Ludo Game

17.01.2026

Îndrumător:

dr. ing. Daniel Morariu

Student:

Puia Elena-Teodora

(C_23/2)

Istoric Versiuni

Data	Versiune	Descriere	Autor
02/12/2025	0.1	Versiunea conține doar corpul claselor, cu câteva metode care urmează să fie folosite mai târziu. Scopul acestei versiuni este de a structura programul pe clase și de a crea legături între ele (încuibărire și moștenire). Structura este gândită astfel încât să fie implementat și polimorfismul.	Puia Teodora
11/12/2025	0.2	Versiunea jocului are niște clase mult mai bine structurate, cu metode complexe, care aparțin algoritmului propriu-zis al jocului. Există o interfață grafică, dar aceasta nu arată funcționalitatea niciunei metode din clasele jocului.	Puia Teodora
16/12/2025	0.3	Interfața începe să ia formă. Totodată, se creează primele obiecte ale claselor jocului, iar metodele claselor sunt apelate prin intermediul acestora, observând funcționalitatea lor. Lipsește implementarea grafică a pionilor, dar există o dezvoltare a logicii de mutare a acestora.	Puia Teodora
03/01/2026	0.4	Implementarea completă a tablei de joc (obiecte căsuță cu coordonate aferente) și afișarea grafică a pionilor. Aceștia se mișcă pe tablă după un algoritm. Totuși, varianta are erori de deplasare a pionilor (sens trigonometric, invers).	Puia Teodora
06/01/2026	0.5	Algoritmul de mutare al pionilor pe căsuțele tablei a fost optimizat, sensul deplasării fiind acum corect. De asemena, au fost tratate și cazurile în care nu există mutări valide, s-au implementat toate regulile jocului care au fost propuse.	Puia Teodora Șerban Maria

11/01/2026	1.0	Jocul local este finalizat. S-a revizuit codul și s-a finalizat testarea tuturor scenariilor de joc.	Puia Teodora
16/01/2026	1.1	Adăugarea corpului clasei JocRețea pentru o posibilă implementare a unor socketuri.	Puia Teodora

Cuprins

1 Specificarea cerințelor software

1.1 Introducere

Proiectul constă în realizarea unei aplicații pentru jocul „Nu te supăra frate”, transpus în varianta englezească „Ludo”. Aplicația este dezvoltată în limbajul C++ folosind mediul de programare Embarcadero C++ Builder cu biblioteca VCL pentru interfața grafică. Proiectul pune accent pe utilizarea corectă a principiilor Programării Orientate pe Obiecte.

1.1.1 Obiective

1. Crearea unei interfețe simple pentru implementarea grafică a jocului.
2. Utilizarea corectă a conceptelor de Programare Orientată pe Obiecte.
3. Crearea unor clase bine structurate, care să bifeze Încapsularea, Moștenirea și Polimorfismul, dar să fie și relevante pentru joc.
4. Separarea logicii jocului de interfață, favorizând utilizarea metodelor din interiorul claselor în detrimentul funcțiilor globale independente
5. Implementarea corectă a regulilor jocului Ludo: ieșirea din casă a pionilor, deplasarea pe traseu, eliminarea adversarilor, intrarea în spațiul de câștig, aflarea câștrigătorului.
6. Elaborarea unui algoritm eficient pentru calcularea destinației pionilor și validarea mutărilor pe o tablă circulară.
7. Crearea aplicației pentru client care face posibilă alegerea tipului de joc (local sau în rețea) cu ajutorul principiului de moștenire.
8. Implementarea unei extensii a aplicației pentru client care să permită funcționalitatea jocului în rețea (obiectiv partial atins).
9. Implementarea serverului functional pentru jocul în rețea (obiectiv incomplet).

