|  |  |
| --- | --- |
| https://aciee.ugal.ro/images/logo.png | **FACULTATEA DE AUTOMATICĂ, CALCULATOARE, INGINERIE ELECTRICĂ SI ELECTRONICĂ** UNIVERSITATEA ”DUNĂREA DE JOS” DIN GALAŢI Str. Științei nr. 2, Tel.: **+40 336 130 236**  800146 - Galaţi, România Fax: **+40 236 470 905**  [*https://aciee.ugal.ro*](https://aciee.ugal.ro) |

**Str. Ştiinţei nr. 2, Galaţi - 800146, România**

**UNIVERSITATEA DUNĂREA DE JOS GALAŢI**

**FACULTATEA DE AUTOMATICA, CALCULATOARE, INGINERIE ELECTRICĂ ȘI ELECTRONICĂ**

**SPECIALIZAREA: CALCULATOARE ȘI TEHNOLOGIA INFORMAȚIEI**

Realizarea jocului Mazze Collapse

utilizand Design Patterns

Indrumator: s.l. Sabina Costache

Disciplina: Baze de Date II

Student: Dudu Florin grupa:22C31-A

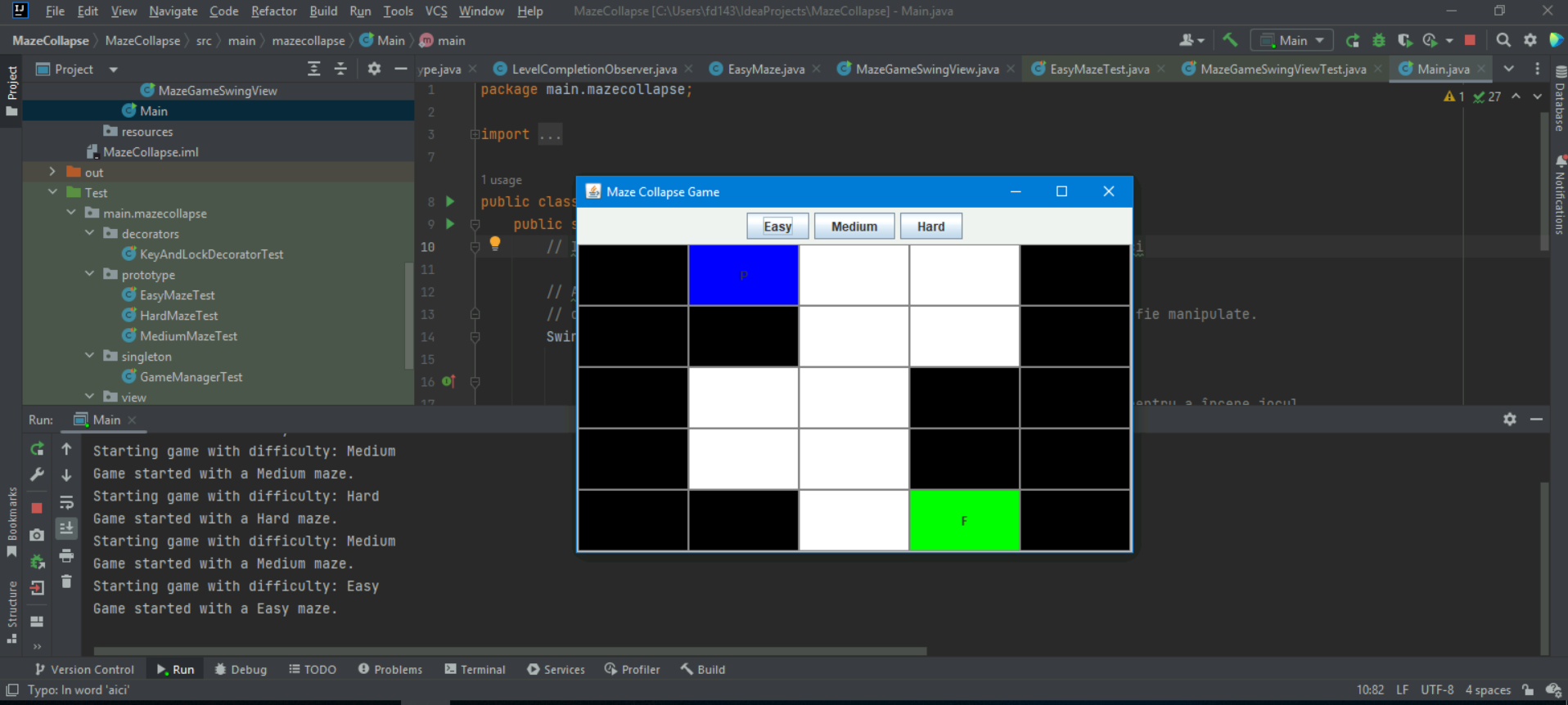
1. **Introducere**

În acest proiect, am abordat provocarea de a recrea și extinde jocul „Maze Collapse” utilizând diverse Șabloane de Proiectare (Design Patterns). Scopul principal a fost de a explora și a implementa principii robuste de design software pentru a construi un joc care nu doar că urmează logica originală, dar și îmbunătățește experiența utilizatorului și facilitață extensibilitatea și întreținerea codului.

Jocul „Maze Collapse” reprezinta un puzzle intrigant bazat pe navigarea printr-un labirint. Scopul jucătorului este de a găsi calea către ieșire, depășind diferite obstacole și capcane. Pentru a ilustra diversitatea și complexitatea jocului, am implementat trei nivele de dificultate: Easy, Medium și Hard. Fiecare nivel aduce un nou set de provocări și mecanici de joc, crescând gradul de dificultate și oferind o experiență unică pentru utilizatori.

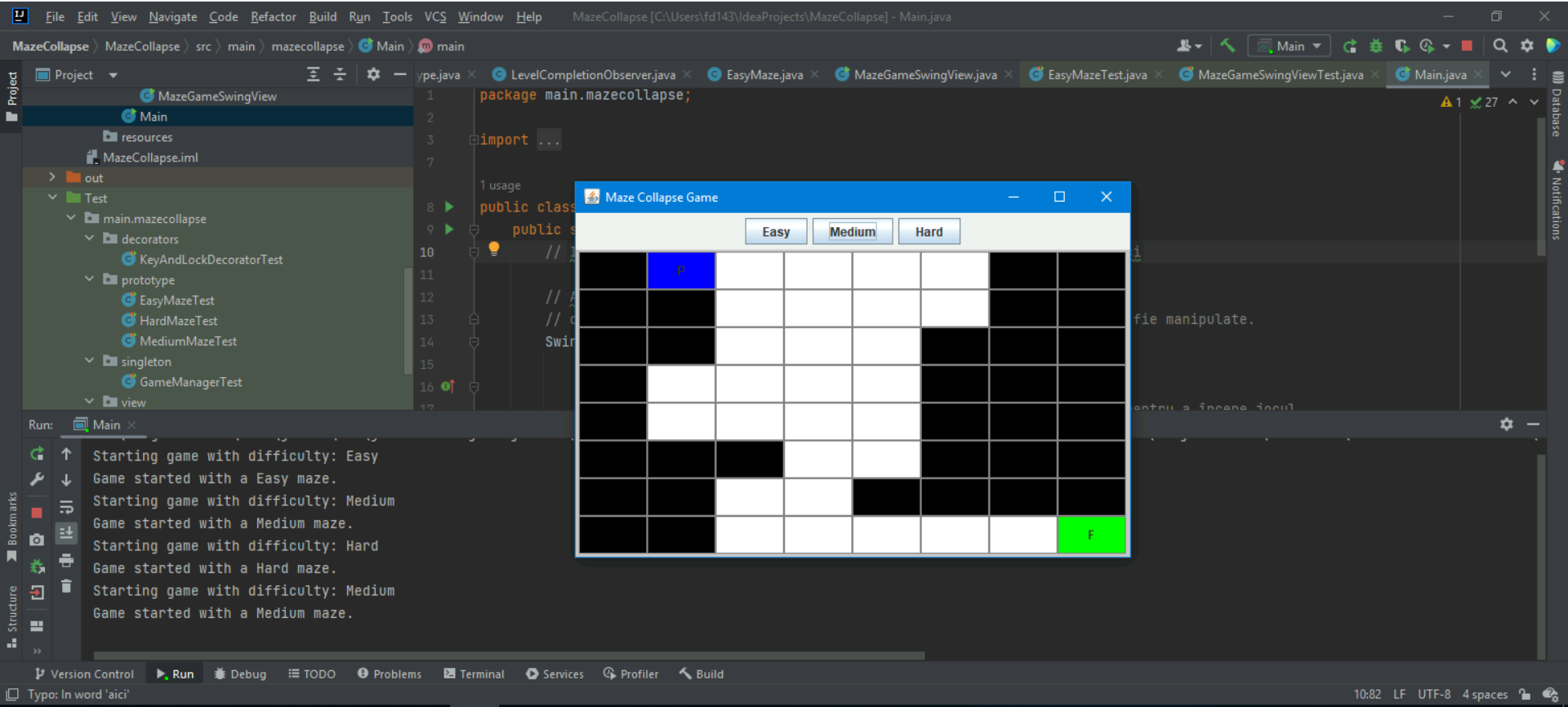
În continuare voi prezenta nivelurile de dificultate redate in cadrul aplicatiei mele:

* Nivelul Easy - Oferă o introducere prietenoasă în mecanicile de bază ale jocului, cu un labirint simplu și direct, ideal pentru începători sau pentru a se obișnui cu controalele jocului.



