Iordache Elena Alexandra
322AC

SQUID GAME

În cadrul acestui proiect, am recreat logica jocului din serialul sud corean “Squid Game”. Acesta este scris în limbajul c++ și folosește concepte de Programare Orientată pe Obiecte.

Aplicația stochează, pentru început, datele oamenilor care vor participa la jocuri. Datele necesare pe care le-am folosit sunt:

- nume
- prenume
- oraș
- datoria de bani pe care o are persoana respectivă - alocată în mod aleator între 10.000 și 100.000
- Greutate - alocată în mod aleator între 50 și 100

În aplicație avem un total de 108 persoane care sunt împărțite în 2 categorii: participanți și supraveghetori. Participanții sunt în număr de 99 și au fiecare câte un număr de concurs, iar celor 9 supraveghetori le-a fost atribuită o formă de mască (cerc, triunghi sau pătrat).

Am realizat o clasele *“Person”, “Players”* și *“Supervisor”* pentru a lucra mai ușor cu datele participanților/ supraveghetorilor.

În *Figura 1* sunt ilustrate diagramele UML ale celor 3 clase menționate mai sus. Se pot observa atributele și metodele fiecărei clase, dar și relația de moștenire dintre clasa “Person” și clasele “Player” și “Supervisor”.

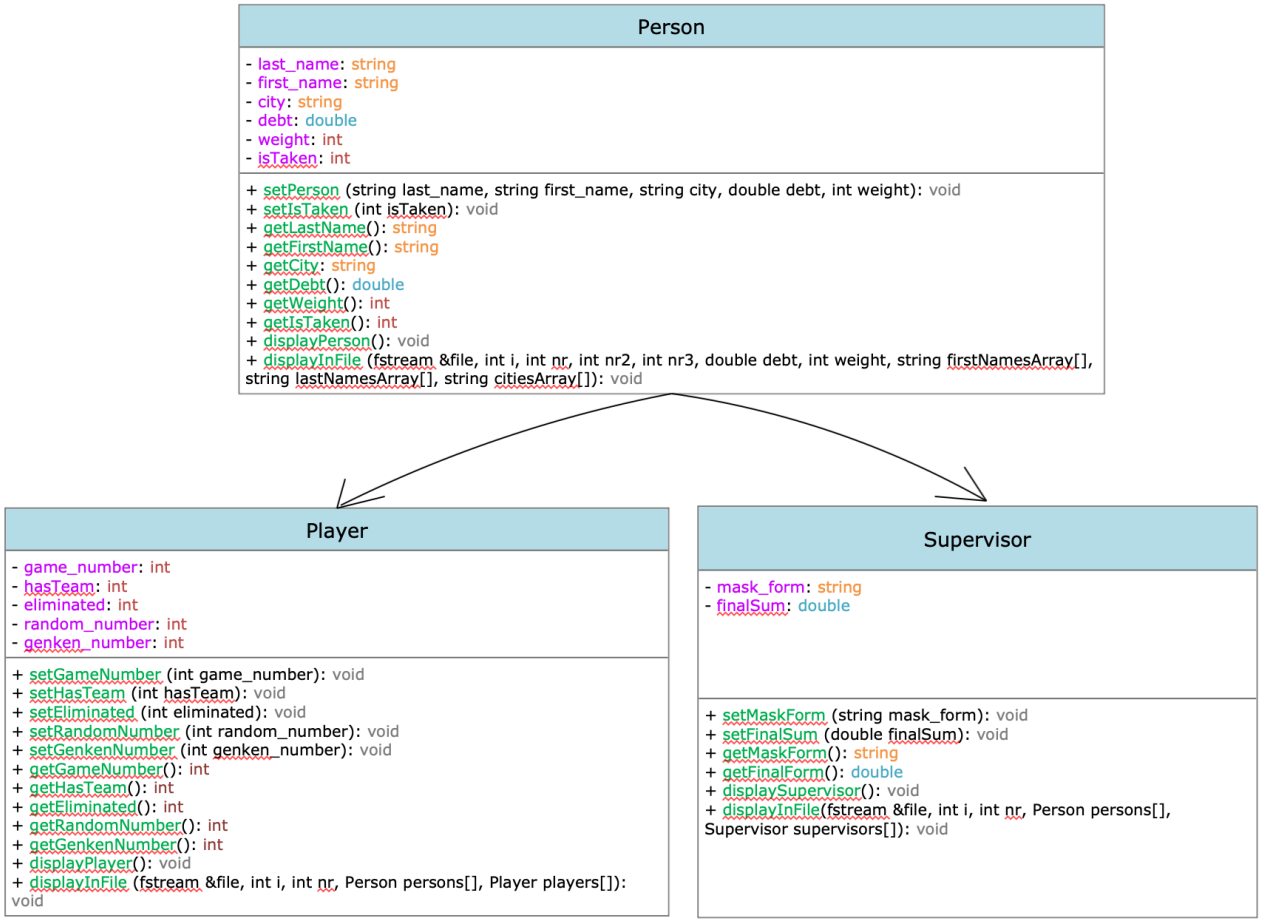

Figura 1

Clasa Person

1. **Atribute**: - last_name: numele persoanei;
   - first_name: prenumele persoanei;
   - city: orașul din care provine;
   - debt: suma de bani pe care o datorează;
   - weight: greutatea persoanei;
   - isTaken: acest atribut este folosit pentru a verifica dacă persoanei respective i-a fost dat deja un rol (participant/ supraveghetor) sau nu (valoarea 1 - i-a fost atribuit; valoarea 0 - nu i-a fost atribuit).

2. **Metode** : - setPerson: metodă care memorează datele despre o persoana;
