**Universitatea "Dunărea de Jos" din Galaţi**  
**Facultatea de Automatică, Calculatoare, Inginerie Electrică şi Electronică**

Specializarea Calculatoare

Proiect Ingineria Programelor

Coordonator: Student:

Costache Sabina Axinte Georgiana Madalina

22C31

2024

Maze Collapse 1

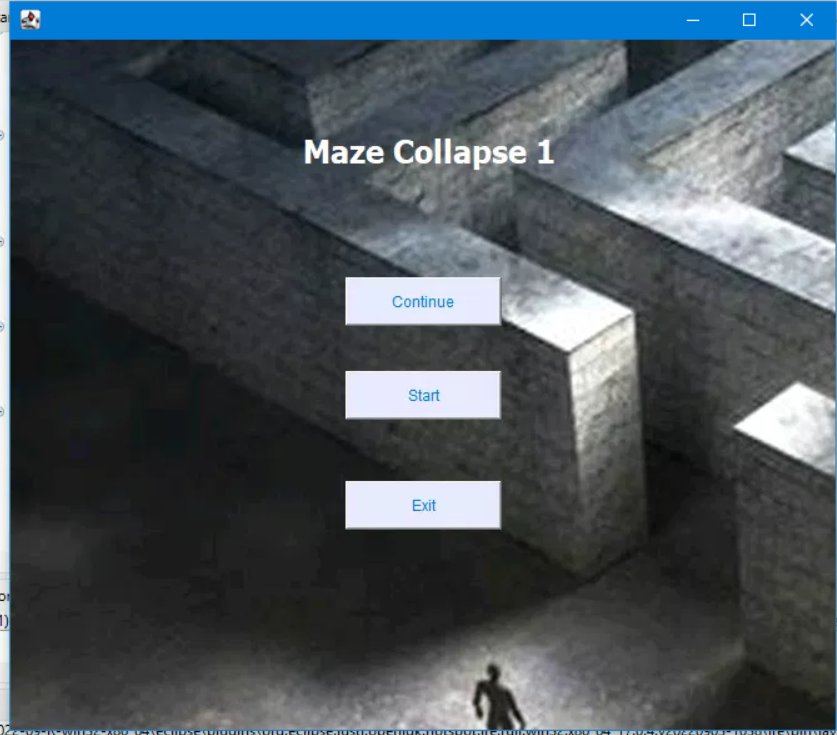
Cuprins

1. Tema proiectului
2. Euristici si principia
3. Sabloane
4. Calculul indicelui de stabilitate
5. Arhitectura sistemului
6. Plan de testare

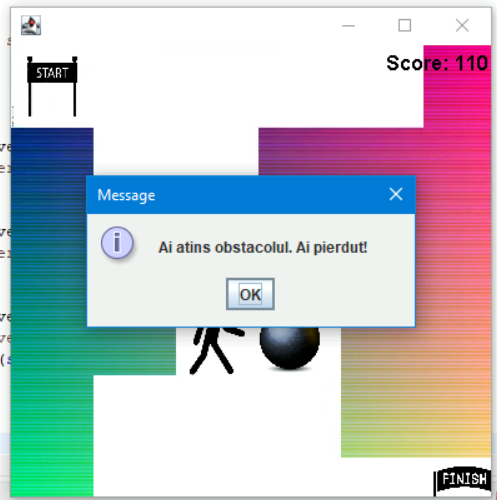
Tema proiectului

Tema proiectului este realizarea unui joc numit Maze Collapse 1 utilizând minim cinci şabloane. Jocul trebuie să fie cât mai apropiat <https://www.mathplayground.com/logic_maze_collapse_1.html>

In acest joc trebuie sa parcurgeti un labirint si sa evitati anumite obstacole. Jucatorul nu va putea sa se intoarca din drum altfel va pierde.



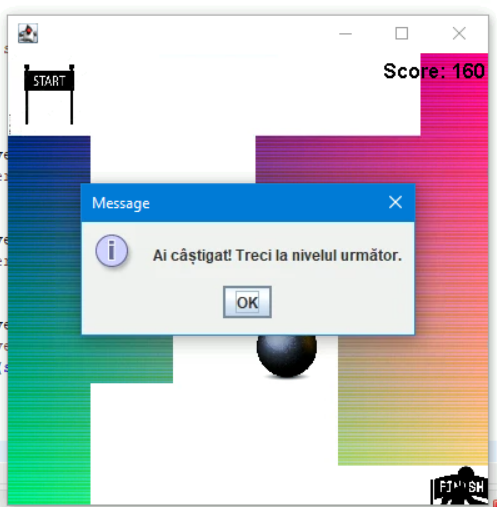
Interfata de Start



Cand jucatorul atinge obstacolul primeste un mesaj in care este avertizat.