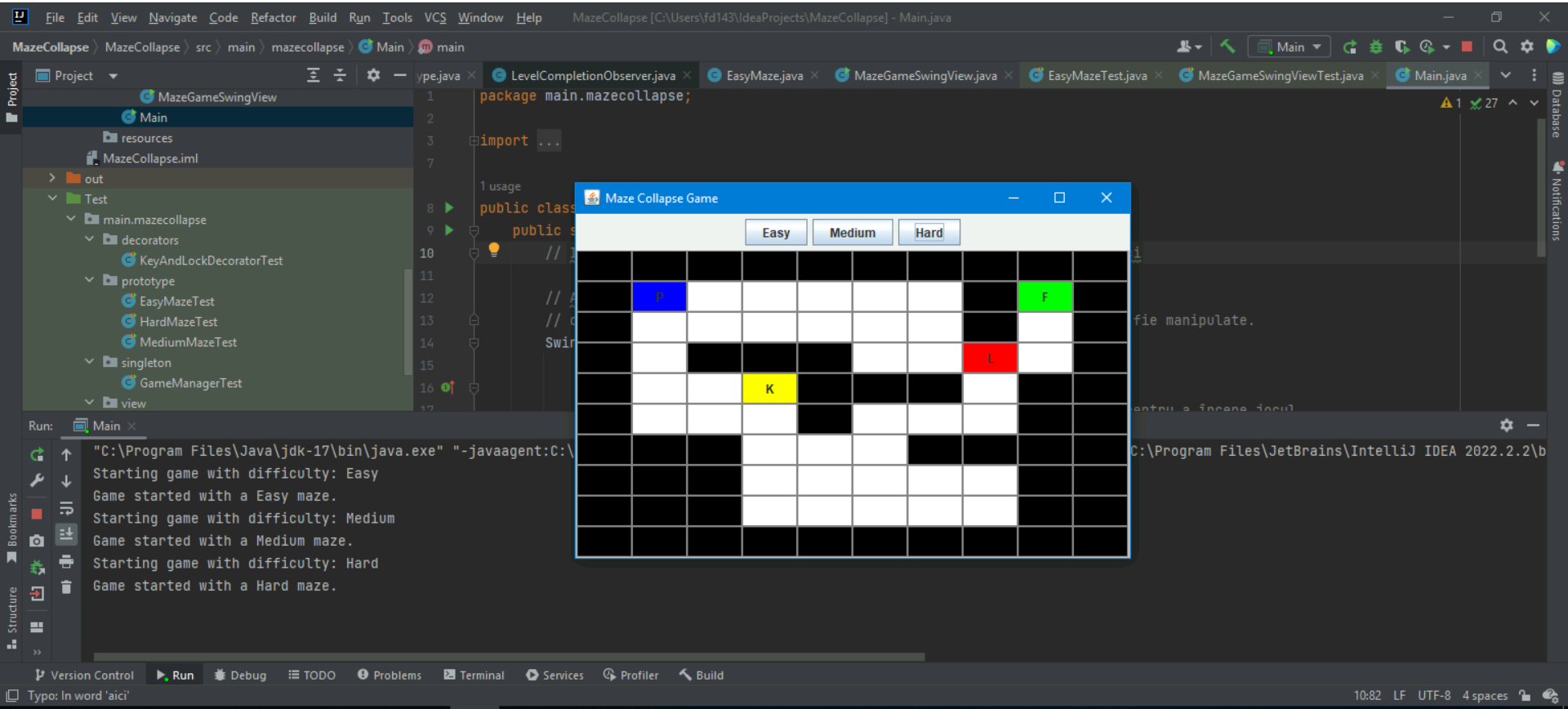
**Fig.1.1.** *Reprezentarea nivelului Easy Maze Collapse*

* Nivelul Medium - Ridică miza cu un labirint mai complex și posibil obstacole sau capcane suplimentare, punând la încercare abilitățile de navigare și gândire strategică ale jucătorilor.



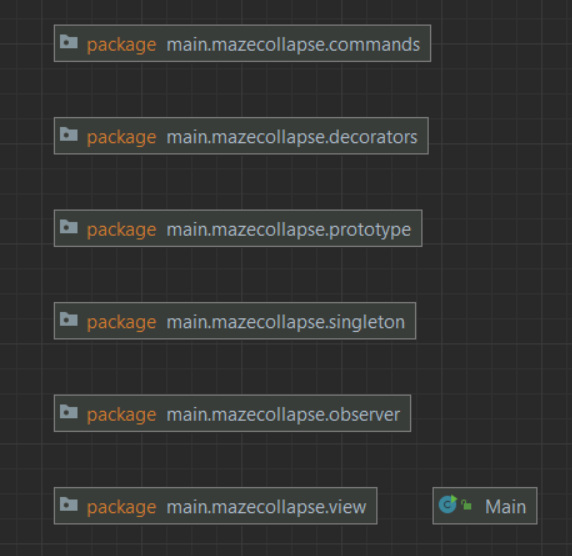
**Fig.1.2.** *Reprezentarea nivelului Medium MazzeCollapse*

* Nivelul Hard - Reprezintă o provocare adevărată, cu un labirint extins și plin de capcane, chei și lacăte, ce necesită gândire critică avansată și abilități de rezolvare a problemelor.