   - setIsTaken: metodă care memorează 1 dacă i-a fost atribuit un rol persoanei sau valoarea 0 dacă nu i-a fost atribuit.
   - getLastName(), getFirstName(), getCity(), getDebt(), getWeight(), getIsTaken(): getters;
   - displayPerson(): afișează informațiile despre persoana respectivă în terminal;
   - displayInfile(): afișează informațiile despre persoană în fișier;

Clasa Player - moștenește clasa “Person”

Atributele și metodele folosite în afara celor moștenite de la clasa “Person” sunt:

1. **Atribute**: - game_number: numărul de concurs al jucătorului ;
- hasTeam: acest atribut este folosit pentru a verifica dacă jucătorul este repartizat într-o echipă sau nu;
- eliminated: acest atribut este folosit pentru a verifica dacă jucătorul este eliminat sau nu;
- random_number: numărul alocat random - folosit la unul dintre jocuri;
- genken_number: numărul folosit pentru ultimul joc.

2. **Metode** : - setGameNumber: memorează numărul de concurs al jucătorului;
- setHasTeam: memorează 1 dacă jucătorul este distribuit într-o echipă sau 0 dacă jucătorul nu are echipă;
- setEliminated: memorează 1 dacă jucătorul este eliminat sau 0 dacă nu a fost eliminat;
- setRandomNumber: memorează numărul alocat random necesar unuia dintre jocuri;
- setGenkenNumber: memorează numărul alocat random necesar ultimului joc;
- getGameNumber(), getHasTeam(), getEliminated(), getRandomNumber(), getGenkenNumber(): getters;
- displayPlayer(): afișează informațiile despre jucătorul respectiv în terminal;
- displayInfile(): afișează informațiile despre jucator în fișier;

Clasa Supervisor - moștenește clasa “Person”

Atributele și metodele folosite în afara celor moștenite de la clasa “Person” sunt:

1. **Atribute**: - mask_form: forma măștii atribuite;
- finalSum: suma totală pe care o va câștiga la finalul jocului.