Jucatorul cand vrea sa revina pe o pozitie anterioara sau se abate de la labirintul predefinit primeste un mesaj.



Cand ajunge la finish primeste un mesaj de confirmare si trece la nivelul urmator.

Euristici si principii

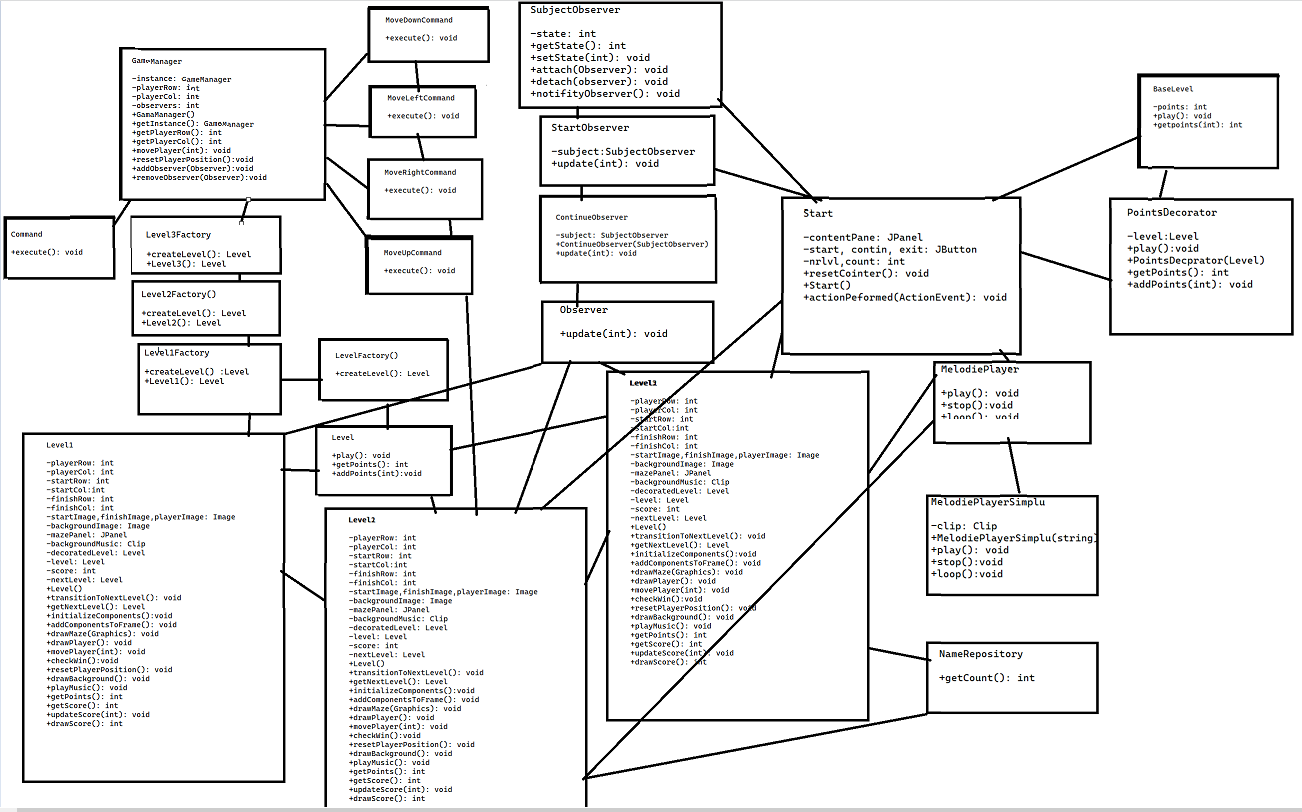
**Principiul deschis și închis** (OCP)

Acest principiu este descris după cum urmează: entitățile software (clase, module, funcții etc.) ar trebui să fie deschise pentru extindere, dar închise pentru modificare . Aceasta înseamnă că ar trebui să fie posibil să se schimbe comportamentul extern al unei clase fără a face modificări codului existent al clasei. Conform acestui principiu, clasele sunt concepute astfel încât ajustarea unei clase pentru a se potrivi unor condiții specifice necesită pur și simplu extinderea acesteia și suprascrierea unor funcții. Aceasta înseamnă că sistemul trebuie să fie flexibil, capabil să funcționeze în condiții schimbătoare fără a schimba codul sursă.

