

**Tema 2 -Arrow Puzzle Game**

Model – View – ViewModel

Problema 35

**Proiectare Sofware**

**-Calculatoare si Tehnologia Informatiei-**

Student: Iamnitchi Bogdan

Grupa: 30235

An: 3

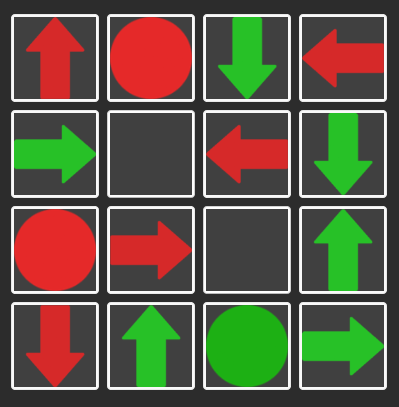
1. **Enuntul problemei**

Dezvoltați o aplicație pentru următorul joc cu 2 jucători: Se consideră săgeți de 2 culori, iar scopul jocului este de a completa zonele din pătrat astfel încât pe aceeași linie, pe aceeași coloană și pe aceeași diagonală să nu se găsească 2 săgeți orientate în aceeași direcție, indiferent de culoarea acestora. Pierde jucătorul care nu mai are nicio posibilitate de a așeza săgeți în zonele libere ale pătratului. Implementarea se va realiza astfel încât un utilizator al aplicației (jocului) să joace cu calculatorul. Se va utiliza o variantă a algoritmului MINIMAX.

**Jucătorii** vor putea să se autentifice și să-și vizualizeze punctajul.

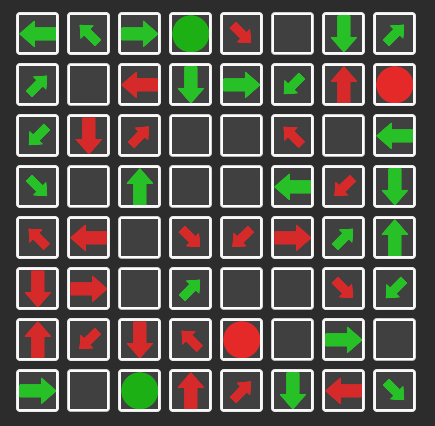
Utilizatorii de tip **administrator** pot efectua următoarele operații după autentificare:

* Operații CRUD pentru informațiile legate de jucătorii care s-au autentificat;
* Vizualizarea listei tuturor jucătorilor autentificați.

**Nivel 1:**

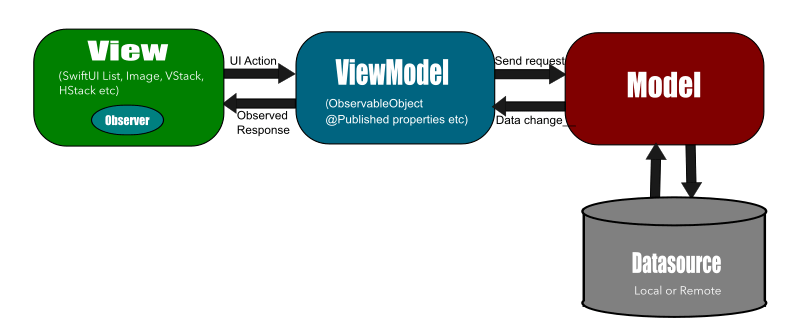
*Figura 1: Nivel 1*

**Nivel 2:**



*Figura 2: Nivel 2*

**Arhitectutura MVVM**

Arhitectura MVVM, cunoscută și sub numele de Model-View-ViewModel, este un model de proiectare de software care separă logica de afișare (UI) de logica de afaceri a unei aplicații. Iată o descriere detaliată a fiecărei componente a arhitecturii MVVM:

*Figura 3: Arhitectura MVVM*

**Model (Modelul):**

* Reprezintă datele și logica de afaceri a aplicației.
* Nu știe nimic despre interfața utilizatorului sau despre cum sunt afișate datele.
* Este responsabil pentru gestionarea datelor, validare, accesul la resurse externe etc.

**View (Vizualizarea):**

* Reprezintă interfața utilizatorului (UI) a aplicației.
* Afișează datele utilizatorului și interacționează cu utilizatorul.
* Nu conține logica de afaceri; este doar responsabilă pentru afișarea informațiilor și colectarea interacțiunilor utilizatorului.

**ViewModel (Modelul de vizualizare):**

* Conectează Modelul și Vizualizarea.
* Conține logica de afaceri a aplicației care nu este strict legată de interfața utilizatorului.
* Transformă datele din Model într-o formă ușor de afișat pentru Vizualizare și invers.
* Poate gestiona și starea vizualizării, cum ar fi vizualizarea sau ascunderea unor elemente, activarea sau dezactivarea unor controale etc.
* În arhitectura MVVM, interacțiunea dintre aceste trei componente este gestionată astfel:

**View este legat de ViewModel** printr-un mecanism de legare a datelor (data binding). Acesta permite actualizarea automată a Vizualizării atunci când datele din ViewModel sunt modificate.

**ViewModel este legat de Model** și poate accesa datele și logica de afaceri din Model pentru a le transforma într-o formă potrivită pentru Vizualizare.

**Modelul poate fi accesat direct de către ViewModel** pentru a obține și actualiza datele și pentru a efectua operațiuni de logică de afaceri.

Arhitectura MVVM încurajează separarea clară a responsabilităților și facilitează testarea unităților, deoarece fiecare componentă poate fi testată separat. De asemenea, promovează reutilizarea codului și modularitatea, deoarece fiecare componentă este independentă și poate fi înlocuită sau extinsă fără a afecta celelalte componente.

1. **Instrumente utilizate**

Am dezvoltat aplicația în C# folosind mediul de dezvoltare integrat (IDE) Visual Studio. Acest IDE mi-a oferit un mediu puternic de dezvoltare, incluzând funcții precum completarea automată a codului, depanare și gestionarea proiectelor, facilitând astfel procesul de dezvoltare.

Pentru a scrie codul aplicației, am folosit limbajul de programare C#. Acesta este un limbaj puternic, orientat pe obiecte, dezvoltat de Microsoft, cu o sintaxă curată și ușor de înțeles, ceea ce mi-a permis să dezvolt aplicația cu eficiență și să obțin rezultatele dorite.

Aplicația desktop pe care am dezvoltat-o a fost construită folosind framework-ul .NET. Acest framework a oferit o colecție bogată de biblioteci și un mediu de rulare pentru dezvoltarea și executarea diferitelor tipuri de aplicații, inclusiv aplicații desktop, ceea ce mi-a permis să creez un produs final robust și funcțional.

Pentru a gestiona interfața utilizatorului și interacțiunile cu utilizatorul, am utilizat biblioteci grafice precum Windows Forms .NET Framework. Aceste biblioteci au facilitat crearea unei interfețe grafice atractivă și intuitivă pentru utilizator, contribuind la experiența generală a jocului.

În ceea ce privește sunetul și muzica din joc, am folosit biblioteci precum NAudio sau alte soluții care permit redarea și manipularea sunetului, adăugând astfel o dimensiune suplimentară experienței de joc.