2. **Metode** : - setMaskForm: memorează forma măștii supraveghetorului;
- setFinalSum: memorează suma totală primită de supraveghetor la finalul tuturor jocurilor;
- getMaskForm(), getFinalSum(): getters;
- displaySupervisor(): afișează informațiile despre supraveghetorul respectiv în terminal;
- displayInfile(): afișează informațiile despre supraveghetor în fișier;

Clasa Display - clasă abstractă

1. **Metode** : - fistGame( ), secondGame( ), thirdGame( ), fourthGame( ): pure virtual functions

Clasa DisplayMessage - moștenește clasa “Person”

1. **Metode** : - fistGame( ), secondGame( ), thirdGame( ), fourthGame( ): void -> fiecare afișează pe ecran un anumit mesaj.

În *Figura 2* sunt ilustrate diagramele UML ale celor 2 clase menționate mai sus:

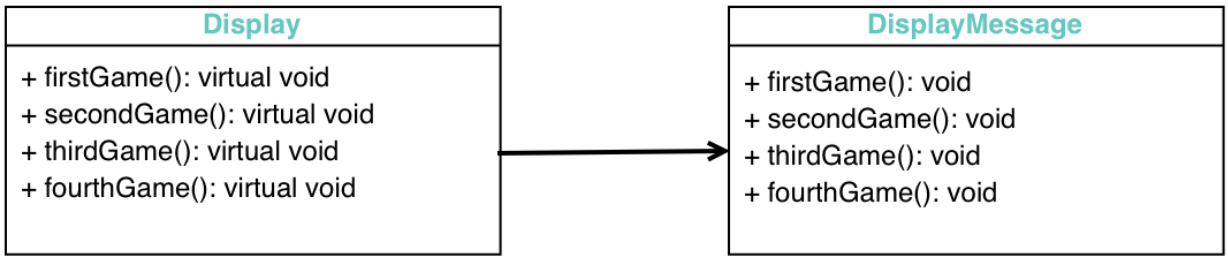

Figura 2

De asemenea, am folosit ***14 fișiere text***:

- **NAMES.txt** -> în acest fișier sunt scrise 108 prenume (fiecare pe câte o linie);
- **LASTNAMES.txt** -> în acest fișier sunt scrise 108 nume (fiecare pe câte o linie);
- **CITIES.txt** -> în acest fișier sunt scrise 108 nume de orașe (fiecare pe câte o linie);
- **PERSONS.txt** -> în acest fișier sunt reținute datele despre fiecare persoană: nume, prenume, oraș, datorie, greutate;
- **PLAYERS.txt** -> în acest fișier sunt reținute datele despre fiecare jucător: nume, prenume, oraș, datorie, greutate, număr de concurs;
- **SUPERVISORS.txt** -> în acest fișier sunt reținute datele despre fiecare supraveghetor: nume, prenume, oraș, datorie, greutate, forma măștii;
- **TEAMS.txt** -> în acest fișier sunt reținute echipele inițiale. Este scrisă câte o persoană pe rând: prenume, nume, oraș, datorie, greutate și numărul de concurs (dacă este participant) sau forma măștii (dacă este supraveghetor). Sunt scrise datele despre un supraveghetor pe un rând, iar pe următoarele 11 linii sunt scrise datele despre jucătorii din echipa acestuia;

