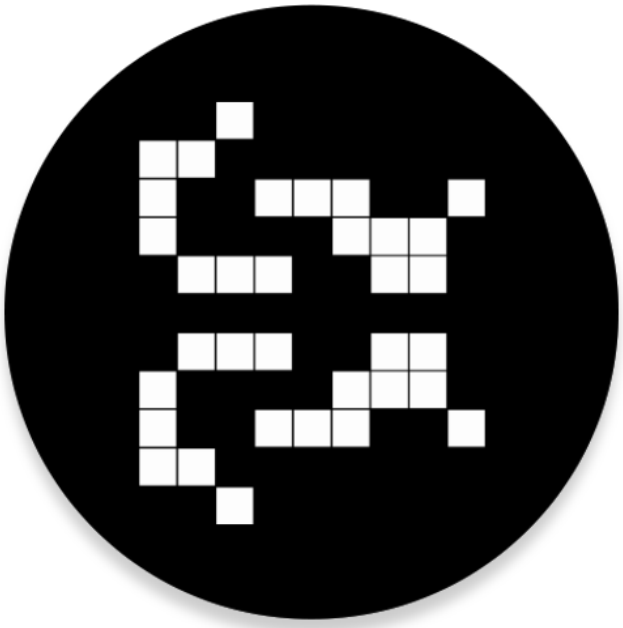
Design Patterns

Conway's Life Game 기능 확장 및 설계 개선



중앙대학교
창의ICT공과대학
소프트웨어학부

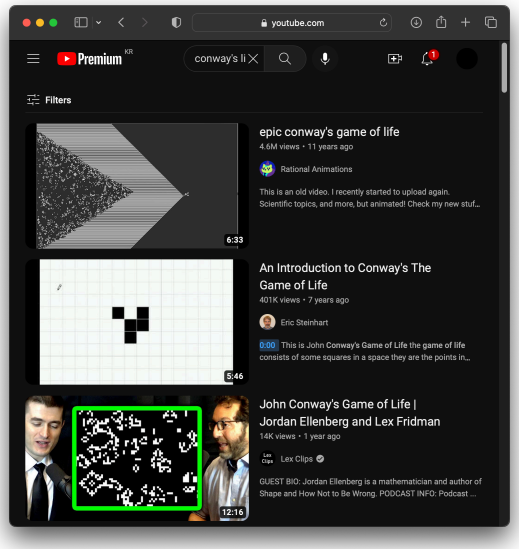
20185659 김혜성
20181267 박서현
20186669 박지수

Contents

Table of contents

1. 요약

1.1. 기능 확장

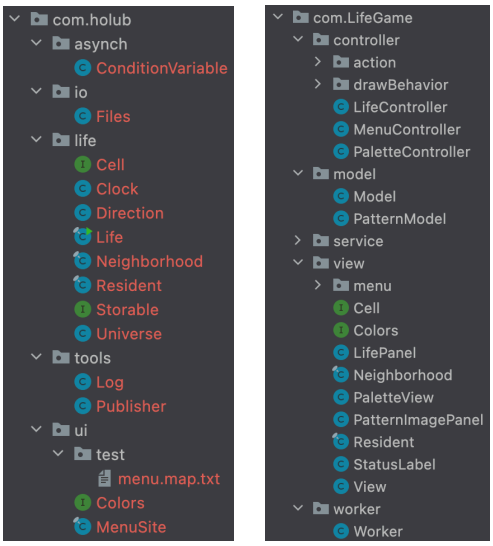


▲ **[picture 01] "Conway's Life Game", YouTube**

[picture 01]과 같이, 기존 Conway's Life Game 프로그램의 흥미 요소를 분석해보았을 때 게임적인 부분이 아닌, cell의 생명 패턴으로 여러 작품들을 만들어내는 예술적인 요소가 가장 크다고 생각했다. 따라서, 새로운 시스템이 아닌 해당 프로그램을 사용하며 느낀 불편함을 해결 할 수 있는 편의 기능을 추가하는 방향으로 기능 확장을 설계하였다.

우선, 작은 크기의 cell을 하나씩 누르며 패턴을 그리는 것이 번거롭고 규모가 큰 패턴을 만들기 어렵다고 느껴졌다. 또한 빠르게 동시다발적으로 cell의 상태가 변화하여 중간 진행 상황을 따라가기 힘들었다. 이러한 불편함을 해결하기 위해 **1) Palette** 기능과 **2) 현재 generation수, live cells** 개수 확인 기능을 구현하여 프로그램을 확장시켰다.

1.2. 설계 개선

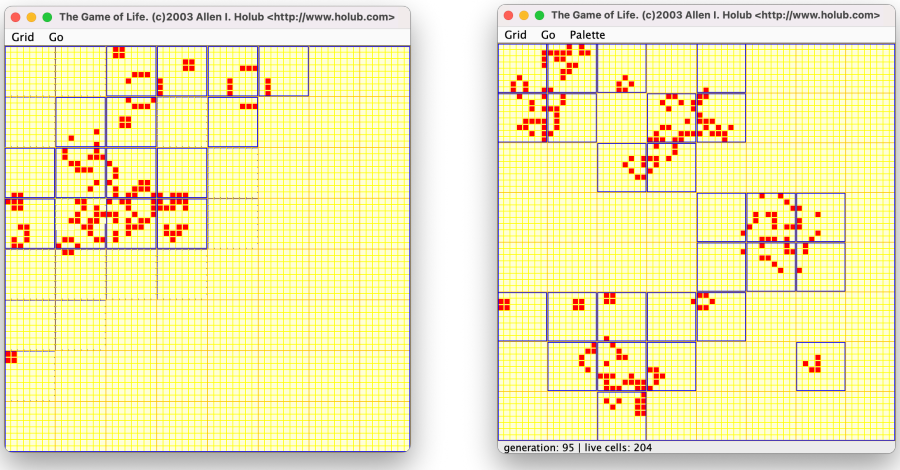


▲ **[picture 02]** 기존 파일 구조(좌), 개선한 파일 구조(우)

기존의 프로그램은 model, view, controller의 구분 없이 view에서 모든 로직을 수행하는 구조를 가지고 있어 여러 책임을 동시에 가져 SRP를 위반하고 있었다. 가장 큰 문제점은 OCP를 위반하여 기능을 확장하고자 하는데 어려움이 있었다. 또한 의존성 주입을 하지 않고 의존관계가 많아 DIP를 위반하고 테스트를 작성하는데 어려움이 많았다.

개선된 설계에서는 Spring Framework를 채택해 UI 컴포넌트들과 컨트롤러를 Spring Bean으로 등록해 **Singleton**으로 구성하고, Spring Container를 이용해 의존성을 주입해 **DIP**를 해소했다. 또 MVC Observer 패턴으로 작성하여 Model-View-Controller의 책임을 분리해 **SRP**를 지키고, View를 Observer로, Model을 Subject로 **Observer Pattern**을 채택하여 **OCP**를 지켰다.

또 Palette 기능을 추가하기 위해 **Strategy Pattern**을 채택하였고, Menu의 OCP를 잘 지키기 위해 **Command Pattern**을 적용하여 세부적으로도 유연한 설계를 완성하였다.

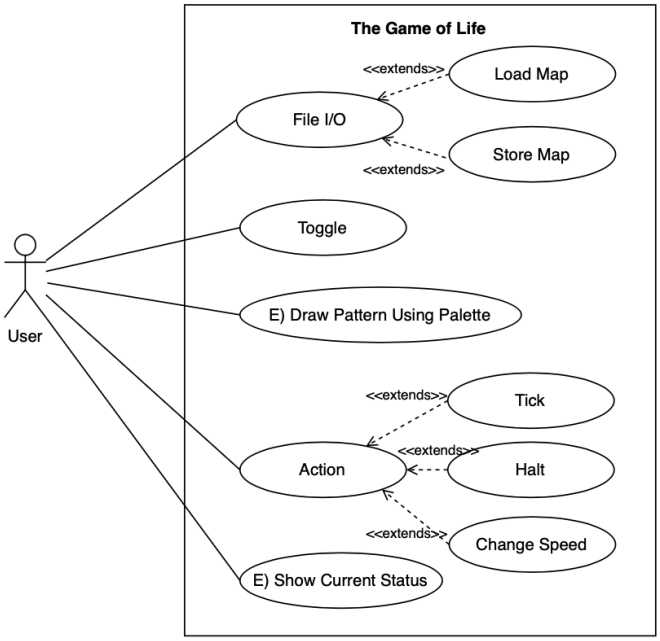


▲ **[picture 03]** 기존 프로그램(좌), 개선된 프로그램(우)

MVC 패턴을 이용한 설계를 적용한 결과, **[picture 03]**과 같이, 원래 프로그램에서는 Neighborhood의 파란색 테두리가 올바르게 그려지지 않았는데, 개선된 설계에서는 깔끔하게 적용된 것을 확인할 수 있었다. 또한 Unit Test를 작성할 때 Mockito를 활용해 쉽게 테스트를 작성할 수 있었고, 새로운 기능을 추가할 때도 OCP를 지켰기 때문에 개발 생산성 면에서도 장점을 보였다.