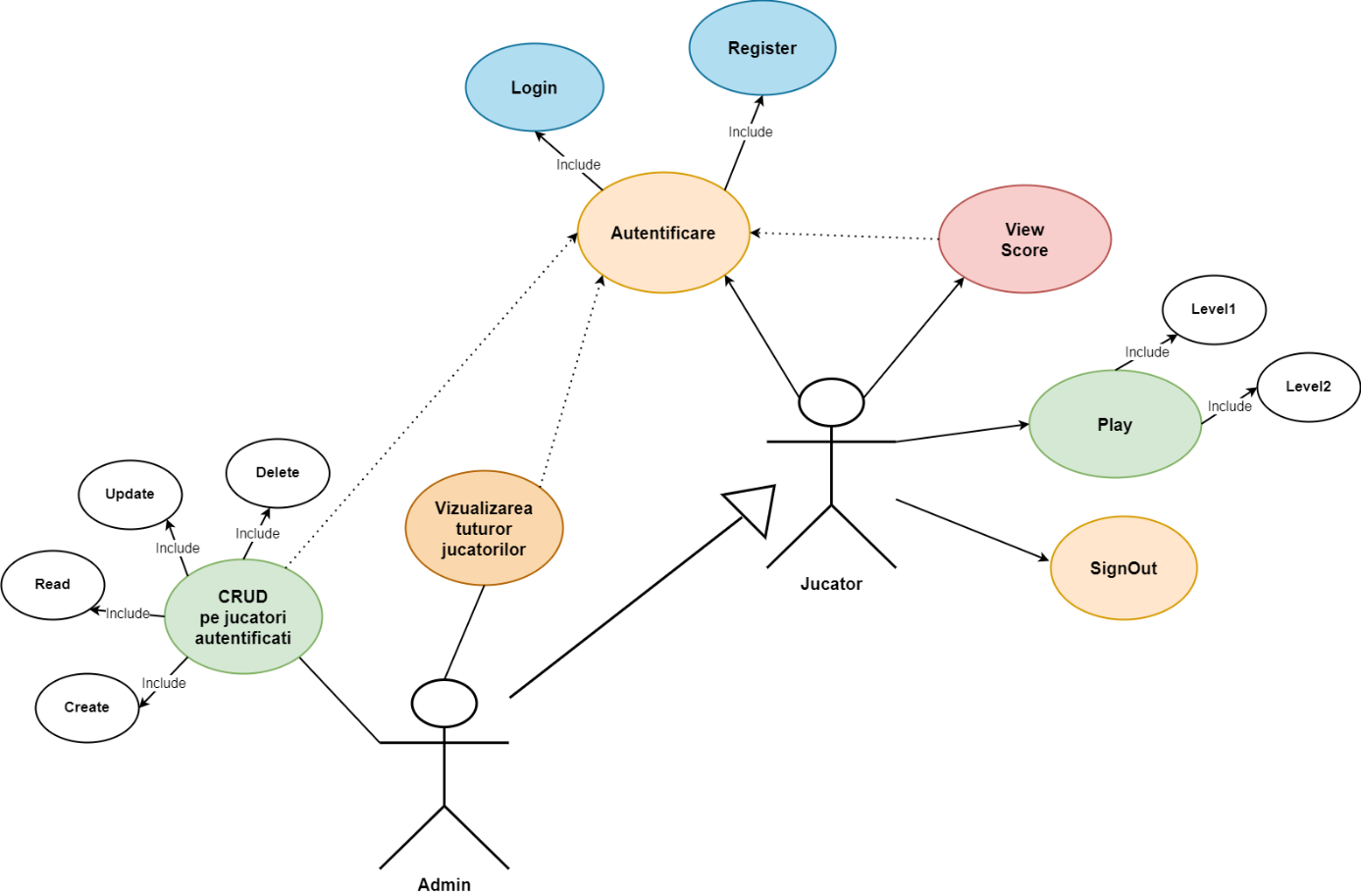
Deși opțional, am integrat și un sistem de bază de date pentru a gestiona salvarea datelor utilizatorului sau a progresului în joc. Am putut folosi baze de date precum SQL Server, SQLite sau alte soluții de stocare a datelor pentru a asigura o experiență personalizată și persistentă pentru utilizatori.

Pentru gestionarea codului sursă al aplicației și pentru a lucra eficient în echipă, am utilizat un sistem de control al versiunilor, cum ar fi Git. Acest lucru mi-a permis să monitorizez și să gestionez modificările în codul sursă al aplicației într-un mod organizat și colaborativ.

Acestea sunt instrumentele principale pe care le-am utilizat în dezvoltarea aplicației mele în C#, contribuind la crearea unui joc captivant și funcțional pentru utilizatori.

1. **Proiectare si implementare**

**Diagrama USE CASE**

Prima etapa in realizarea acestei teme, a fost creearea diagramei use-case in aplicatia Drawio unde am pus actorii principali precum si cazurile de utilizare cele mai importante pentru a avea o privire de ansamblu asupra a ceea ce urmeaza sa implementez.

*Figura 4: Diagrama UseCase*

Aici se afla Use-Case-ul problemei. Drept actori, avem un administrator si un Jucator, acestia putand face diverse actiuni.

**Autentificare Administrator:**

* Actor Principal: Administratorul
* Descriere: Acest caz de utilizare permite administratorului să se autentifice în sistem pentru a avea acces la funcționalitățile administrative.
* Acțiuni:
* Administratorul introduce numele de utilizator și parola în interfața de autentificare.
* Sistemul verifică credențialele și autentifică administratorul.
* Dacă autentificarea reușește, administratorul are acces la funcționalitățile administrative (CRUD pe jucători etc.).
* Dacă autentificarea eșuează, administratorul primește un mesaj de eroare și nu are acces la funcționalitățile administrative.

**CRUD Player (Create, Read, Update, Delete Player):**

* Actor Principal: Administratorul
* Descriere: Acest caz de utilizare permite administratorului să creeze, să vizualizeze, să actualizeze și să șteargă informații despre jucători în sistem.
* Acțiuni:
  + Administratorul accesează secțiunea de administrare a jucătorilor din interfața de administrare.
  + Administratorul poate crea un nou jucător, specificând detalii precum nume, scor etc.
  + Administratorul poate vizualiza detalii despre un jucător existent, inclusiv scorul său.
  + Administratorul poate actualiza informațiile unui jucător existent, cum ar fi scorul acestuia.
  + Administratorul poate șterge un jucător din sistem.

**Autentificare Jucător:**

* Actor Principal: Jucătorul
* Descriere: Acest caz de utilizare permite jucătorului să se autentifice în sistem pentru a avea acces la funcționalitățile destinate jucătorilor.
* Acțiuni:
  + Jucătorul introduce numele de utilizator și parola în interfața de autentificare.
  + Sistemul verifică credențialele și autentifică jucătorul.
  + Dacă autentificarea reușește, jucătorul are acces la funcționalitățile destinate jucătorilor (vizualizarea scorului, jocul etc.).
  + Dacă autentificarea eșuează, jucătorul primește un mesaj de eroare și nu are acces la funcționalitățile destinate jucătorilor.

**Vizualizare Scor:**

* Actor Principal: Jucătorul
* Descriere: Acest caz de utilizare permite jucătorului să vizualizeze scorul său în sistem.
* Acțiuni:
  + Jucătorul accesează secțiunea destinată vizualizării scorului din interfața de joc.
  + Sistemul afișează scorul jucătorului în interfața de joc.