- **FIRSTGame.txt** -> în acest fișier sunt reținute datele despre jucătorii care au rămas in concurs după primul joc (prenume, nume, număr concurs) - fiecare pe câte o linie;
- **beforeSECONDGame.txt** -> în acest fișier sunt reținute echipele pentru jocul 2. Este afișat numărul echipei, urmat de datele (prenume, nume, număr concurs) jucătorilor din echipa respectivă;
- **SECONDGame.txt** -> în acest fișier sunt reținute datele despre jucătorii care au rămas în concurs după al doilea joc (prenume, nume, număr concurs) - fiecare pe câte o linie;
- **beforeTHIRDGame.txt** -> în acest fișier sunt reținute echipele pentru jocul 3. Este afișat numărul echipei, urmat de datele (prenume, nume, număr concurs și numărul alocat random) jucătorilor din echipa respectivă;
- **THIRDGame.txt** -> în acest fișier sunt reținute datele despre jucătorii care au rămas în concurs după al treilea joc (prenume, nume, număr concurs) - fiecare pe câte o linie;
- **FOURTHGame.txt** -> în acest fișier sunt reținute datele jucătorilor rămași în concurs: prenume, nume, număr concurs și un număr ales aleatoriu (între 1 și 3 inclusiv), care este folosit pentru ultimul joc;
- **FINAL.txt** -> în acest fișier sunt reținute rezultatele finale ale concursului: numele și prenumele jucătorului câștigător împreună cu suma totală obținută; numele și prenumele supraveghetorilor împreună cu sumele câștigate (afișați în ordine descrescătoare în funcție de acestea); echipa de supraveghetori care a câștigat cea mai mare sumă de bani.

✳ **MAIN**:

- Am apelat funcția *srand(time(NULL))* pentru a genera numere random;
- Am creat 6 vectori:
    - Vector de tipul *Person* care stochează datele necesare a 108 persoane
    - Vector de tipul *Player* care stochează datele necesare a 99 de jucatori
    - Vector de tipul *Supervisor* care stochează datele necesare a 9 supraveghetori
    - Vector de tipul *string* care stochează 108 prenume
    - Vector de tipul *string* care stochează 108 nume
    - Vector de tipul *string* care stochează 108 nume de orașe
- Am citit prenumele, numele și orașele din fișierele în care sunt reținute;

- Am adăugat in fișierul *“PERSONS”*, precum și în vectorul *“persons”* datele persoanelor:
    - pentru fiecare persoană în parte am generat 3 numere random folosind funcția *“rand()”* pentru a se alege aleatoriu un nume, prenume, respectiv oraș din fișierele anterioare;
    - în aceeași maniera am procedat și pentru alegerea greutății și a datoriei;
    - am setat atributul de *“isTaken”* cu valoarea 0 pentru că nicio persoană nu are vreun rol la începutul jocului;
    - am afișat informațiile în fișier.

- Am adăugat în fișierul *“PLAYERS”,* precum și în vectorul *“players”* datele jucătorilor:
    - am generat un număr random folosind funcția *“rand()”* pentru a alege aleatoriu o persoană din vectorul *“persons”;*
    - am verificat dacă acesta are deja un rol sau nu;
    - dacă nu avea un rol, atunci l-am adăugat în vectorul *“players”* și am setat atributul *“isTaken”* la valoarea 1;
    - i-am atribuit un număr de concurs;
    - am setat atributele *“hasTeam”* și *“setEliminated”* cu valoarea 0 (jucătorul nu este repartizat în vreo echipa și nici nu este eliminat din joc);
    - am afișat informațiile în fișier.

- Am adăugat în fișierul *“SUPERVISORS”*, precum și în vectorul *“supervisors”* datele supraveghetorilor:
    - Am luat fiecare persoană în parte și am verificat dacă are deja rolul de jucător sau nu;
    - dacă nu avea un rol, atunci l-am adăugat în vectorul “supervisors” și am setat atributul *“isTaken”* cu valoarea 1;
    - I-am atribuit un tip de mască (cerc/ triunghi/ pătrat);
    - am afișat informațiile în fișier.

- Am adăugat în fișierul *“TEAMS”* echipele inițiale (fiecare jucător are în echipa sa câte 11 jucatori) - folosind funcția “makeGroups”.