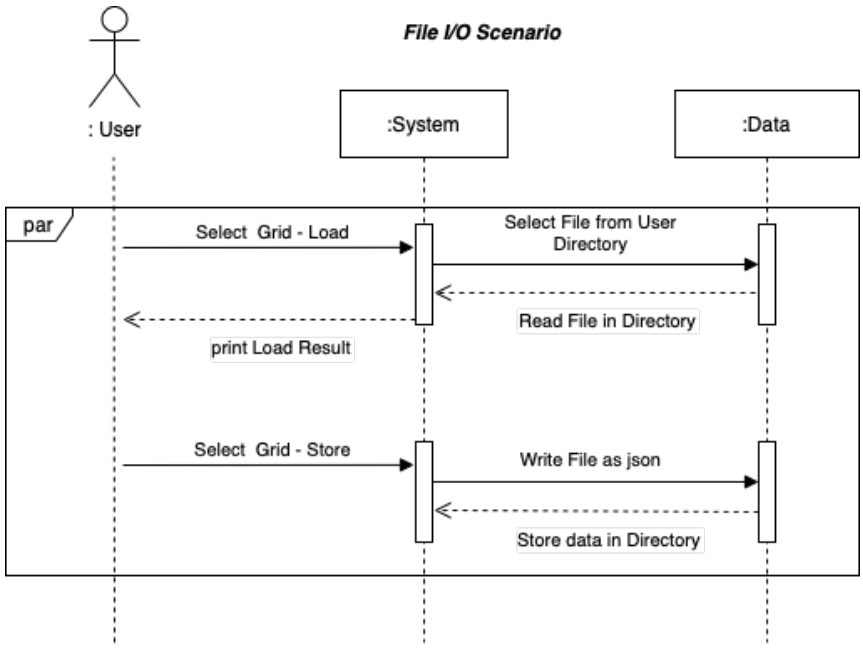
2. 분석

2.1. Use-case Diagram

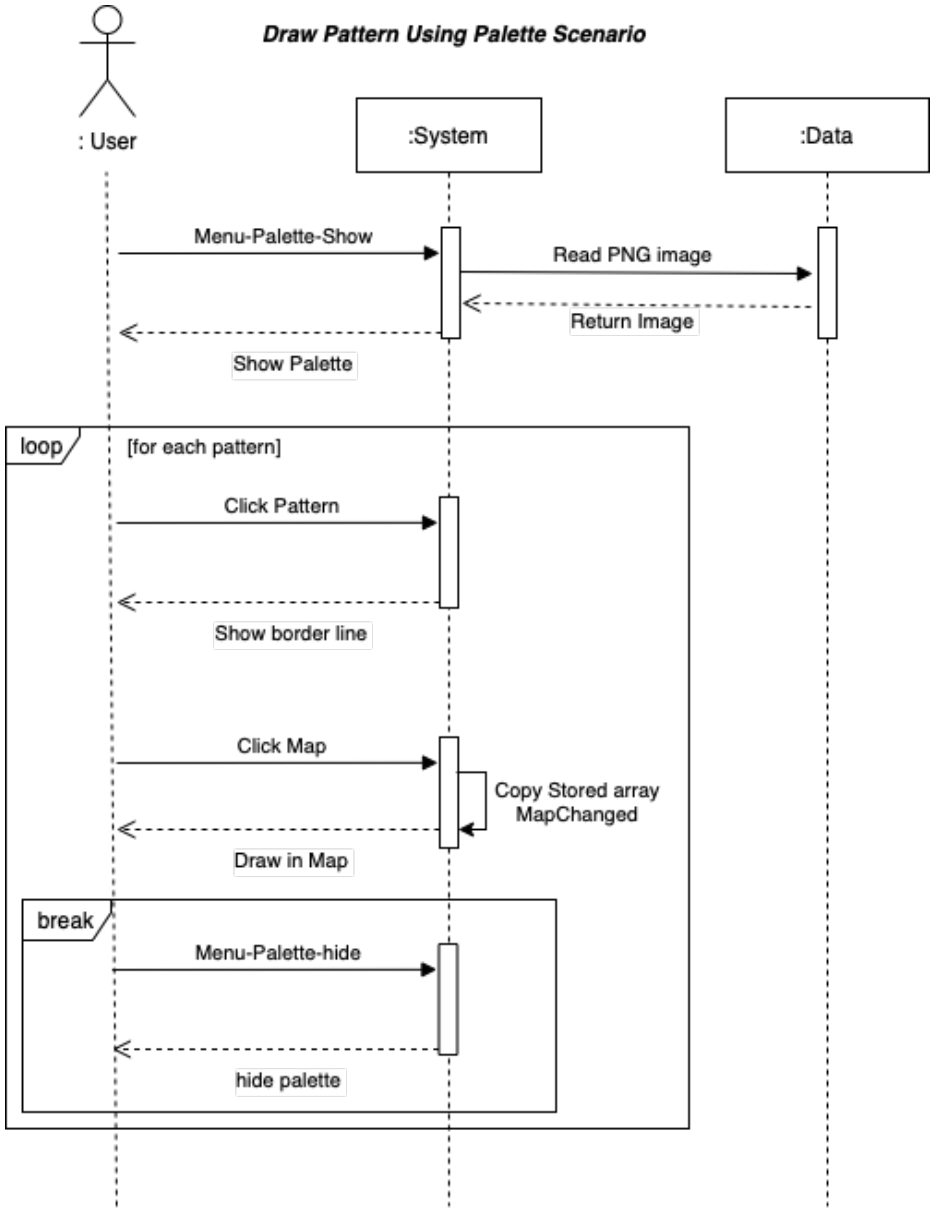


▲ [picture 04] Use Case Diagram

2.2. System Sequence Diagram



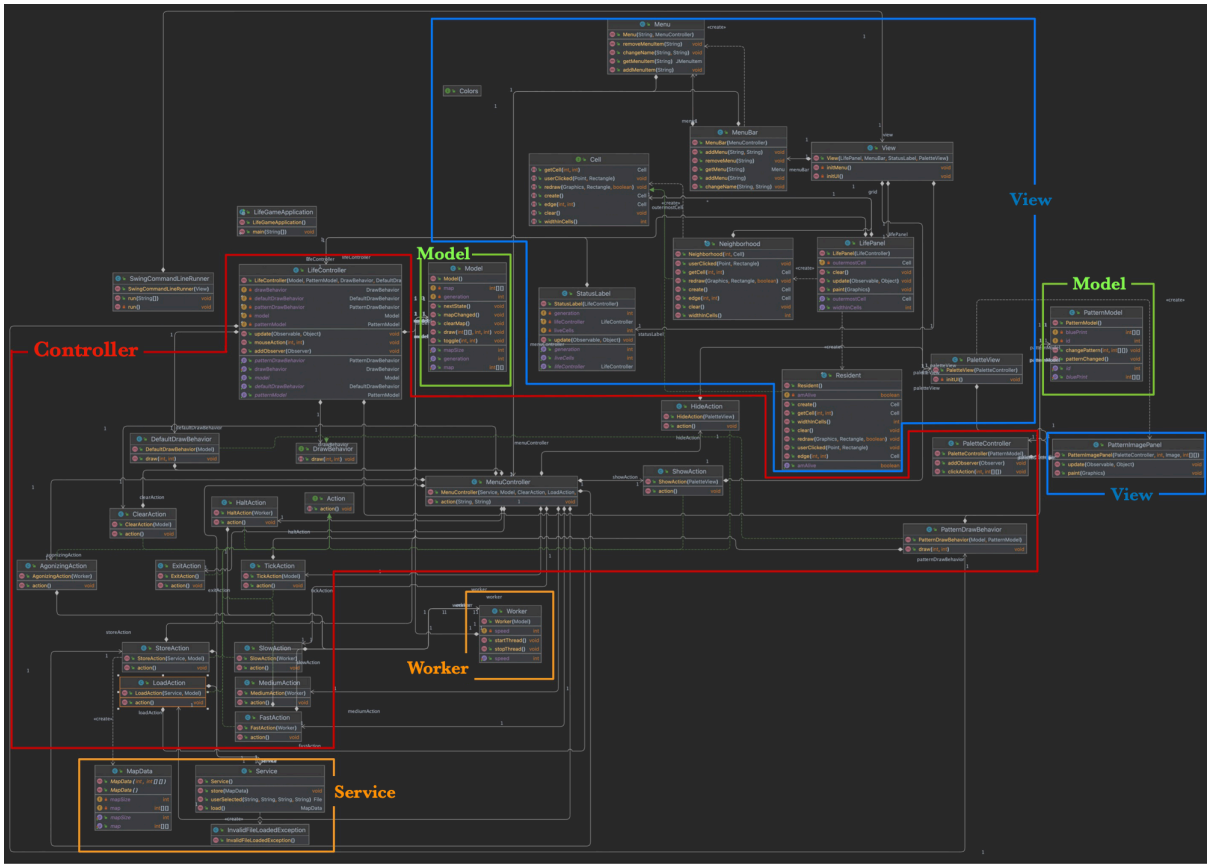
▲ [picture 05] System Sequence Diagram (File I/O Scenario)



▲ [picture 06] System Sequence Diagram (Draw Pattern Using Palette Scenario)

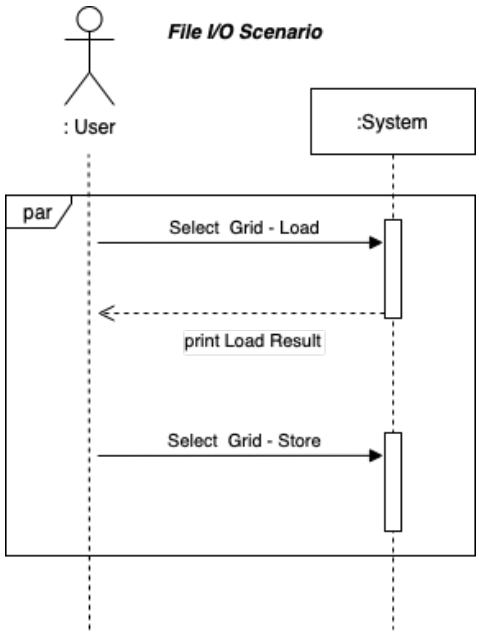
3. 아키텍처

3.1. Design Class Diagram

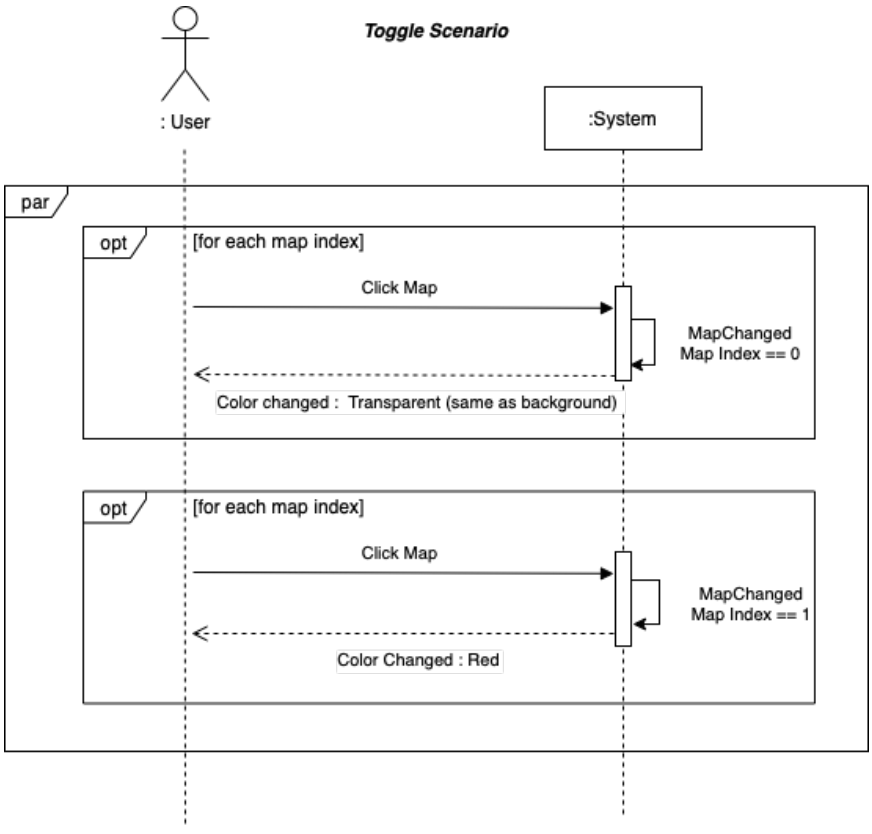


▲ [picture 07] Class Diagram Design

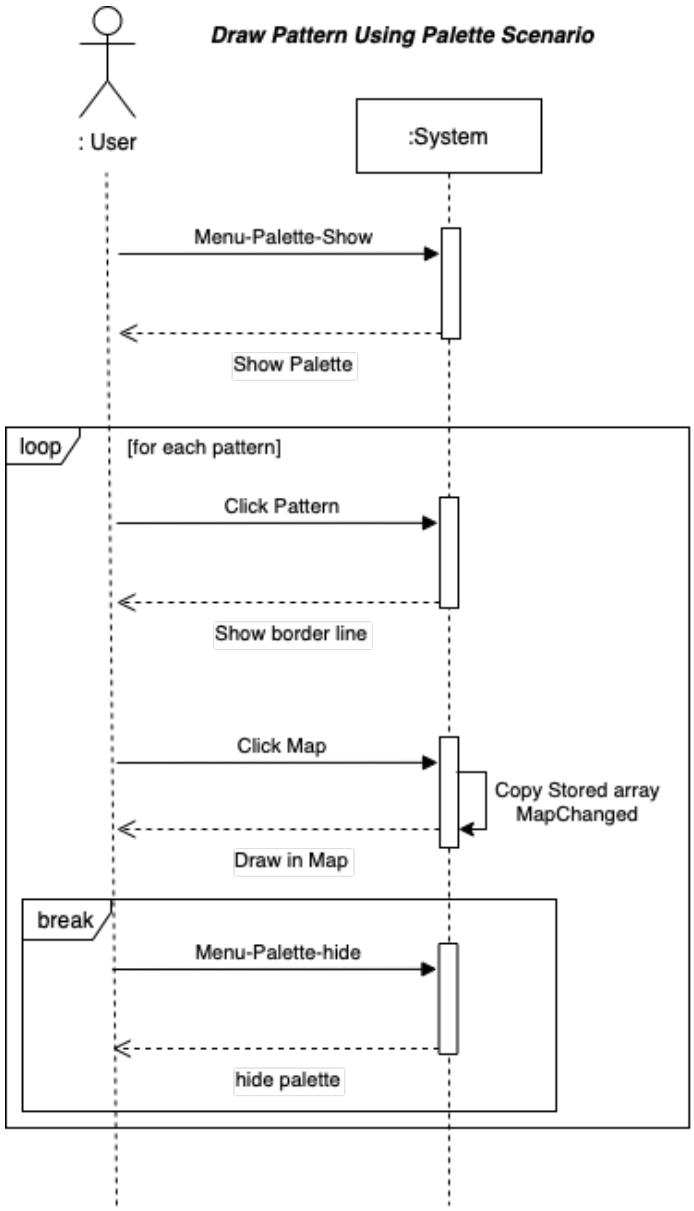
3.2. Sequence Diagram



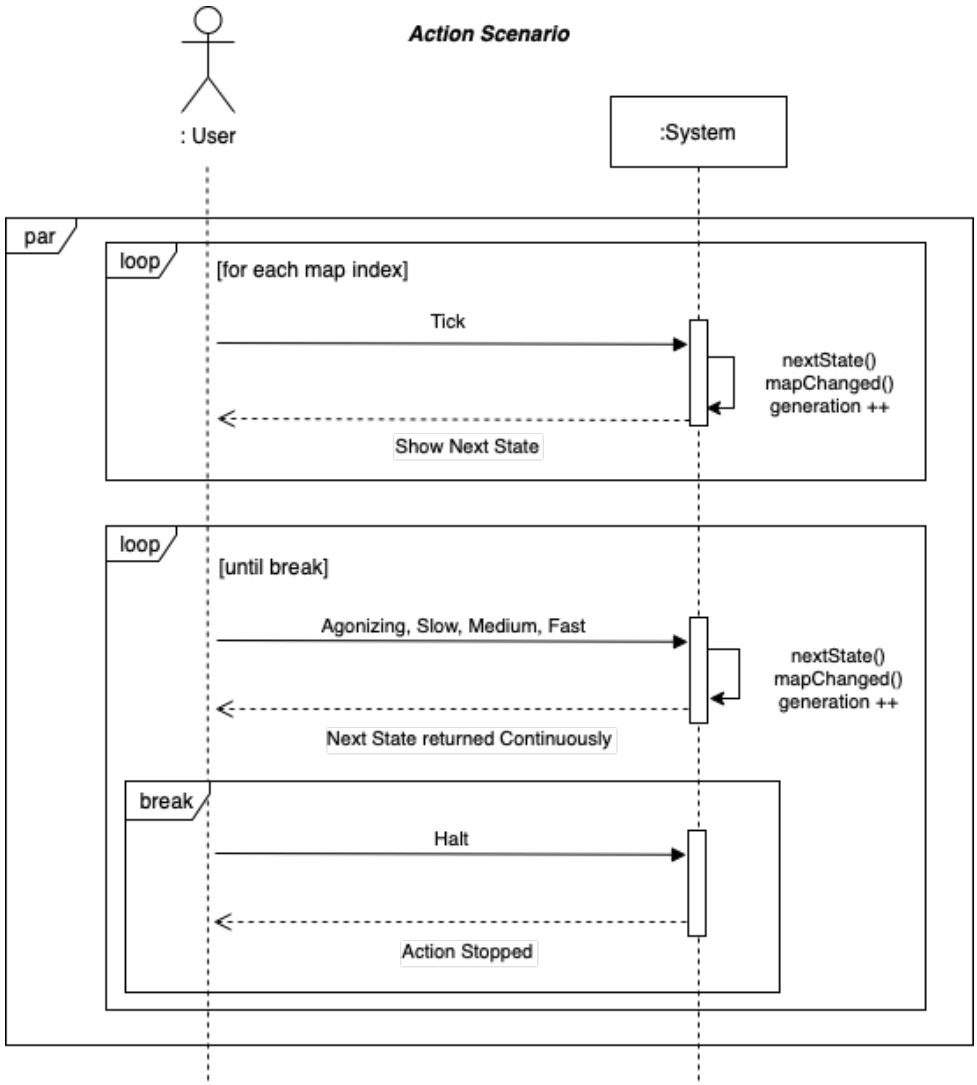
▲ [picture 08] Sequence Diagram (File I/O Scenario)



