cu un scalar se face prin scăderea din fiecare proprietate.

```
public static Vector2 operator -(Vector2 original, float scale)
{
    return new Vector2(original.x - scale, original.y - scale);
}
```

Mai sunt și alți operatori suprascriși și toate funcționează pe același principiu de, a modifica proprietățile fiecăruia.

Lista operatorii suprascriși:

- `public static bool operator !=(Vector2 A, Vector2 B)`
- `public static bool operator ==(Vector2 A, Vector2 B)`
- `public static Vector2 operator +(Vector2 A, Vector2 B)`
- `public static Vector2 operator +(Vector2 original, float scale)`
- `public static Vector2 operator -(Vector2 A, Vector2 B)`
- `public static Vector2 operator -(Vector2 original, float scale)`
- `public static Vector2 operator *(Vector2 original, float scale)`
- `public static Vector2 operator *(Vector2 A, Vector2 B)`
- `public static Vector2 operator /(Vector2 original, float scale)`

2.6 Clasa Prefs

Această clasă are identificatorii de acces public și static pentru că, conține valorile și setările generale care trebuiesc folosite în timpul jocurilor, de asta fiecare proprietate este marcat astfel. O parte din proprietățile din această clasă se poate edita din Formularul de Setări la rularea programului.

```
public static class Prefs
{

    #region About game
    public static float ballSize = 80;
    public static Color tableColor = Color.SeaGreen;
    public static float friction = 0.002f;

    public static int timeScale = 10;
    #endregion

    #region About AI
    public static int countInGeneration = 50;
    public static int mutationChance = 30;

    public static int minShootsCount = 5;
```

```
    public static int maxShootsCount = 7;
    #endregion

    public static int GetRandomAngle() {...}
    public static float GetRandomForce() {...}

}
```

- `public static float ballSize = 80;`
  Dimensiunea bilelor cât și a găurilor. Se poate edita din setări.

- `public static Color tableColor = Color.SeaGreen;`
  Culoarea de fundal al mesei. Nu se poate edita din setări pentru că este relevantă și ar strica aspectul plăcut al jocului.

- `public static float friction = 0.002f;`
  Forța de frecare care se va aplica bilelor. Nu se poate edita din setări pentru că ar strica uzabilitatea și experiența utilizatorului.

- `public static int timeScale = 10;`
  Este un scalar care definește cât de repede să se execute jocurile. Se poate edita din setări.

- `public static int countInGeneration = 50;`
  Numărul total de generații pe tură.

- `public static int mutationChance = 30;`
  Șansa de mutație a unei lovituri dintre o generație.

- `public static int minShootsCount = 5;`
  Numărul minim de lovituri al fiecărei generații.

- `public static int maxShootsCount = 7;`
  Numărul maxim de lovituri al fiecărei generații.

- `public static int GetRandomAngle() {...}`
  Cu ajutorul acestei funcții pot obține un unghi aleator, în grade.

```
public static int GetRandomAngle()
{
    return Engine.rnd.Next(360);
}
```

- `public static float GetRandomForce() {...}`
  Cu ajutorul acestei funcții pot obține o putere de lovit aleatoare. Intervalul valorii va fi [0.7, 7]. Este un interval ales arbitrar și consider că este suficient pentru acest program. Valoarea 0 reprezintă că nu este putere de lovit iar o valoare foarte mare ar fi făcut jocul nejucabil.

```
public static float GetRandomForce()
{
    return 0.7f + (float)Engine.rnd.NextDouble() * 7f;
}
```

2.7 Clasa Ross

Clasa aceasta este responsabilă de desenarea obiectelor. Are identificatorii de acest public și static pentru că reține variabile și metode globale, folosite de toate celelalte clase. Orice apel la variabilele acestea se face prin Ross.NumeVariabilă.

```
public static class Ross
{

    public static float Width { get { return canvas.Width; } }
    public static float Height { get { return canvas.Height; } }

    static Graphics grp;
    static Bitmap bmp;
    static PictureBox canvas;

    public static void Init(PictureBox pictureBox) {...}

    public static void Draw(Hole hole) {...}
    public static void Draw(Ball ball) {...}

    public static void Clear() {...}

    public static void BeginDraw() {...}
    public static void EndDraw() {...}
}
```

- `public static float Width { get { return canvas.Width; } }`
  Lățimea pânzei pe care se poate desena. Este și lățimea mesei de joc.