1.1.2 Definiţii, Acronime şi Abrevieri

După cum sunt obișnuită în urma laboratoarelor, orice variabilă pentru formă am notat-o începând cu litera „f”, orice variabilă pentru o componentă de tip Button începe cu „btn”, orice variabilă pentru o componenta de tip Label începe cu „lbl”, orice variabilă pentru o componenta de tip Panel începe cu „pnl”, orice variabilă pentru o componenta de tip Edit începe cu „edit”. În construcția claselor, numele variabilelor de tip membru încep cu litera „m”.

1.1.3 Tehnologiile utilizate

- Limbaj de programare: C++;
- Mediu de dezvoltare: Embarcadero C++ Builder;
- Framework GUI: VCL (Visual Component Library);
- Biblioteci adiționale: System.Win.ScktComp.hpp (pentru socketi), Vcl.Imaging.pngimage.hpp (pentru imagini PNG).

1.2 Cerințe specifice

În această secțiune sunt detaliate funcționalitățile efectiv implementate în versiunea curentă a aplicației:

1. Configurarea jocului: Această funcționalitate permite utilizatorului să aleagă tipul jocului (Joc Local sau Joc Rețea) și să seteze numărul de jucători și numele acestora.
2. Reprezentarea grafică a tablei și a pionilor: Instanțierea tablei cu căsuțe și afișarea pionilor pe căsuțele din casă corespunzătoare culorilor.
3. Aruncarea zarului: Generarea aleatoare a valorilor zarului (1-6) și afișarea rezultatului în interfață.
4. Validarea și calculul mutărilor: Sistemul verifică automat dacă o mutare este permisă conform regulilor (ex: ieșirea din casă doar cu zarul 6, interzicerea de creare a turnurilor, imposibilitatea de a depăși căsuța de start) și calculează poziția finală a pionului.
5. Executarea mutării și gestionarea coliziunilor: Se asigură deplasarea grafică și logică a pionului. Include mecanica de „bătaie” (eliminare), prin care un pion adversar aflat pe căsuța destinație este trimis înapoi în baza sa.
6. Gestionarea fluxului de joc (turele): Schimbarea automată a jucătorului activ (în sensul acelor de ceasornic) și acordarea unei aruncări suplimentare în cazul obținerii zarului 6 sau a eliminării unui pion advers.
7. Verificarea condițiilor de câștig: Monitorizarea constantă a stării pionilor pentru a determina momentul în care un jucător a reușit să ducă toți cei 4 pioni în zona finală, declanșând câștigul și oprirea jocului.

2 Validarea și calculul mutărilor

2.1 Descriere

Această funcționalitate prezintă mecanismul central al jocului, responsabil de transformarea valorii aleatoare a zarului într-o mișcare validă pe tabla de joc. Aceasta este esențială pentru integritatea aplicației, deoarece asigură respectarea regulamentului Ludo.

Se asigură manipularea corectă a pionilor pe traseu, mișcarea circulară a acestora, ieșirea din casă doar la îndeplinirea condiției de zar egal cu 6, interzicearea suprapunerii pionilor, gestionarea mutărilor în funcție de stările pionilor (în casă, pe traseu, în zona finală).

Algoritmul implementat se folosește de repetate ori de funcția "calculeazaDestinatia" care gestionează centralizat toate scenariile

2.2 Fluxul de evenimente

2.2.1 Fluxul de bază: Determinarea destinației

Procesul de validare este apelat în trei momente distincte ale unei ture pentru a ghida utilizatorul:

1. Utilizatorul, având zarul aruncat, află dacă are mutări posibile. Sistemul verifică apelând funcția de calcul al destinației pentru toți cei 4 pioni ai jucătorului. Dacă funcția returnează null pentru toți, tura se încheie automat, economisind timp. Dacă există cel puțin o mutare validă, jocul așteaptă inputul utilizatorului (alegerea pionului).
2. Utilizatorul dă click pe un pion, iar sistemul validează alegerea. Se apelează funcția strict pentru pionul selectat. Dacă rezultatul este null (ex: încearcă să iasă din casă cu zar 3), selecția trebuie schimbată. Dacă rezultatul este o adresă validă (Casuta*), se permite trecerea la execuția mutării.
3. Funcția mai este apelată odata pentru a trece ulterior la funcționalitatea care se ocupă de mutarea pionilor.