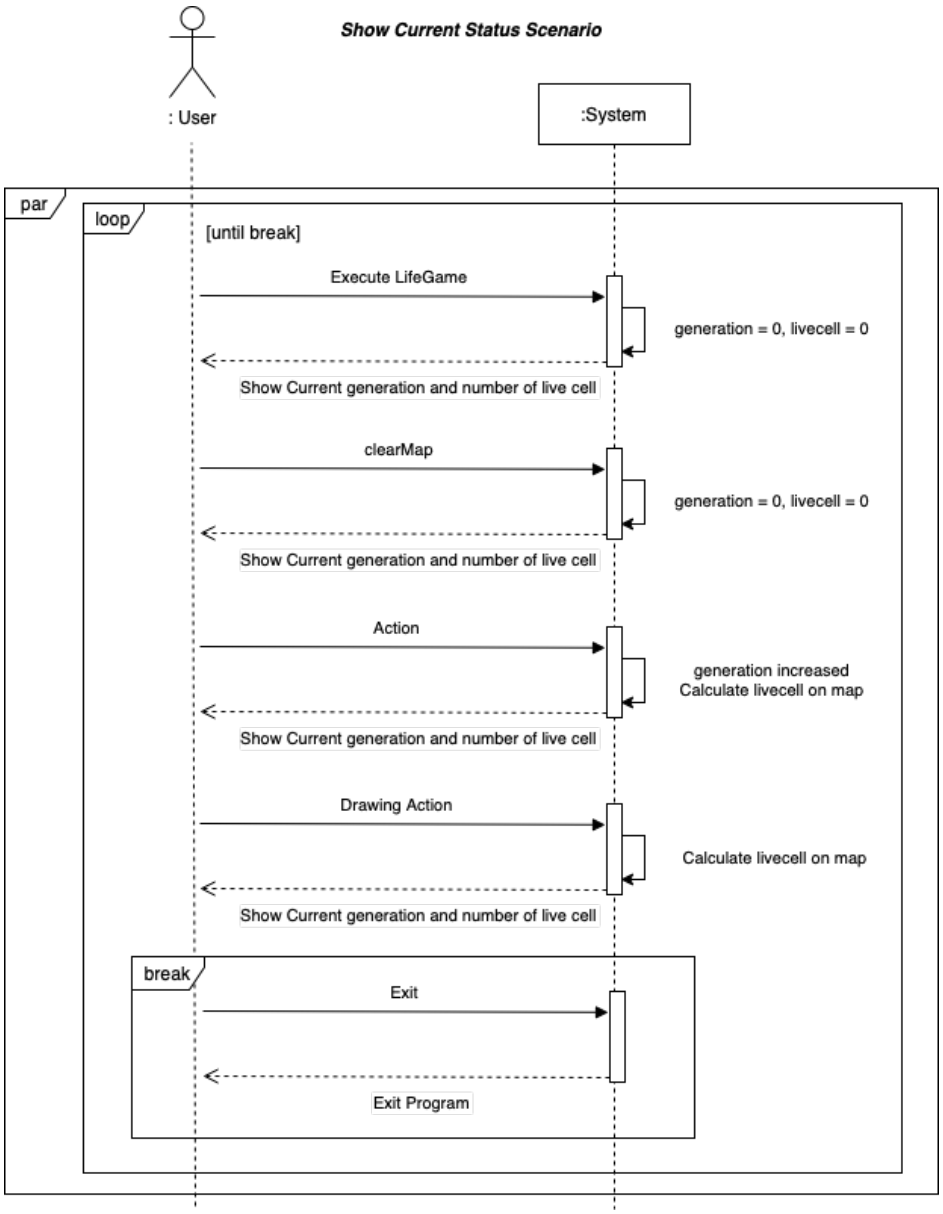
▲ [picture 09] Sequence Diagram (Toggle Scenario)



▲ [picture 10] Sequence Diagram (Draw Pattern Using Palette Scenario)



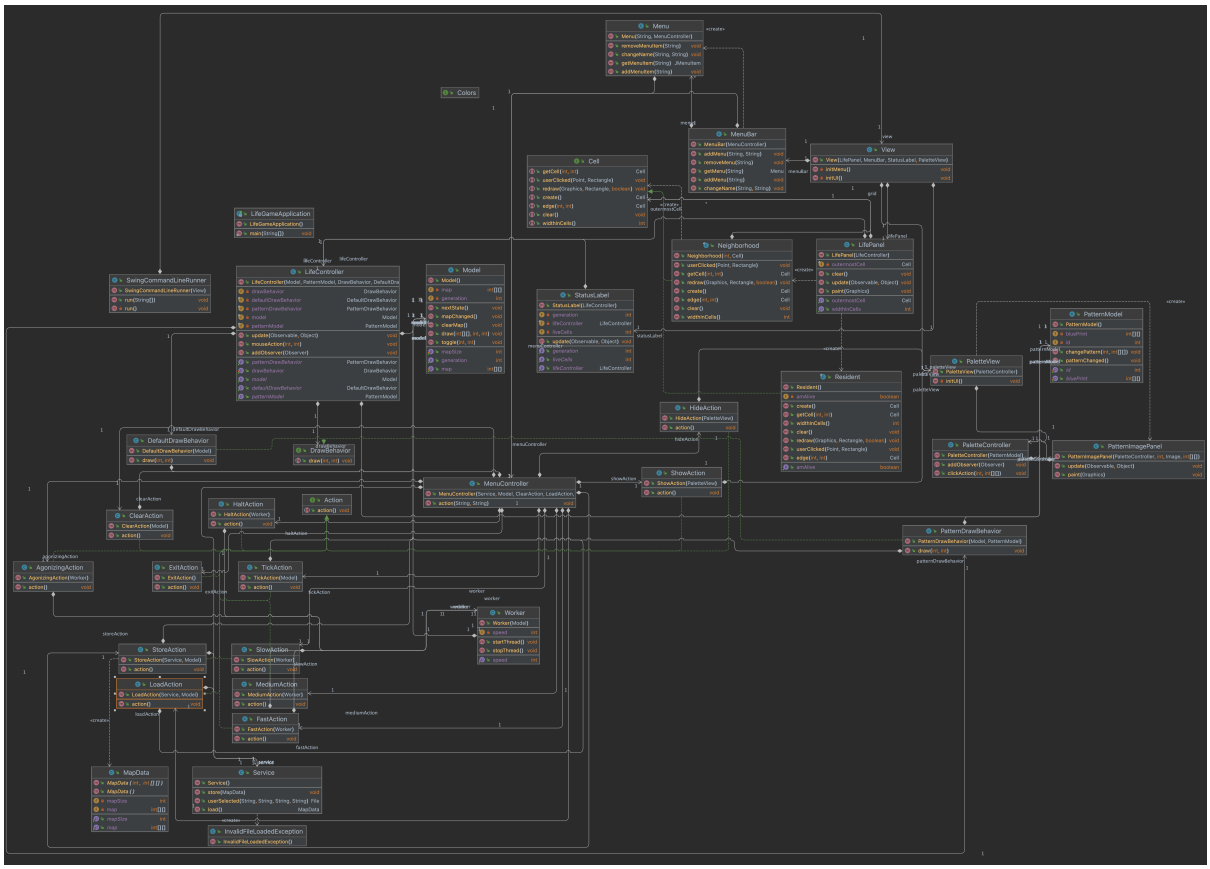
▲ [picture 11] Sequence Diagram (ActionScenario)



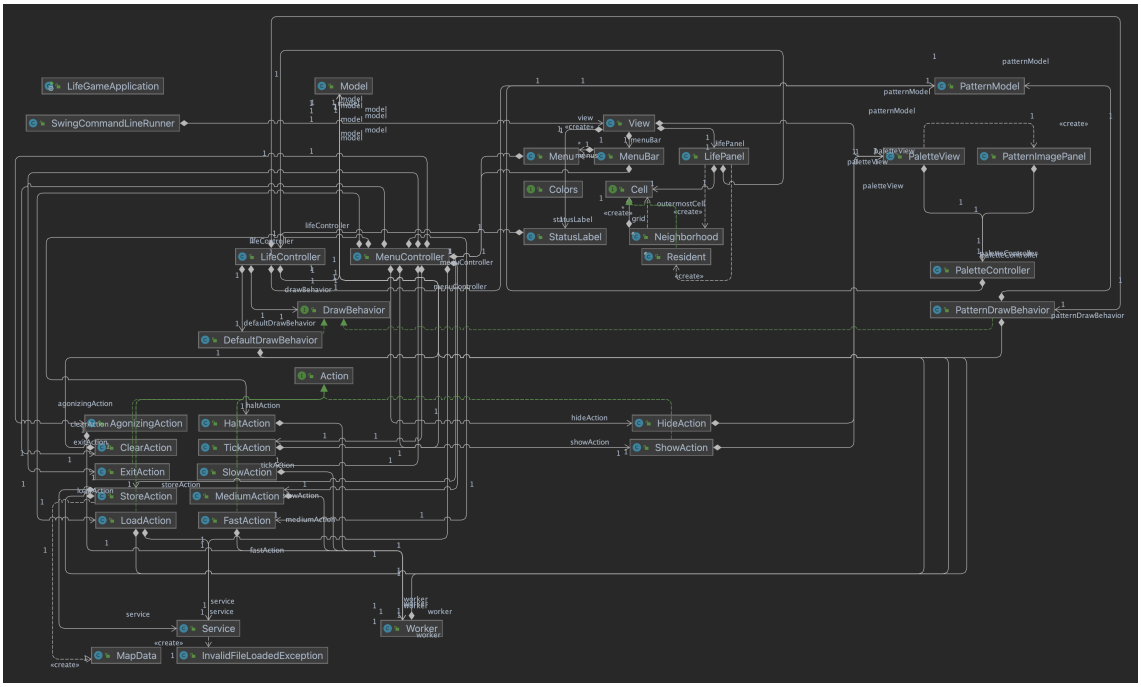
▲ [picture 12] Sequence Diagram (Show Current Status Scenario)

4. 개발

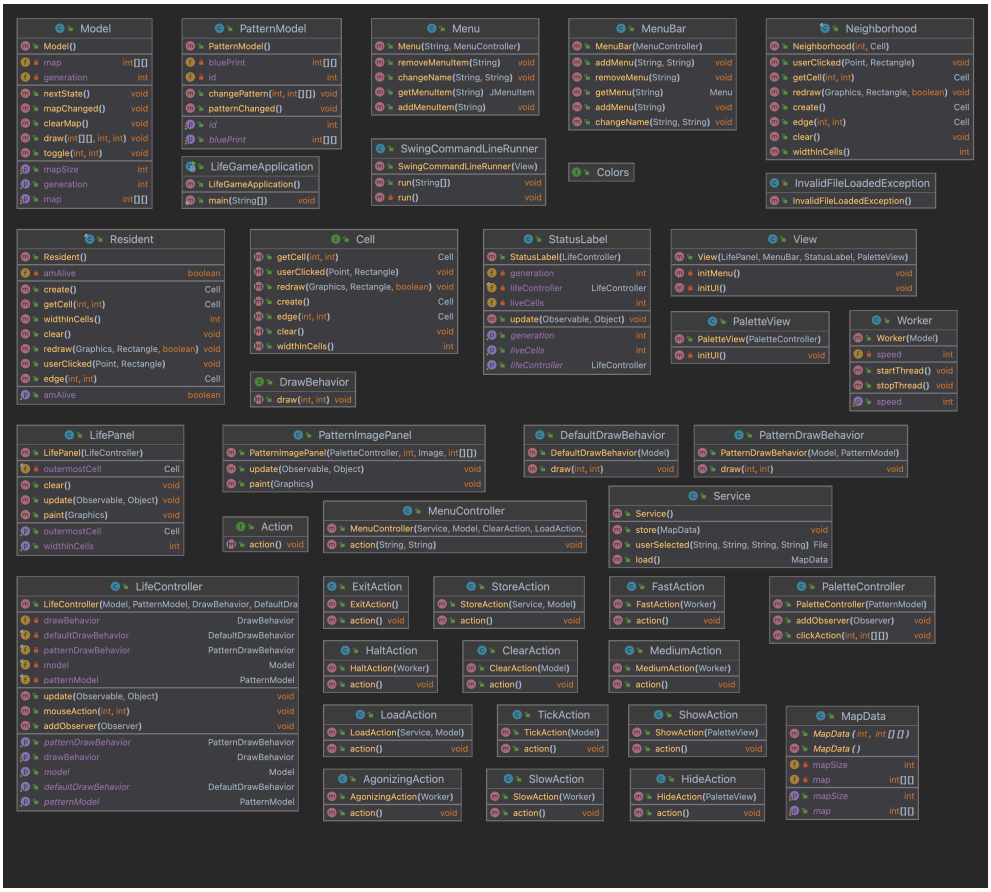
4.1. Class Diagram



▲ [picture 13] Class Diagram



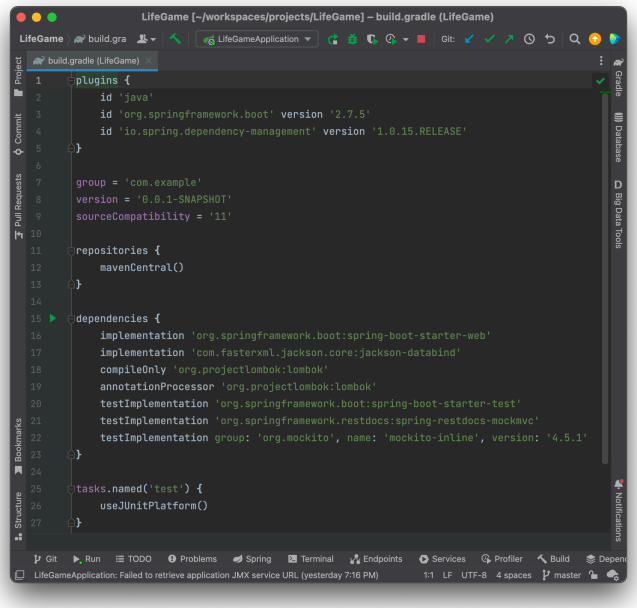
▲ [picture 14] Summarized Class Diagram



▲ [picture 15] Classes

4.2. Gradle

Gradle을 이용해 빌드하고 패키징하였다.



▲ [picture 16] build.gradle

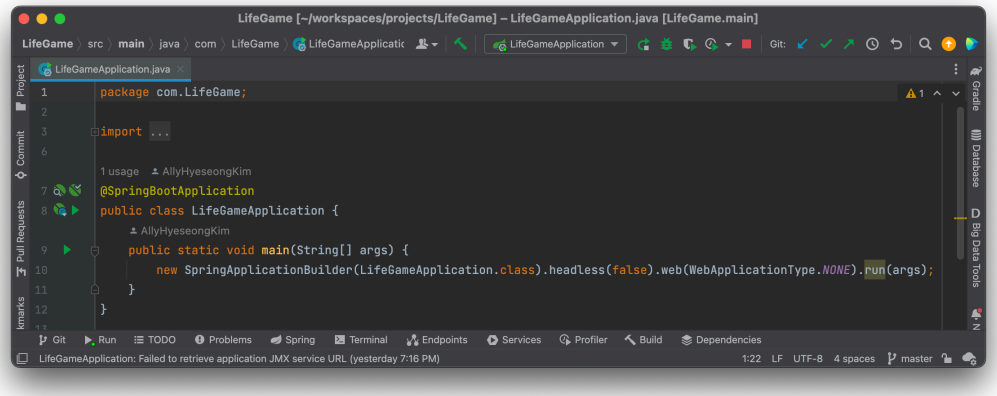
Spring Framework를 이용하기 위해 Spring에 관련된 의존성을 추가하였고, Getter, Setter를 편하게 추가하기 위해 lombok을 의존성에 추가하였다.

Load 및 Save 기능을 구현하기 위해 Java Object를 JSON으로 marshal/unmarshal할 수 있는 ObjectMapper를 이용하기 위한 Jackson의 의존성을 추가하였다.

JUnit을 이용해 테스트할 때 Spring으로 개발한 장점을 살려 쉽게 Mock 객체를 주입할 수 있도록 하기 위해 Mockito 의존성을 추가하였다.