În următoarea parte a codului, m-am ocupat de cele **4 jocuri:**

- Am actualizat vectorul *“players”* și fișierul *“FIRSTGame”* cu jucătorii care nu au fost eliminați după primul joc;

- Am creat echipele pentru jocul 2, folosind funcția *“makeGroupsForSecondGame”*;
- Am adăugat în matricea “m” numerele de concurs ale jucătorilor în funcție de echipa din care făcea parte, iar in fișierul *“beforeSECONDGame”* echipele formate;
- Am actualizat vectorul *“players”* și fișierul *“SECONDGame”* cu jucătorii care nu au fost eliminați după al doilea joc joc;

- Am creat echipele pentru al treilea joc, folosind funcția *”makeGroupsForThirdGame”*;
- Am adăugat în matricea “m2” numerele de concurs ale jucătorilor în funcție de echipa din care făcea parte, iar în fișierul *“beforeTHIRDGame”* echipele formate;
- Am actualizat vectorul *“players”* și fișierul *“THIRDGame”* cu jucătorii care nu au fost eliminați după al treilea joc joc;

- Apelez funcția *“GENKEN”* - funcția pentru ultimul joc - până când rămâne un singur câștigător pe care îl afișez pe ecran;

- Am calculat suma pe care a câștigat-o la finalul concursului fiecare supraveghetor;

- Am adăugat în fișierul *“FINAL”* rezultatele finale: câștigătorul împreună cu suma totală câștigată, supraveghetorii afișați în ordine descrescătoare în funcție de suma primită, precum și echipa de supraveghetori care a strâns cea mai mare suma de bani.

✳ **verifyFileExists**

- Tipul funcției: int;
- Parametrii funcției: string *fileName*;
- Rolul funcției: verifică dacă un fișier se poate deschide sau nu; dacă acesta nu se poate deschide, atunci se va afișa un mesaj corespunzător pe ecran.

✳ **readFromFile**

- Tipul funcției: void;
- Parametrii funcției: string *fileName*, string *array[ ]*;
- *Rolul funcției*: citește linie cu liniei informațiile din fișierul dat, adăugându-le în vectorul *“array”*.

✳ **displayNotEliminated**

- Tipul funcției: void;
- Parametrii funcției: fstream *&file*, Player *players[ ]*, string *order*;
- Rolul funcției: afișează în fișierul trimis ca parametru jucătorii care nu au fost eliminați din joc (nume, prenume, număr de concurs). Aceasta poate să îi afișeze crescător în funcție de numerele de concurs ale jucătorilor dacă *order* este *“ascending”* sau descrescător dacă *order* este *“descending”*.

✳ **randomMaskForm**

- Tipul funcției: string;
- Parametrii funcției: int *nb2*, int **nb_circles*, int **nb_triangles*, int **nb_squares*;
- Conceptul jocului: celor 9 supraveghetori trebuie să li se atribuie câte o mască. Ei sunt împărțiți în mod egal, astfel că vom avea 3 supraveghetori cu mască în formă de cerc, 3 supraveghetori cu mască în formî de triunghi și 3 supraveghetori cu mască în formă de pătrat;
- Rolul funcției: returnează tipul măștii pentru un supraveghetor;
- Am considerat: 1 = “circle”, 2 = “triangle”, 3 = “square”;
- Dacă se trimite ca parametru al funcției numărul 1, iar numărul de cercuri nu este 3 încă, atunci se returnează “circle”. Se procedează analog pentru “triangle” și “square”;
- Dacă se trimite ca parametru al funcției numărul 1, dar numărul de cercuri este deja 3, verificam dacă numărul de triunghiuri este atins. Dacă este și el 3, afișăm “square”, dacă nu este, afișăm “triangle”. Se procedează analog atât pentru cazul de triunghi, cât și pentru cazul de pătrat.