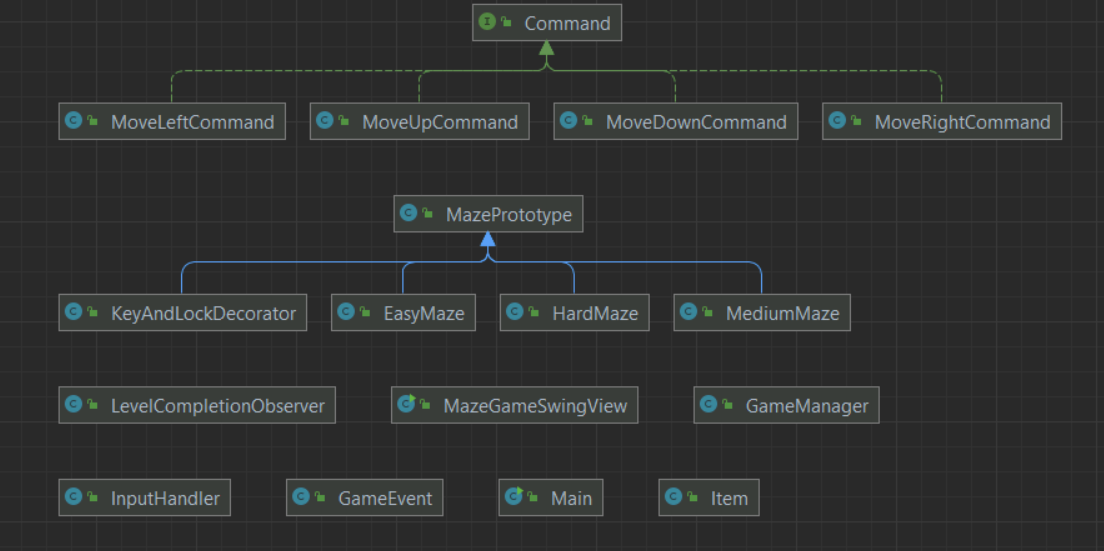


**Fig.1.3.** *Reprezentarea nivelului Hard MazzeCollapse*

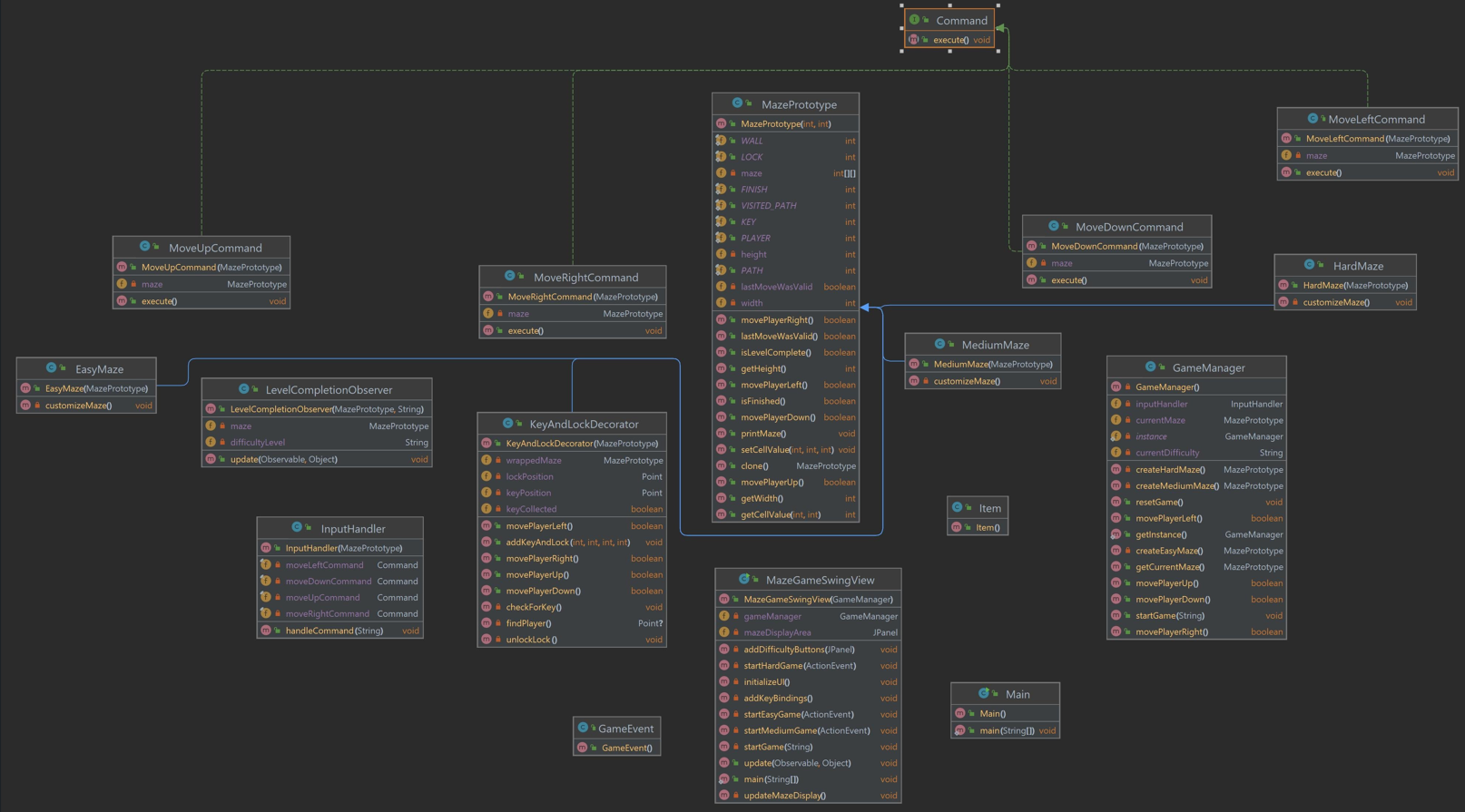
1. **Diagrama de clase şi de pachete, cu evidenţierea separată a şabloanelor, principiilor, şi euristicilor folosite**



**Fig.2.1.** *Reprezentarea pachetelor jocului a jocului Maze Collapse*



**Fig.2.2.** *Reprezentarea diagramei de clase a jocului Maze Collapse*



**Fig.2.3.** *Reprezentarea diagramei de clase si a campurilor acestora*

1. **Calculul indicelui de stabilitate pentru un pachet** **(sau o clasă)**

In programare calculul indicelui de stabilitate se referă în general la măsura în care acesta este susceptibil la schimbări în timpul dezvoltării software. Un pachet stabil este unul care are multe dependențe către el, dar depinde de foarte puține alte pachete.

Conceptul se bazează pe principiul că pachetele care sunt mai des utilizate nu ar trebui să fie schimbate frecvent, deoarece astfel de schimbări ar avea un impact larg asupra întregii aplicații.

Indicele de stabilitate (I) se poate calcula folosind formula:

unde:

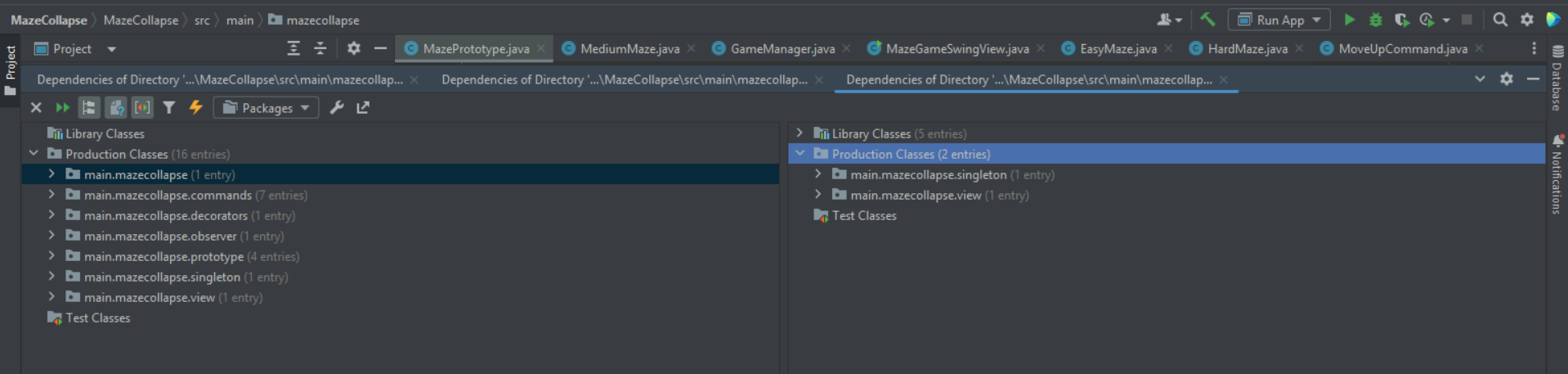
* Ce -> reprezinta numărul de clase din alte pachete care depind de clasele din pachetul analizat (efferent couplings).
* Ca -> reprezinta numărul de clase din pachetul analizat care depind de clase din alte pachete (afferent couplings).

Indicele de stabilitate variază între 0 și 1, unde:

* 0 înseamnă că pachetul este complet instabil (nu are dependențe către el).
* 1 înseamnă că pachetul este complet stabil (nu depinde de alte pachete).



**Fig.3.1.** *Reprezentarea dependentelor claselor*

**Fig.3.2.** *Analiza dependentelor pachetelor Intrari si Iesiri*

In urma analizei intrarilor si iesirilor specific fiecarui pachet, rezultatele calculelor pentru indicile de stabilitate sunt dupa cum urmeaza:

* Pentru pachetul Commands avem:

Ce=1

Ca=6

* Pentru pachetul Observer avem:

Ce=1

Ca=1

* Pentru pachetul Decorator

Ce=1

Ca=1

* Pentru pachetul Prototype:

Ce=1

Ca=4

* Pentru pachetul Singleton:

Ce=1

Ca=7

* Pentru pachetul View:

Ce=1

Ca=2

1. **Utilizarea sabloanelor de proiectare**

Pentru realizarea jocului Maze Collapse am stabilit inca de la inceput ca voi utiliza urmatoarele Sabloane de proiectare:

* 1. **Singleton (Sablon de tip creational)**

Rolul acestuia este de a se asigura că o clasă are o singură instanță și oferă un punct global de acces la aceasta.

În cadrul proiectului meu, clasa GameManager reprezinta un exemplu de Singleton, aceasta fiind folosită pentru a gestiona starea globală a jocului, inclusiv începerea și resetarea jocului, păstrând o singură instanță activă pe parcursul rulării.

public class GameManager extends Observable {

private static GameManager instance;

private GameManager() {}

public static GameManager getInstance() {

if (instance == null) {

instance = new GameManager();

}

return instance;

}

}

* 1. **Command (Sablon de tip comportamental)**

Acesta permite encapsularea unei cereri ca un obiect, permițând astfel parametrizarea clienților cu diferite cereri, cozi sau operații și suport pentru operații anulabile.