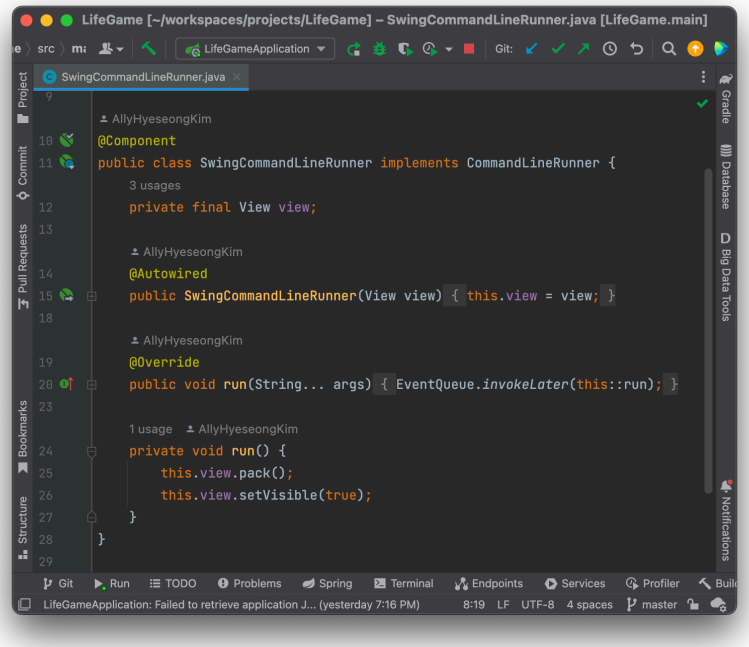
4.3. Spring

MVC Pattern으로 프로젝트를 리팩토링 하면서 객체 간 느슨한 결합과 IoC를 위해 Spring Framework를 활용하여 개발하였다.



▲ [picture 17] LifeGameApplication class

@SpringBootApplication을 main이 있는 클래스에 붙여주고 main에서Application을 run 해준다. Spring Web을 사용하지 않기 때문에 WebApplicationType.NONE을 설정해 주었다.



▲ [picture 18] LifeGameApplication class

SwingCommandLineRunner는 CommandLineRunner를 구현하는 클래스이다. 모든 컴포넌트들이 초기화되고 난 후에 메인 JFrame의 setVisible method를 실행하기 위해 작성되었다.

Spring Framework에서 SpringBootApplication이 실행되고 나면 CommandLineRunner를 구현한 컴포넌트를 찾아 run(String… args) method를 찾아 실행시킨다. EventQueue.invokeLater method를 이용해 Spring Component 초기화 작업이 완료되고 난 뒤에 View Component의 setVisible method를 실행해 Swing 화면을 표시한다.

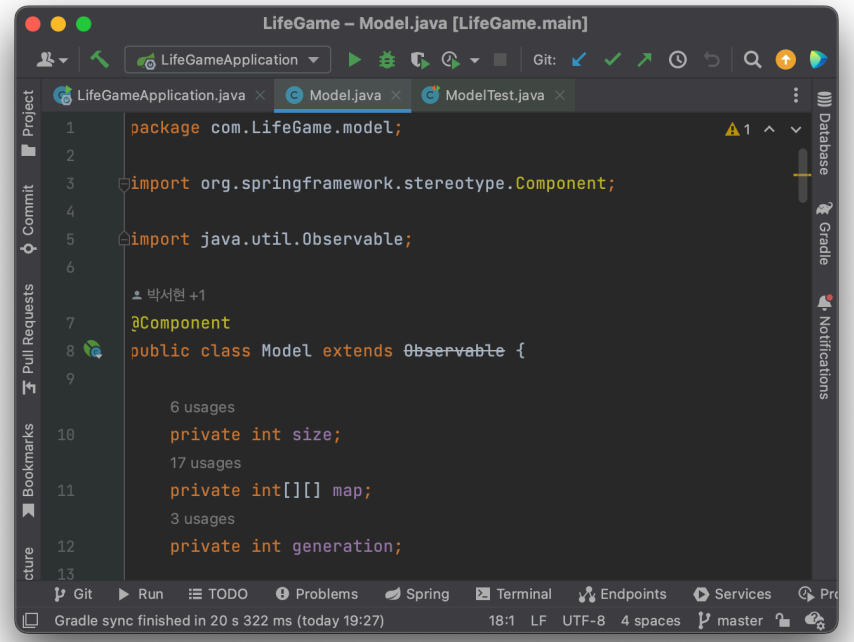
4.4. MVC Architecture Pattern

4.4.1. Model

model은 cell의 정보와 기능을 담고 있는 Model class와 palette에 들어가는 pattern에 대한 정보와 기능을 담고 있는 PatternModel class로 구성되어 있다. 각 model은 Observable class를 상속받아 Observer pattern의 subject 역할을 하여 각 observer에 update된 정보가 notify된다.

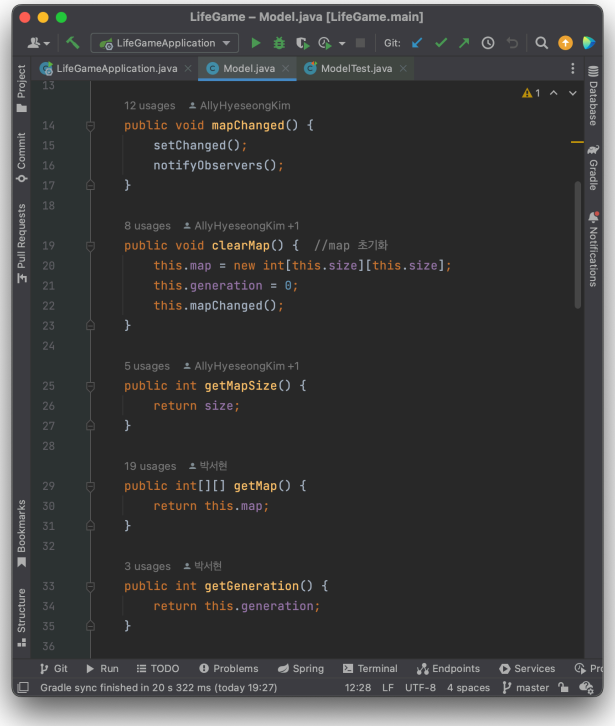
4.4.1.1. [Model] Model class

Model class는 Observer pattern의 subject 역할을 하며 Observable class를 상속받고 cell의 정보와 기능을 담고 있다.



▲ **[picture 19] Model class (1)**

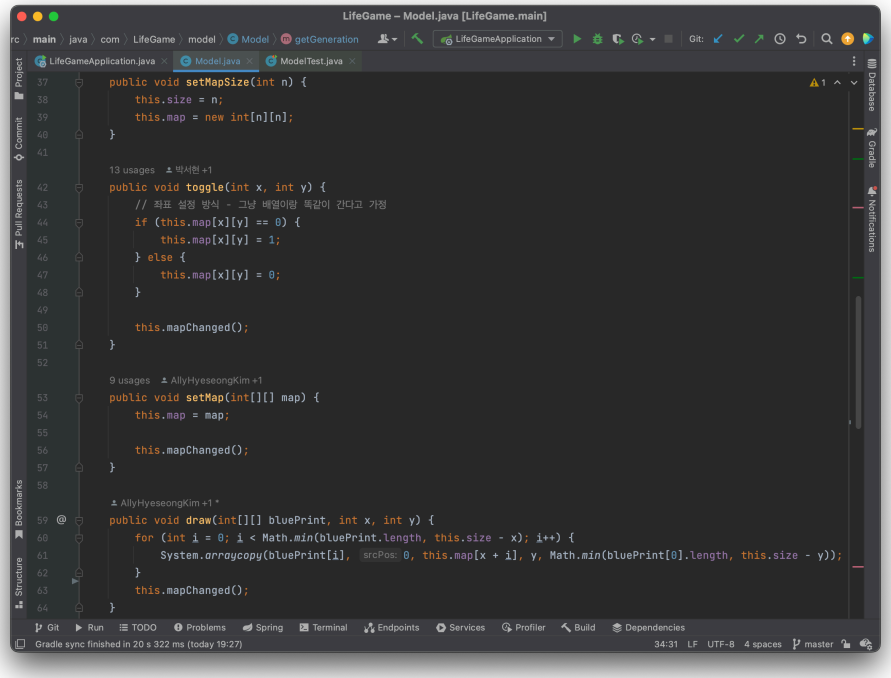
Model class는 2차원 배열 형태의 map과 int형의 size, int형의 generation 변수를 가진다.



▲ **[picture 20] Model class** (2)

mapChanged를 통해 map이 바뀐 경우 Observer에게 notify할 수 있다.

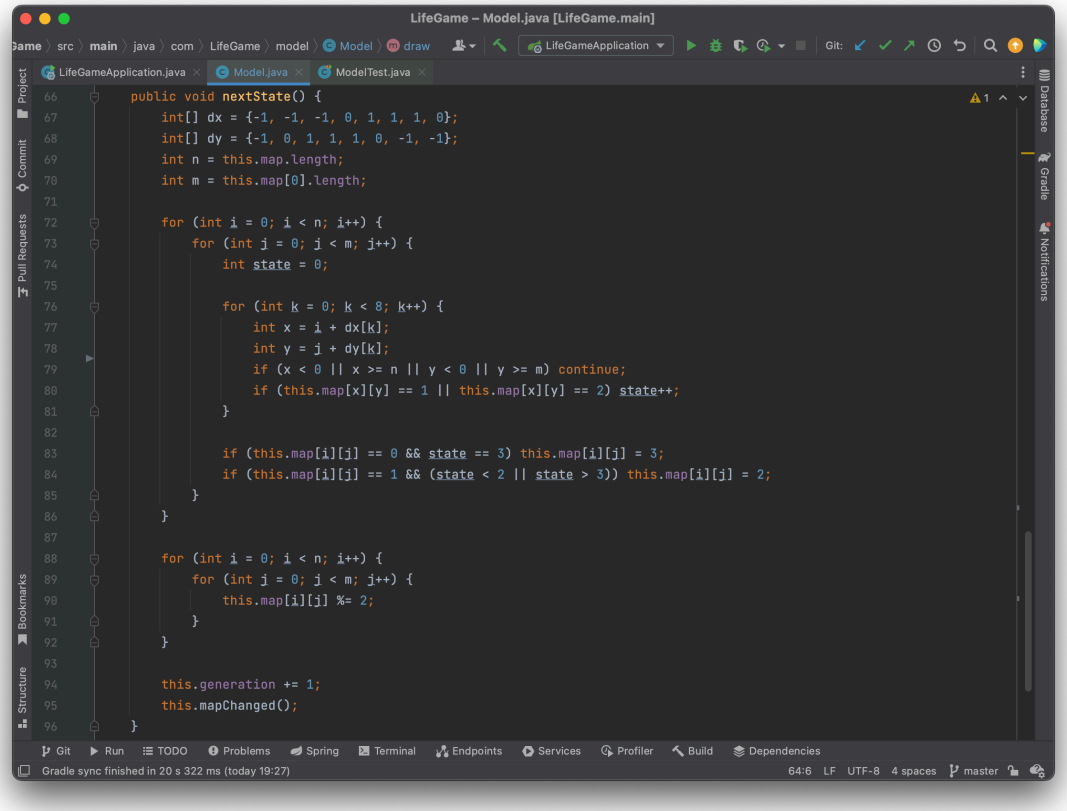
clearMap을 통해 현재 map과 generation을 초기화시키고 mapchanged를 호출한다. getMapSize에서는 현재 map의 size를 return하고, getMap을 통해 현재 map을 2차원 배열 형태로 반환한다. getGeneration을 통해 현재 세포의 세대 정보를 얻는다.



▲ **[picture 21] Model class** (3)

setMapSize로 map의 크기를 설정할 수 있다.

toggle을 통해서 map을 클릭했을 때 살아있는 세포(1)가 있었다면 0으로, 죽은 세포가 있었다면 1로 상태를 변경한다. setMap으로 map을 임의로 설정할 수 있다. draw 통해 팔레트의 패턴을 map에 그릴 수 있다. toggle, setMap, draw는 모두 mapChanged를 호출하여 상태 변화를 notify시킨다.

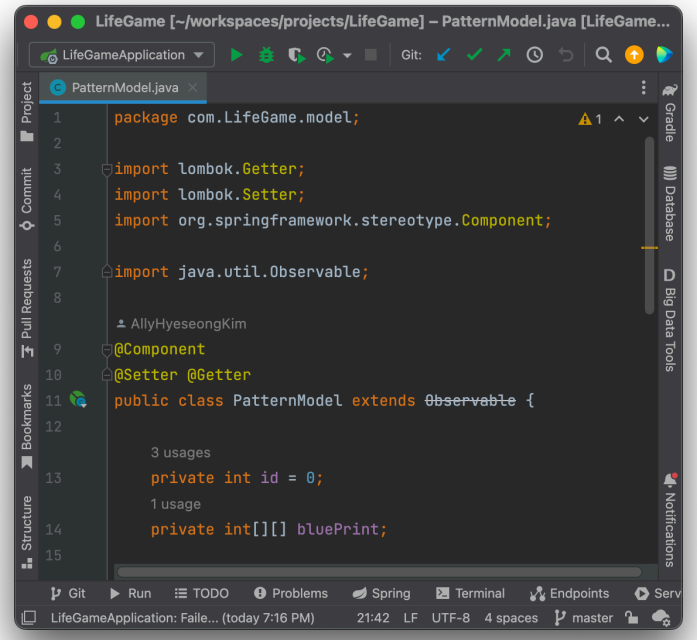


▲ **[picture 22] Model class (4)**

nextState를 호출하여 현재 map에 있는 세포들을 다음 상태로 변화시킨다. 죽은 세포의 이웃 중 세 개가 있으면 그 세포는 살아나고, 살아 있는 세포의 이웃 중 두 개나 세 개가 살아 있다면 해당 세포는 살아있는 상태를 유지한다. 2차원 배열을 순회하며 모든 방향에 대해 탐색하며, 탐색이 끝나고 해당하는 배열의 인덱스에 변화가 필요하다면 0이나 1을 사용하는 대신 2와 3을 사용한 후 모든 탐색이 끝나고 2로 나머지 연산을 한다. nextState를 할 때마다 generation이 증가하고, mapChanged가 호출된다.