**Joc:**

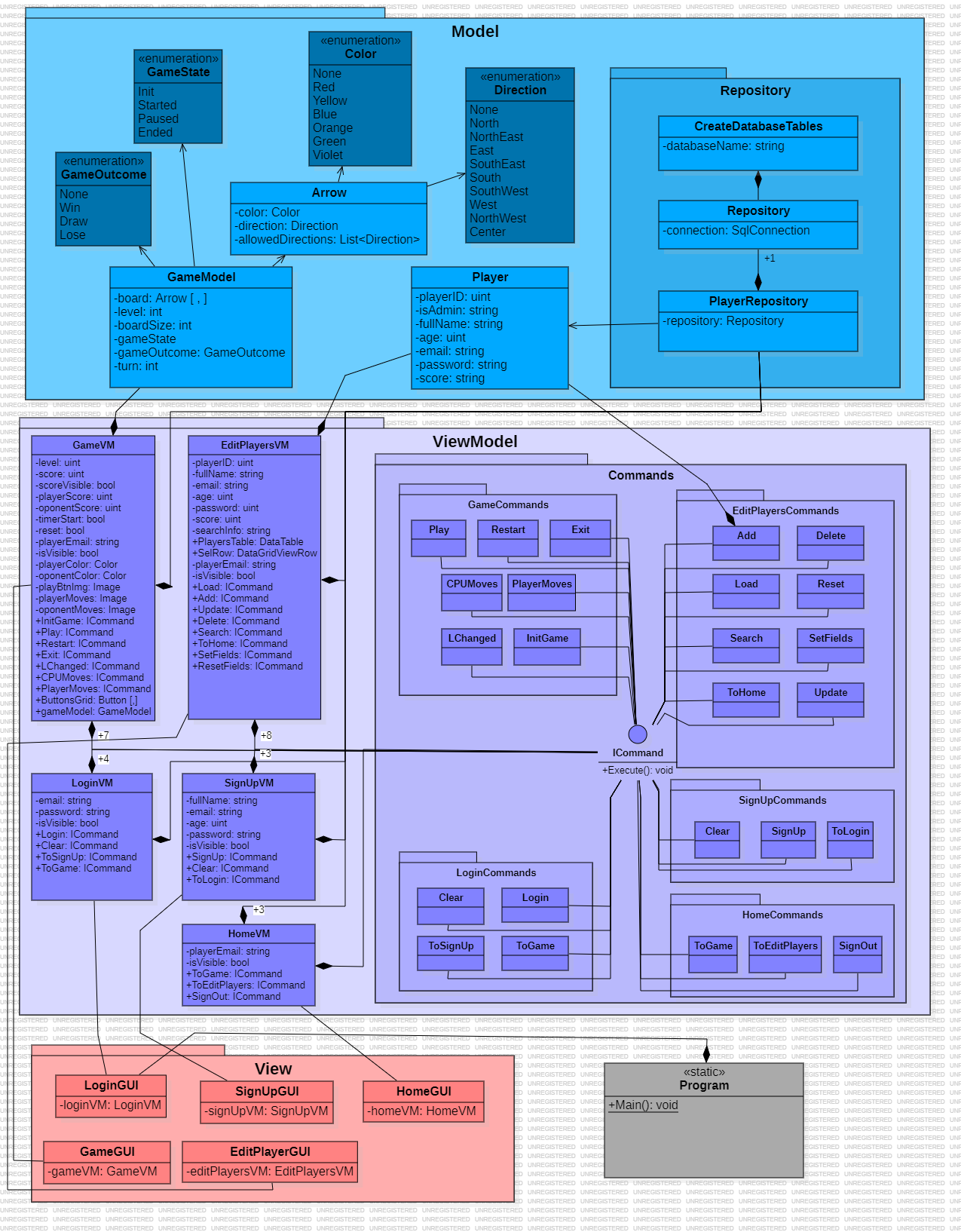
* Actor Principal: Jucătorul
* Descriere: Acest caz de utilizare permite jucătorului să joace jocul.
* Acțiuni:
  + Jucătorul accesează secțiunea de joc din interfața de joc.
  + Poate selecta ce nivel de joc doreste sa joace
  + Jucătorul interacționează cu jocul conform mecanicilor și regulilor stabilite.
  + Sistemul actualizează scorul jucătorului în timpul jocului conform performanței sale.

**SignOut:**

* Actor Principal: Jucătorul
* Descriere: Acest caz de utilizare permite jucătorului să joace jocul.
* Acțiuni: jucatorul se deconecteaza din contul curent

**Diagrama de clase**

Pentru diagrama de clase am folosit aplicatia StarUML. Voi prezenta o diagrama de clase in care am pus doar atributele ca sa se poata vedea cu usurina legatura dintre clase si pachete. Acestea au un nume sugestiv: Model, View si ViewModel. Fiecare pachet are mai multe clase implementate in aplicatie, acestea avand atributele si operatiile necesare functionarii aplicatiei.



*Figura 5: Diagrama de clase*

În arhitectura MVVM (Model-View-ViewModel), diagrama de clase ilustrează modul în care cele trei componente - Modelul, Vizualizarea și Modelul de Vizualizare - sunt conectate și interacționează între ele. Iată o descriere generală a diagrammei de clase în cadrul arhitecturii MVVM:

**Model (Modelul):**

• Clasa reprezentând Modelul conține datele și logica de afaceri a aplicației.

• Aceasta include structurile de date și metodele necesare pentru gestionarea jucătorilor, cum ar fi numele, punctajele etc.

• Modelul poate conține, de asemenea, metode pentru a efectua operațiuni CRUD (Create, Read, Update, Delete) pe datele jucătorilor.

**View (Vizualizarea):**

• Clasa View reprezintă interfața utilizatorului și este responsabilă pentru afișarea informațiilor și interacțiunea cu utilizatorul.

• În diagrama de clase, aceasta poate include elementele UI, cum ar fi butoanele, etichetele, casetele de text etc.

• Vizualizarea este pasivă și nu conține logica de afaceri. În schimb, comunica cu Modelul de Vizualizare pentru a obține și a afișa datele necesare.

**ViewModel (Modelul de Vizualizare):**

• Clasa ViewModel acționează ca un intermediar între Model și Vizualizare. Este responsabilă pentru transformarea datelor din Model într-o formă adecvată pentru Vizualizare și gestionarea stării acesteia.

• În diagrama de clase, Modelul de Vizualizare poate conține referințe către Model și Vizualizare, permițându-i să interacționeze cu acestea.

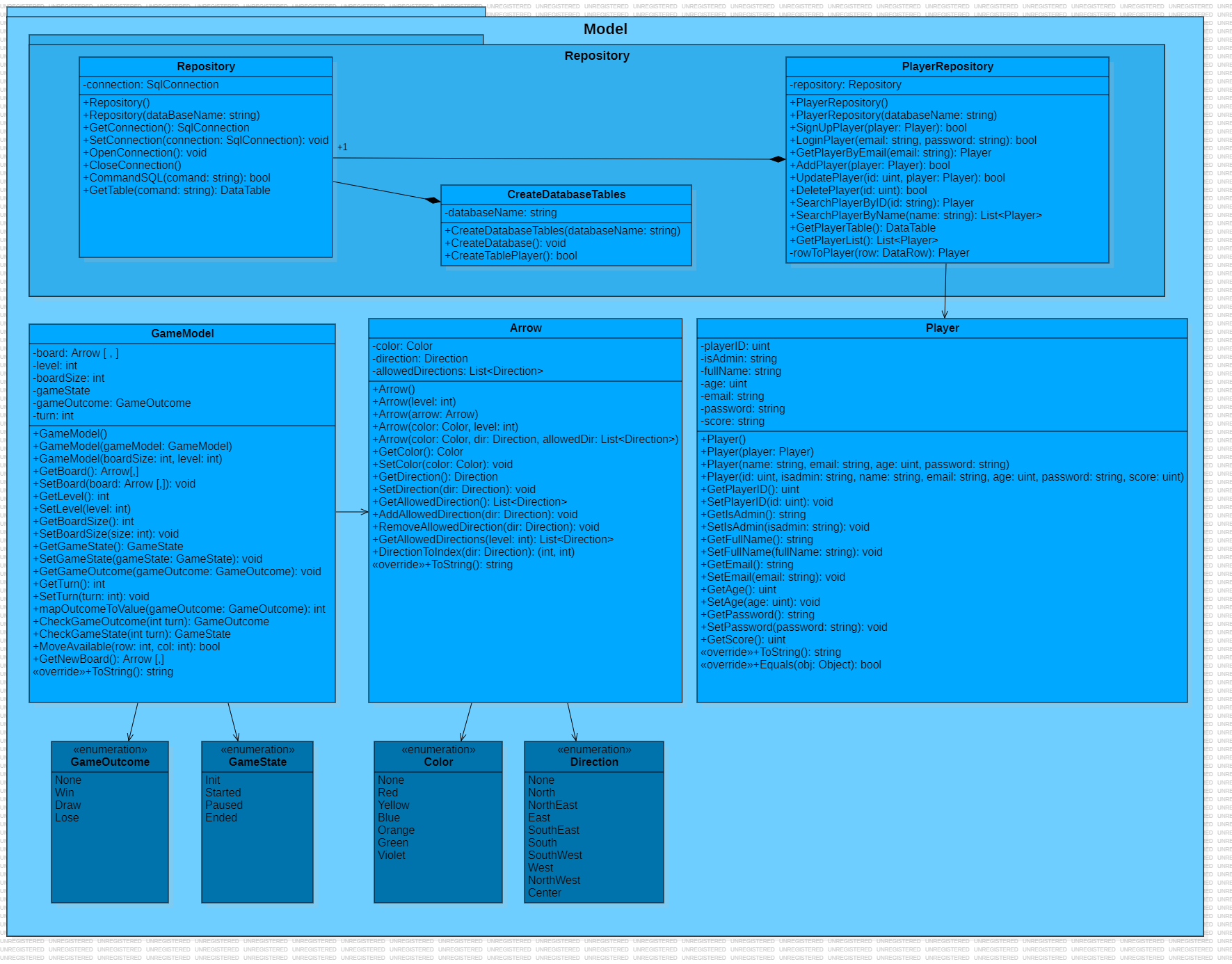
• De asemenea, poate conține metode pentru a gestiona acțiunile utilizatorului și pentru a comunica cu Modelul pentru a efectua operațiuni CRUD sau alte operațiuni de procesare a datelor.