**Polimorfismul** este dat prin comportamentul promis in intefata care este functional.

**Imobilitatea** se prezinta prin codul care nu este amestecat si incalcit.

**Aplicarea legii Demetrei** intr-o forma slaba prin folosirea lui this, super, variabile globale.



Evidentierea sabloanelor utilizate

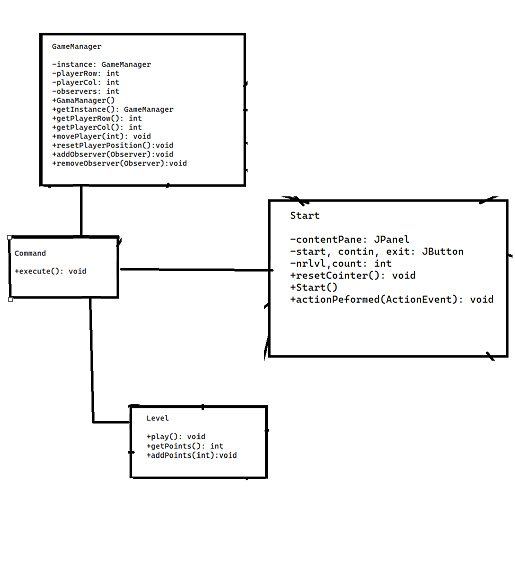
* Singleton

Un şablon creaţional de clasă foloseşte moştenirea pentru a varia clasa care este instanţiată, iar un şablon creaţional de obiect foloseşte un alt obiect pentru a realiza instanţierea.

Şablonul asigură faptul că o clasă are o singură instanţă şi că este oferit un punct global de acces la această instanţă unică.

Pattern-ul Singleton este cuprins în categoria şabloanelor creaţionale, deşi pare mai mult un şablon noncreaţional. Există multe situaţii în programare cănd este necesar ca o clasă să aibă o instanţă şi numai una: un singur manager de ferestre, un singur sistem de fişiere, sau un singur punct de acces la o bază de date.

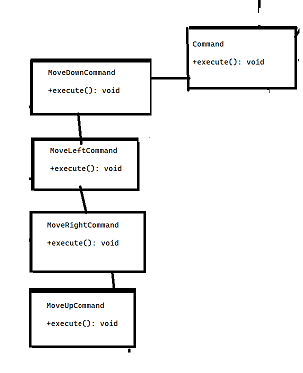
Am folosit acest sablon in scopul generarii unei singure instante a clasei GaemManager la apasarea butonului de Start. In UML apar si clasele Command deoarece prin intermediul lor dau functionalitate butonului de Start si-mi creez instanta de GameManager.



* Command

Şablonul comportamental se ocupă cu algoritmi şi cu atribuirea responsabilităţilor între orice obiecte. Pe lângă şabloanele de obiecte sau de clase, acestea descriu şi şabloanele de comunicare între clase/obiecte. Aceste şabloane ajută la concentrarea mai mult asupra modului în care sunt interconectate obiectele şi mai puţin asupra fluxului complex de control dificil de umărit la execuţie. Şablonul Command permite obiectelor din seturile de instrumente sa genereze cereri către obiecte de aplicaţie neprecizate, prin transformarea cererii însăşi într-un obiect. Acest obiect poate fi stocat şi transmis la fel ca alte obiecte. Cheia acestui şablon este o clasă abstractă Command, care declară o interfaţă pentru executarea operaţiilor. În cea mai simplă formă, această interfaţă include o operaţie abstractă Execute. Subclasele Concrete ale clasei Command precizează o pereche destinatar–acţiune, stocând destinatarul ca o variabilă de instanţă şi implementând operaţia Execute ca să invoce cererea. Destinatarul are cunoştinţele necesare pentru a îndeplini cererea.

Am folosit acest sablon pentru deplasarea jucatorului cu ajutorul claselor MoveUpCommand, MoveDownCommand, MoveLeftCommand, MoveRightCommand.