În pachetul commands, clasele MoveUpCommand, MoveDownCommand, MoveLeftCommand și MoveRightCommand implementează interfața Command și sunt utilizate pentru a abstractiza acțiunile de deplasare ale jucătorului, permițând o mai bună flexibilitate și extensibilitate.

public class InputHandler {

private final Command moveRightCommand;

private final Command moveLeftCommand;

// alte comenzi...

public void handleCommand(String input) {

if ("MOVE\_RIGHT".equalsIgnoreCase(input)) {

moveRightCommand.execute();

}

// alte condiții...

}

}

Clasa InputHandler utilizează șablonul Command pentru a abstractiza acțiunile de deplasare. Comenzile concrete sunt encapsulate și pot fi executate dinamic

* 1. **Observer (Sablon de tip comportamental)**

Rolul acestui sablon consta in definirea unei dependențe între obiecte astfel încât atunci când unul se schimbă, starea sa este notificată și actualizată automat în toate obiectele dependente.

In cadrul proiectului meu clasa LevelCompletionObserver este un observer care așteaptă să fie notificat despre completarea unui nivel. Acesta este legat de MazePrototype, care atunci când se schimbă (de exemplu, când nivelul este completat), notifică LevelCompletionObserver.

public class LevelCompletionObserver implements Observer {

@Override

public void update(Observable o, Object arg) {

if (o instanceof MazePrototype) {

// Logica de răspuns la schimbare...

}

}

}

LevelCompletionObserver implementează interfața Observer și răspunde la schimbările din MazePrototype, exemplificând utilizarea șablonului Observer.

* 1. **Decorator (Sablon de tip structural)**

Acest sablon de proiectare adaugă dinamic noi funcționalități unui obiect fără a altera structura sa. Este o alternativă flexibilă la subclasarea pentru extinderea funcționalităților.

In cadrul proiectului meu clasa KeyAndLockDecorator extinde comportamentul unui MazePrototype adăugând conceptul de chei și lacăte, fără a modifica clasa originală MazePrototype.

public class KeyAndLockDecorator extends MazePrototype {

private MazePrototype wrappedMaze;

public KeyAndLockDecorator(MazePrototype wrappedMaze) {

this.wrappedMaze = wrappedMaze;

// alte inițializări...

}

@Override

public boolean movePlayerRight() {

// Decorarea comportamentului...

return wrappedMaze.movePlayerRight();

}

// alte metode suprascrise...

}

KeyAndLockDecorator extinde funcționalitatea unui MazePrototype, adăugând conceptul de chei și lacăte, fără a modifica clasa originală.

* 1. **Prototype (Sablon de tip creational)**

Rolul acestui sablon de proiectare consta in specificarea tipurilor de obiecte de

creat folosind o instanță prototip și creeand noi obiecte prin copierea acestui prototip.

Acest tip de sablon l-am utilizat prin derivarea claselor EasyMaze, MediumMaze și

HardMaze din MazePrototype. Acestea folosesc prototipul pentru a crea labirinte cu diferite niveluri de dificultate, copiind și personalizând configurația inițială.

public class EasyMaze extends MazePrototype {

public EasyMaze(MazePrototype prototype) {

super(prototype.getWidth(), prototype.getHeight());

// Personalizare bazată pe prototip...

}

}

EasyMaze derivă din MazePrototype, folosind șablonul Prototype pentru a crea diverse versiuni ale labirintului, personalizându-l în funcție de nivelul de dificultate.

Fiecare dintre aceste șabloane îndeplinește un rol specific în cadrul proiectului, contribuind la structura, organizarea și flexibilitatea codului.

Implementarea acestor șabloane facilitează menținerea și extinderea proiectului, oferind o separare clară a responsabilităților și permițând modificările într-o manieră controlată și previzibilă.

1. **Evidenţierea arhitecturii sistemului.**

Arhitectura sistemului de joc "Maze Collapse" este structurată pe mai multe nivele, fiecare cu roluri și responsabilități specifice. Iată o evidențiere generală a arhitecturii:

1. **Interfața cu utilizatorul (UI - User Interface)**

Componente: MazeGameSwingView

Rol: Gestionează prezentarea și interacțiunea directă cu utilizatorul. Afișează labirintul și stările jocului, primește input de la utilizator și afișează mesaje relevante.

1. **Logica de joc (Game Logic)**

Componente: GameManager, MazePrototype, EasyMaze, MediumMaze, HardMaze, KeyAndLockDecorator

Rol: Centralizează regulile și starea jocului, gestionează ciclul de viață al jocului (start, reset), și aplică regulile jocului, cum ar fi mișcarea jucătorului și interacțiunea cu chei și lacăte.

1. **Managementul inputurilor (Input Management)**

Componente: InputHandler, Command și clasele concrete ale comenzilor (MoveUpCommand, etc.)

Rol: Abstrage acțiunile utilizatorului și le transformă în comenzi concrete care vor fi executate de logica de joc.

1. **Observerii (Observers)**

Componente: LevelCompletionObserver

Rol: Așteaptă evenimente specifice (cum ar fi completarea unui nivel) și reacționează la aceste schimbări (afișând mesaje sau actualizând UI).

1. **Modelul de date (Data Model)**

Componente: MazePrototype și extinderile/decorațiile acestuia.

Rol: Păstrează structura datelor jocului, cum ar fi reprezentarea labirintului și starea diferitelor celule (perete, cale, cheie, lacăt).

1. **Plan de testare şi lista cu cazuri de test (Rezultatele testelor cu Junit)**
   1. **Planul de Testare pentru EasyMaze:**

Scopul Testării: Verifica corectitudinea inițializării unui labirintului EasyMaze, care este esențială pentru funcționarea corespunzătoare a jocului. Obiectivul este să ma asigur că labirintul este configurat corect cu locațiile de start, de sfârșit și traseul definit în mod adecvat.

Inițializarea corectă a locațiilor de start și de sfârșit: Acesta este un aspect fundamental al gameplay-ului. Fără aceste locații setate corect, jocul nu ar putea funcționa așa cum este intenționat.

Verificarea traseului labirintului: Ma asigur că traseul este creat corect și că jucătorii pot naviga prin el. Traseul trebuie să fie logic și să respecte regulile jocului.

Metode de Testare:

Verificare valorilor celulelor în coordonate specifice: Acest test utilizează assertEquals pentru a verifica dacă celulele individuale ale labirintului sunt setate cu valorile corecte.

1. Date de Intrare:

Labirintul este inițializat cu dimensiuni prestabilite (în acest caz, 5x5) și este configurat în setUp pentru fiecare test.

Condiții de Validare a Rezultatelor:

Se așteaptă ca valorile celulelor la coordonatele specificate să corespundă cu cele definite în metoda customizeMaze a clasei EasyMaze.

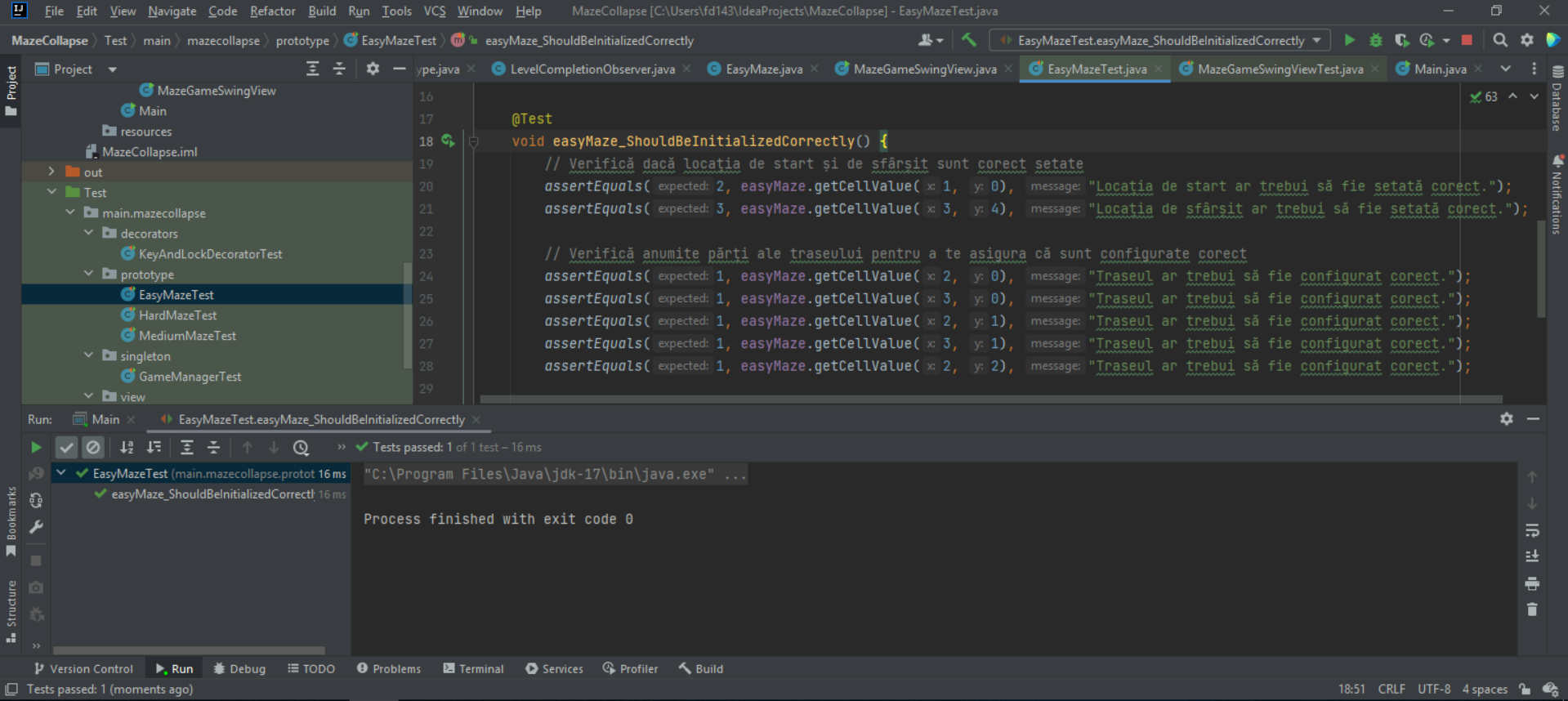
1. Justificare:

Importanța logicii de inițializare: Testarea inițializării corecte a labirintului este critică, deoarece o eroare aici ar putea face jocul imposibil de jucat sau de finalizat. De exemplu, dacă locația de start sau de sfârșit este greșită, jucătorul ar putea începe într-o poziție nevalidă sau nu ar putea termina nivelul.

Asigurarea conformității cu designul jocului: Verificarea traseului labirintului asigură că designul jocului este respectat și că labirintul oferă experiența intenționată. Este important să testăm aceste aspecte pentru a asigura integritatea și coerența gameplay-ului.

1. Rezultatele Testelor cu JUnit:

Testul easyMaze\_ShouldBeInitializedCorrectly a trecut cu succes, indicând că locațiile de start și de sfârșit sunt corect setate și că traseul labirintului este configurat conform așteptărilor. Aceasta confirmă că inițializarea EasyMaze funcționează corect și creează un labirint valid pentru joc.



**Fig.6.1.** Testarea labirintului Easy Maze Junit

* 1. **Planul de testare pentru MediumMaze:**

Scopul Testării:Acest plan de testare este destinat să verifice corectitudinea inițializării și configurării labirintului MediumMaze în jocul MazeCollapse. Testarea se concentrează pe validarea traseului labirintului și a locațiilor critice (start și sfârșit) pentru a asigura o experiență de joc consistentă și provocatoare la nivel mediu.

Metode de Testare:

Verificarea Locațiilor de Start și Sfârșit: Este crucial ca aceste locații să fie setate corect, deoarece poziționarea incorectă poate influența negativ jucabilitatea.

Verificarea Configurării Traseului: Testarea asigură că labirintul este navigabil și îndeplinește cerințele de dificultate medie.

Mod de Realizare:

Utilizarea assertEquals: Această metodă este folosită pentru a verifica dacă celulele labirintului în coordonatele specifice corespund cu valorile așteptate, conform designului jocului.

Date de Intrare:

Labirintul MediumMaze este inițializat cu dimensiuni specifice (8x8) și este configurat în setup pentru fiecare test. Verificăm dacă labirintul respectă cerințele unui nivel de dificultate medie, cu un traseu navigabil și punctele critice (start și sfârșit) corect configurate.

Condiții de Validare a Rezultatelor:

Se așteaptă ca valorile celulelor în coordonatele testate să corespundă cu cele stabilite în metoda customizeMaze a clasei MediumMaze. Verificăm conformitatea traseului și corectitudinea locațiilor de start și de sfârșit.

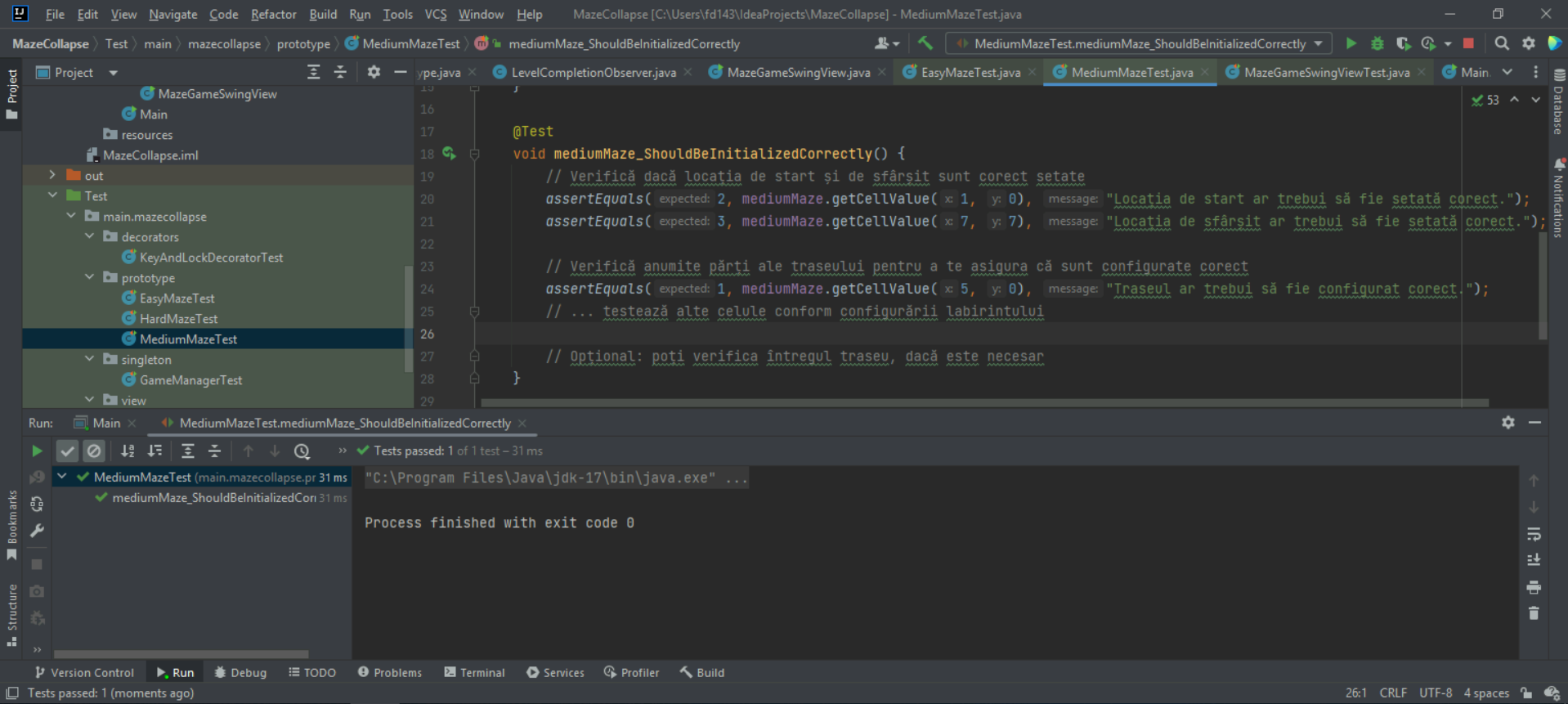
Justificare:

Importanța Inițializării Corecte: O inițializare incorectă a labirintului ar putea afecta negativ experiența jucătorului și funcționalitatea jocului.

Conformitatea cu Designul de Joc: Asigurăm că labirintul MediumMaze oferă un nivel de dificultate adecvat și respectă structura și logica jocului.

Rezultatele Testelor cu JUnit:

Testarea MediumMaze indică faptul că labirintul este inițializat corect și respectă cerințele de design pentru un nivel mediu, asigurând astfel o experiență de joc echilibrată și provocatoare.



**Fig.6.2.** *Rezultatul testului pentru clasa Medium Maze*

**6.3. Planul de Testare pentru HardMaze**

Scopul Testării: Acest plan este destinat să verifice corectitudinea inițializării și configurării labirintului HardMaze în jocul MazeCollapse. Testarea are ca scop asigurarea conformității labirintului cu specificațiile unui nivel dificil, verificând locațiile critice și traseul.

