UNIVERSITATEA "STEFAN CEL MARE", SUCEAVA
FACULTATEA DE INGINERIE ELECTRICA SI STIINTA CALCULATOARELOR
SPECIALIZAREA CALCULATOARE

PROIECT DISCIPLINA POO

[JOCUL SNAKE în C++]

Autor
Studentul CIOBAN I. BENIAMIN

SUCEAVA 2023

TEMA PROIECT

TEMA SI MOTIVAȚIA ALEGERII

Tema proiectului este de a realiza jocul clasic 2d Snake. Jocul constă în a mânca cât mai multe fructe, care apar pe poziții aleatorii și libere de pe mapă.

Pentru fiecare fruct mâncat, lungimea șarpelui crește cu un segment, astfel spațiul disponibil în care se poate mișca șarpele scade și trebuie evitate ciocnirile cu marginile mapei și coada șarpelui. Scopul fiind de a ocupa toată mapa.

CUPRINS

Cuprins:

CAPITOLUL I

1. ELEMENTE TEORETICE

1.1. DESCRIEREA PROBLEMEI

Acest subcapitol se ocupă cu problematica realizării și implementării aplicației dorite, adică jocul Snake. Astfel aplicația dorită necesită următoarele implementări:

- Prezentarea grafică a jocului (mapă, șarpe, fruct) și a meniurilor (texte, butoane) ;
- Preluarea datelor de intrare (input-urilor) de la tastatură;
- Prelucrarea datelor de intrare și actualizarea stării jocului ;
- Procesarea stării jocului (Start, Game Over, Pause, Exit, jocul în sine) ;
- Salvarea progresului, scorului (Highscore).

Aplicația trebuie să ofere utilizatorului control asupra începerii jocului, opririi, pauzei sau a terminării aplicației, astfel sunt necesare meniuri pentru aceste operații. De asemenea, aplicația trebuie să țină seama și de scorul obținut, astfel trebuie salvat cel mai mare scor într-un fișier.

Având în vedere jocul în sine, putem stabili elementele principale care definesc jocul Snake. Prin urmare în componența jocului avem: o mapă, un șarpe și un fruct. Aceste elemente au un rol semnificativ și bine determinat, astfel:

- Mapa:
  - reprezintă locul unde are loc acțiunea
  - este sub formă matriceală
- Șarpele:
  - reprezintă un șir de segmente
  - se mișcă pe mapă
  - nu are voie să se suprapună sau să iasă înafara mapei
  - lungimea crește când mănâncă un fruct
- Fruct:
  - apare aleatoriu pe mapă
  - dispare când este mâncat de șarpe și reapare imediat, în alt loc liber
  - trebuie să existe un singur fruct pe mapă în orice moment

Mai multe detalii despre implementare și abordare sunt prezentate în subcapitolul următor.

1.2. ABORDAREA TEORETICA A PROBLEMEI

Privind problema din punct de vedere al datelor, putem asocia componentele principale enumerate în subcapitolul anterior cu: matrice, vectori , poziții și valori numerice.

Prin urmare putem asocia elementele astfel:

- Mapa este asociată cu o matrice, unde valorile 0 reprezintă gol/liber, 1 reprezintă șarpele, 2 reprezintă fructul. Un exemplu ar fi:

0 2 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 1 1 1 0
0 0 0 1 1 1 1 1 1 1 1 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0

Din această matrice putem să observăm că fructul se află pe linia 0, coloana 1, putem vedea ce poziții ocupă șarpele însă nu putem descifra sensul în care merge sau continuitatea acestuia.

Prin urmare, matricea mapei este folosită pentru a salva pozițiile șarpelui și a fructului, pentru verificări de ciocniri atunci când șarpele se mișcă sau pentru a verifica locuri libere când reapare fructul.

- Șarpele fiind un șir de segmente, putem asocia acest șir cu un array de poziții (Vector2) din matricea mapă. Pe baza exemplului anterior avem:

Poziție array	0	1	2	3	4	5	6	7	8	...
Valoare	(4,1)	(5,1)	(5,2)	(6,2)	(7,2)	(8,2)	(8,3)	(8,4)	(9,4)	...

Poziția 0 din array este considerată capul șarpelui iar modificările de mișcare aduse șarpelui sunt procesate doar pe valorile acestei poziție, restul pozițiilor din array preiau valorile de pe pozițiile precedente, astfel, ultima poziție ia valoarea de pe penultima poziție și tot așa, pană se ajunge la poziția 0, unde în loc să preia o valoare, adunăm o unitate în sensul în care merge șarpele.

- Fructul este reprezentat printr-o poziție (Vector2), poziție care se schimbă atunci când fructul este mâncat.

Pentru a implementa meniurile, putem folosi niște variabile booleene care salvează starea aplicației, de exemplu când aplicația se află în meniul principal, avem variabila „inMainMenu” setată pe „true”, în caz contrar e setată pe „false”, când aplicația rulează jocul avem „inGame” setată pe true, etc.

Pentru a implementa „score” și „best score”, folosim un contor „score” care pornește de la 0, când șarpele mănâncă fructul, acest contor crește cu o unitate, apoi când jocul se termină (Game Over), citim dintr-un fișier (dacă există) „best score”, comparăm cu aceste două valori și salvăm în același fișier noul „best score”.

Pentru a implementa cerințele problemei sunt utilizate:

I. Librărie grafică cum ar fi SFML bazată pe OpenGL.
   Această librărie oferă multe posibilități. Cum ar fi de a crea o fereastră în care putem afișa elementele grafice (Texturi, texte, imagini, etc), oferă un sistem de preluare a input-urilor de la tastatură sau mouse, și multe alte facilități, însă pe care nu le vom folosi.
II. Texturi pentru: text, șarpe, fruct și mapă;
III. Utilizarea limbajului de programare C++, mediul de programare Visual Code în Linux;

1.3. ELEMENTE SPECIFICE POO

Pentru problema descrisă anterior, în rezolvarea ei se vor crea clase si metode specifice pentru: „snake”, „fruit” , „map”, „game” și „input”. Pe lângă metodele specifice fiecărei clase, se vor folosi și constructorii și deconstructorii claselor.

Folosind o librărie pentru partea grafică, se vor folosi și clase deja făcute, de exemplu clasa „Vector2f”, care reprezintă un vector de lungime 2, cu parametrii „x” și „y”, de tipul „float”, clasa „Texture” care reprezintă o imagine, citită din memorie sau creată chiar din cod. Clasa „Sprite” este o clasă care se folosește de clasa „Texture”, poate chiar selecta doar porțiune din textură pentru afișare, obiectele acestei clase sunt afișate. Clasa „RenderWindow” reprezintă fereastra pe care afișăm sprite-urile, etc.