✳ **makeGroups**

- Tipul funcției: void;
- Parametrii funcției: Supervisor *supervisors[ ],* Player *players[ ]*, fstream *&file*, string *maskForm*, int *teams[10][12]*, int **nbTeams*, string *supervisorsTeams[10][3]*;
- Conceptul jocului: toți jucătorii trebuie repartizați în mod egal supraveghetorilor. Astfel, vor fi 9 echipe cu câte 11 jucători fiecare;
- Rolul funcției: formează grupele inițiale;
- Pentru fiecare supraveghetor în parte, am afișat în fișierul primit ca parametru numele, prenume și forma măștii ale acestuia, urmând ca sub el sa fie scrișii jucătorii care fac parte din echipa acestuia;
- Am stocat într-o matrice *"superisorsTeams"* pe prima poziție numele supraveghetorului, pe a doua poziție numele, iar pe a treia forma măștii;
- Am generat aleatoriu un număr între 1 și 99. Dacă jucătorul cu numărul de concurs respectiv nu face parte dintr-o echipă, atunci îl afișăm în fișier, adăugăm în matricea *"teams"* numărul de concurs al acestuia și setăm atributul *"hasTeam"* la valoarea 1.

✳ **RED_LIGHT_GREEN_LIGHT**

- Tipul funcției: void;
- Parametrii funcției: Player *players[ ]*, fstream *&file*;
- Conceptul jocului: pentru primul joc se iau în considerare doar numerele de concurs ale jucătorilor. Astfel, jucătorii cu numere de concurs pare sunt eliminați din joc;
- Rolul funcției: afișează jucătorii rămași după primul joc;
- Verific numerele de concurs ale fiecărui jucător. Dacă acesta este impar, atunci îl afișăm în fișierul trimis ca parametru. Dacă acesta este par, setăm atributul de *"eliminated"* al jucătorului respectiv la valoarea 1.

✳ **makeGroupsForSecondGame**

- Tipul funcției: void;
- Parametrii funcției: Player *players[ ]*, fstream *&file,* int m[5][13];
- Conceptul jocului: pentru al doilea joc, jucătorii sunt împărțiți aleatoriu în 4 grupe. Cum după primul joc au rămas doar 50 de participanți, echipele vor avea fiecare câte 12 jucatori. Jucătorii care nu sunt repartizați în nicio echipă, trec automat la jocul următor;
- Rolul funcției: formează echipe pentru al doilea joc;
- Actualizăm atributul de *"hasTeam"* al fiecărui jucător la valoarea 0 (niciun jucător nu are echipă);
- Generăm un număr random între 1 și 99. Dacă jucătorul cu numărul de concurs respectiv nu este eliminat și nu este deja repartizat într-o echipă, atunci îl afișăm în fișierul primit ca parametru, actualizăm atributul *"hasTeam"* la valoarea 1 și adăugăm numărul de concurs al jucătorului în matricea *"m"*;
- Matricea *m* reține numerele de concurs al jucătorilor în funcție de echipa la care a fost repartizat: pe prima linie sunt 12 numere, care reprezintă numerele de concurs ale jucătorilor din prima echipă șamd.

✳ **calculateMaxWeight**

- Tipul funcției: void;
- Parametrii funcției: int *weight1*, int *weight2*, int **maxWeightRound*, int **winner*, int *index1*, int *index2*;
- Rolul funcției: calculează greutatea maximă;
- Atribuie lui *maxWeightRound* maximul dintre *weight1* și *weight2*;
- Atribuie lui *winner index1*, dacă *weight1* este mai mare sau *index2* în caz contrar.

✳ **TUG_OF_WAR**

- Tipul funcției: void;
- Parametrii funcției: Player *players[ ]*, fstream *&file*, int *m[5][13]*;
- Conceptul jocului: Se calculează greutatea totală a fiecărei echipe. Se duelează două câte două echipe și câștigă echipa cu greutatea cea mai mare. Toți jucătorii din celelalte echipe sunt eliminați;
- Rolul funcției: afișează jucătorii rămași după al treilea joc;
- Se calculează greutatea totală a fiecărei echipe și se adaugă valorile obținute în vectorul *“sum[5]”*;
- Afișez pe ecran greutățile echipelor;
- Apelez funcția *“calculateMaxWeight”* pentru primele două echipe;
- Apelez funcția *“calculateMaxWeight”* pentru a treia și a patra echipă;
- Apelez funcția *“calculateMaxWeight”* pentru echipele câștigătoare din fiecare duel;
- Afișez pe ecran greutatea maximă și echipa căreia îi aparține;
- Parcurgem fiecare echipă în parte. Dacă aceasta nu este echipa câștigătoare, actualizăm atributul de *“eliminated”* la valoarea 1 pentru jucătorii din echipa respectivă;
- Afișăm în fișierul primit ca parametru jucătorii care nu au fost eliminați.