Focusul Testelor:Verificarea Configurării Corecte a Labirintului: Asigurarea că locațiile de start și de sfârșit sunt setate corect și că labirintul respectă structura unui nivel dificil.

Testarea Integrității Structurii Labirintului: Verificarea dacă matricea labirintului este configurată corect, fără erori sau incoerențe.

Testarea Cazurilor Limită și a Traseului: Examinarea diferitelor segmente ale labirintului pentru a confirma că sunt configurate în mod adecvat pentru un nivel dificil.

Executarea și Validarea Testelor:

După implementarea și rularea testelor, e important să se evalueze rezultatele pentru a verifica dacă toate condițiile sunt îndeplinite. Eșecul unui test poate indica probleme în inițializarea labirintului sau în designul acestuia.

Metode de Testare:

Utilizarea assertEquals și altor metode JUnit: Aceste metode sunt folosite pentru a verifica valorile celulelor în coordonate specifice și pentru a asigura că structura labirintului corespunde cu cerințele unui nivel dificil.

Date de Intrare:

Prototipul de labirint HardMaze cu dimensiuni prestabilite, utilizat pentru a seta și verifica diferite aspecte ale structurii.

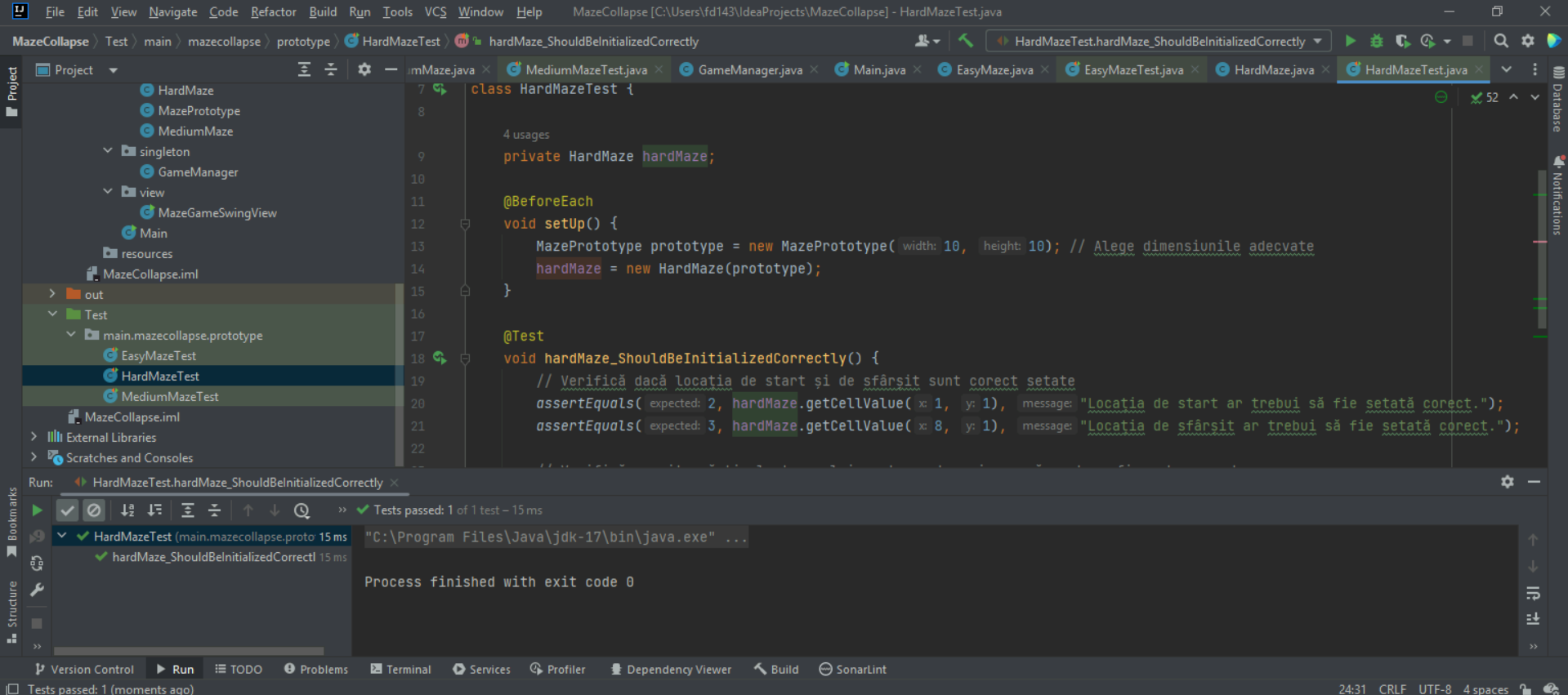
Justificare pentru Alegerea Testului:

Asigurarea Calității Jocului: Testarea detaliată a nivelului dificil contribuie la menținerea standardelor înalte de calitate și la prevenirea erorilor de gameplay.

Verificarea Conformității cu Nivelul Dificil: Este esențial să se asigure că labirintul dificil îndeplinește așteptările de complexitate și provocare.

Rezultatele Testelor cu JUnit:

Rezultatele ar trebui să indice că HardMaze este inițializat și configurat corect, validând astfel că labirintul oferă nivelul de dificultate și provocare așteptat în acest nivel de joc.



**Fig.6.3.** *Realizarea Testelor pentru clasa HardMaze*

* 1. **Plan de Testare pentru Clasa GameManager:**

Scopul Testării: Verifica funcționalitatea și comportamentul corect al clasei GameManager în diferite scenarii de joc. Acest lucru include inițializarea corectă a labirintului în funcție de dificultate, actualizarea corespunzătoare a stării jocului în urma acțiunilor jucătorului și capacitatea de a reseta jocul.

Focusul Testelor: Inițializarea Labirintului: Asigurarea că labirintul corect este inițializat în funcție de dificultatea selectată.

Actualizarea Labirintului: Verificarea dacă mișcările jucătorului sunt reflectate corect în starea labirintului.

Resetarea Jocului: Confirmarea că jocul poate fi resetat și că starea inițială este restaurată corespunzător.

Metode de Testare:

Testul Inițializării Jocului (startGame\_ShouldInitializeCorrectMazeBasedOnDifficulty): Folosește assertTrue pentru a verifica dacă labirintul corespunzător este inițializat la selectarea dificultății.

Testul Actualizării Labirintului (movePlayer\_ShouldUpdateMazeCorrectly):

Verifică prin assertTrue dacă mișcările jucătorului sunt executate corect și dacă labirintul se actualizează corespunzător.

Testul Resetării Jocului (resetGame\_ShouldRestartTheGame):

Utilizează assertNotNull și alte afirmații relevante pentru a evalua dacă jocul este resetat corespunzător.

Justificare:

Importanța Inițializării Corecte: Inițializarea corectă a labirintului este esențială pentru o experiență de joc adecvată. Alegerea labirintului greșit ar putea duce la confuzie sau frustrare.

Verificarea Comportamentului de Mișcare: Mișcările jucătorului sunt o componentă centrală a jocului. Este crucial să se asigure că acestea sunt gestionate corect.