- `public static float Height { get { return canvas.Height; } }`
  Înălțimea pânzei pe care se poate desena. Este și înălțimea mesei de joc.

- `static PictureBox canvas;`
  Este pânza/masa noastră care de fapt este o referință către controlul care se află pe formularul de joc și va conține obiectele desenate.

- `static Bitmap bmp;`
  Pe acesta se vor face desenele efective și se va seta ulterior ca sursă pentru pânza noastră.

- `static Graphics grp;`
  Cu ajutorul ei putem desena pe variabila bmp.

- `public static void Init(PictureBox pictureBox) {...}`
  Metoda care se apelează la începutul programului pentru a inițializa variabilele necesare și mediul nostru grafic.

```
public static void Init(PictureBox pictureBox)
{
    canvas = pictureBox;
    bmp = new Bitmap(canvas.Width, canvas.Height);
    grp = Graphics.FromImage(bmp);
    Clear();
}
```

- `public static void Draw(Hole hole) {...}`
  Metoda aceasta ia ca parametru o instanță a clasei de gaură și o afișează pe ecran la poziția corespunzătoare.

```
public static void Draw(Hole hole)
{
    grp.FillEllipse(Brushes.Black, hole.position.x - Prefs.ballSize / 2,
hole.position.y - Prefs.ballSize / 2, Prefs.ballSize, Prefs.ballSize);
}
```

- `public static void Draw(Ball ball) {...}`
  Metoda aceasta ia ca parametru o instanță a clasei de bilă și o afișează pe masă la poziția și culoarea proprie. Menționez că am lăsat codul responsabil cu alte informații cum sunt direcția și puterea de lovit cu care se mișcă bila comentată. Pentru că am considerat că este o informație de mai puțină valoare. Dar la nevoie acesta poate fi din nou implementate cu ușurință.

```
public static void Draw(Ball ball)
{
    grp.FillEllipse(new SolidBrush(ball.color), ball.position.x - Prefs.ballSize / 2,
ball.position.y - Prefs.ballSize / 2, Prefs.ballSize, Prefs.ballSize);

    //grp.DrawString("Direction: " + ball. direction.ToString(), new Font("Arial",
10f),Brushes.Black, ball.position.x, ball.position.y-20);

    //grp.DrawString("Force: "+ball.force.ToString(), new Font("Arial",
10f),Brushes.Black, ball.position.x, ball.position.y);
}
```

- `public static void Clear() {...}`
  Golește toată suprafața grafică, lăsând doar culoarea mesei la vedere. Am nevoie de această funcție fiindcă, mediul grafic necesită ștergerea elementelor vechi pentru a le afișa pe cele noi.

```
public static void Clear()
{
    BeginDraw();
    EndDraw();
}
```

- `public static void BeginDraw() {...}`
  Aceasta este prima parte din ștergerea suprafeței grafice. Cu ajutorul ei golesc tot ce se vede pe masă.

```
public static void BeginDraw()
{
    grp.Clear(Prefs.tableColor);
}
```

- `public static void EndDraw() {...}`
  Aceasta este cea de a doua parte din ștergerea suprafeței grafice. Cu ajutorul ei pot aplica schimbările făcute pe ecran.

```
public static void EndDraw()
```

```
{
    canvas.Image = bmp;
}
```

2.8 Clasa Hole

Aceasta este o clasă auxiliară simplă pentru a reprezenta găurile mese din joc.

```
public class Hole
{
    public Vector2 position;

    public Hole(float x, float y){...}

    public void Draw(){...}
}
```

- `public Vector2 position;`
  Coordonatele propriei de pe masă.