Logica algoritmului de calcul al destinației se face prin funcția Tabla::calculeazaDestinatia, care gestionează patru scenarii pentru a returna o căsuță validă sau nullptr:

1. Ieșirea din casă: Permite mutarea pe căsuța de Start doar dacă starea este PION_INCASA și zarul este 6.
2. Deplasarea pe Traseu: Calculează noua poziție scăzând valoarea zarului din ID-ul curent (id-urile căsuțelor sunt luate în sens invers față de traseul pionilor) și aplică o corecție matematică (adaugă 52 dacă rezultatul este negativ) pentru a asigura circularitatea tablei.
3. Zona Finală: Detectează trecerea prin „poarta" culorii și redirecționează calculul în vectorul zonei finale, verificând ca pionul să nu depășească triunghiul de câștig.
4. Coliziuni: Returnează nullptr dacă destinația este ocupată de un pion propriu (evită suprapunerea), dar permite mutarea peste adversari (pentru eliminare), mai puțin când aceștia se află pe căsuța proprie de start.

2.2.2 Pre-condiţii

Pentru ca algoritmul de determinare a destinației să poată fi apelat și executat corect, sistemul trebuie să se afle într-o stare specifică, iar utilizatorul trebuie să fi efectuat anumite acțiuni premergătoare:

1. Generarea aleatoare (Zarul): Utilizatorul trebuie să fi apăsat butonul de Zar, iar variabila mValoareZarCurent trebuie să aibă o valoare validă între 1 și 6. Funcția nu poate fi apelată fără o valoare a zarului.
2. Selecția Jucătorului Activ: Trebuie să fie rândul jucătorului care deține pionul selectat.
3. Selecția Vizuală: Utilizatorul trebuie să dea click exact pe componenta TImage (imagine) a unui pion.

Limitări și condiții netratate:

- Aplicația nu blochează vizual pionii care nu pot fi mutați. Toți pionii rămân activi vizual, iar validarea se face doar după click. Totuși, utilizatorul primește mesaje care îi spune atunci când nu apasă pe pionul potrivit.

2.2.3 Post-condiţii

În urma rulării funcționalității de determinare a destinației, sistemul și utilizatorul ajung în una dintre următoarele stări:

1. Identificarea unei Destinații Valide:

- Sistem: Funcția returnează un pointer valid către un obiect de tip Căsuță. Acest pointer este folosit imediat de funcția de mutare pentru a actualiza legăturile logice (mPionPeCasuta).
- Utilizator: Utilizatorul observă că pionul selectat își schimbă poziția instantaneu pe tablă, ocupând noua căsuță calculată.

2. Respingerea Mutării (Destinație Invalidă):

- Sistem: Funcția returnează nullptr. Starea internă a tablei nu se modifică.
- Utilizator: Pionul rămâne nemișcat pe poziția inițială. În label-ul de informații (lblInfo) se afișează un mesaj de tipul „Mutare imposibilă” sau „Alege alt pion”, semnalând că algoritmul a rulat dar nu a găsit o destinație validă conform regulilor.

3 Executarea mutării și gestionarea coliziunilor

3.1 Descriere

Această funcționalitate reprezintă acțiunea concretă de schimbare a stării jocului. Dacă funcționalitatea anterioară (Capitolul 2) doar interoga posibilitățile de mutare a pionilor, această funcționalitate modifică datele.

Este relevantă deoarece sincronizează logica jocului (poziția în memorie) cu interfața grafică (poziția vizuală pe ecran) și implementează cea mai importantă mecanică competitivă a jocului Ludo: eliminarea de pe traseu al unui pion al adversarului. De asemenea, gestionează tranzițiile de stare ale pionului (intrarea pe tablă, avansarea în zona finală sau finalizarea cursei).

Logica este încapsulată în metoda Tabla::mutaPion, apelată prin polimorfism din JocLocal::executaMutare.