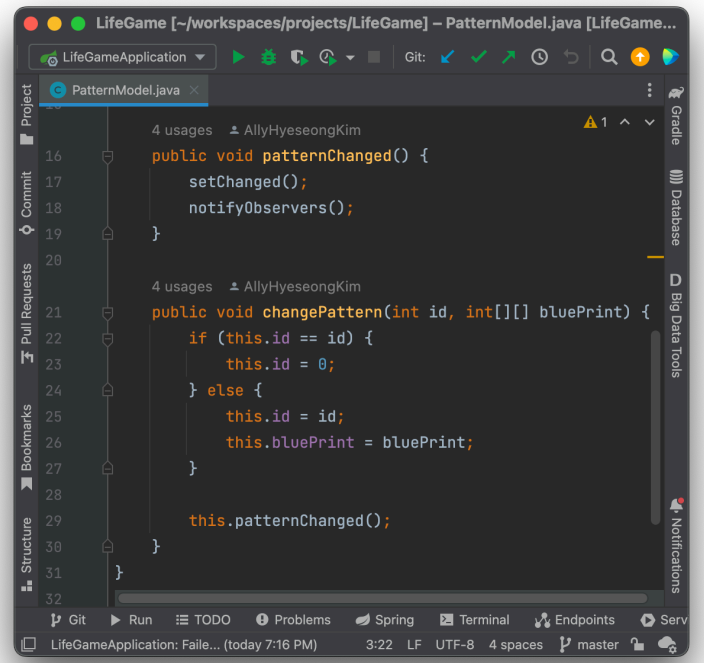
4.4.1.2. [Model] PatternModel class

PatternModel class는 Observer pattern의 subject로 Observable class를 상속받고 현재 선택된 패턴의 id와 그림 정보를 가지고 있다.



▲ **[picture 23] PatternModel class** (1)

이 클래스를 Singleton 객체로 생성하기 위해 Spring의 @Component annotation을 통해 Spring Bean에 등록하였고, lombok의 @Getter @Setter annotation을 통해 controller에서 현재 선택된 field들을 설정하고 observer에서 현재 선택된 field 값을 확인할 수 있도록 하였다.



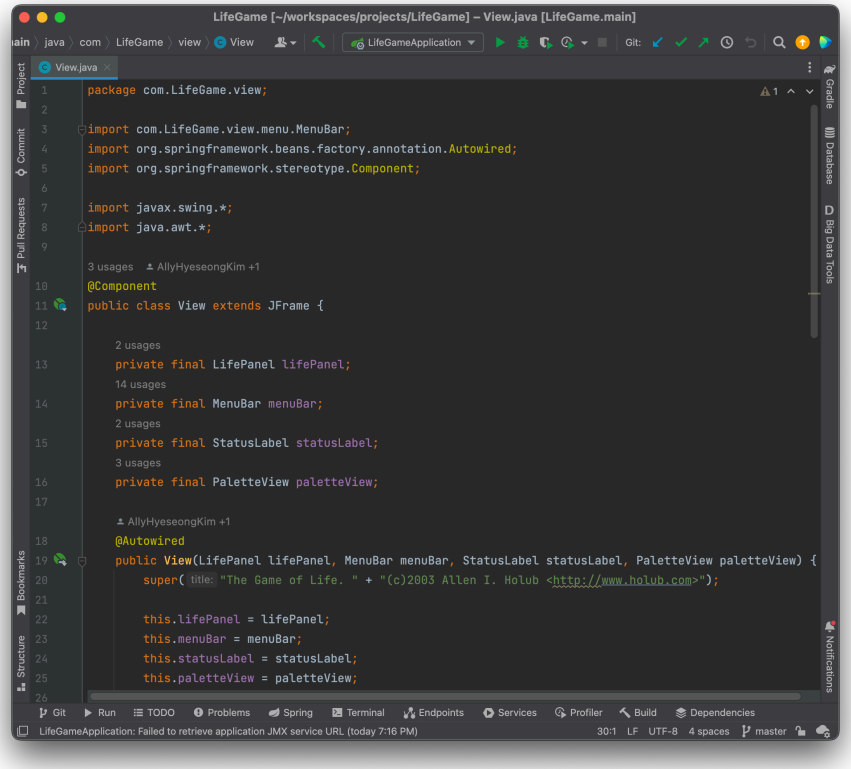
▲ **[picture 24] PatternModel class** (2)

changePattern method를 호출하여 현재 선택된 패턴을 등록하면 patternChanged method를 호출하여 observer에게 notify하도록 하였다.

4.4.2. View

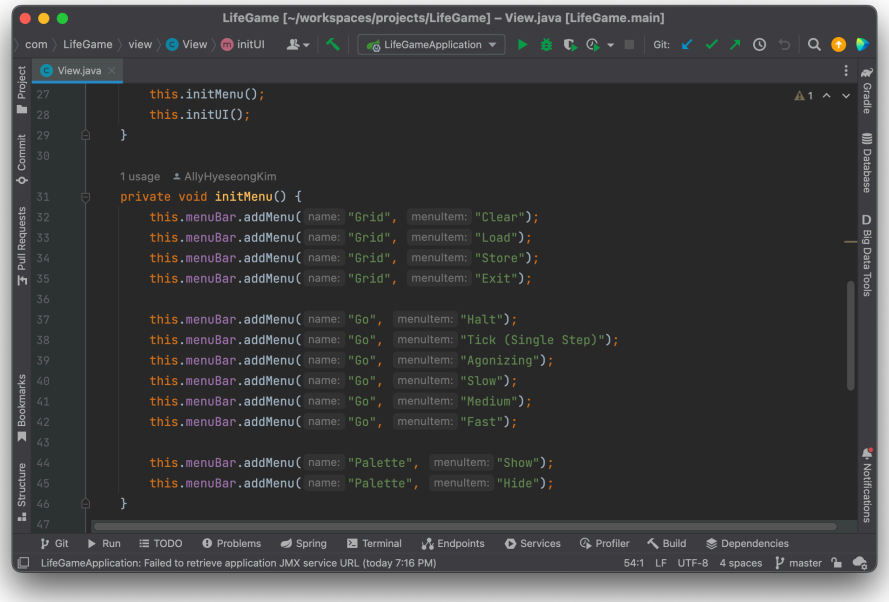
4.4.2.1. [View] View class

View class는 JFrame을 상속받는다. View 생성 시 menu bar를 등록하고, LifePanel, StatusLabel을 등록하도록 하였다.



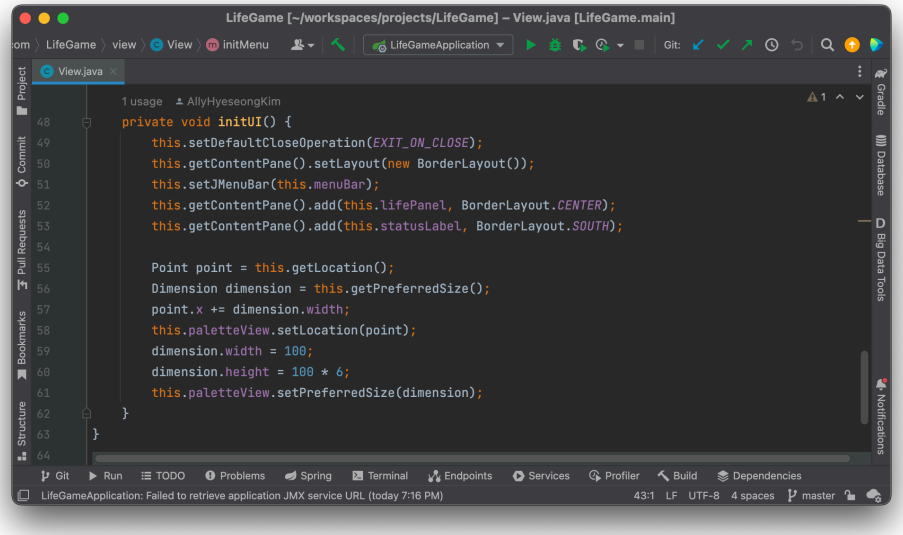
▲ **[picture 25] View class (1)**

이 클래스를 Singleton 객체로 생성하기 위해 Spring의 @Component annotation을 통해 Spring Bean에 등록하였고, View에서 사용하는 LifePanel, MenuBar, StatusLabel, PaletteView는 Spring의 @Autowired annotation을 통해 Bean에서 자동으로 주입받도록 하였다.



▲ **[picture 26] View class** (2)

MenuBar에 들어갈 Menu들을 View 생성 시 만들어지도록 하였다.

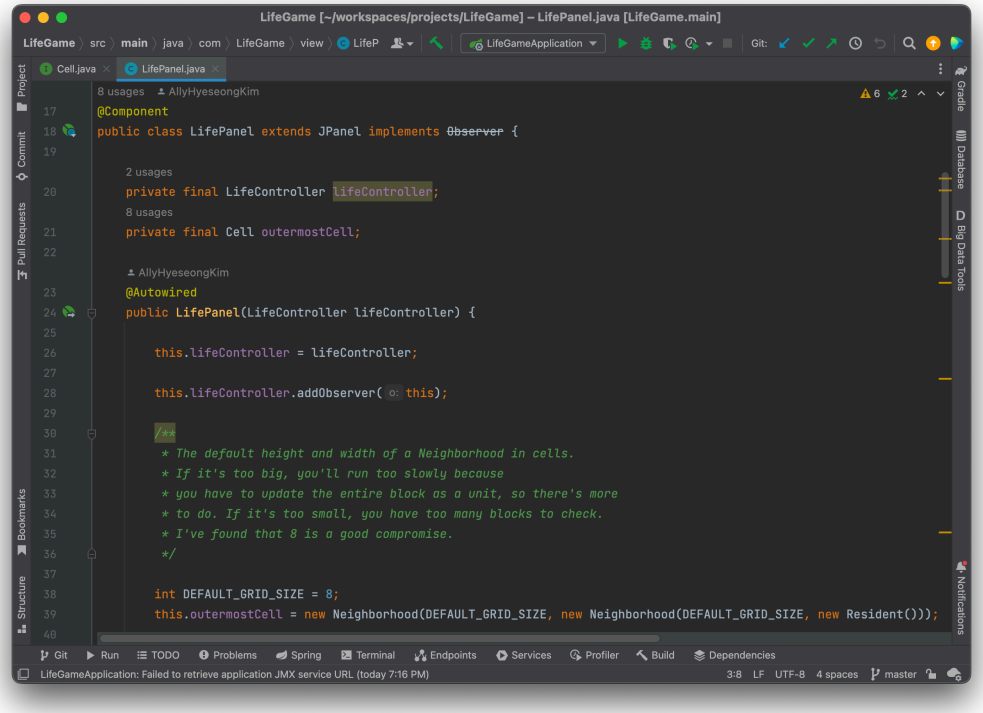


▲ **[picture 27] View class** (3)

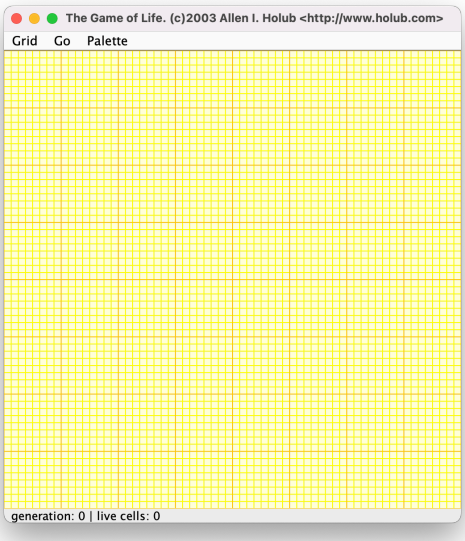
JFrame을 상속받는 View에 메뉴를 가지는 menu bar, cell들을 그리는 LifePanel, 현재 cell의 정보를 가지는 StatusLabel을 등록하였다. 또한, View가 그려지는 창의 바로 오른쪽에 PaletteView가 띄워지도록 설정하였다.

4.4.2.2. [View] LifePanel class

LifePanel class는 JPanel을 상속받고 Observer interface를 구현한다. Cell들이 있는 판을 그리는 panel로 cell을 가지고 있고, LifePanel에서 발생하는 로직은 LifeController로 위임한다.



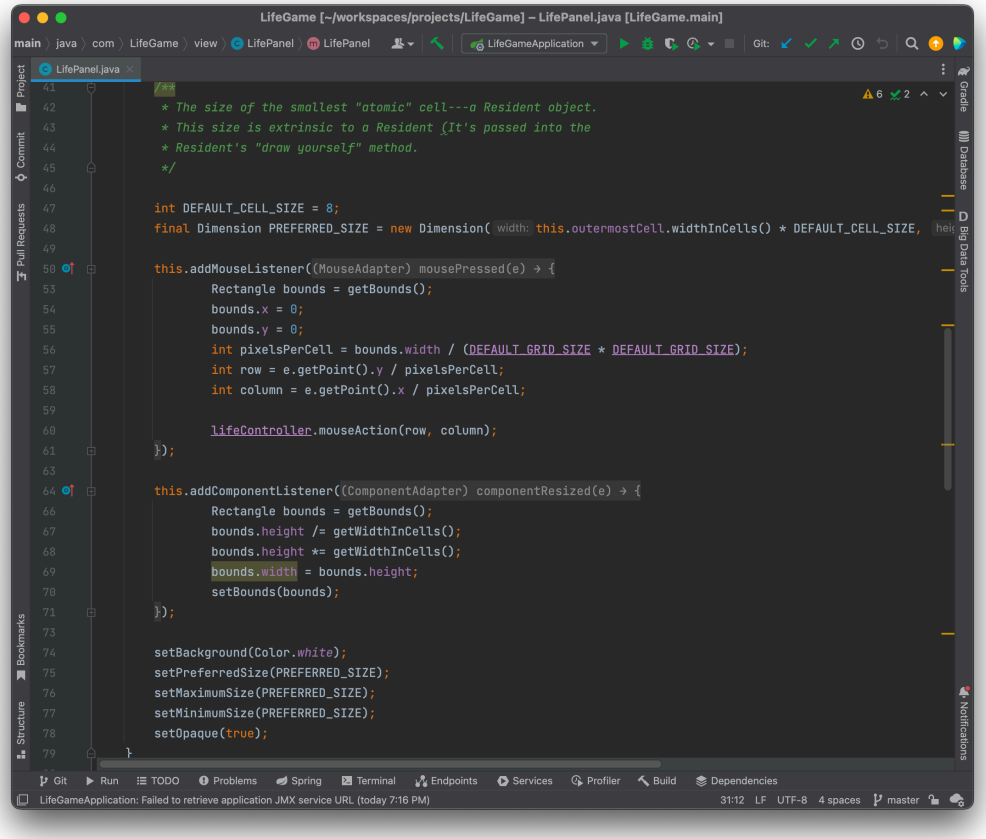
▲ [picture 28] LifePanel class (1)



▲ [picture 29] LifeGame

이 클래스를 Singleton 객체로 생성하기 위해 Spring의 @Component annotation을 통해 Spring Bean에 등록하였고, LifePanel에서 사용하는 LifeController, Cell은 Spring의 @Autowired annotation을 통해 Bean에서 자동으로 주입받도록 하였다.