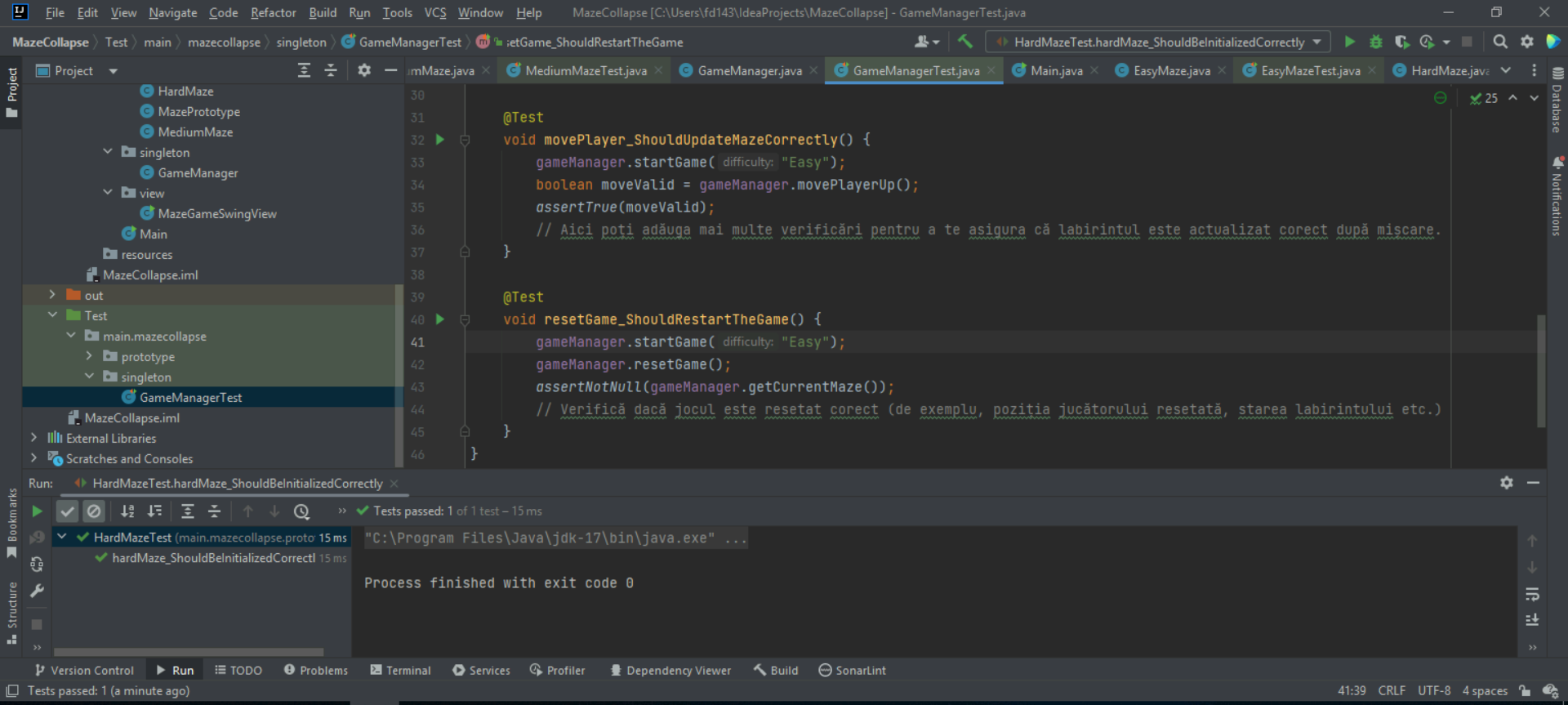
Resetarea Jocului: Capacitatea de a reseta jocul oferă o experiență de utilizare mai bună, permițând jucătorilor să înceapă din nou rapid.

Rezultatele Testelor cu JUnit:

Testul Inițializării Jocului: Trebuie să treacă cu succes, indicând că labirintul corespunzător este selectat pentru fiecare nivel de dificultate.

Testul Actualizării Labirintului: Ar trebui să confirme că mișcările jucătorului sunt gestionate corect.

Testul Resetării Jocului: Ar trebui să valideze că jocul este resetat corect, pregătit pentru o nouă rundă.



**Fig.6.4.** *Realizarea Testelor pentru clasa GameManager*

* 1. **Plan de testare pentru clasa Main**

Scopul Testării:Verifica dacă aplicația se inițializează corect fără a genera excepții. Testul este conceput pentru a asigura că procesul de pornire al jocului MazeCollapse este robust și nu confruntă probleme critice la lansare.

Focusul Testelor:

Inițializarea Aplicației: Testarea se axează pe verificarea faptului că metoda main poate fi executată fără erori sau excepții.

Stabilitatea Sistemului: Confirmarea că aplicația poate începe procesul de rulare fără a întâmpina probleme fundamentale.

Metode de Testare:

Testul Inițializării Aplicației (main\_initializesWithoutExceptions):

Folosește assertDoesNotThrow pentru a verifica dacă aplicația poate fi inițializată fără a arunca excepții.

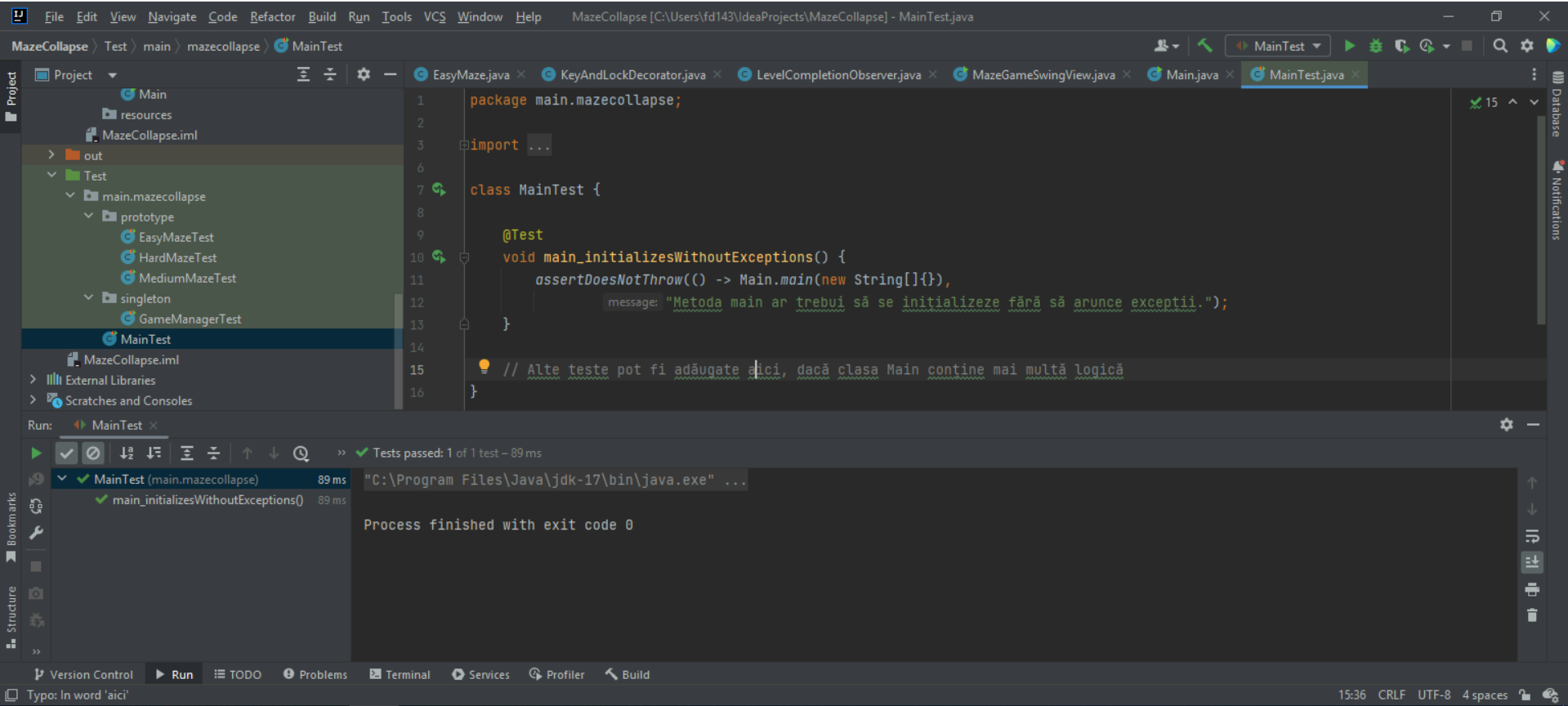
Justificare pentru Alegerea Testelor:

Verificarea Lansării Aplicației: Este esențial să ne asigurăm că aplicația poate începe fără probleme, deoarece acesta este primul pas în experiența utilizatorului.

Testarea Robustetii: Asigurarea că aplicația este stabilă și funcționează corect chiar de la început crește încrederea în calitatea și fiabilitatea acesteia.

Rezultatele Testelor cu JUnit:

Testul Inițializării Aplicației: Ar trebui să treacă cu succes, indicând că metoda main se inițializează corect fără a genera excepții.



**Fig.6.5.** *Realizarea Testelor pentru clasa Main*

* 1. **Planul de testare pentru clasa MazeGameSwingView**

Planul de testare pentru clasa MazeGameSwingView a fost conceput pentru a verifica aspecte esențiale ale interfeței grafice Swing a jocului Maze Collapse. Scopul principal a fost de a asigura funcționalitatea corectă și aspectele vizuale ale UI-ului. Iată planul detaliat:

1. Testarea Inițializării Componentelor UI

Testul mazeDisplayAreaShouldBeInitialized: Verifică dacă zona principală de afișare a labirintului (mazeDisplayArea) este inițializată corect. Aceasta este esențială pentru joc, deoarece afișează labirintul jucătorilor. O inițializare corectă asigură că jucătorii vor vedea labirintul pe ecran.