- `public Hole(float x, float y){...}`
  Constructorul de bază și singurul de care este nevoie, pentru a crea o instanță a găurii.

```
public Hole(float x, float y)
{
    this.position = new Vector2(x, y);
}
```

- `public void Draw(){...}`
  Funcție care va afișa gaură

```
public void Draw()
{
    Ross.Draw(this);
}
```

3 Dezvoltarea aplicației din punct de vedere al programării vizuale

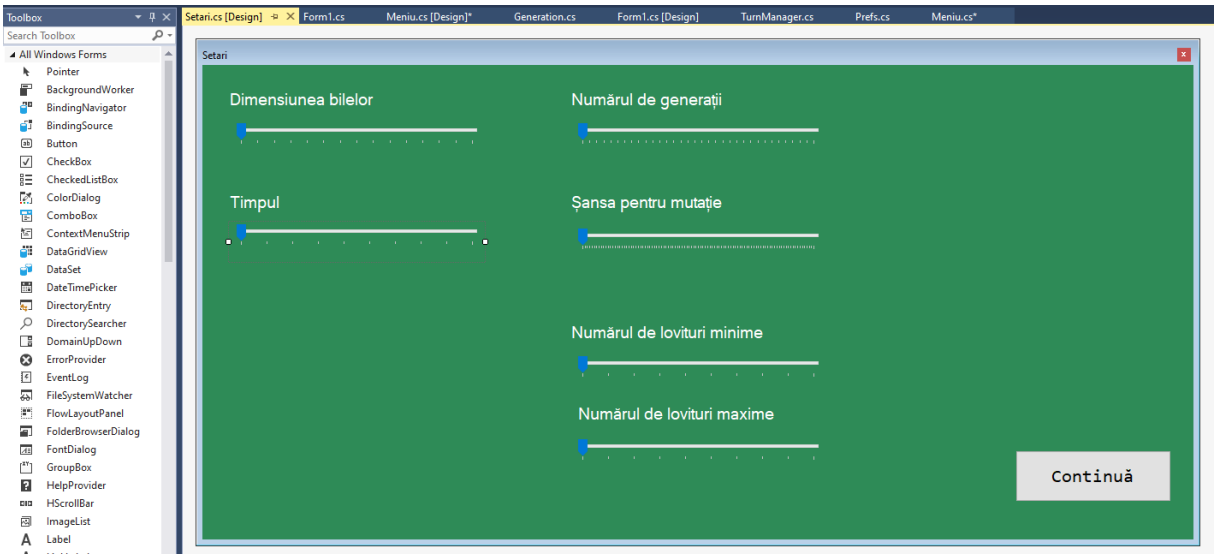
3.1 Formularul de Meniu

Figură 4 Interfața formularului de meniu al programului

Acesta este primul formular care apare la rularea aplicației, conținând titlul și două butoane pentru navigare.

```
public partial class Meniu : Form
{
        public Meniu()
        {
                InitializeComponent();
        }

        private void buttonStart_Click(object sender, EventArgs e)
        {
                Setari setari = new Setari();
                setari.ShowDialog();
                this.Close();
        }
        private void buttonExit_Click(object sender, EventArgs e)
        {
                this.Close();
        }
}
```

3.2 Formularul de Setări



Figură 5 Interfața formularului de setări ale jocului

Acest formular are ca scop să ofere utilizatorului un mediu de unde va putea schimba datele de intrare al jocurilor.

Utilizatorii au la îndemână o varietate de configurări, toate ajustabile cu ajutorul trackbar[12]-urilor. Acesta este cea mai bună controală, permițând să seteze un minim și un maxim cu ușurință, oferind în același timp o uzabilitate foarte simplă pentru utilizator. Aceste configurări sunt virtuali infinite. Singurul motiv pentru care sunt limitate este pentru că, folosind unele valori, jocului ar fi imposibil de jucat, spre exemplu dacă am seta mărimea bilelor mai mare decât cea a mesei.