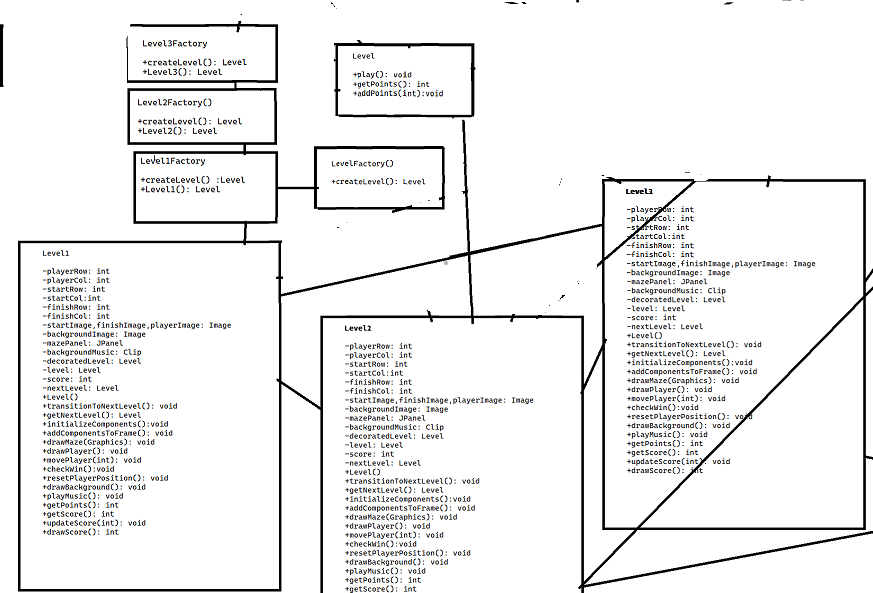


* Factory

Un şablon creaţional de clasă foloseşte moştenirea pentru a varia clasa care este instanţiată, iar un şablon creaţional de obiect foloseşte un alt obiect pentru a realiza instanţierea.

Şablonul Factory este un sablon creational care oferă o interfață pentru creare obiectelor dintr-o superclasă, dar permite subclaselor să modifice tipul de obiecte care vor fi create.

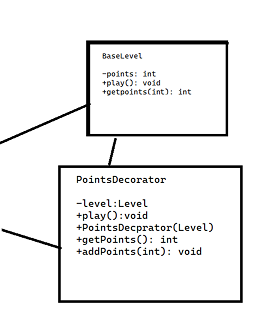
Acest sablon este utilizat in scopul creeri de leveluri in joc.



* Decorator

Un şablon structural care reprezintă o alternativă mai flexibilă faţă de derivarea claselor în scopul extinderii funcţionalităţii. Acest şablon permite ataşarea în mod dinamic de funcţiuni adiţionale unui obiect. Se mai numeşte înfăşurător.

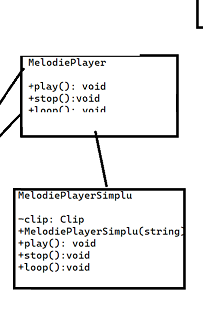
In cadrul proiectului este utilizat pentru afisarea,resetarea si actualizarea scorului.



* Adapter

Sablonul de proiect "Adapter" este un sablon structural care permite interfațelor incompatibile să lucreze împreună. Scopul său este de a oferi o interfață comună între două clase incompatibile. În Java, acest sablon este implementat printr-un adaptor.

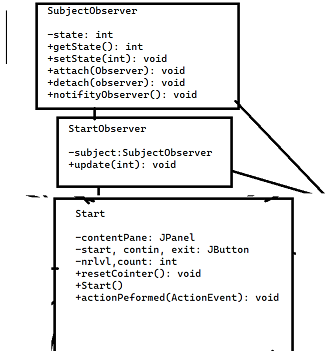
Eu l am folosit pentru integrarea unei melodii pe fundal in cadrul jocului.



* Observer

Sablonul "Observer" este un sablon comportamental care definește o relație unu-la-mulți între obiecte, astfel încât atunci când starea unui obiect se schimbă, toate dependențele sale sunt notificate și actualizate automat. Acesta este utilizat pentru a implementa un mecanism de subscriere-observare în care un obiect (numit subiect sau subiect observabil) menține o listă de obiecte dependente (numite observatori) care trebuie notificate la modificări ale stării subiectului.

In cadrul jocului este folosit pentru actualizarii starii jocului si a pozitiei jucatorului.