✳ **makeGroupsForThirdGame**

- Tipul funcției: void;
- Parametrii funcției: Player *players[ ]*, fstream *&file*, int *m[8][3]*;
- Conceptul jocului: pentru al treilea joc, jucătorii sunt împărțiți aleatoriu în echipe de câte doi. Cum după al doilea joc au rămas doar 14 participanți, vor fi 7 echipe;
- Rolul funcției: formează echipe pentru al treilea joc;
- Actualizăm atributul de *“hasTeam”* al fiecărui jucător la valoarea 0 (niciun jucător nu are echipă), dacă acesta nu a fost eliminat din concurs;
- Generăm un număr random între 1 și 99. Dacă jucătorul cu numărul de concurs respectiv nu este eliminat și nu este deja repartizat într-o echipă, atunci îi acordăm un număr random între 1 și 99 și îl afișăm în fișierul primit ca parametru;
- Actualizăm atributul *“hasTeam”* cu valoarea 1 și adăugăm numărul de concurs al jucătorului în matricea *“m”*;
- Matricea *m* reține numerele de concurs ale jucătorilor în funcție de echipa la care a fost repartizat: pe prima linie sunt 2 numere, care reprezintă numerele de concurs ale jucătorilor din prima echipă șamd.

✳ **MARBLES**

- Tipul funcției: void;
- Parametrii funcției: Player *players[ ]*, fstream *&file*, int *m[8][3]*;
- Conceptul jocului: fiecărui jucător îi este asociat un număr random. Jucătorul cu numărul mai mare câștigă;
- Rolul funcției: afișează jucătorii rămași după al treilea joc;
- Pentru fiecare dintre cele 7 echipe formate, verific ce jucător are numărul mai mare și îl elimin pe celălalt;
- Afișez în fișierul trimis ca parametru jucătorii care nu au fost eliminați, în ordinea descrescătoare a numerelor de concurs.

✳ **RulesOfGenken**

- Tipul funcției: int;
- Parametrii funcției: int a, int b;
- Rolul funcției: afișează un număr în funcție de regulile de la jocul “piatră foarfecă hârtie”;
- Considerăm: 1 = piatră, 2 = hârtie, 3 = foarfecă;
- Aplicăm regulile jocului. Spre exemplu, dacă *“a”* este 1 și *“b”* este 2 (adică se “bate” piatra cu hârtia), câștigă hârtia, deci se afișează 2. Se procedează analog și pentru celelalte cazuri.

✳ **GENKEN**

- Tipul funcției: void;
- Parametrii funcției: Player *players[ ]*, fstream *&file*, int *finalPlayers[ ]*, int **nbFinalPlayers*;
- Conceptul jocului: plecând în ordine descrescătoare a numerelor de concurs, jucătorii se vor duela (1vs1) pe rând. Fiecare mutare are un număr asociat, piatră = 1, hârtie = 2, foarfece = 3. Numărul ales de fiecare jucător este random. Se joacă până când un jucător este eliminat și se merge la următorul până rămâne un singur învingător;
- Rolul funcției: afișează jucătorii rămași după al patrulea joc;
- Luăm fiecare jucător în parte și verificăm dacă acesta a fost eliminat sau nu;
- Dacă nu a fost, îi atribui un număr random între 1 și 3 și îl afișăm în fișierul primit ca parametru (împreună cu numele și numărul de concurs al acestuia). În același timp, adăugăm numărul lui de concurs în vectorul *“finalPlayers”*;
- Aplic funcția *“RulesOfGenken”* jucătorilor rămași în concurs; când un jucător pierde, acesta este eliminat;
- Actualizăm numărul de jucători finaliști (*nbFinalPlayers*).