1. Clasa „Fruit”.

```
class Fruit{
private:
    sf::Sprite sprite;
    sf::Vector2f position;
public:
    ///Constructor
    Fruit(sf::Texture&,int);

    ///Metode
    sf::Vector2f& GetPosition();
    void UpdatePosition(sf::Vector2f);
    void DrawFruit(sf::RenderWindow&, int);

    ///Deconstructor
    ~Fruit();
};
```

În aceasta clasă ne folosim de clasele menționate anterior pentru a crea atributele:

- „sprite” – reprezintă un obiect creat din o textură, acestui obiect îi putem modifica scala, poziția, rotația, culoarea, etc.
- „position” – reprezintă poziția pe care o va avea fructul în scenă/ în fereastră;

Constructorul primește o textură care va fi prelucrată pentru a crea sprite-ul, și o valoare de tipul „int” care reprezintă rezoluția la care trebuie dimensionată. În metoda `GetPosition ()` returnăm variabila „position”, În metoda `UpdatePosition()` actualizăm poziția curentă cu poziția primită în parametru. În metoda `DrawFruit()` primim ca parametru fereastra pe care trebuie afișată sprite-ul folosind `sf::RenderWindoW::Draw()`, și un parametru de tipul „int” care reprezintă rezoluția în pixeli a unui pătrățel, datele mapei fiind sub formă matriceală, pozițiile salvate în obiecte, cum ar fi „position” pentru „Fruit”, sunt numere întregi de la 0 la 11, prin urmare daca folosim valorile din „position”, fructul este afișat pe pixelii de la 0 – 11 pe coordonata X și Y. Prin urmare înmulțim poziția cu rezoluția în pixeli a unui pătrățel.

2. Clasa „Snake”

```
class Snake{
private:
    const int tilePixelSize=64;
    const int tilesPerRow=5;
    const int snakeMaxLenght=144;
    int lenght;

    sf::Texture snakeTexture;

    int *bodyParts;
    sf::Vector2f *body;

    sf::Vector2f direction;
public:
    ///Constructor
    Snake(sf::Texture&);

    ///Metode
    int GetSnakeLenght();
    sf::Vector2f GetSnakeBodyPosition(int);
    void MoveSnake(sf::Vector2f&);
    void DrawSnake(sf::RenderTarget &, int);
    bool IncreaseSnake();

    ///Deconstructor
    ~Snake();
};
```

În clasa „snake” avem atributele următoare:

- Constante pentru:
  - dimensiunea rezoluției unui segment grafic al șarpelui,
  - lungimea maximă a șarpelui;
- „lenght” – parametru ce indică lungimea curentă a șarpelui;
- „snakeTexture” – acest parametru salvează textura din care se vor crea „sprite-uri” pentru diferitele părți ale șarpelui;
- „direction” – reprezintă un vector cu 2 valori între [-1, 1], care indică direcția curentă a șarpelui;
- „body” și „bodyParts” este perechea care reprezintă șarpele. „body” este un array de poziții, pentru fiecare segment al șarpelui, iar „bodyParts” este un array de aceiași lungime, în care este salvat valoarea care arată ce parte grafică trebuie afișată, folosind enumeratorul următor:

În acest enumerator sunt reprezentate poziția dintr-o matrice astfel : i * rows + j

```
enum SnakeBody{
    HeadUp=3,
    HeadLeft=4,
    HeadDown=9,
    HeadRight=8,
    TailDown=13,
    TailRight=14,
    TailUp=19,
    TailLeft=18,
    LeftDown=0,
    Horisontal=1,
    RightDown=2,
    LeftUp=5,
    Vertical=7,
    RightUp=12
};
```

Aceste valori transformate în pereche [i, j] și înmulțită cu „tilePixelSize” va rezulta bucata din textura principală care trebuie decupată pentru a afișa bucata de șarpe corespunzătoare după cum se poate observa în textura următoare, segmentată și numerotată

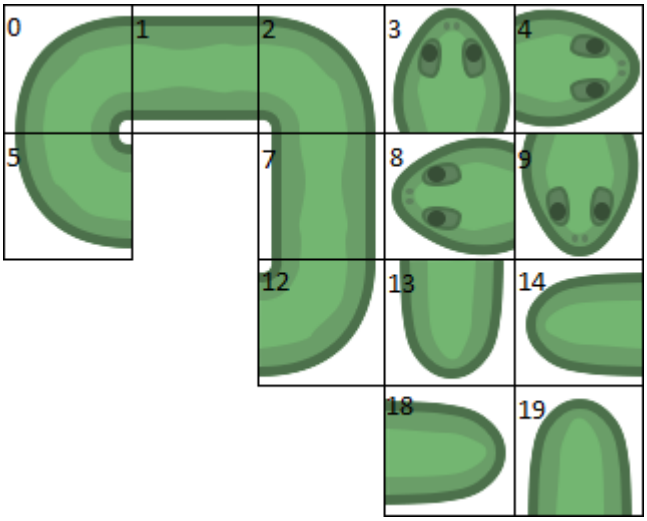

Fig. 1. Textura segmentată

În constructor se inițializează array-urile, se setează textura și se creează un șarpe de lungime 3.

Metoda `GetSnakeLenght()` returnează lungimea șarpelui, metoda `GetSnakeBodyPosition()` returnează poziția unui segment ales. Metoda `MoveSnake()` primește un vector2f, care reprezintă o nouă direcție. În funcție de noua direcție și de vechea direcție, se calculează noul segment grafic care trebuie reprezentat. Capul, coada, și primul segment după cap sunt calculate, restul sunt preluate de la segmentele precedente. . Metoda `DrawSnake()` primește ecranul și rezoluția la care trebuie scalate sprite-urile și afișează toate segmentele. . Metoda `IncreaseSnake()` mărește lungimea, array-urile și calculează o nouă poziție pentru segmentul nou, dacă nu se mai poate adăuga un segment (a atins lungimea maximă) returnează „false”, dacă mărirea a reușit returnează „true”.

În deconstructor se eliberează memoria folosită.

3. Clasa „Map”

```
class Map{
private:
    const int EMPTY = 0;
    const int WALL = 1;
    const int FRUIT = 2;
    const int height=12, width=12;

    sf::Sprite sprite;

    int **map;

    void CreateMapData();

public:
    ///Constructor
    Map(sf::Texture&, int);

    ///Metode
    void SetInfo(int, int, int);
    int GetInfo(int, int);
    bool UpdateMap(Snake&);
    void UpdateMap(Fruit&);
    sf::Vector2f RandomFreePosition();
    bool IsFull();
    void DrawMap(sf::RenderWindow&);

    ///Deconstructor
    ~Map();
};
```

În clasa „Map” avem atributele următoare:

- Constante pentru:
  - dimensiunea mapei (width și height),
  - identificatori valori din matrice (gol, obstacol, fruct);
- „sprite” – parametru în care este salvat sprite-ul fructului;
- „map” – matrice în care sunt salvate entitățile;

Metoda `CreateMapData()` este apelată de constructor pentru a aloca și a pune valori în matricea „map”, de dimensiunile primite ca parametru în constructor. Metoda `SetInfo()` setează pe poziția specificată valoarea trimisă în parametru, respectiv `GetInfo()` returnează valoarea. Metoda `UpdateMap()` are două variante, una în care este primit ca parametru șarpele și una în care este primit fructul, diferența dintre cele două este că atunci când este primit șarpele ca parametru, matricea este resetată, când este primit fructul, matricea nu este resetată. Metoda `IsFull()` returnează „true” dacă nu mai sunt „goluri”/ spații libere în matrice, și „false” în caz contrar. Metoda `RandomFreePosition()` returnează un „Vector2f” care indică către o poziție aleatorie și liberă din matrice. Pentru a obține o poziție aleatorie se folosește `std::rand()` pentru a genera numere aleatorii, astfel, pentru a obține o valoare dintr-un interval [0, n] se apelează în modul următor `val = rand() % n` Metoda `DrawMap()` afișează mapa pe ecranul primit ca parametru. În deconstructor se eliberează memoria alocată pentru matrice.