• Modelul de Vizualizare poate utiliza un mecanism de notificare pentru a actualiza automat Vizualizarea atunci când datele din Model sunt modificate.

În diagrama de clase MVVM, conexiunile dintre aceste componente arată cum acestea interacționează între ele. De exemplu, Vizualizarea poate fi legată de Modelul de Vizualizare prin intermediul mecanismului de legare a datelor, iar Modelul de Vizualizare poate comunica cu Modelul pentru a obține și actualiza datele necesare.

Prin mecanismul de legare a datelor (data binding) din arhitectura MVVM, Vizualizarea și Modelul de Vizualizare sunt legate între ele într-un mod declarativ și bidirecțional. Acest lucru înseamnă că orice modificare a datelor în Modelul de Vizualizare este reflectată automat în Vizualizare, și invers. Astfel, atunci când utilizatorul interacționează cu elementele UI din Vizualizare, aceste interacțiuni sunt propagate către Modelul de Vizualizare, care le procesează și poate actualiza Modelul subiacent. În același timp, orice actualizare a datelor în Model este automat reflectată în Vizualizare, astfel încât să se mențină întotdeauna consistența între datele afișate și starea reală a aplicației.

**Pachetul Model:**



*Figura 6: Pachetul Model*

În cadrul pachetului model al arhitecturii MVVM, ai un subpachet numit "repository" care se ocupă de interacțiunea cu baza de date. Iată o descriere mai detaliată a claselor din acest subpachet:

**CreateDatabaseTables:**

* Această clasă este responsabilă pentru crearea tabelelor din baza de date necesare pentru stocarea datelor jucătorilor sau altor informații relevante pentru joc.
* Poate conține metode pentru a executa scripturi SQL pentru crearea tabelelor sau alte operațiuni necesare pentru inițializarea bazei de date.

**PlayerRepository:**

* Această clasă este responsabilă pentru gestionarea datelor jucătorilor din baza de date. Ea oferă operații CRUD (Create, Read, Update, Delete) pentru a interacționa cu datele jucătorilor.
* Metodele acestei clase pot include:
  + addPlayer(): pentru a insera un nou jucător în baza de date.
  + getPlayer(): pentru a obține informații despre un jucător specific din baza de date.
  + updatePlayer(): pentru a actualiza informațiile unui jucător existent în baza de date.
  + deletePlayer(): pentru a șterge un jucător din baza de date.

**Repository:**

* Această clasă este responsabilă pentru gestionarea conexiunii cu baza de date. Ea oferă metode pentru a stabili și a închide conexiunea cu baza de date.
* În plus, poate conține și metode pentru a executa interogări SQL și a gestiona tranzacțiile în cadrul bazei de date.
* Prin intermediul acestei clase, celelalte clase din pachetul repository pot comunica cu baza de date și pot efectua operațiile necesare.
* Prin intermediul acestor clase, pachetul repository din cadrul modelului oferă o abstracție eficientă pentru gestionarea interacțiunii cu baza de date. Acest lucru permite modelului să rămână independent de detaliile specifice ale bazei de date și să ofere o interfață simplificată pentru a efectua operațiile CRUD necesare pentru gestionarea datelor jucătorilor în cadrul jocului tău.

Ceea ce priveste modelul in sine avem uramtoarele:

**Clasa Arrow:**

* Această clasă reprezintă o săgeată în jocul cu săgeți. Fiecare săgeată are o culoare și o direcție specifică, precum și o listă de direcții permise pentru deplasare.
* Arrow se ocupă de definirea caracteristicilor individuale ale săgeții, inclusiv culoarea și direcția, și de specificarea direcțiilor în care poate fi mutată în timpul jocului.

**Clasa GameModel:**

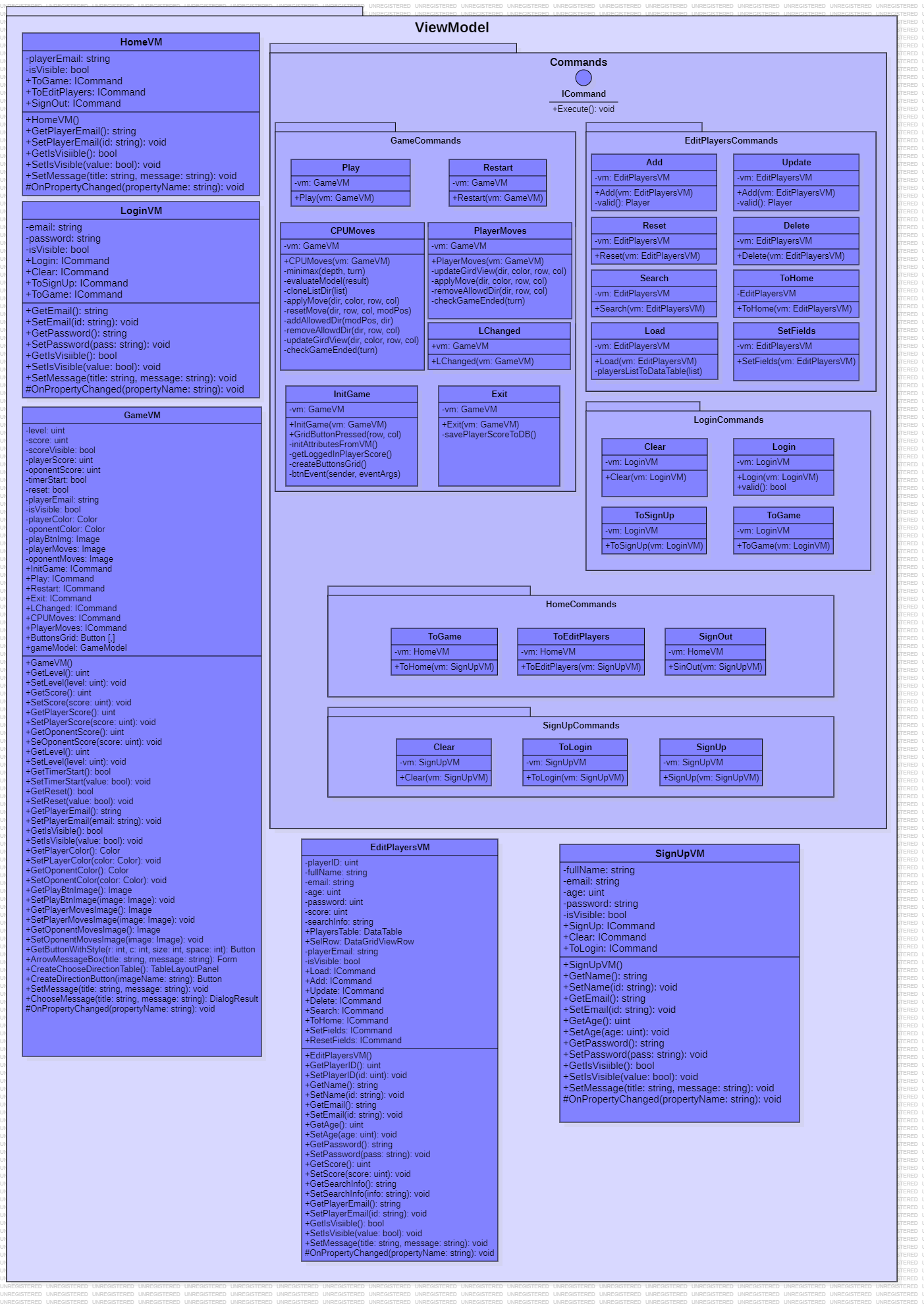
* Această clasă gestionează starea și logica generală a jocului. Ea se ocupă de lucruri precum tabla de joc, nivelul curent, starea jocului, rezultatul și al cui este rândul.
* GameModel se ocupă de inițializarea și gestionarea stării generale a jocului, inclusiv gestionarea tablei de joc și a evoluției partidei de-a lungul timpului.

**Clasa Player:**

* Această clasă reprezintă un jucător în cadrul jocului. Ea gestionează informații specifice despre fiecare jucător, precum numele, vârsta, scorul și drepturile de administrator.
* Player se ocupă de definirea și gestionarea datelor individuale ale fiecărui jucător, inclusiv informații personale și progresul în joc.