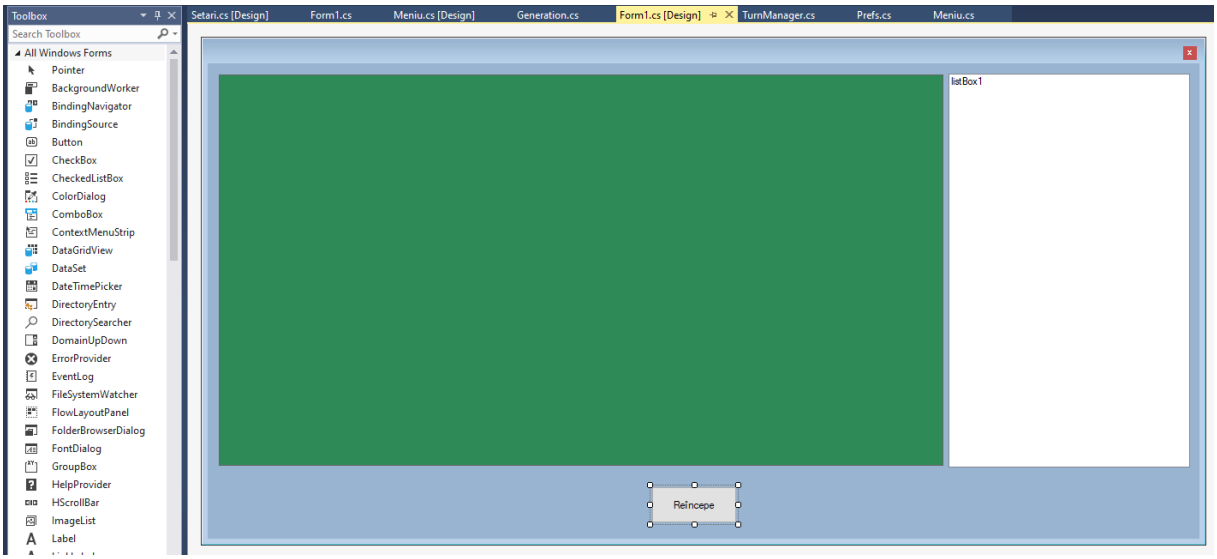
Valorile sunt setate implicit pe niște date acceptabile, schimbarea lor este doar cu scop experimental.

```
public partial class Setari : Form
{
	public Setari()
	{
		InitializeComponent();
	}
	private void Setari_Load(object sender, EventArgs e)
	{
		trackBarBallSize.Value = (int)Prefs.ballSize;
		trackBarTimeScale.Value = Prefs.timeScale;

		trackBarGenerationsCount.Value = Prefs.countInGeneration;
		trackBarMutationChance.Value = Prefs.mutationChance;

		trackBarMinShootCount.Value = Prefs.minShootsCount;
		trackBarMaxShootCount.Value = Prefs.maxShootsCount;
	}

	private void buttonBack_Click(object sender, EventArgs e)
	{
		if(trackBarMinShootCount.Value> trackBarMaxShootCount.Value)
```

[12] Control Windows al cărei valoare se alege prin tragerea la stânga sau la dreapta al unui obiect.

```
            {
                MessageBox.Show("Numărul de lovituri minime trebuie să fie mai
puțin sau egal, cu numărul maxim de lovituri!", "A apărut o problemă!!");
                return;
            }
            Form1 form1 = new Form1();
            form1.ShowDialog();
            this.Close();
        }

        private void trackBarBallSize_Scroll(object sender, EventArgs e)
        {
            Prefs.ballSize = trackBarBallSize.Value;
        }
}
```

3.3 Formularul de Joc



Figură 6 Interfața formularului de joc

În acest formular se va găsi conținutul aplicației și rularea algoritmului, pe baza datelor de intrare configurate pe formularul de deasupra.