4. Clasa „MyInput”

```
class MyInput{
public:
    bool onceUp, onceDown, onceLeft, onceRight, onceEnter, onceEsc;
    bool up, down, left, right, enter, esc;

    ///Constructor
    MyInput();

    ///Metode
    void GetInput();

    ///Deconstructor
    ~MyInput();
};
```

În clasa „MyInput” avem atributele următoare:

- parametri de tipul „bool” pentru indicarea apăsării unor butoane specifice.

Parametrii „up”, „down”, „left”, „right”, „enter”, „esc” sunt „true” cât timp sunt apăsate butoanele specifice, în schimb parametrii „onceUp”, „onceDown”, „onceLeft”, „onceRight”, „onceEnter”, „onceEsc” sunt „true” doar în primul frame în care sunt apăsate butoanele. Apelarea metodei `GetInput()` verifică dacă butoanele sunt „up”, „down”,... etc. sunt false și dacă butoanele tastaturii sunt apăsate, atunci butoanele „onceUp”, ..etc. devin true și pe urmă se fac true și butoanele „up”,... etc. În caz contrar, dacă deja erau true, butoanele „once” devin false.

5. Clasa „Game”

```
class Game{
private:
    sf::RenderWindow window;
    sf::Texture mapTexture;
    sf::Texture snakeTexture;
    sf::Texture fruitTexture;
    sf::Texture menuTexture;
    sf::Texture scoreTexture;
    sf::Texture numbersTexture;

    char saveFile[MAX_FILE_PATH];

    int pixelsPerUnit, buttonSelected, score, bestScore;
    bool inStartingMenu, inPauseMenu, inGame, inGameOverMenu;

    Map* map;
    Fruit* fruit;
    Snake* snake;

    sf::Vector2f direction;

    bool DrawNumber(sf::Vector2f,int);
    bool ShowInGameScore();
    bool ShowMainMenu();
    bool ShowPauseMenu();
    bool ShowGameOverMenu();
public:
    ///Constructor
    Game(int, char*);

    ///Metode
    bool LoadTextures(char*, char*, char*, char*, char*, char*);

    bool Start();
    bool StartGame();
    bool UpdateGame(MyInput&);
    bool ContinueGame();
    bool PauseGame();
    bool RestartGame();
    bool ReturnToMenu();
    bool GameOver();
    bool Exit();

    ///Deconstructor
    ~Game();
};
```

În clasa „Game” avem atributele următoare:

- „window” – parametru în care este salvat ecranul,
- Texturile folosite pentru mapă, fruct, șarpe, meniuri, etc.,
- Locația fișierului în care este salvat cel mai bun scor,
- „pixelsPerUnit” - rezoluția unei poziții,
- „buttonSelected” – un contor care salvează ce opțiune este selectată,
- „score” și „bestScore”,
- „inStartingMenu”, „inPauseMenu”, „inGame”, „inGameOverMenu” – parametrii ce indică starea jocului,
- „direction” – vector ce indică noua direcție selectată de utilizator,
- Parametrii ce salvează instanțe ale claselor „Map”, „Fruit”, „Snake”

Metodele `ShowInGameScore()`, `ShowMainMenu()`, `ShowPauseMenu()` și `ShowGameOverMenu()` doar afișează pe ecran textele corespunzătoare meniurilor active. Metoda `DrawNumber()` afișează pe ecran la poziția dată ca parametru, numărul corespunzător parametrului primit. Constructorul primește rezoluția dorită a aplicației și locația fișierului în care este salvat scorul. Metoda `LoadTextures()` primește ca parametrii o serie de locații de unde citește imaginile dorite pentru texturi. Metoda `Start()` inițializează componentele necesare cum ar fi: „window” , „MyInput” , verifică și încarcă cel mai bun scor din fișier dacă există, setează starea aplicației pe meniul principal, etc. Metoda `StartGame()` setează aplicația pe starea de joc, instanțiază obiectele necesare cum ar fi mapa, șarpele și fructul. Iar în loop-ul aplicației este apelată metoda `UpdateGame()` , metodă ce prelucrează inputul și aplică modificări corespunzătoare mapei, șarpelui și fructului și le afișează pe ecran. Metoda `PauseGame()` poate fi apelată doar dacă starea aplicației este pe starea de joc, metoda păstrează starea de joc dar activează și starea de meniu de pauză, setând „inPauseMenu” pe „true”. Metodele `ContinueGame()`, `RestartGame()`, `ReturnToMenu()` și `GameOver()` setează starea aplicației în mocul corespunzător fiecărei metode. Pentru fiecare stare este apelată afișarea specifică. Metoda `Exit()` oprește aplicația, deconstructorul este apelat și se eliberează memoria.

CAPITOLUL II

2. IMPLEMENTARE

Modul de rezolvare a acestei probleme este prin crearea unei clase principale „Game”, trebuie instanțiată doar o dată în „main”, obiectul instanțiat trebuie să primească locațiile asset-urilor, a fișierului în care se salvează scorul, și rezoluția ecranului. Astfel, în clasă trebuie să existe atribute pentru păstrarea texturilor și a fișierului. După ce s-au încărcat, se dă „start la aplicație”, în start se inițializează fereastra, un obiect pentru input, și se setează starea aplicației. Cât timp fereastra este deschisă se execută un „loop” în care are loc prelucrarea inputului, a stărilor, a meniurilor, a jocului, a afișărilor grafice și a calculelor.

Clasa „Game” trebuie să conțină atribute booleene care reprezintă starea aplicației, iar în funcție de aceste stări, în loop fie se afișează meniuri fie rulează jocul în sine. Dacă aplicația este în starea „inMainMenu”, starea de început când pornește aplicația, se afișează meniul principal și se așteaptă să fie selectate unul din cele două butoane „Play” sau „Exit”. Dacă se selectează „Exit”, atunci se termină programul, dacă se selectează „Play”, se setează starea „inGame” și se pornește rularea jocului.