Am mai introdus si cateva enumeratii in ceea ce priveste **Directia** , **Culoarea**, **Starea Jocului** (initiala, in desfasurare, terminata) si **Rezultatul Joucului** (castig, pierdere, sau egalitate).

**Pachetul ViewModel:**



*Figura 7: Pachetul ViewModel*

În cadrul arhitecturii pe care am dezvoltat-o, pachetul ViewModel este esențial pentru gestionarea logicii de afaceri și comunicarea cu interfețele de utilizator. Am organizat ViewModel-urile pentru fiecare ecran sau flux de lucru din aplicație, incluzând clase precum HomeVM, LoginVM, GameVM și SignUpVM, EditPlayersVM. Acestea sunt strâns legate de View-urile corespunzătoare, cum ar fi HomeGUI, LoginGUI etc., pentru a asigura o comunicare eficientă între logica de afaceri și afișare.

Am proiectat atributele acestor ViewModel-uri pentru a se conecta dinamic cu View-urile asociate, folosind legarea datelor (data binding), astfel încât să poată fi actualizate și sincronizate în timp real. De asemenea, pentru a gestiona acțiunile utilizatorului, am creat un subpachet Commands în cadrul pachetului ViewModel, unde am definit toate comenzile asociate interacțiunilor utilizatorului.

Acest subpachet Commands este organizat în subpachete specifice pentru fiecare ViewModel, cum ar fi GameCommands, LoginCommands, EditPlayerCommands etc. Această structură modulară și clară permite o gestionare eficientă a comenzilor și logicii asociate acțiunilor utilizatorului pentru fiecare ecran sau flux de lucru din aplicație.

HomeVM (Modelul de Vizualizare pentru ecranul principal):

* Atribuțiile din HomeVM sunt strâns legate cu View-ul asociat, HomeGUI.
* Atributele din HomeVM, cum ar fi listele de jucători și punctaje, sunt legate prin legarea datelor cu elementele UI corespunzătoare din HomeGUI.
* În ceea ce privește comenzile specifice, HomeVM creează și gestionează GameCommands, LoginCommands etc., în pachetele corespunzătoare pentru a trata acțiunile specifice ecranului principal.

LoginVM (Modelul de Vizualizare pentru ecranul de autentificare):

* Atributele din LoginVM, precum numele utilizatorului și parola, sunt legate cu View-ul asociat, LoginGUI, prin legarea datelor.
* De asemenea, LoginVM creează și utilizează comenzi specifice pentru a gestiona acțiunile asociate autentificării, cum ar fi LoginCommands pentru procesul de conectare și validare a datelor introduse de utilizator.

GameVM (Modelul de Vizualizare pentru ecranul de joc):

* Atributele din GameVM, cum ar fi starea jocului și mutările jucătorilor, sunt legate cu View-ul asociat, GameGUI, pentru a actualiza și a afișa starea jocului.
* Pentru a gestiona acțiunile și comenzile specifice jocului, GameVM creează și utilizează GameCommands, care sunt definite în pachetul GameCommands și care sunt asociate cu acțiunile și evenimentele din cadrul jocului.

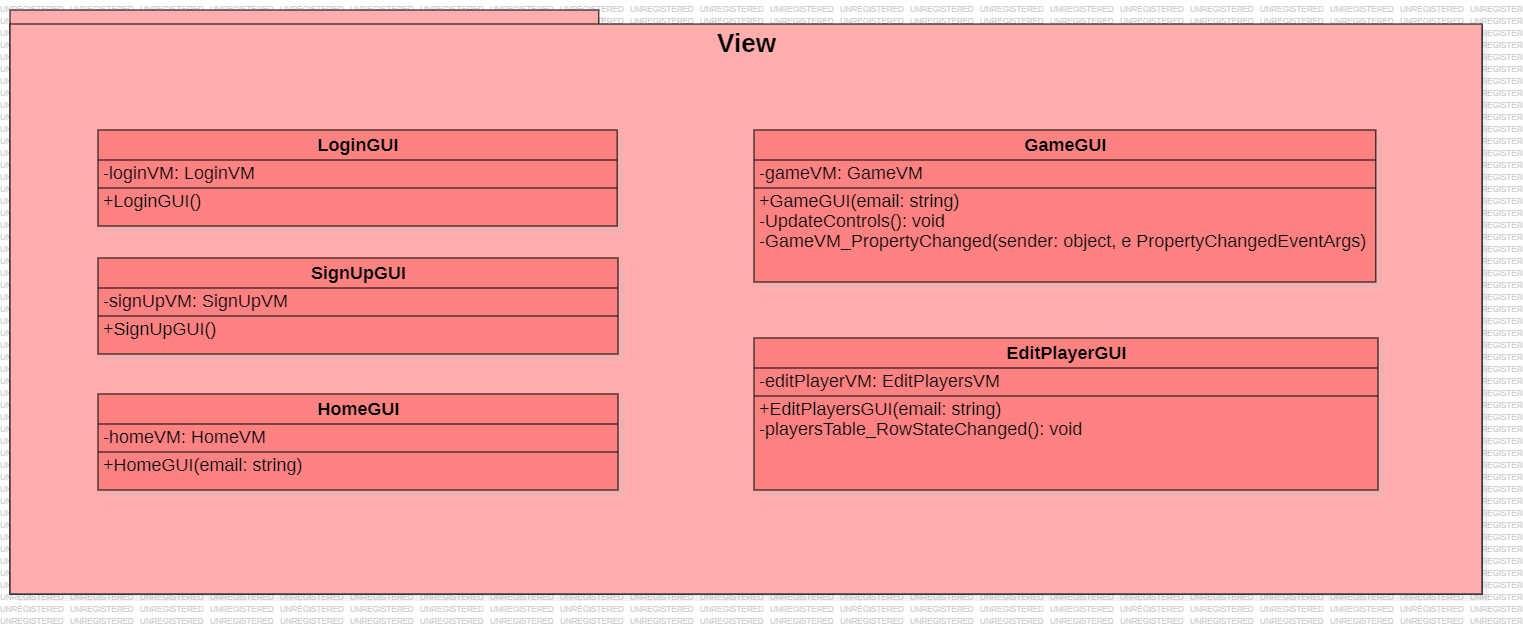
SignUpVM (Modelul de Vizualizare pentru ecranul de înregistrare):

* Atributele din SignUpVM, precum datele introduse pentru înregistrare, sunt legate cu View-ul asociat, SignUpGUI, pentru a valida și a afișa aceste date.
* Pentru gestionarea procesului de înregistrare și a acțiunilor asociate, SignUpVM creează și utilizează EditPlayerCommands sau alte comenzi specifice definite în pachetul EditPlayerCommands.

EditPlayersVM (Modelul de Vizualizare pentru gestionarea jucătorilor):

* Responsabil pentru gestionarea operațiilor CRUD (Create, Read, Update, Delete) pe datele jucătorilor și pentru efectuarea căutărilor după nume.
* Poate să conțină proprietăți precum lista de jucători, câmpuri pentru introducerea datelor noului jucător etc.
* Metodele din această clasă sunt concepute pentru a permite administratorului să adauge, să editeze, să șteargă sau să caute jucători în sistem.
* Interacționează strâns cu interfața utilizatorului asociată, EditPlayersGUI, pentru a afișa și a actualiza corespunzător datele și rezultatele operațiilor CRUD.

**Pachetul View:**



*Figura 8: Pachetul View*

GameGUI:

* Acest View este responsabil pentru afișarea și gestionarea interfeței grafice asociate jocului.
* Implementează interfața IGameGUI, care definește metodele necesare pentru afișarea și actualizarea elementelor grafice specifice jocului.
* Atributul GameVM reprezintă ViewModel-ul asociat cu acest View și este folosit pentru a lega și actualiza datele și comportamentul din spatele interfeței grafice.
* GameGUI oferă metode pentru adăugarea și modificarea componentelor GUI specifice jocului, cum ar fi tabla de joc sau elementele pentru afișarea scorului.

HomeGUI:

* Acest View prezintă ecranul principal al aplicației și este responsabil pentru afișarea regulilor jocului și a opțiunilor de navigare.
* Implementează interfața IHomeGUI, care definește metodele necesare pentru afișarea și actualizarea elementelor grafice specifice ecranului principal.
* Atributul HomeVM reprezintă ViewModel-ul asociat cu acest View și este folosit pentru a lega și actualiza datele și comportamentul din spatele interfeței grafice.
* HomeGUI oferă metode pentru adăugarea și modificarea componentelor GUI specifice ecranului principal, precum și pentru gestionarea navigării între diferitele ecrane ale aplicației.