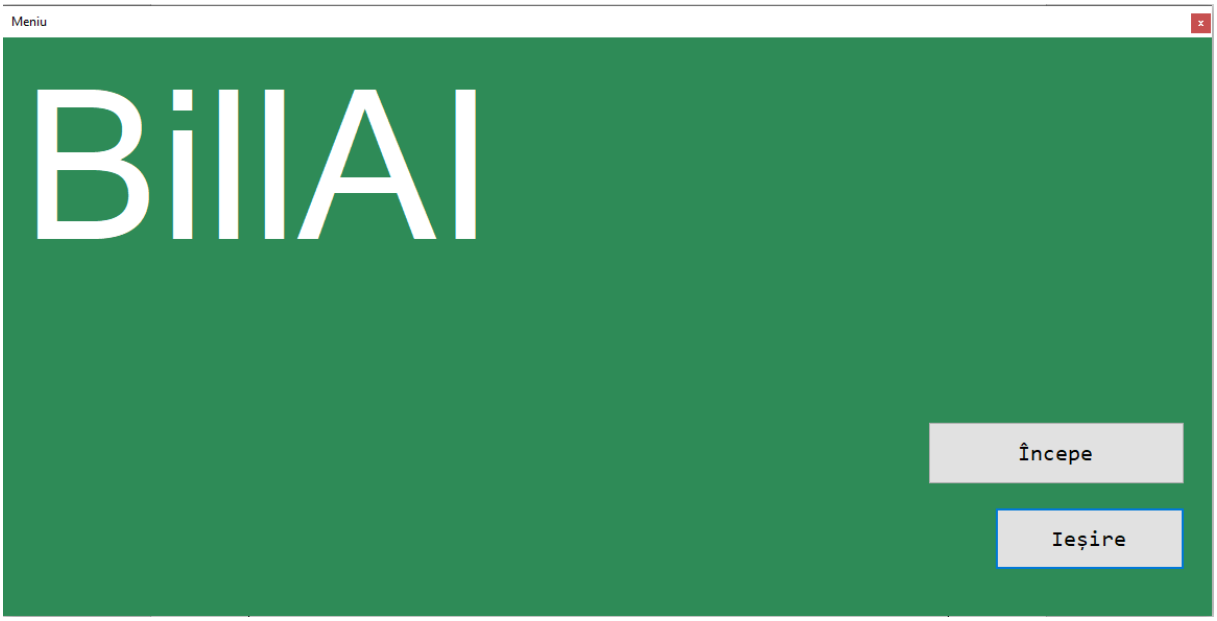
În partea stângă găsim controlul de picturebox[13] în care se vor randa [14]jocurile.
În partea dreaptă se află un listbox, folosit pentru afișarea mesajelor către utilizator.
Iar în partea de jos centrat, se găsește un buton care va reporni toată aplicația, în cazul în care dorim să utilizăm programul cu alte date de intrare.

```
public partial class Form1 : Form
{
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
```

[13] Control Windows pentru afișarea unei imagini fie încărcată fie generată din program.
[14] Originat din limba engleză, În cazul nostru procesul de randare înseamnă desenarea obiectelor pe ecranul utilizatorului.

```
            Ross.Init(pictureBox1);
            Engine.Init(listBox1);
        }

        private void pictureBox1_Paint(object sender, PaintEventArgs e)
        {
            Engine.Update();
        }

        private void buttonRestart_Click(object sender, EventArgs e)
        {
            System.Diagnostics.Process.Start(Application.ExecutablePath);
            this.Close();
        }
}
```