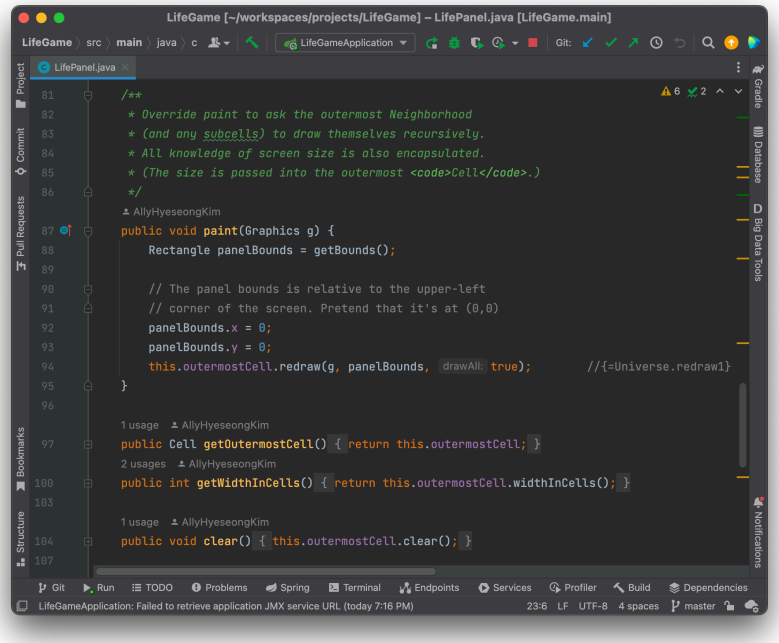
Cell들이 있는 판은 8*8개의 Resident cell이 있는 작은 Neighborhood cell이 8*8개 있는 형태로 이루어져 있다.



▲ **[picture 30] LifePanel class** (2)

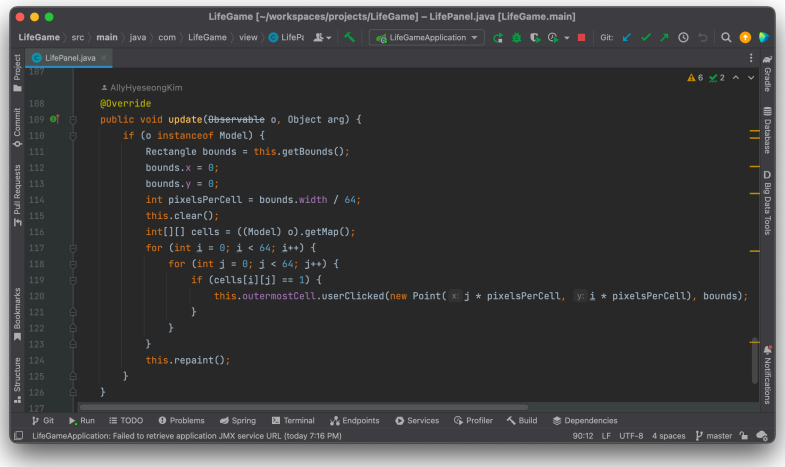
LifePanel에서 Cell을 누를 경우, MouseListener로 눌린 Cell의 좌표 위치를 계산하여 LifeController의 mouseAction method를 호출하도록 하였다. 모델의 조작 및 실제 로직은 LifeController가 담당하도록 하여 책임을 분리하고 느슨한 결합 구조로 구성하였다.

마우스로 JFrame 창의 크기를 조절하는 경우 LifePanel 그래픽 크기도 자동으로 조절해 주기 위해 ComponentListener를 붙이고 ComponentAdapter를 이용해 componentResized 이벤트가 발생할 때 마다 bound를 재설정하도록 하였다.



▲ **[picture 31] LifePanel class** (3)

paint method는 기존에 있떤 코드를 그대로 사용하였다. DFS 알고리즘을 이용해 각 Resident의 redraw method를 실행하여 화면을 업데이트한다. clear, getWidthInCells method도 마찬가지로 DFS 알고리즘을 이용하는 방식으로 동작한다.



▲ **[picture 32] LifePanel class** (4)

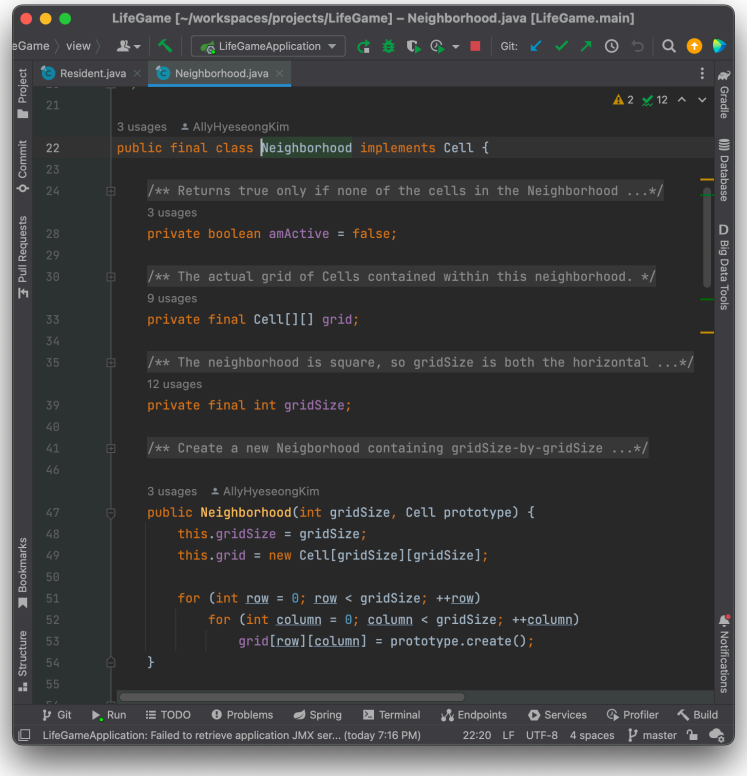
java.util.Observer interface를 구현하여 override한 method로, Model이 업데이트 될 때 마다 notify받고 화면에 Model대로 그려주도록 하였다. Model과 View에서 map배열을 저장하는 방식이 다르기 때문에 이를 변환하는 연산이 포함되며, 화면에 그리는 기능은 원래 Cell interface에 정의되어 있던 userClicked method를 이용하였다.

4.4.2.3. [View] Cell interface

기존 코드를 그대로 활용하여 개발하였으며, Neighborhood와 Resident가 Cell Interface를 구현한다.

4.2.2.4. [View] Neighborhood class

Cell interface를 구현한 클래스로, 하위 Cell들의 묶음 역할을 한다.



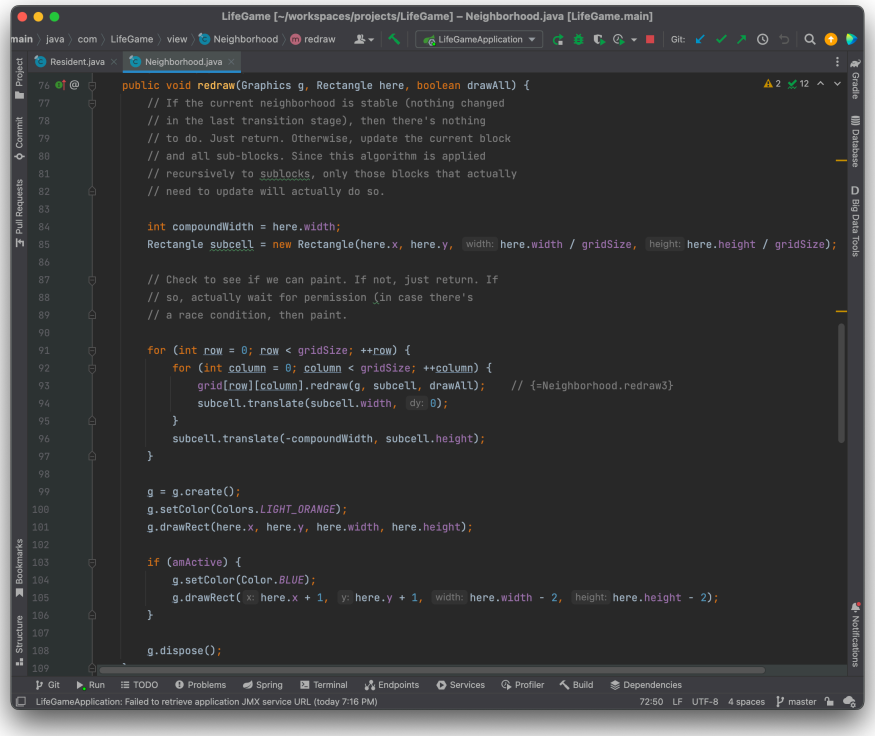
▲ **[picture 33] Neighborhood class** (1)

boolean형 amActive 필드는 현재 Cell의 활성화 여부를 저장하며, 내부에 활성화 된 Resident가 하나라도 있는 경우 활성화 된 것으로 본다. 이후 redraw method를 호출하면 amActive 필드를 참조하여 화면에 표시한다.

grid 필드는 하위 Cell들을 저장하는 필드이다.

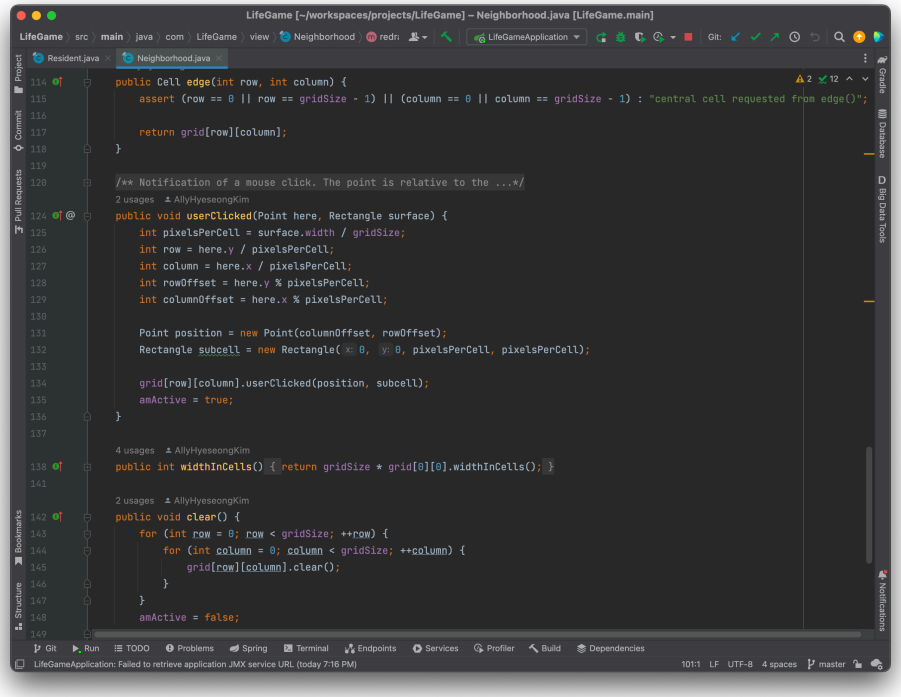
gridSize 필드는 grid 이차원 배열의 크기를 결정하는 필드로 생성자에서 참조한다.

생성자에서는 gridSize대로 grid를 초기화하고 parameter로 받은 prototype을 복사하여 grid의 모든 칸을 채운다.



▲ **[picture 34] Neighborhood class** (2)

redraw method는 LifePanel에서 자신에게 맞는 위치에 칸을 그리고 하위 Cell들을 DFS 알고리즘으로 순회하며 모든 하위 Cell들의 redraw method를 호출한다. amActive 필드의 값을 참조하여 칸 내부 색을 지정하여 Live Cell을 표시한다.



▲ **[picture 35] Neighborhood class** (3)

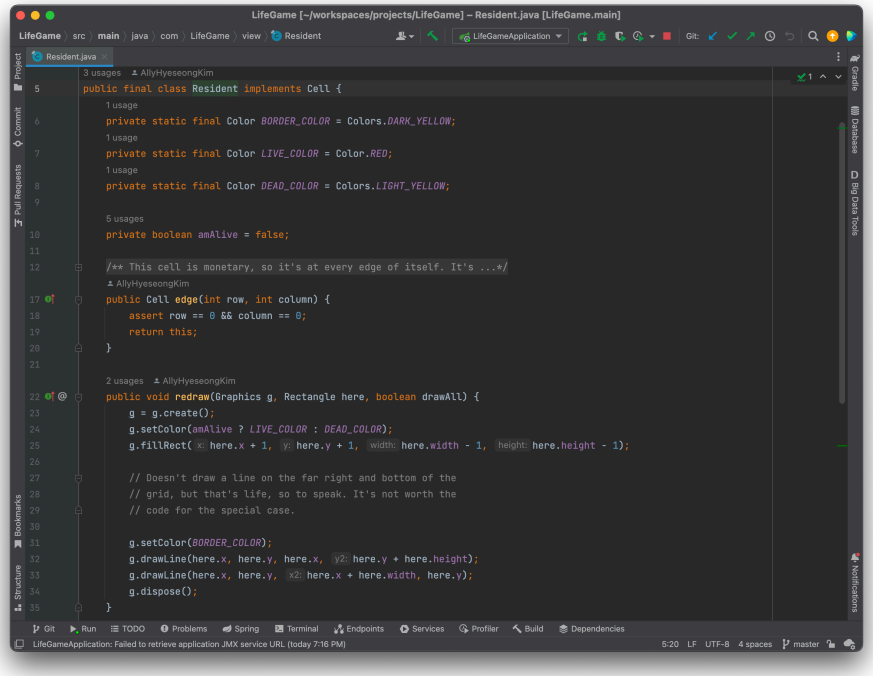
edge method는 row, column에 위치한 하위 Cell 인스턴스를 반환하는 getter로서 기능한다.

userClicked method는 화면에 클릭 이벤트가 발생했을 때 화면의 픽셀값을 토대로 활성화해야 하는 Resident를 찾아 해당 Resident의 amActive를 true로 설정한다.

widthInCells method는 해당 Cell이 화면에서 차지하는 width 픽셀 값을 반환한다.

clear method는 DFS 알고리즘을 이용해 모든 하위 Cell들을 순회하며 amActive 값을 false로 업데이트한다.