Calculul indicelui de stabilitate

⦁ Cuplajul aferent : numarul de clase din afara pachetului care depind de pachet

⦁ Cuplajul eferent : numarul de clase din interiorul pachetul care depind de clase din exteriorul pachetului

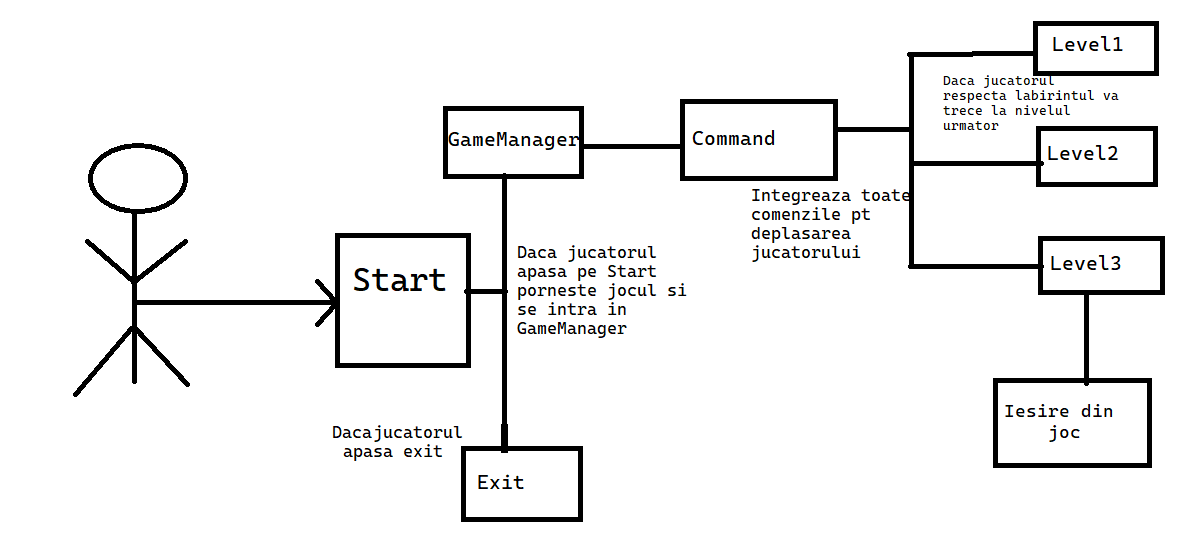
⦁ Factorul de instabilitate: Ce/(Ce+Ca) ;

Pachetul pro:

Ca=0 , Ce= 0

Factorul de instabilitate=0.

Arhitectura Sistemului



Plan de testare

Testarea a fost realizata in Junit.

1.1 Obiectivele Testării:

Asigurarea funcționării corecte a fiecărei funcționalități în cadrul proiectului.

Identificarea și remedierea erorilor.

Validarea performanței și a securității proiectului.

1.2 Strategii de Testare:

Testare unitară pentru fiecare clasă și metodă.

Testare integrată pentru a verifica interacțiunile dintre diferite componente.

Testare de performanță pentru a evalua răspunsul sistemului sub sarcini variate.

Testare de securitate pentru a identifica și remedia potențiale vulnerabilități.

2. Listă cu Cazuri de Test:

2.1 Testare Unitară:

Caz de Test 1: Verificarea corectitudinii funcției play() în cadrul clasei BaseLevel.

Justificare: Această metodă conține logica principală a jocului.

Caz de Test 2: Validarea funcționalității decoratorului PointsDecorator.

Justificare: Asigurarea că punctele bonus sunt adăugate corect la nivel.

2.2 Testare Integrată:

Caz de Test 3: Verificarea interacțiunilor dintre niveluri (transiția de la un nivel la altul).

Justificare: Asigurarea unei tranziții fără probleme între niveluri.

2.3 Testare de Performanță:

Caz de Test 4: Evaluarea timpului de răspuns al jocului sub o sarcină intensivă.

Justificare: Verificarea dacă jocul rămâne responsiv sub sarcini intense.

2.4 Testare de Securitate:

Caz de Test 5: Identificarea și remedierea potențialelor vulnerabilități de securitate.

Justificare: Asigurarea că jocul nu este expus la atacuri sau exploatarea deficiențelor de securitate.

În concluzie, după rularea suitei de teste putem garanta funcționarea corectă a tuturor componentelor solicitate.