✳ **displayWinner**

- Tipul funcției: void;
- Parametrii funcției: Player *players[ ]*;
- Rolul funcției: parcurge vectorul de *“players”* și afișează pe ecran singurul jucător care nu a fost eliminat;

✳ **displayWinnerInFile**

- Tipul funcției: void;
- Parametrii funcției: Player *players[ ]*, fstream *&file*;
- Conceptul jocului: câștigătorul primește suma tuturor datoriilor celorlalți concurenți;
- Parcurgem vectorul de *"players"* și adunăm într-o variabilă *"sumWinner"* datoriile jucătorilor care au fost eliminați;
- Afișăm în fișierul primit ca parametru câștigătorul, împreună cu suma totală primită.

✳ **calculateSupervisorsSum**

- Tipul funcției: void;
- Parametrii funcției: Player *players[ ]*, Supervisor *supervisors[ ]*, int *teams[10][12]*, string *supervisorsTeams[10][3]*, double *arrayOfSums[10]*, int **nbOfSums;*
- Conceptul jocului: fiecare supraveghetor primește suma aferentă jucătorilor eliminați care îi aparțineau. Supraveghetorul care îl are pe câștigător va primi datoria lui înmulțită cu 10. Din sumele primite se scade datoria lor inițială și astfel se află cât le-a rămas la fiecare.
- Rolul funcției: adaugă în vectorul *"arrayOfSums"* suma primită de fiecare supraveghetor la finalul concursului;
- Pentru fiecare supraveghetor, calculăm într-o variabilă *"sum"* suma datoriilor jucătorilor pe care i-a avut în echipă, iar în variabila *"n"* contorizăm numărul de jucatori care au fost eliminați din echipa lui;
- Dacă la final, *n* este diferit de 11, adică supraveghetorul are câștigătorul în echipă, acesta primește datoria lui înmulțită cu 10 din care se scade datoria lui inițială;
- Dacă supraveghetorul nu îl are pe câștigător în echipă, se scade din suma calculată anterior (*"sum"*) datoria lui inițială;
- Adăugăm sumele finale primite de fiecare supraveghetor în vectorul *"arrayOfSums"*.

✳ **displayArray**

- Tipul funcției: void;
- Parametrii funcției: T *v[ ]*, int *n*; (unde T este un template)
- Rolul funcției: afișează vectorul *"v"* de lungime *"n"*.

✳ **selectionSort**

- Tipul funcției: void;
- Parametrii funcției: X *v[ ]*, int *n*; (unde X este un template)
- Rolul funcției: folosește metoda "selection sort" pentru a ordona descrescător elementele vectorului *"v"*.

✳ **displaySupervisorsSum**

- Tipul funcției: void;
- Parametrii funcției: Supervisor *supervisors[ ]*, double *arrayOfSums[10],* fstream *&file*;
- Rolul funcției: afișează în fișierul primit ca parametru numele, prenumele și suma totală primită ale fiecărui supraveghetor.

✳ **displayTeamWithBiggestSum**

- Tipul funcției: void;
- Parametrii funcției: Supervisor *supervisors[ ]*, fstream *&file*;
- Rolul funcției: afișează în fișierul primit ca parametru echipa de supraveghetori care a câștigat cea mai mare suma de bani, precum și suma primită;
- Calculăm, folosind variabilele *“sumCircle”*, *“sumTriangle”* și *“sumSquare”* suma câștigată de echipa cerc, triunghi, respectiv pătrat;
- Dacă *“sumCircle”* are cea mai mare valoare dintre cele 3, afișez în fișier că echipa “cerc” a câștigat, precum și valoarea lui *“sumCircle”*. Se procedează analog și pentru celelalte două cazuri.