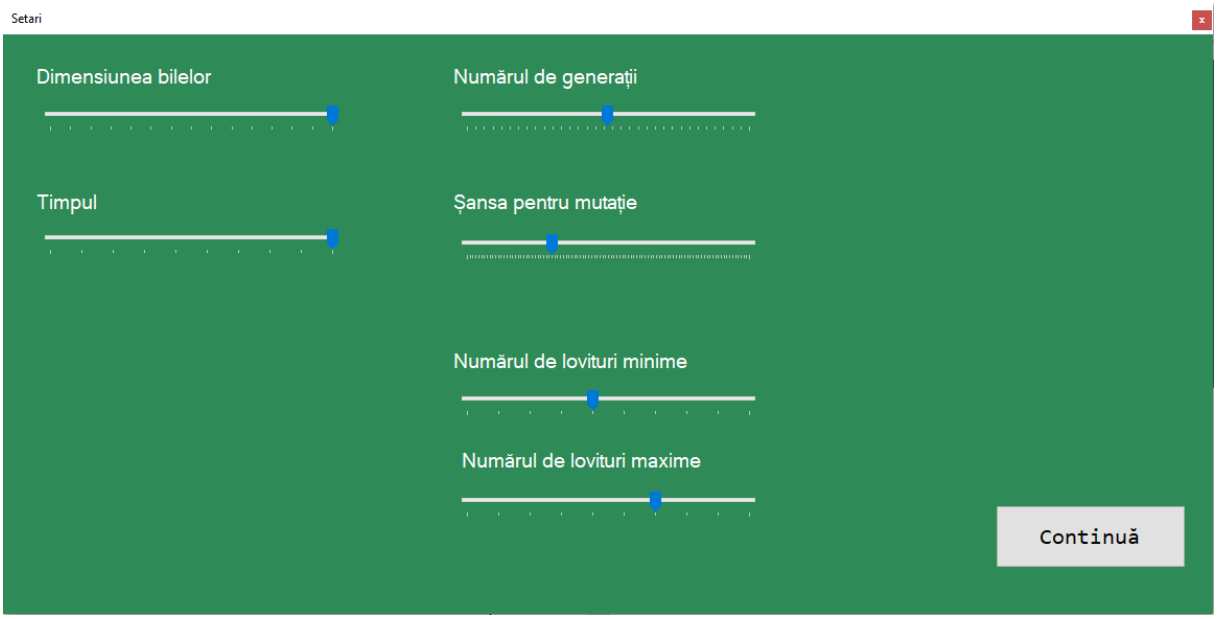
4 Utilizarea aplicației



Figură 7 Fereastra de meniu

La pornirea programul acest formular se va deschide conținând:

- butonul de „Începe” care te va trimite către formularul cu setările jocului.
- butonul de „ieșire” care va închide aplicația.



Figură 8 Fereastra de setări

Pe acest formular utilizatorul are la dispoziție următoarele configurări:

- ***Dimensiunea bilelor***
  Reprezintă mărimea bilelor de biliard de pe masă. De asemenea și mărimea găurilor.

- ***Timpul***
  Este un scalar care definește cât de repede să se execute pașii. Minimul și implicitul este de 1, cea ce înseamnă că totul va fi la o viteză normal . Acesta poate crește până la 10. Orice valoare mai mare decât 1 va face ca lucrurile să se execute mult mai repede. Mecanism inspirat de variabila timeScale[15] din clasa Time[16] din Unity[17].