3.2 Fluxul de evenimente

3.2.1 Fluxul de bază: Actualizarea poziției și tratarea coliziunilor

Acest flux descrie succesiunea de evenimente care are loc imediat după ce o mutare a fost validată și calculată.

Pașii de implementare și execuție:

1. Inițiere: Utilizatorul a selectat un pion valid. Interfața grafică a apelat metoda executaMutare din clasa Joc, care a validat dacă mutarea este posibilă. Apoi, a fost apelată funcția de mutare a pionului.
2. Identificarea Destinației: Se apelează intern funcția calculeazaDestinatia (cea de a treia oară) pentru a obține pointerul exact către căsuța unde va ajunge pionul.
3. Gestionarea Coliziunii (Eliminarea): Înainte de a muta pionul curent, sistemul verifică dacă pe căsuța destinație există deja un pion (destinatie->esteOcupata()). Deoarece validarea a permis mutarea, știm sigur că pionul existent este un adversar. Sistemul accesează pionul adversar, îi resetează starea la PION_INCASA și îi mută coordonatele grafice înapoi în zona de bază a culorii sale. Căsuța destinație este eliberată logic (eliminaPion()).

4. Actualizarea Logică (Memorie): Pionul curent este deconectat de la căsuța veche (vecheaCasuta->eliminaPion()). Pionul este conectat la noua căsuță (nouaCasuta->setPion(pion)).
5. Actualizarea Stării Pionului: În funcție de tipul căsuței destinație, starea pionului se actualizează: dacă ajunge pe Start - PION_PETABLA, dacă intră pe culoarul colorat - PION_INZONAFINALA, dacă ajunge pe căsuța de câștig - PION_FINALIZAT.
6. Sincronizarea Grafică: Metoda Pion::muta citește noile coordonate X și Y ale căsuței destinație și actualizează proprietățile Left și Top ale componentei TImage asociate, realizând animația vizuală a săriturii.

3.2.2 Pre-condiții

Pentru ca executarea mutării să aibă loc fără erori de memorie sau logice, trebuie îndeplinite următoarele:

1. Validarea anterioară: Pionul selectat trebuie să fi trecut deja de procesul de validare, garantând că destinația nu este nullptr.
2. Integritatea Grafică: Obiectul TImage asociat pionului trebuie să fie valid (creat), deoarece metoda de mutare va încerca să îi modifice coordonatele.
3. Coordonate Inițializate: Toate căsuțele de pe tablă trebuie să aibă coordonatele X și Y setate corect (realizat la inițializarea Tabla).

Limitări și condiții netratate:

- Nu există o animație fluidă a deplasării. Mutarea este instantanee (teleportare), ceea ce este funcțional, dar estetic simplist.

3.2.3 Post-condiții

După execuția acestei funcționalități, starea aplicației se modifică astfel:

1. Modificarea Poziției:

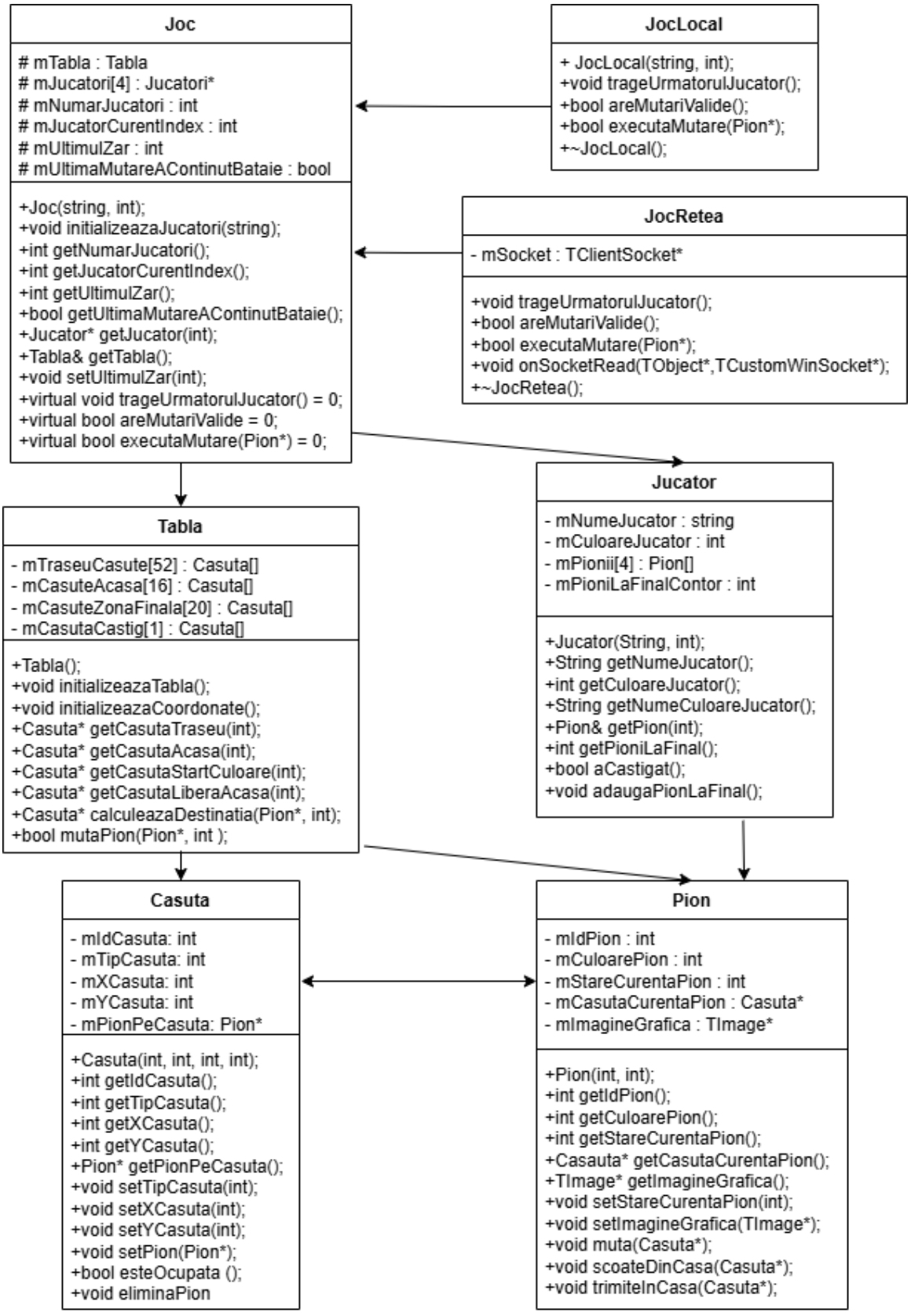
- Utilizator: Vede pionul pe noua căsuță.
- Sistem: Pionul are referința actualizată către noua căsuță. Vechea căsuță este liberă.

2. Eventuala Eliminare: Dacă a existat o coliziune, utilizatorul vede pionul adversarului înapoi în zona de start a bazei sale. Numărul de pioni activi ai adversarului scade.