Când rulează jocul pentru prima dată, se setează scorul 0, se instanțiază un obiect de tipul clasei „Map”, care conține o matrice în care sunt salvate pozițiile șarpelui și a fructului, se instanțiază un obiect de tipul clasei „Snake” care conține un șir de poziții de pe mapă, este de lungime 3 și se poziționează pe mapă, se instanțiază un obiect de tipul clasei „Fruit” care conține o poziție de pe mapă.

După ce s-au instanțiat obiectele jocului, se updatează matricea mapei cu pozițiile șarpelui și folosind o funcție de „random”, căutăm în matrice o poziție liberă unde să poziționăm fructul iar acum putem spune că jocul a început. Folosind o clasă pentru input, procesăm direcția în care să meargă șarpele.

Când se termină jocul, se deschide meniul de „Game Over” unde este afișat scorul si „Best Score”, iar dacă noul scor este mai mare decât „Best Score” atunci salvăm în fișier noul scor.

O prezentare grafică a modului de lucru este în figura următoare:

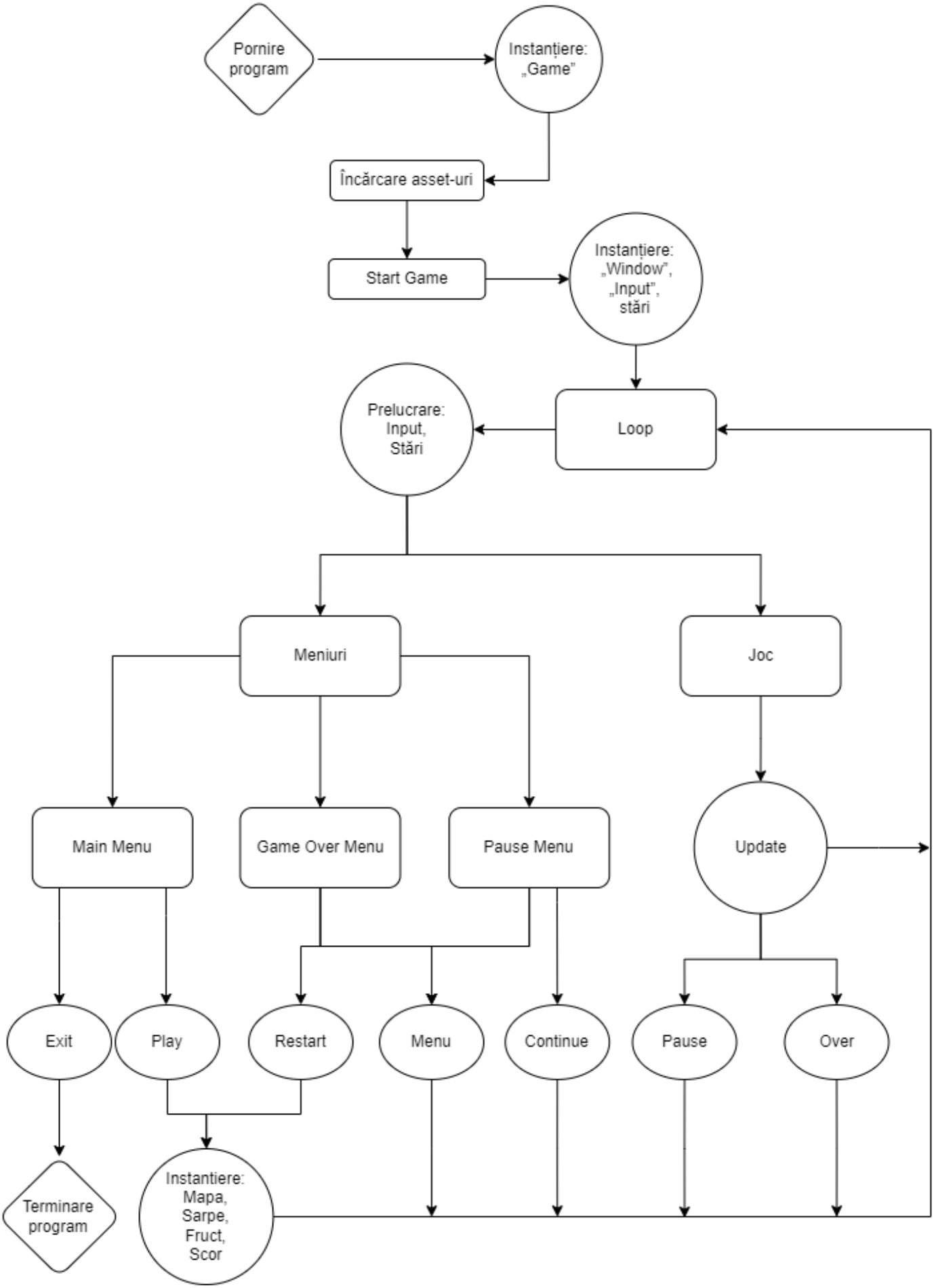

Diagrama 1. Modul de lucru al aplicației

Modul de prezentare propus inițial arăta astfel:

Fig. 2. Meniul principal. Butonul „PLAY” selectat

Fig. 3. Meniul principal. Butonul „EXIT” selectat

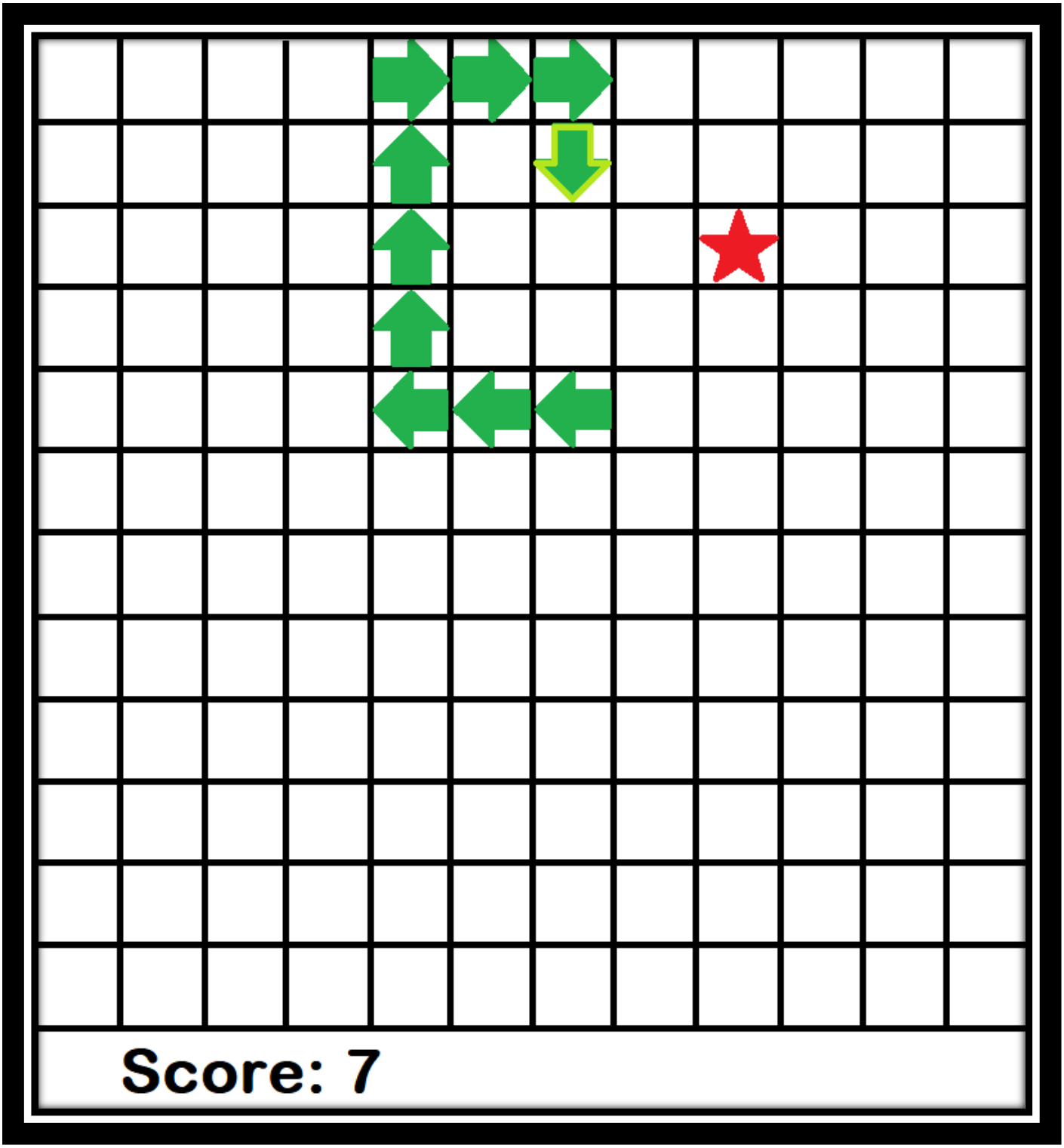

Fig. 4. Jocul

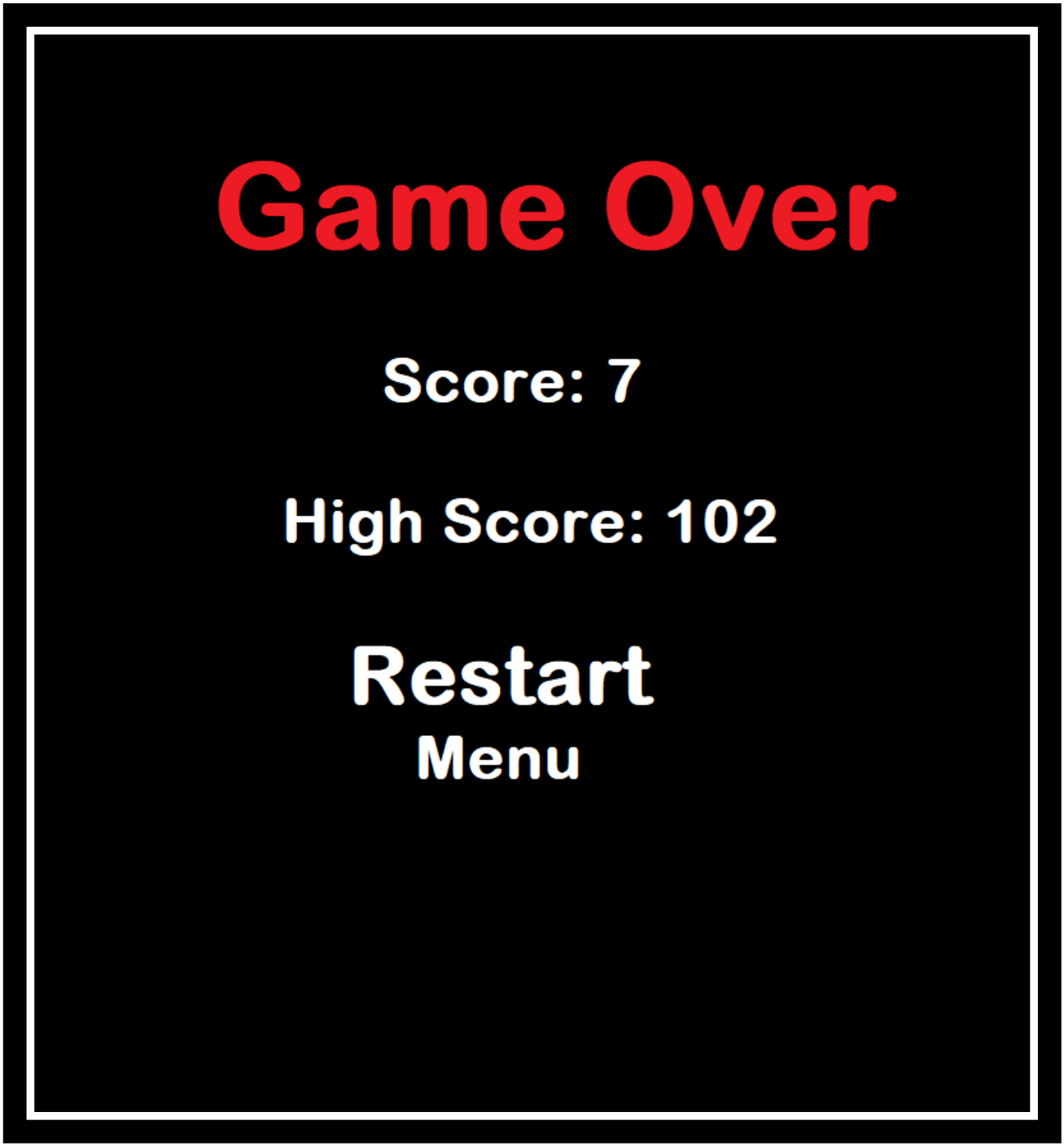

Fig. 5. Meniul GameOver. Butonul „Restart” selectat

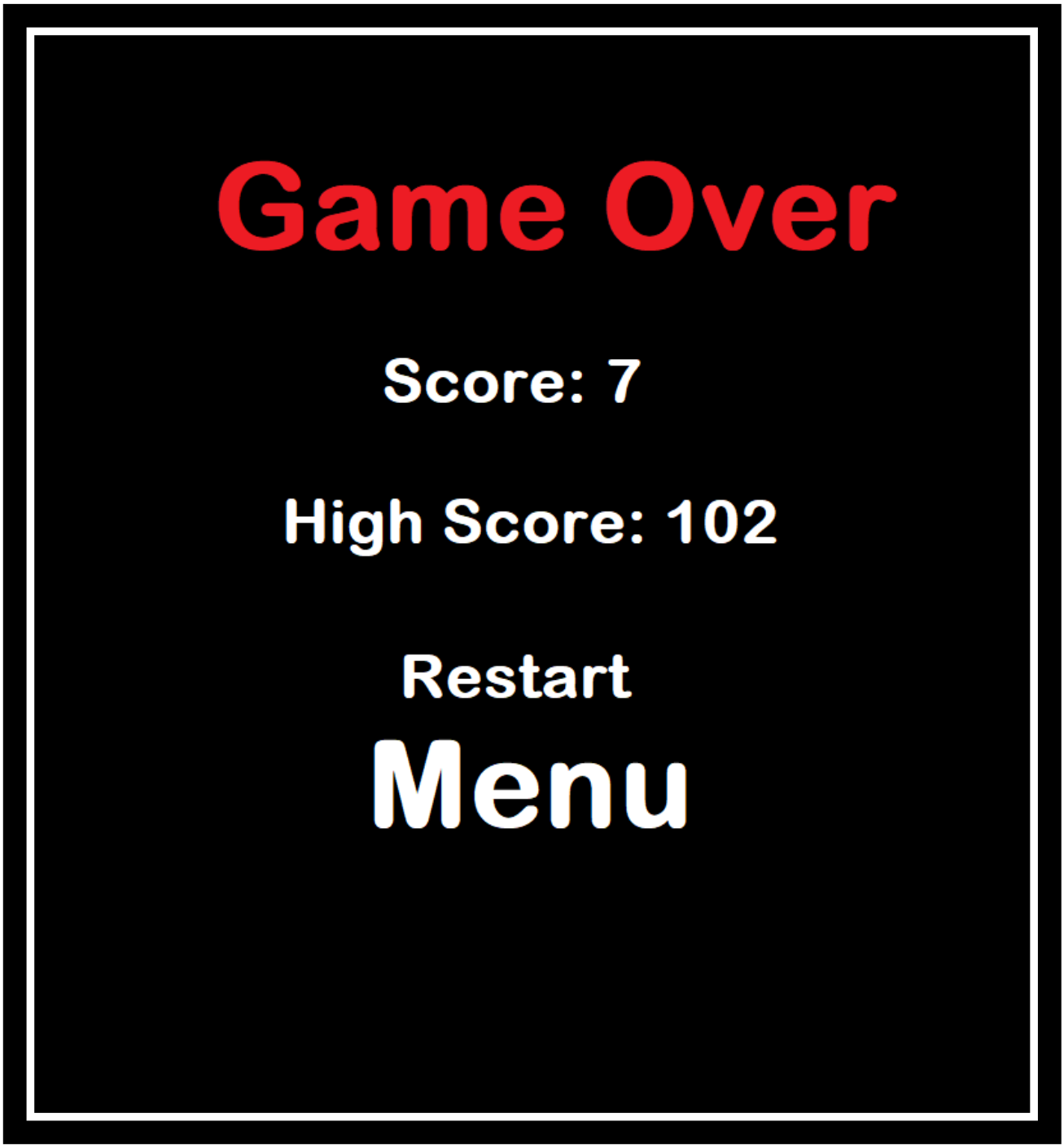

Fig. 6. Meniul GameOver. Butonul „Menu” selectat

2.1. TEHNOLOGII FOLOSITE

Pentru a realiza proiectul, pe linux, se va folosi limbajul de programare „C++”, împreună cu librăria grafică „SFML”. Ca mediu IDE utilizat în construcția proiectului este folosi ”Visual Studio Code” iar pentru compilarea programului se va folosi utilitarul „make”, care necesită un fișier „makefile”. Acest fișier definește un set de sarcini de executat prin care putem să construim executabilul și să gestionăm proiectul.

2.2. DIAGRAMA DE CLASE, SCHEMA BLOC, WORKFLOW

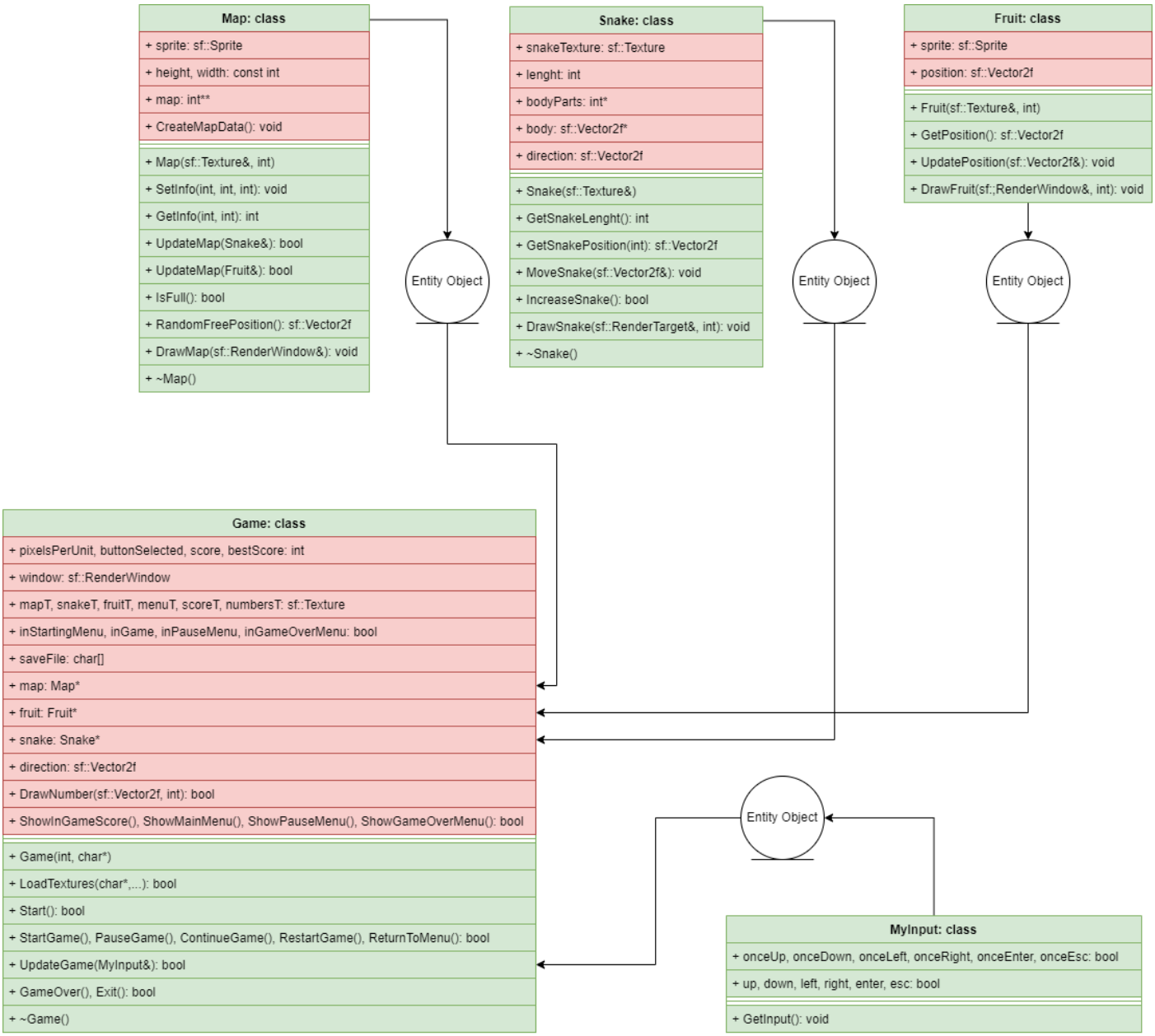

Fig. 7. Diagrama UML de clase a aplicației

Modul de lucru este în felul următor:

Este instanțiată clasa „Game”, este apelată metoda `LoadTextures()` pentru a încărca texturile necesare pentru aplicație apoi este apelată metoda `Start()` care instanțiază atributul „window”, un obiect de tipul clasei „MyInput,”, setează starea pe meniul principal și începe un loop cât timp fereastra „window” este deschisă. În acest loop fereastra este resetată, sunt procesate stările aplicației și dacă aplicația este în meniuri se apelează metodele pentru afișare a meniurilor, și se prelucrează input-ul pentru a selecta butoanele

respective. Dacă este selectată începerea jocului, starea este pusă pe „inGame”, se apelează metoda `StartGame()` care instanțiază mapa, un șarpe și fructul. Apelează metoda mapei pentru `UpdateMap()` cu parametrul șarpe, apoi apelează metoda mapei `RandomFreePosition()` pentru a poziționa fructul pe un loc liber, după care apelează metoda mapei `UpdateMap()` cu parametrul fruct. Astfel matricea mapei este completă și se revine în loop. Acum fiindcă este starea de joc, se apelează metoda `UpdateGame()` în care sunt prelucrate inputurile pentru a determina noua direcție a șarpelui, se verifică ciocnirile, dacă a fost mâncat fructul (poziția nouă a capului coincide cu poziția fructului, se incrementează scorul), dacă trebuie să fie mărit șarpele (se mărește șarpele, se updatează mapa cu pozițiile șarpelui, se caută o poziție liberă pentru fruct, se updatează poziția fructului, apoi se updatează mapa cu fructul), apoi se afișează în ordine mapa, fructul, șarpele și mesajul pentru scor și valoarea sa. Dacă nu au fost îndeplinite condițiile de terminare, este returnat „true” și se efectuează loop-ul. În caz contrar, este returnat „false”, se apelează metoda `GameOver()` unde se compară noul scor cu cel mai bun scor, se salvează scorul în fișier, se șterg instanțele mapei, șarpelui și a fructului, și se setează starea pe meniul de game over „inGameOverMenu”.

2.3. GRAFICA ÎN MOD TEXT

Pentru reprezentarea textului în modul grafic se folosește o textură care conține textul respectiv, din care decupăm cuvintele necesare pentru afișare.

Decuparea se face prin selectarea unei porțiuni (dimensiune, poziție) din textura principală.

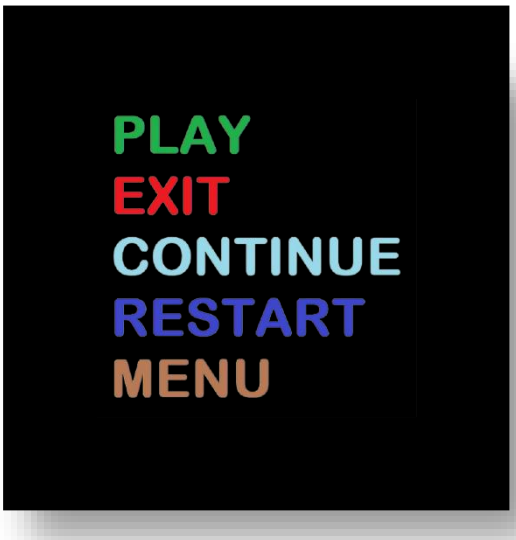

Fig. 8. Textură cu texte pentru meniuri.

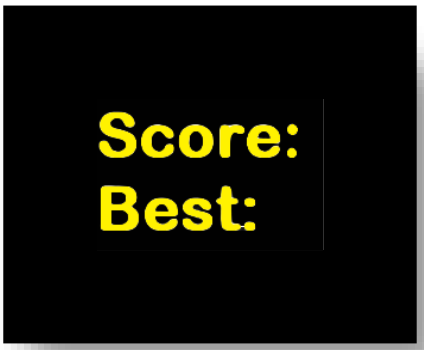

Fig. 9. Textură cu texte pentru scor.

Fig. 10. Textură cu cifre pentru valoarea scorului.

Pentru afișarea numerelor folosim o textură prezentată în Fig. 9, din care decupăm cifra corespunzătoare care trebuie afișată, decuparea se face prin crearea unui „Sprite” care face referință la textură și cu un „Rectangle” selectăm poziția și dimensiunea în pixeli a zonei care trebuie decupate.

Exemplu afișarea cifrei „8”: Textura este pe 640x64 pixeli, astfel fiecărei cifră îi corespunde 64x64 de pixeli, această valoare (64, 64) este pusă în dimensiunea „rectangle”-ului. Cifrele fiind în ordine, iar poziția de decupare începe din colțul stânga sus al „rectangle”-ului, poziția selectată este (0, 8*64). Deci pentru o cifră „i”, poziția este (0, i*64).

CAPITOLUL III

3. ANALIZA SOLUTIEI IMPLEMENTATE

3.1. FORMATUL DATELOR DE I/O

Datele de intrare pentru aplicație sunt apăsările de taste de la tastatură, reprezentate sub formă booleene. Astfel, tastele pentru datele de intrare sunt următoarele: a /left arrow, w /up arrow, d /right arrow, s /down arrow, esc, enter.

Datele de ieșire sunt reprezentări grafice ale elementelor jocului, bazate pe operațiile corespunzătoare selectate, de asemenea datele salvate în fișier reprezintă valoarea numerică a celui mai bun scor obținut.

3.2. STUDII DE CAZ

Așa cum a fost prezentat în subcapitolul anterior, datele de intrare sunt percepute ca „true” sau „false”, pentru apăsarea sau ne apăsarea unor taste. Datorită faptului că aplicația rulează un loop, inputul este preluat în fiecare ciclu, lucru care poate fi dăunător pentru unele scopuri, prezentate ulterior.