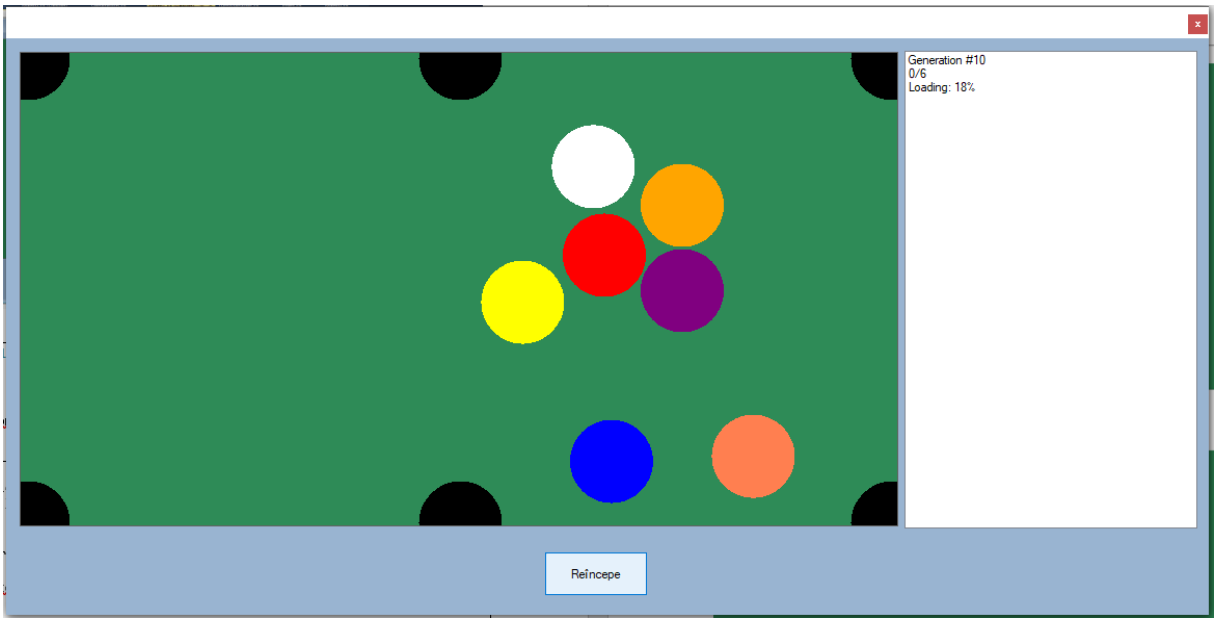
- ***Numărul de generații***
  Totalul de generații care sa ruleze per tura.
  Fiindcă este vorba despre un algoritm genetic cu cât mai multe generații cu atât mai repede vom ajunge la rezultatele dorite. Dar această setare poate să ajungă până la 100 de generații pe tură și având un minim de o generație.

- ***Șansa pentru mutație***
  Când o generație se mutează [18]fiecare lovitură din acesta are posibilitatea de a se schimba la o lovitură nouă, complet aleatoare. Această configurare reprezintă o șansă ca o lovitură să se schimbe. Setată la maxim toate loviturile din generație se vor schimba. Setată la minim nici o lovitură nu se va schimba.

- ***Numărul de lovituri minime***
  Reprezintă câte lovituri, cel puțin, trebuie să execute fiecare generație.

- ***Numărul de lovituri maxime***
  Reprezintă câte lovituri, cel mult, poate să execute fiecare generație.
  Acesta trebuie să fie mai mare sau egală, cu numărul de lovituri minime.

[15] Variabilă din mediul Unity, care se referă la cât de repede sau încet trece timpul din joc comparativ cu cea din lumea reală.
[16] Clasa din mediul Unity, care se ocupă de timpul din jocuri.
[17] Program de dezvoltarea jocurilor video.
[18] Schimbare de formă.



Figură 9 Fereastra de joc

Din acest moment utilizatorul aplicației poate să urmărească decursul jocurilor și progresul de învățare ale algoritmului. La început acesta va juca ca un nepriceput, similar cu un om care joacă pentru prima oară. Dar în timp acest program va fi capabil să joace mult mai bine, iar după multe generații acesta va putea executa lovituri extraordinare, băgând mai multe bile dintre singură lovitură.

Este important să menționez că rezultatul final și viteza cu care va ajunge acolo va diferi aproape de fiecare dată, datorită a doi factori externi:

1. Configurările făcute de utilizator de pe formularul anterior.
   Spre exemplu: un joc in care dimensiunea bilelor foarte mare și cu multe generații, îi va fi mai ușor și va da rezultate mult mai repede decât un joc în care bilele sunt mici și numărul maxim de lovituri este scăzută.