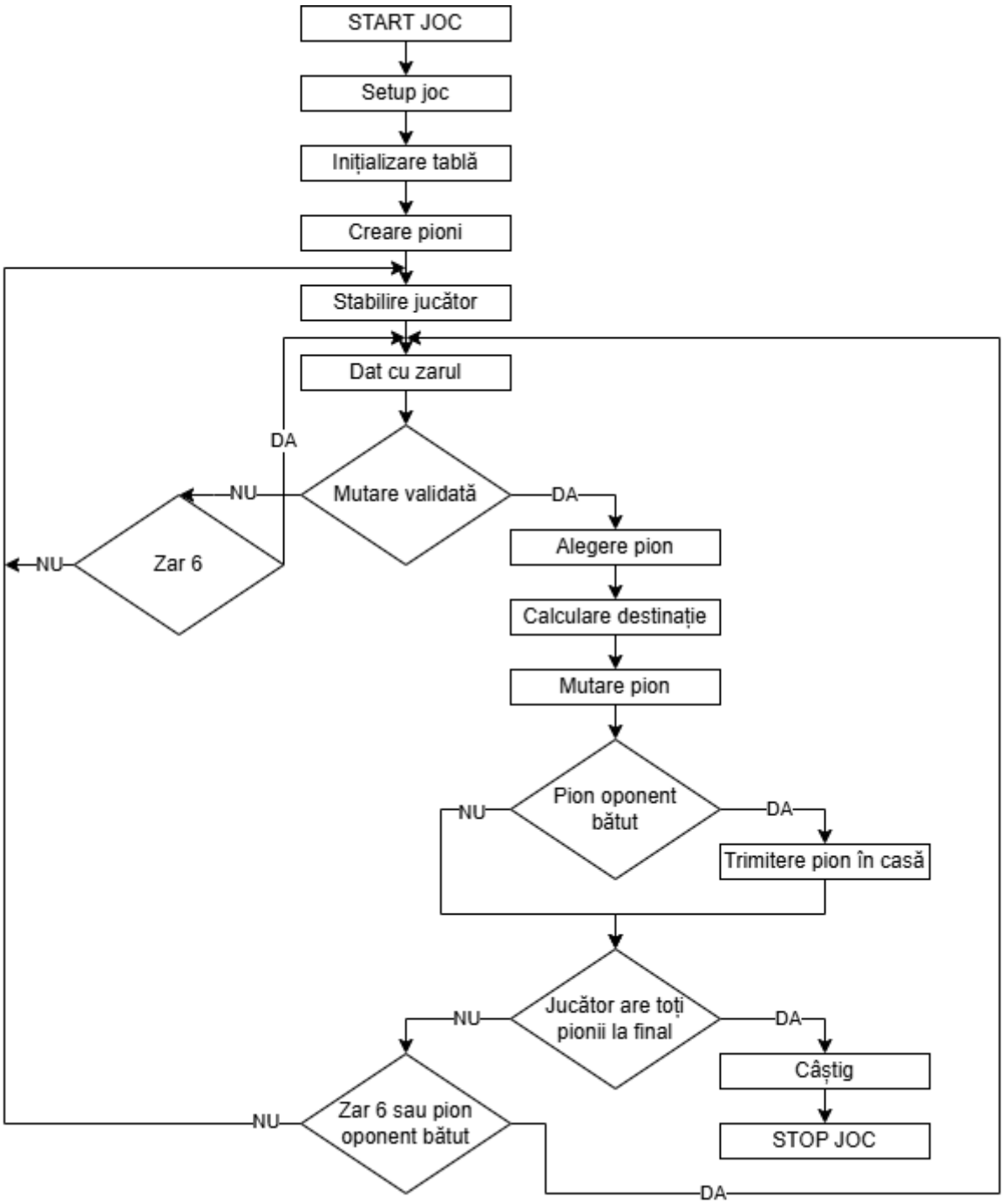
3. Actualizarea Scorului: Dacă pionul a ajuns pe căsuța de Câștig, el devine PION_FINALIZAT și dispare de pe traseul comun. Contorul de pioni finalizați al jucătorului crește.
4. Feedback pentru Tura Următoare: Funcția returnează true dacă s-a mâncat o piesă sau false altfel. Acest rezultat este folosit de interfață pentru a decide dacă jucătorul mai primește o aruncare cu zarul.

4 Implementare

4.1 Diagrama de clase



4.2 Descriere detaliată



5 Bibliografie

- Programare Orientată pe Obiecte. Principii – Macarie Breazu
- Materiale laborator - Programare Orientată pe Obiecte, Dr. ing. Daniel Morariu
- https://docwiki.embarcadero.com/RADStudio/Florence/en/Tutorials - Dezvoltare aplicație cu interfață grafică
- https://gemini.google.com/ - Google Gemini (AI Assistant) - Asistență pentru debugging