Testul difficultyButtonsShouldBeInitialized: Verifică dacă butoanele pentru selectarea dificultății jocului ("Easy", "Medium", "Hard") sunt inițializate corespunzător. Aceste butoane sunt cruciale pentru permiterea jucătorilor de a alege nivelul de dificultate al jocului, influențând experiența de joc.

2. Testarea Configurației și Aspectului UI

Testul titleShouldBeSetCorrectly: Verifică dacă titlul ferestrei este setat corect la "Maze Collapse Game". Acest lucru este important pentru identificarea ferestrei jocului de către utilizatori.

Testul mazeDisplayAreaLayoutShouldBeSet: Se asigură că layout-ul pentru mazeDisplayArea este setat corespunzător. Corectitudinea layout-ului este esențială pentru a aranja corespunzător componentele vizuale ale labirintului pe ecran.

Justificare

Aceste teste au fost alese pentru a verifica aspectele critice ale UI-ului care afectează direct experiența utilizatorului și funcționalitatea jocului. Prin aceste teste, ne asigurăm că:

Interfața grafică este inițializată corespunzător și este pregătită pentru interacțiunea cu utilizatorul.

Elementele vizuale cheie, cum ar fi zona de afișare a labirintului și butoanele de dificultate, sunt prezente și funcționale.

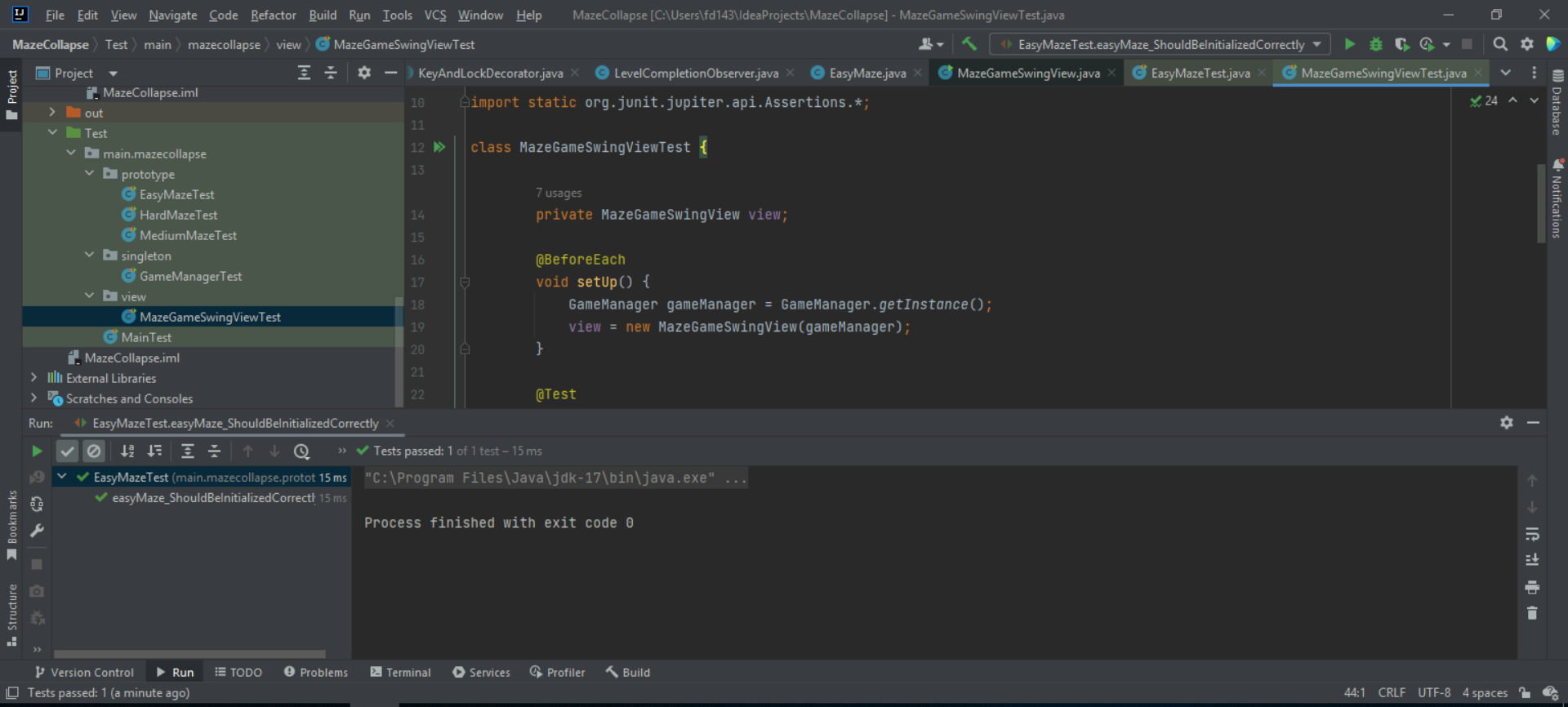
Aspectul general și configurația UI-ului sunt conforme cu așteptările și contribuie la o experiență de joc plăcută și intuitivă.

Rezultatele Testelor cu JUnit

Toate testele de mai sus trec cu succes, indicând că componentele UI sunt inițializate corespunzător și că aspectul UI-ului este setat conform așteptărilor.

Dacă un test eșuează, acest lucru indică o posibilă problemă în inițializarea componentelor UI sau în configurarea aspectului UI, necesitând o investigație și o corecție ulterioară.

Aceste teste formează o bază solidă pentru asigurarea calității interfeței grafice a jocului Maze Collapse. În continuare, se pot adăuga teste mai avansate pentru a acoperi interacțiunile utilizatorului și răspunsurile dinamice ale UI-ului, dar acestea pot necesita tehnici de testare mai complexe sau utilizarea unor biblioteci specializate.



**Fig.6.6.** *Realizarea Testelor pentru clasa MazeGameSwingView*

* 1. **Planul de testare pentru clasa KeyAndLockDecorator**

Planul de testare pentru clasa KeyAndLockDecorator se concentrează pe verificarea funcționalităților esențiale ale decoratorului în contextul jocului Maze Collapse. Acest decorator adaugă elemente suplimentare de joc - o cheie și un lacăt - care influențează navigarea jucătorului în labirint. Iată planul detaliat cu justificările pentru fiecare test:

1. Testarea Plasării Corecte a Cheii și a Lacătului (addKeyAndLockShouldPlaceKeyAndLockCorrectly)

Scop: Verifică dacă metoda addKeyAndLock plasează corect cheia și lacătul în labirint.

Justificare: Această funcționalitate este crucială pentru dinamica jocului, deoarece prezența cheii și a lacătului modifică modul în care jucătorul navighează prin labirint. Testul asigură că aceste elemente sunt plasate în locațiile dorite, ceea ce este fundamental pentru experiența de joc.

2. Testarea Colectării Cheii prin Mișcarea Jucătorului (movingPlayerToKeyShouldCollectIt)

Scop: Verifică dacă mișcarea jucătorului în poziția cheii rezultă în colectarea acesteia.

Justificare: Colectarea cheii este un pas esențial pentru progresul în joc, deoarece permite jucătorului să deblocheze lacătul. Acest test asigură că logica de detectare a colectării cheii funcționează corect și că starea internă a decoratorului este actualizată corespunzător.

3. Testarea Deschiderii Lacătului când Cheia este Colectată (lockShouldUnlockWhenKeyIsCollected)

Scop: Verifică dacă lacătul este deblocat (transformându-se într-o cale) atunci când cheia este colectată.

Justificare: Deblocarea lacătului este un aspect cheie al jocului, permițând jucătorului să progreseze sau să finalizeze nivelul. Acest test se asigură că labirintul răspunde corect la colectarea cheii, schimbând starea celulei lacătului într-o cale navigabilă.

Aceste teste formează o bază solidă pentru asigurarea funcționalității corecte a elementelor suplimentare introduse de KeyAndLockDecorator. Prin aceste teste, ne asigurăm că jocul răspunde corespunzător la acțiunile jucătorului și că elementele suplimentare de joc - cheia și lacătul - funcționează așa cum sunt prevăzute în designul jocului. Testele sunt esențiale pentru a asigura o experiență de joc coerentă și provocatoare pentru utilizatori.



**Fig.6.7.** *Realizarea Testelor pentru clasa KeyAndLockDecorator*

Prin intermediul acestui proiect, am demonstrat cum aplicarea eficientă a șabloanelor de proiectare poate conduce la crearea unui software modular, ușor de întreținut și de extins. Următoarele capitole vor detalia arhitectura sistemului, implementarea șabloanelor de proiectare specifice și modul în care acestea contribuie la funcționalitatea și flexibilitatea jocului „Maze Collapse”.