4.4.2.5. [View] Resident class



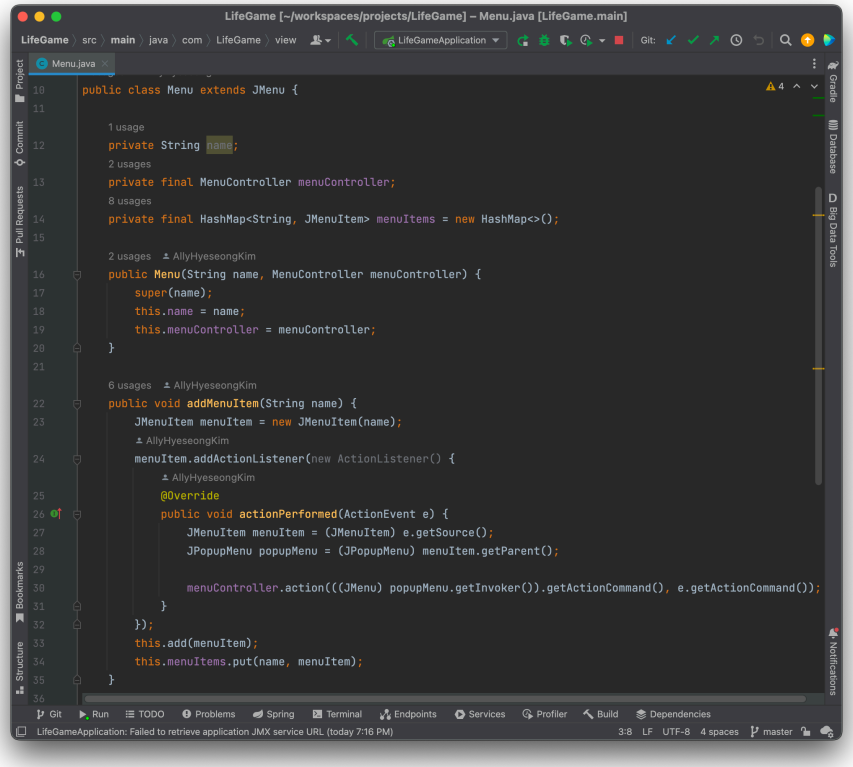
▲ **[picture 36] Resident class** (1)

edge method는 자기 자신을 반환한다.

redraw method는 amAlive 필드 값에 따라 화면의 적절한 위치에 붉은색 혹은 노란색 정사각형을 그리는 기능을 한다.

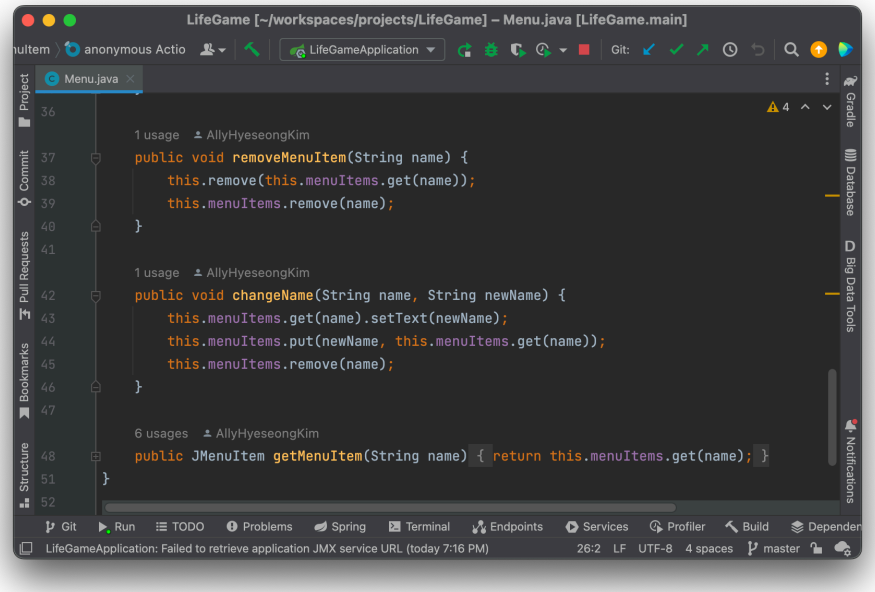
4.4.2.7. [View] Menu class

Menu class는 JMenu class를 상속받는 클래스로, 상단 메뉴바의 메뉴 내 기능 하나하나를 표시한다.



▲ [picture 37] Menu class (1)

addMenuItem은 메뉴 내부에 JMenuItem을 등록하는 기능을 한다. 각 JMenuItem은 ActionListener를 통해 클릭 이벤트를 받고, MenuController에게 기능을 위임한다.

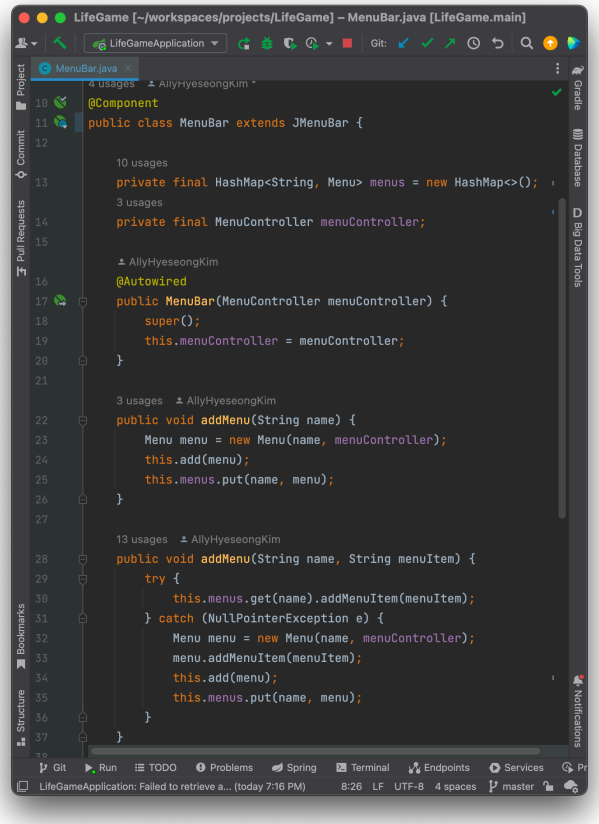


▲ [picture 38] Menu class (2)

removeMenuItem, changeName, getMenuItem은 각각 메뉴 내부 JMenuItem을 제거, 수정, get 하는 method 들이며, 기존 코드에서 추가로 작성한 기능이다.

4.4.2.8. [View] MenuBar class

MenuBar Class는 JMenuBar 클래스를 상속한다.

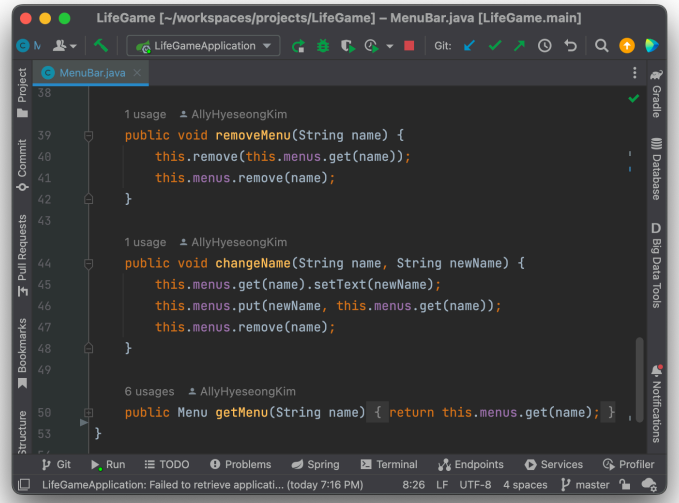


▲ **[picture 39] MenuBar class** (1)

menus 필드는 하위 Menu들을 담고 있는 필드이다.

이 클래스를 Singleton 객체로 생성하기 위해 Spring의 @Component annotation을 통해 Spring Bean에 등록하였고, menuController는 MenuController Bean을 주입받는 필드이다.

addMenu method는 method overloading을 이용해 작성하였고, Menu만 생성하여 추가하는 method와 Menu 내부에 MenuItem을 추가하는 method 두 가지가 존재한다.

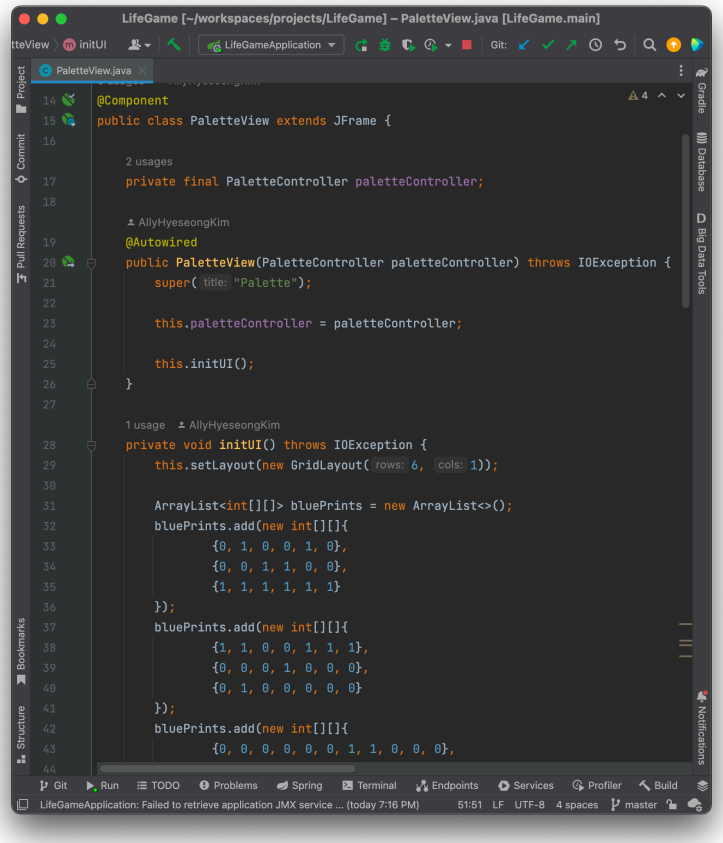


▲ **[picture 40] MenuBar class** (2)

Menu 클래스와 마찬가지로, removeMenu, changeName, getMenu method를 갖고 있고, 각각 메뉴 제거, 메뉴 수정, get 기능을 수행한다.

4.4.2.9. [View] PaletteView class

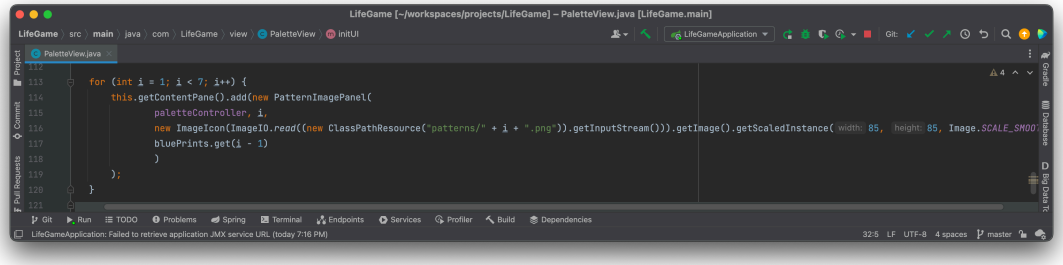
PaletteView Class는 JFrame을 상속받는다.



▲ **[picture 41] PaletteView class** (1)

@Component Annotation을 이용해 Singleton으로 만들어 Spring Bean Container에 Bean으로 등록하고, @Autowired Annotation을 통해 PaletteController 의존성을 주입받는다.

InitUI method에서 int형 2차원 배열을 담은 ArrayList를 작성하였다.

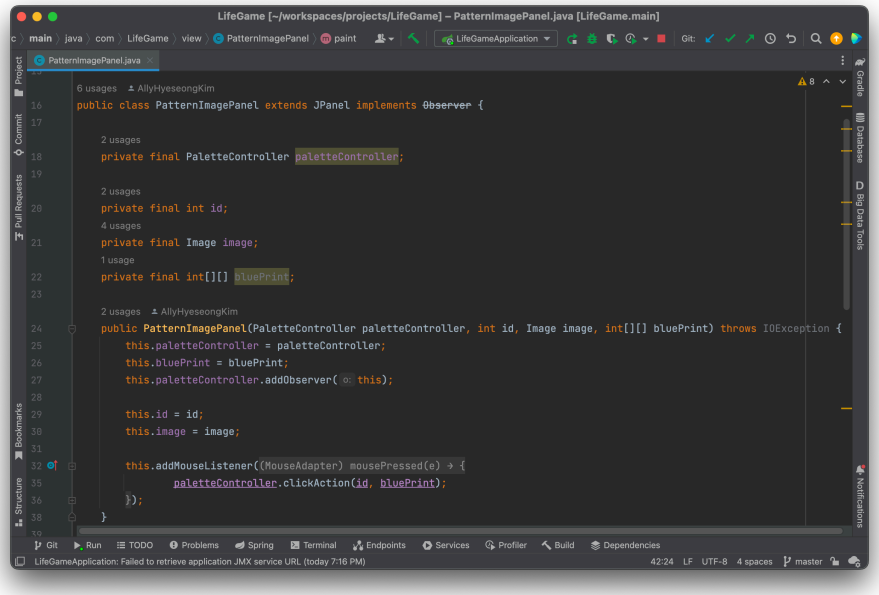


▲ **[picture 42] PaletteView class** (2)

bluePrints ArrayList내 각각의 2차원배열 패턴과 패턴에 대응하는 이미지를 parameter로 넘겨 PatternImagePanel을 생성하도록 하였다.

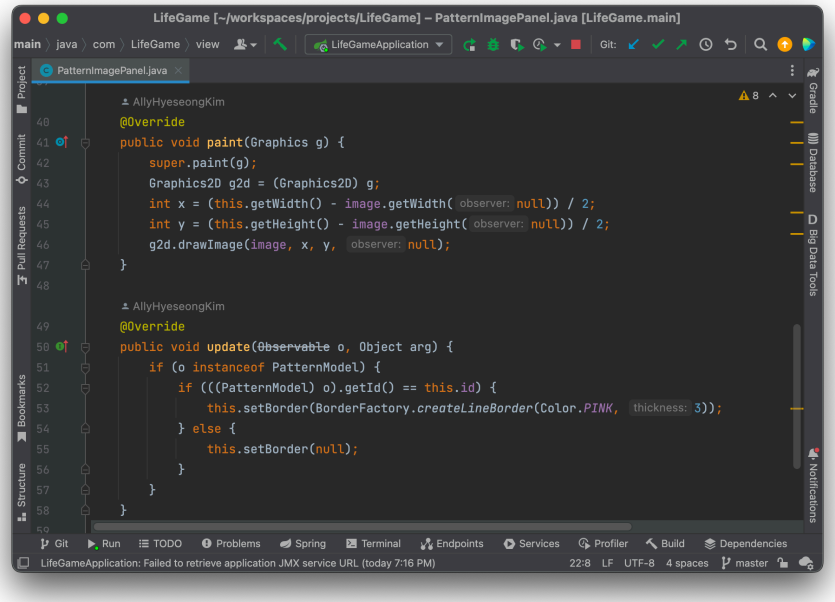
4.4.2.10. [View] PatternImagePanel class

PatternImagePanel은 JPanel을 상속하고, java.util.Observer interface를 구현한다.



▲ **[picture 43] PatternImagePanel class** (1)

PaletteController Bean 의존성을 주입받고, 마우스 이벤트 발생 시 작동하는 로직을 PaletteController에 위임한다.