2. Valorile fiecărei lovituri din prima generație
   Datorita faptului că, programul are la bază algoritmul genetic, direcțiile și puterile de lovit din prima generație sunt alese aleator de către program. Acestea vor fi modificate pe parcursul jocului, dar în prima generație totul este aleator.

Pe panoul din partea stângă vom vedea în timp real, cel mai bun joc jucat până la generația actuală. Și celelalte jocuri sunt jucate în fundal, dar afișându-le pe toate în același timp nu ar fi fost atât de ușor pentru utilizator să înțeleagă ce se întâmplă și evoluția programului

În partea dreaptă putem observa cele trei informații cheie:

- Prima fiind textul *„Generation #10”*, care înseamnă că la momentul actual se află la generația cu numărul 10. 9 generație au fost deja jucate și o urmărim pe a zecea.

- Al doilea text, adică „0/6” reprezintă punctajul curent al celui mai bun jucător de până acum. Prima valoare, „0” , este numărul de bile pe băgate, iar cea de a doua, „6”, este numărul total de bile . Punctajul se schimbă la fiecare băgare de bilă, crescând cu un punct. În cazul in care programul reușește să bage bila albă, punctajul se va seta automat „-1/6”.
- „Loading: 18%” este al treilea text și arată numărul de generații care s-au terminat de jucat în fundal.
  0% înseamnă că toate jocurile din fundal plus cea pe care o avem în față sunt încă în mișcare, iar 100% va fi atunci când toate jocurile s-au terminat. Un joc se consideră terminat când s-au efectuat numărul maxim de lovituri posibile sau bilă albă a fost băgată.
  Doar după ce a ajuns la 100% vor veni noile generații va bazate pe cele vechi
  Cu cât mai mare a fost aleasă setarea numărul de generații din Formularul de Setări cu atât mai mult va dura să ajungă la 100%.

5 Bibliografie

- Cursurile şi laboratoarele Facultăţii de Ştiinţe ("Algoritmi şi structuri de date", "Programare orientată pe obiecte" şi "Medii vizuale de programare")
- Laslo E, Ionescu V.S. Algoritmică C++, MatrixRom Bucureşti 518 pagini 2010, ISBN 978-973-755-640-0
- David WELLER, Alexandre Santos LOBÃO, Ellen HATTON, .NET Game Programming in C#, Apress ISBN(pbk) 1-59059-319-7, New-York, USA
- https://msdn.microsoft.com/en-us/library/dd492132.aspx
- https://msdn.microsoft.com/en-us/library/windows/desktop/dd145203(v=vs.85).aspx
- http://csharp.net-informations.com/gui/cs_forms.htm

DECLARAȚIE DE AUTENTICITATE
A
LUCRĂRII DE FINALIZARE A STUDIILOR

Titlul lucrării BillAI

Autorul lucrării: Dőmőtőr Zsolt - Béla

Lucrarea de finalizare a studiilor este elaborată în vederea susținerii examenului de finalizare a studiilor organizat de către Facultatea ____DE INFORMATICĂ ȘI ȘTIINȚE___ din cadrul Universității din Oradea, sesiunea _____24-26 IUNIE_____ a anului universitar___2019-2020_____.

Prin prezenta, subsemnatul (nume, prenume C.N.P.) _ Dőmőtőr Zsolt - Béla___ ____1971015055071_____

Declar pe proprie răspundere că această lucrare a fost scrisă de către mine, fără niciun ajutor neautorizat și că nicio parte a lucrării nu conține aplicații sau studii de caz publicate de alți autori.

Declar, de asemenea, că în lucrare nu există idei, tabele, grafice, hărți sau alte surse folosite fără respectarea legii române și a convențiilor internaționale privind drepturile de autor.

Oradea, Semnătura

Data ___16.06.2020___