EditPlayersGUI:

* Acest View este responsabil pentru afișarea și gestionarea interfeței grafice asociate editării jucătorilor din joc.
* Implementează interfața IEditPlayersGUI, care definește metodele necesare pentru afișarea și actualizarea elementelor grafice specifice editării jucătorilor.
* Atributul EditPlayersVM reprezintă ViewModel-ul asociat cu acest View și este folosit pentru a lega și actualiza datele și comportamentul din spatele interfeței grafice.
* EditPlayersGUI oferă metode pentru adăugarea și modificarea componentelor GUI specifice editării jucătorilor, cum ar fi lista jucătorilor sau opțiunile de editare.

LoginGUI:

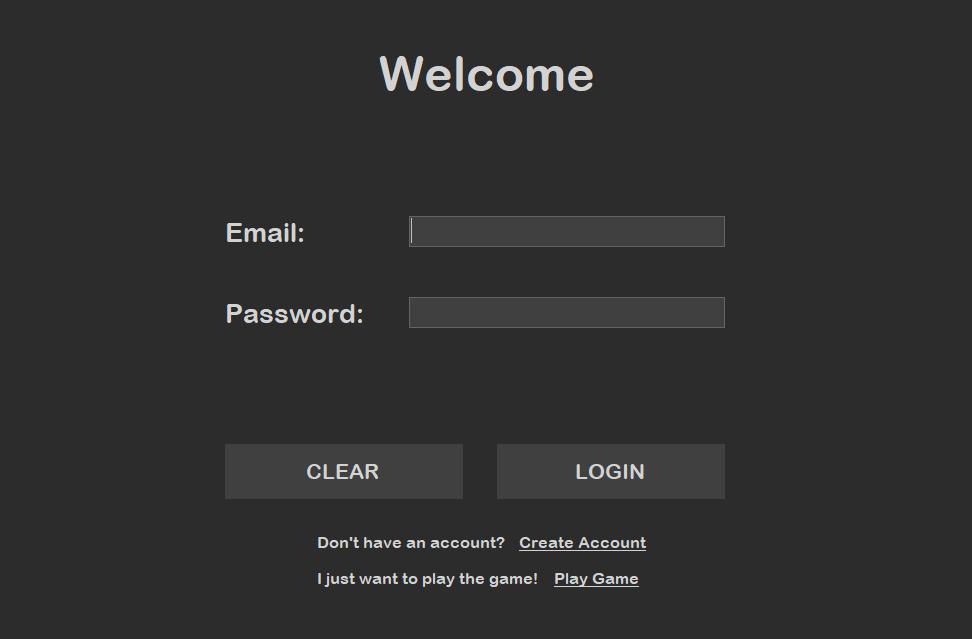
* Acest View este responsabil pentru afișarea și gestionarea interfeței grafice asociate procesului de autentificare a jucătorilor.
* Implementează interfața ILoginGUI, care definește metodele necesare pentru afișarea și actualizarea elementelor grafice specifice autentificării.
* Atributul LoginVM reprezintă ViewModel-ul asociat cu acest View și este folosit pentru a lega și actualiza datele și comportamentul din spatele interfeței grafice.
* LoginGUI oferă metode pentru adăugarea și modificarea componentelor GUI specifice procesului de autentificare, cum ar fi câmpurile pentru introducerea numelui de utilizator și a parolei.

SignUpGUI:

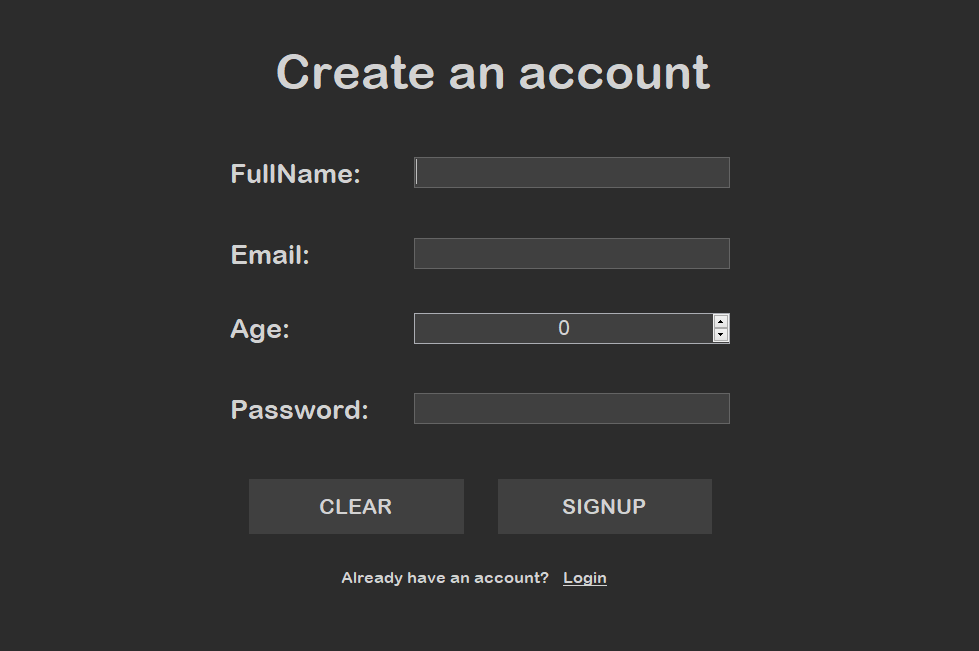
* Acest View este responsabil pentru afișarea și gestionarea interfeței grafice asociate procesului de înregistrare a unui nou jucător în aplicație.
* Implementează interfața ISignUpGUI, care definește metodele necesare pentru afișarea și actualizarea elementelor grafice specifice înregistrării.
* Atributul SignUpVM reprezintă ViewModel-ul asociat cu acest View și este folosit pentru a lega și actualiza datele și comportamentul din spatele interfeței grafice.

Aceste modificări evidențiază integrarea strânsă între View și ViewModel în cadrul arhitecturii MVVM, unde ViewModel-ul este responsabil pentru logica și datele din spatele interfeței grafice, iar View-ul este responsabil pentru afișarea și interacțiunea cu utilizatorul, legându-se dinamic de ViewModel-ul asociat.

1. **Descrierea aplicatiei:**

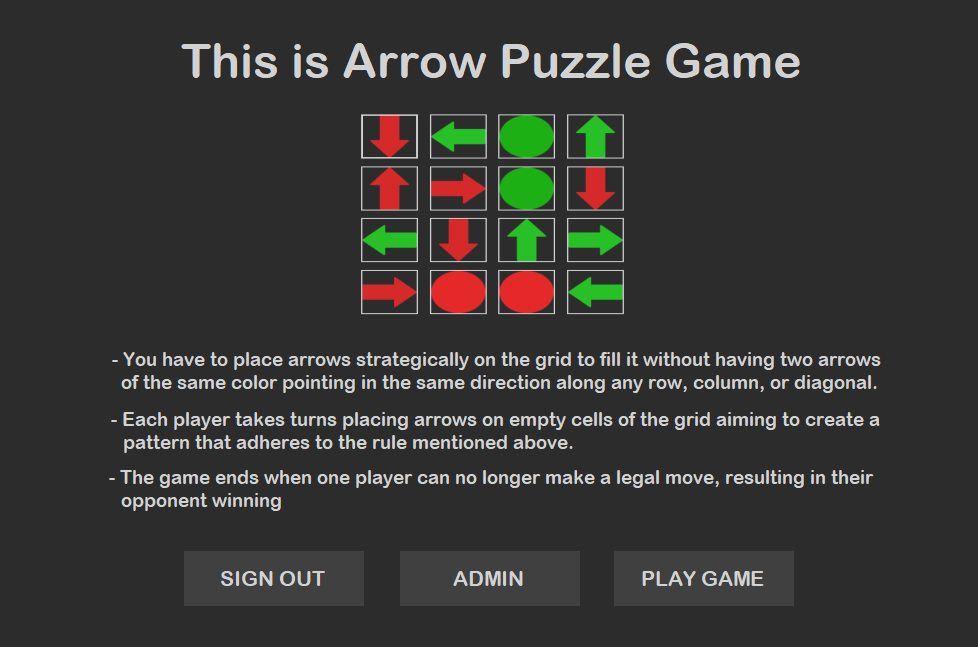
Cand pornim aplicatia clasa program va instantia clasa LoginGUI care ne va intampina cu o fereastra in care sa ne introducem credentialele pentru a ne putea autentifica. Daca nu avem cont putem apasa mai jos pe buttonu de create account care ne va duce pe pagina de SignUp pentru a ne creea un cont. Desigur user-ul poate trece direct la jocul propriu zis fara sa isi faca cont insa scorul acestuia nu va fi retinut astfel. Desigur mai exista si butonul de exit care inchide aplicatia, si buton de clear care sterge campurile completate.