În meniuri, pentru a alege un buton, trebuie selectate cu ajutorul butoanelor de w /up arrow și s /down arrow, în meniurile cu 3 opțiuni, la apăsarea butoanelor nu se poate alege opțiunea din mijloc datorită loop-ului care rulează 60 de cicluri pe secundă și trece peste a doua opțiune foarte rapid, ar fi trebuit ca tasta să fie apăsată timp doar de 1/60 secundă pentru a putea alege a doua opțiune, de aceia s-au introdus parametrii „once[TASTA]” care sunt „true” doar în ciclul în care au fost apăsate după cum se poate observa in graficul următor:

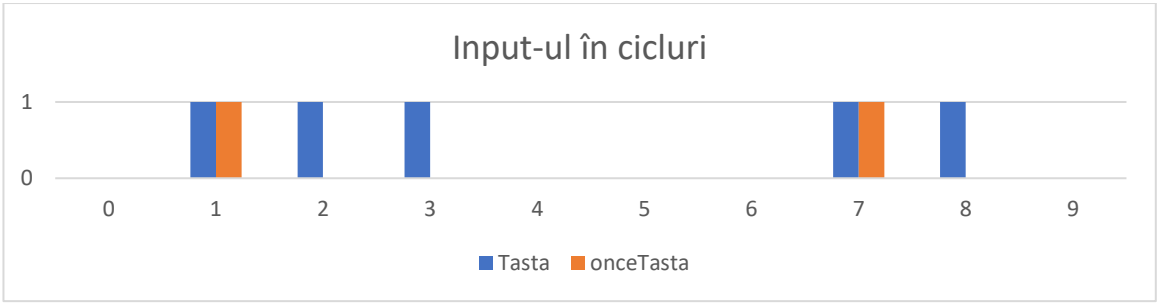

Fig. 11. Valorile datelor de intrare.

Atunci când „Tasta” este 0 („false”) și „onceTasta” este 0. Când „Tasta” devine 1 („true”), „onceTasta” devine și el 1, însă în ciclul următor devine 0. Schimbarea aceasta este datorată verificării dacă butonul de la tastatură a fost apăsat și dacă „Tasta” este 0, doar în acest caz „onceTasta” devine 1, în restul cazurilor este 0.

De asemenea aceste apăsări de taste nu oferă nici-un răspuns în aplicație dacă sunt apăsate într-o stare a aplicației care nu au nevoie de ele, de exemplu:

- în meniul principal doar butoanele w/s sau up/down arrow și enter cauzează efecte în aplicație, astfel: w/s selectează butoanele și enter apelează butoanele
- în joc, butonul esc pune pauză (deschide meniul de pauză) și butoanele wasd / up down left right arrow mișcă șarpele (în funcție de cum sunt apăsate creează un vector2f de lungime absolută 1, care indică direcția de mișcare)

CAPITOLUL IV

4. MANUAL DE UTILIZARE

Compilare/rulare aplicație

Pentru a rula aplicația, trebuie să mergeți în directorul `./Joc_Snake/`
Rulați comanda `make` pentru a crea executabilul, apoi rulați comanda `./bin/GAME.exe` pentru a porni aplicația.
Se poate alege dimensiunea ferestrei prin adăugarea liniei de comandă anterioare unul din următorii parametrii **small**, **medium** sau **big**. Exemplu `./bin/GAME.exe medium`

Interfața aplicației și utilizarea sa este prezentată în descrierea următoare:

- Meniul principal:

Fig. 12. Meniu principal

Meniul prezintă două butoane „Play” și „Exit”. După cum se observă, în Fig. 12, butonul „Play” are dimensiunea mai mare față de butonul „Exit”, datorită faptului că momentan butonul „Play” este selectat, de asemenea dacă ar fi selectat butonul „Exit”, atunci dimensiunea i-ar crește și dimensiunea celuilalt buton ar scădea.

Pentru a selecta un buton se apasă tastele W / up arrow pentru a muta selecția sus și respectiv S / down arrow pentru a muta selecția jos. Pentru a activa butonul ales, se apasă tasta ENTER. Această selecție determină pornirea jocului sau terminarea programului.

- Interfața jocului:

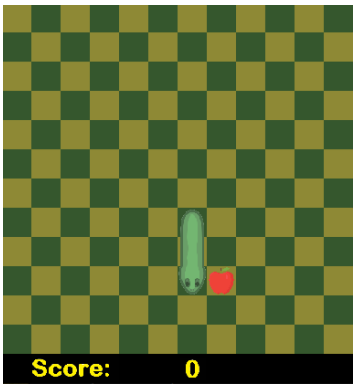

Fig. 13. Interfața joc

În Fig. 13 este prezentată interfața jocului, după cum se observă, jocul are formă matriceală, ceea ce indică o mișcare în 4 direcții. Mișcarea se efectuează prin apăsarea tastelor WASD sau a săgeților de la tastatură, există și constrângeri ale mișcării astfel, nu se poate alege direcția opusă a direcției curente în care merge șarpele.

Pentru a pune pauză jocului trebuie apăsată tasta ESC, ceea ce oprește actualizarea jocului și activează meniul de pauză.

De asemenea se poate observa și scorul obținut, în josul imaginii.

- Meniul de pauză:

Fig. 14. Interfața meniu pauză

În Fig. 14 este prezentată interfața meniului de pauză. După cum se observă, sunt prezentate 3 butoane: „Continue”, „Restart” și „Menu”. Butonul „Continue” închide meniul de pauză și reia actualizarea jocului. Butonul „Restart” închide complet jocul și pornește o rulare nouă iar butonul „Menu” oprește jocul complet și returnează aplicația în meniul principal. Selectarea și activarea butoanelor se face exact ca în meniul principal.

- Meniul de game over:

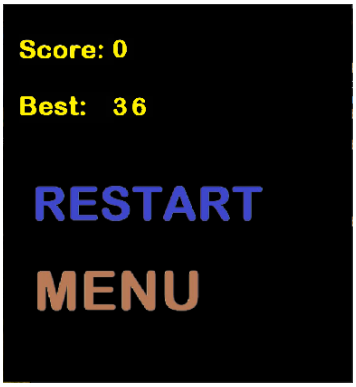

Fig. 15. Interfața meniu game over

În meniul de game over este afișat scorul obținut și cel mai bun scor obținut, iar dedesubt sunt prezentate două butoane: „Restart” și „Menu”. Selectarea și activarea este aceiași ca în meniul principal, iar efectul acestor butoane sunt prezentate în meniul de pauză.

CAPITOLUL V

5. CONCLUZII

Modul abordat pentru rezolvarea problemei este foarte simplă, însă s-ar fi putut implementa mult mai eficient, prin utilizarea moștenirii claselor (s-ar fi putut crea o clasă „Drawable” care) să fie moștenită de clasele „Fruit”, „Map” și „Snake”.

Proiectul poate fi continuat prin adăugarea unui „Scoreboard” care să salveze cele mai bune scoruri pentru diferiți utilizatori, creând astfel o competiție. De asemenea s-ar putea adăuga diferite moduri de joc cum ar fi diferite dimensiuni ale mapei, diferite obstacole pe mapă, diferite efecte ale fructelor (mărește viteza șarpelui sau scade dimensiunea șarpelui, dezorientează sensul în care merge șarpele, etc.), făcând jocul mai interesant.