*Figura 9: Pagina de Login*

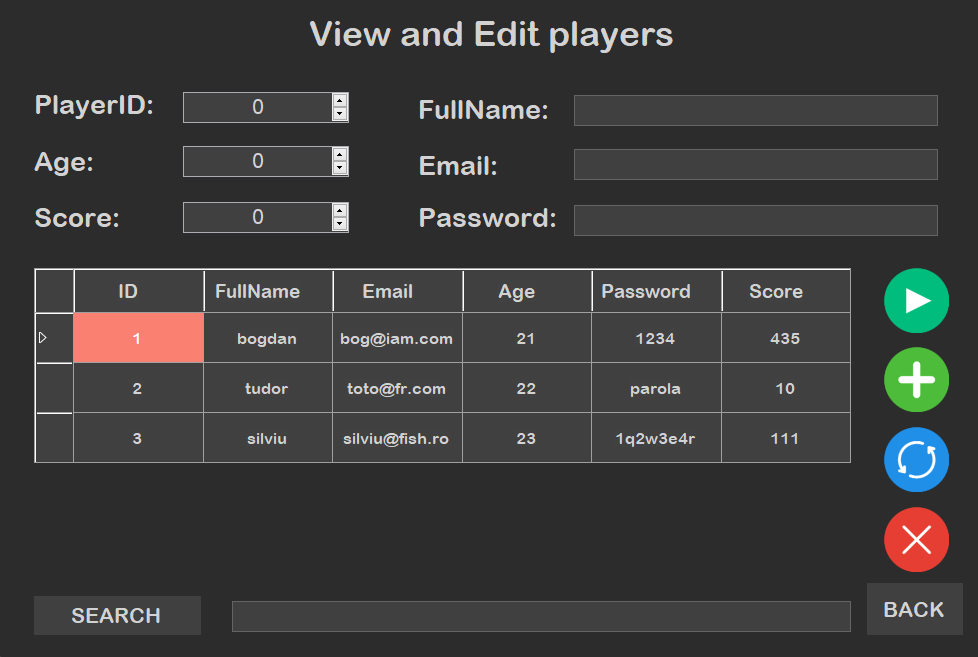
Daca algem optiuna de create account aceasta ne va duce spre pagina de SignUp in care o sa ni se ceara mai multe detalii care vor putea identifica unic jucatorul, de exemplu emai-ul. Si alte detalii precum varsta numele, scorul. Observam ca avem si optiunea de a naviga inapoi la pagina de Login daca am intrat din greseala.

*Figura 10: Pagina de SignUp*

Si aici avem buton de exit care inchide aplicatia Clear pentru a sterge toate campurile completate si buton-ul de signup care ne va inregistra in baza de date iar dupa ne va duce inapoi pe login ca sa ne autentificam cu noile credentiale.

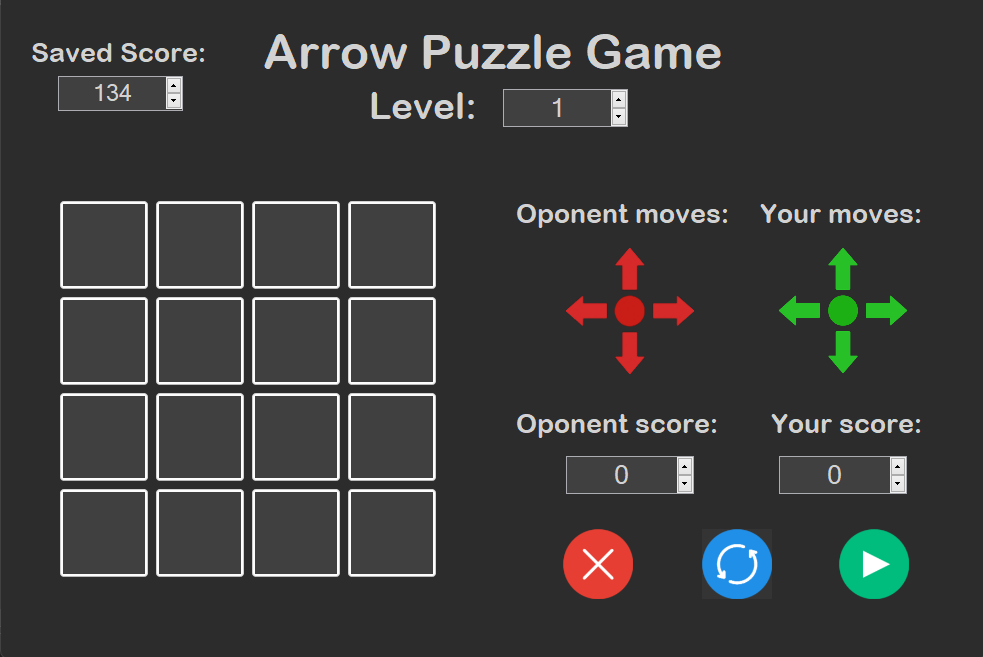
Daca acesti pasi au avut loc cu succes o sa ne apara un MessageBox care ne arata ca ne am autentificat si totul este in regula si vom ajunge astfel pe pagina de Home unde ne sunt explicate regulile jocului si de asemenea si o ilustratie. Desigur mai avem si aici butonul de SignOut care ne va deconecta de la contul curent si ne da optiunea sa ne logam din nou.

*Figura 11: Pagina de Home*

Exista doua tipuri de jucatori, jucatori simpli care isi pot face cont si se pot juca si cei de tip admin care sunt administratori, acestia au un cont special gata creat inainte ca aplicatia sa ruleze. Acestia trebuie sa detina email-ul si parola corespunzatoare si vor fi logati ca si admin. Ce pot dace ei in plus este sa acceseze pagina ADMIN unde vor putea sterge adauga sau actualiza datele tuturor jucatorilor care si au facut cont in aplicatie.

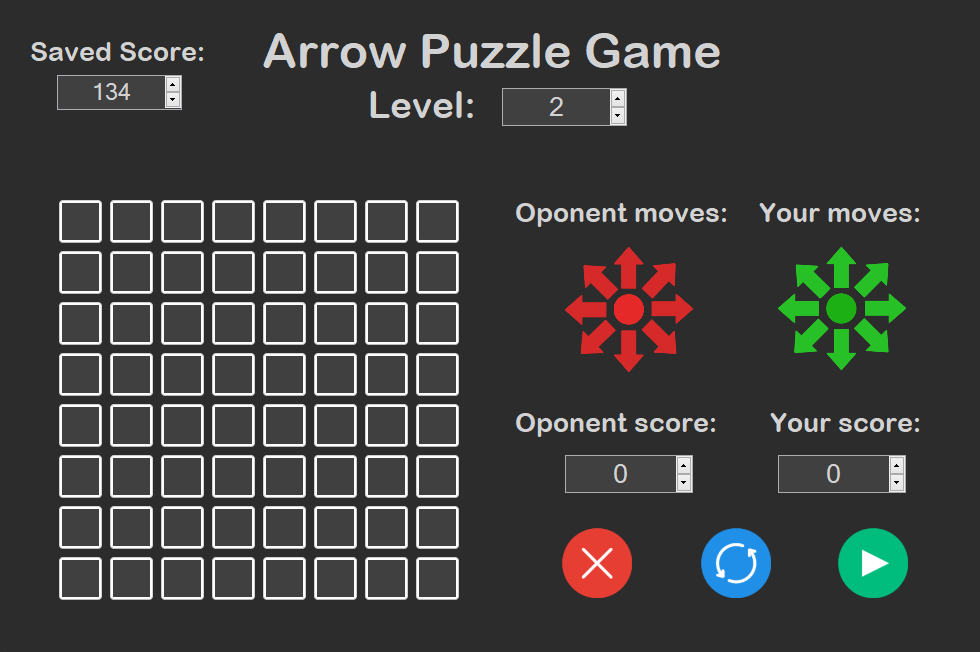
*Figura 12: Pagina EditPlayers*

Din aceasta pagina se pot aduga noi jucatori completand campurile de sus si selectand “+” sau a sterge si actualiza jucatori deja existenti acest lucru se face prin selectarea randului corespunzator jucatorului dorit, astfel se vor completa toate campurile de sus si vom putea modifica ce camp dorim. Admin-ul poate desigur si sa caute jucatori dupa numele sau id ul acestora folosind campul Search din josul pagini.

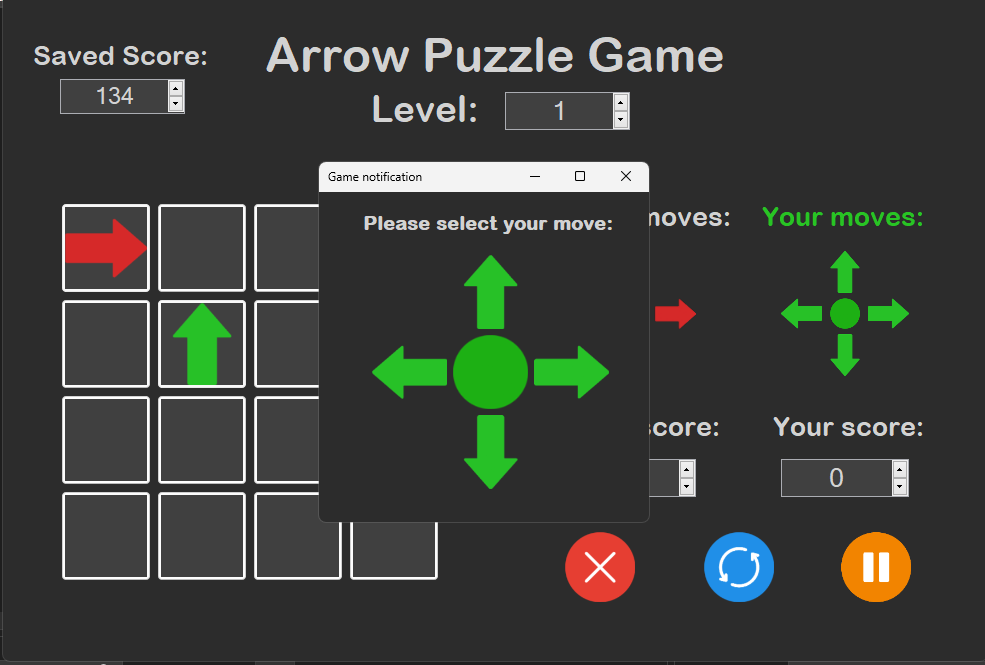
Atat admin-ul cat si jucatorul normal pot acesa pagina de Play Game. Cand aceasta pagina este accesata se creeaza un grid corespunzator nivelului: pentru nivelul 1 este 4x4 iar pt nicelul 2 este 8x8, initial gol. Daca player-ul are cont isi poate vedea scorul salvat in baza de date in stanga sus si scorul din sesiunea curenta in dreapta jos. Jocul se desfasoara impotriva calculatorului si acesta are un scor in sesiunea curenta de joc. La fiecare meci castigat, castigatorul primeste un punct, iar la egalitate primesc ambele parti cate un punct.

*Figura 13: Pagina Game cu nivel 1*

In partea din dreapta jucatorul poate vedea pt fiecare casuta ce miscari posibile are. Si acestea sunt in functie de nivel: nivelul 1 jucatorul are doar 5 miscari iar in nivelul 2 jucatorul are 9 miscari posibile. Mai gasim in partea de jos butoane precum play care incepe jocul, restart pt a reste sesiunea de joc curenta (se reseteaza si scorul), si parasirea sesiunei curente de joc, unde jucatorul va fi intrebat daca vrea sa salveze nou scor obtinut sau nu?

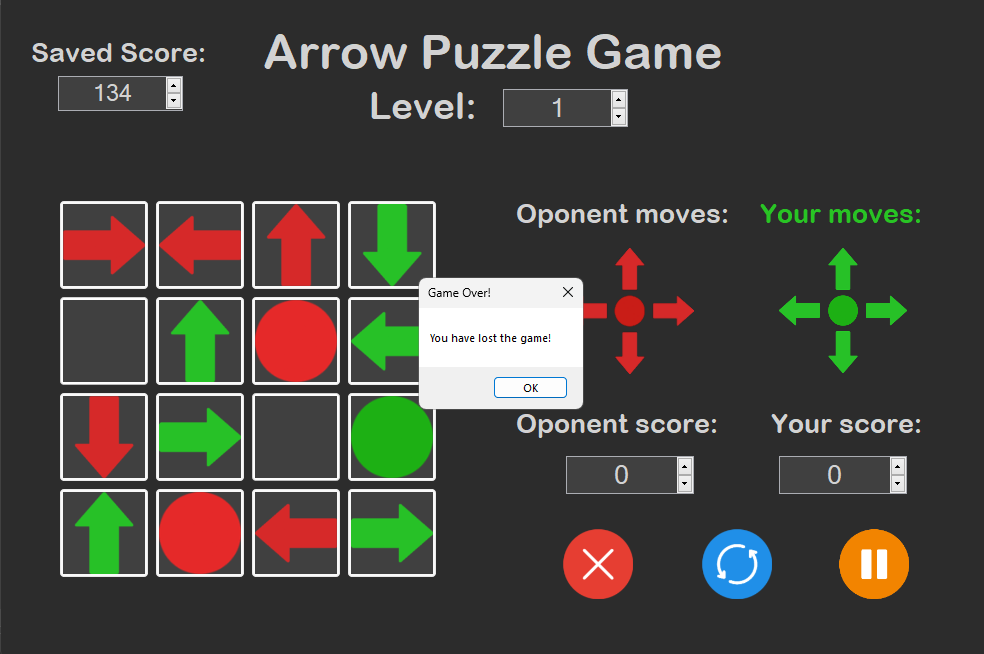


*Figura 14: Pagina Game cu nivel 2*

Dupa ce apasam Play, jocul incepe si se coloreaza cu o culaore specifica a cui e randul sa mute, sau sa faca o miscare. Initial incepe jucatorul iar apoi calculatorul, care implementeaza un algoritm de inteligenta artificiala numit minmax pt a determina cea mai buna miscare de la un moment respectiv, uitandu-se la mai multe configuratii. Odata ce ne vine randu apasam cu mouse-ul unul din pozitile din grid dupa care se va deschide o alta fereastra care ne va zice sa selectam sageata cu directia dorita. Odata o sageata cu o directie pusa aceasta nu se mai poate repeta nici pe aceasi linie nici pe aceasi coloana, iar in nicelul 2 nici pe aceasi diagonala.

*Figura 15: Pagina Game, exemplu de joc*

Jocul se incheie atunci cand nu mai exista miscari posibile care pot fi puse. Pierde cel care trebuie sa mute insa nu mai are casute in care poate pune sageti. Dupa ce se termina o runda incepe automat alta.



*Figura 16: Pagina Game, exemplu final de joc*

1. **Concluzie:**

În concluzie, arhitectura Model-View-ViewModel (MVVM) oferă o modalitate eficientă și scalabilă de a dezvolta aplicații software, facilitând separarea clară a preocupărilor între componentele logice și cele de afișare. Prin intermediul MVVM, fiecare ViewModel gestionează logica de afaceri și interacțiunile cu datele, în timp ce View-urile sunt responsabile pentru prezentarea și interacțiunea cu utilizatorul. Legătura dinamică între View și ViewModel prin mecanisme precum legarea datelor (data binding) permite actualizarea automată a interfeței utilizatorului în funcție de modificările din ViewModel, fără a fi necesară intervenția directă asupra elementelor de interfață. Această separare a preocupărilor și cuplarea slabă între componentele aplicației facilitează dezvoltarea, testarea și întreținerea codului, contribuind la creșterea eficienței și a calității produsului final. Prin implementarea MVVM, se obține o arhitectură bine organizată, ușor de înțeles și de extins, care susține creșterea și evoluția aplicației în timp.