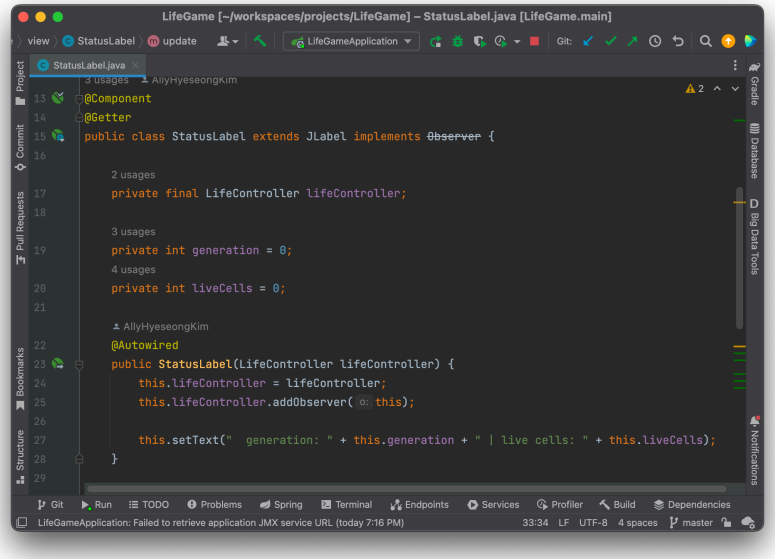
▲ **[picture 44] PatternImagePanel class** (2)

paint method는 화면에 패널을 그리는 기능을 수행한다.

update method는 PatternModel이 변화할 때마다 호출되어 해당 Pattern이 활성화되었는지를 체크하여 Border에 반영한다.

4.4.2.11. [View] StatusLabel class

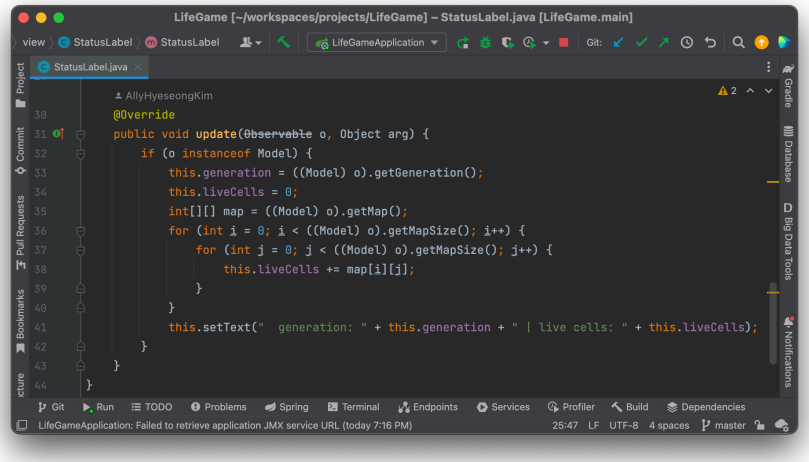
StatusLabel은 JLabel을 상속하고 java.util.Observer interface를 구현한다.



▲ **[picture 45] StatusLabel class** (1)

@Component Annotation을 이용해 Spring Bean으로 등록하여 Singleton 객체로 구현하였다.

LifeController Bean 의존성을 주입받아 addObserver method를 통해 Model에 옵저버로 등록한다.

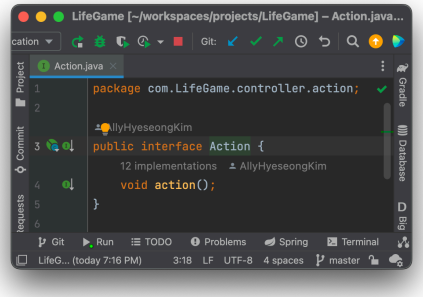


▲ **[picture 46] StatusLabel class** (2)

update method는 Model이 변화할 때 마다 실행되어 Model의 map에서 Live Cell의 개수를 count하고, generation 정보와 함께 Text를 업데이트한다.

4.4.3. Controller

4.4.3.1. [Controller] Action interface

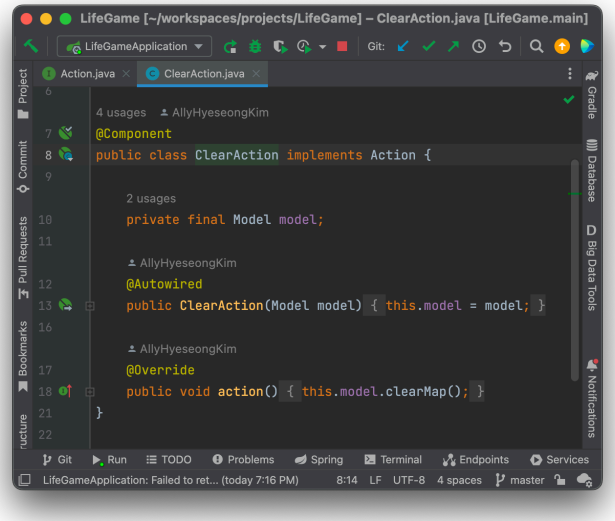


▲ **[picture 47] Action interface**

Command Pattern을 이용해 Menu를 클릭할 때 동작하는 Command들의 프로토타입을 정의한 interface이다. 이후 다형성을 활용해 MenuController에서 적절히 로직을 수행한다.

4.4.3.2. [Controller] ClearAction class

ClearAction class는 Action interface를 구현한다.



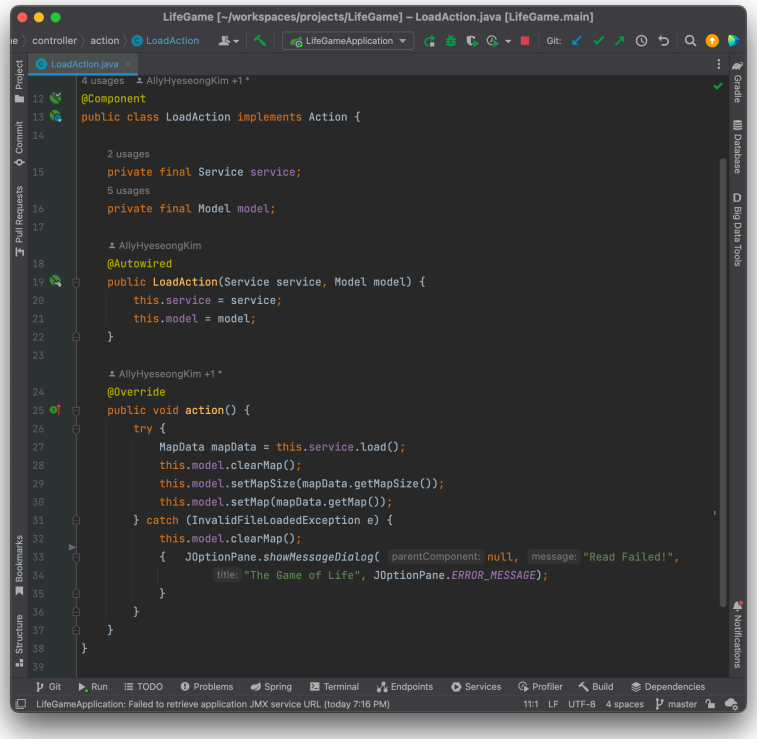
▲ [picture 48] ClearAction class

@Component Annnotation을 이용해 Spring Bean으로 등록해 Singleton으로 구현하였고, 생성자에서 @Autowired를 이용해 Model Bean을 주입받았다.

action method를 실행하면 map의 clearModel method를 호출하여 map을 초기화한다.

4.4.3.3. [Controller] LoadAction class

LoadAction class는 Action interface를 구현한다.



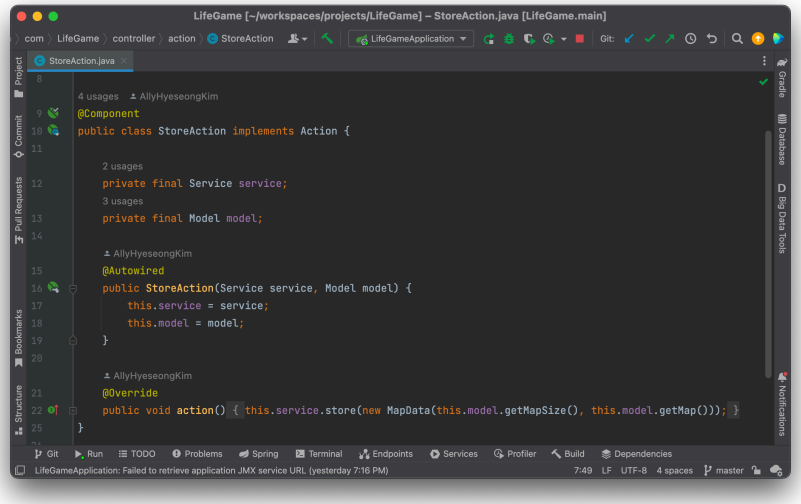
▲ [picture 49] LoadAction class

@Component Annnotation을 이용해 Spring Bean으로 등록해 Singleton으로 구현하였고, 생성자에서 @Autowired를 이용해 Model, Service Bean을 주입받았다.

action method를 실행하면 Service로부터 Map Data를 읽어 Model에 업데이트한다.

4.4.3.4. [Controller] StoreAction class

StoreAction class는 Action interface를 구현한다.



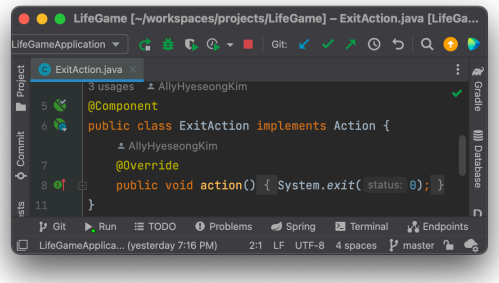
▲ [picture 50] StoreAction class

@Component Annnotation을 이용해 Spring Bean으로 등록해 Singleton으로 구현하였고, 생성자에서 @Autowired를 이용해 Model, Service Bean을 주입받았다.

action method를 실행하면 Model로부터 Map Data를 받아 Service에게 저장 기능을 요청한다.

4.4.3.5. [Controller] ExitAction class

ExitAction class는 Action interface를 구현한다.

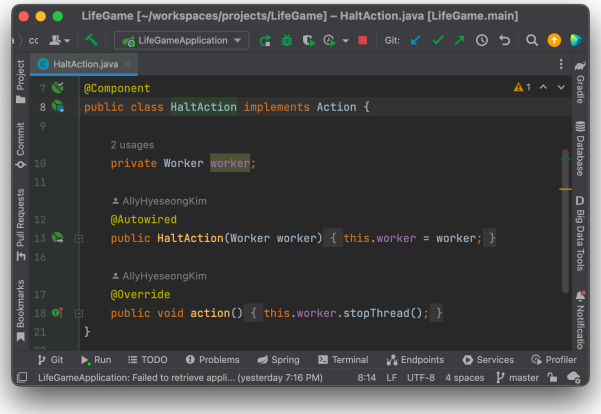


▲ [picture 51] ExitAction class

@Component Annnotation을 이용해 Spring Bean으로 등록해 Singleton으로 구현하였고, action method를 실행하면 프로세스가 종료된다.

4.4.3.6. [Controller] HaltAction class

HaltAction class는 Action interface를 구현한다.



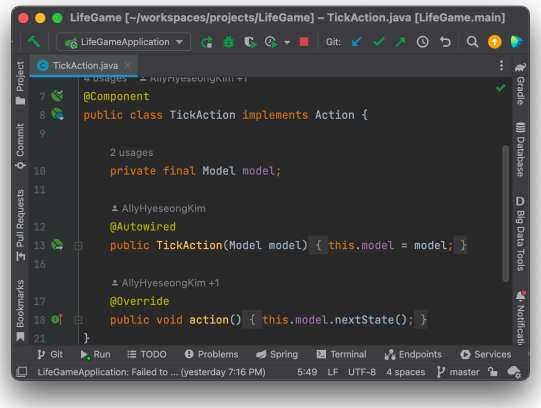
▲ [picture 52] HaltAction class

@Component Annnotation을 이용해 Spring Bean으로 등록해 Singleton으로 구현하였고, 생성자에서 @Autowired를 이용해 Worker Bean을 주입받았다.

action method를 실행하면 Worker의 현재 실행 중인 스레드를 종료하는 stopThread method를 호출한다.

4.4.3.7. [Controller] TickAction class

TickAction class는 Action interface를 구현한다.



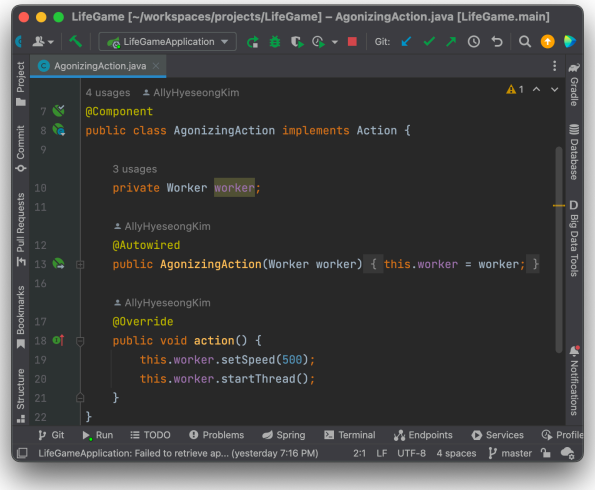
▲ [picture 53] TickAction class

@Component Annnotation을 이용해 Spring Bean으로 등록해 Singleton으로 구현하였고, 생성자에서 @Autowired를 이용해 Model Bean을 주입받았다.

action method를 실행하면 Model을 다음 상태로 업데이트하는 nextState method를 호출한다.

4.4.3.8. [Controller] AgonizingAction class

AgonizingAction class는 Action interface를 구현한다.



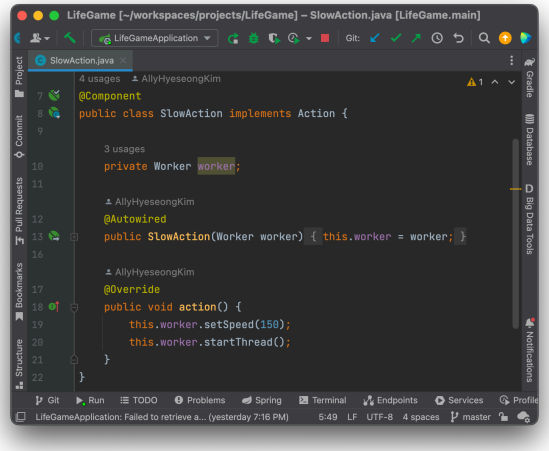
▲ [picture 54] AgonizingAction class

@Component Annnotation을 이용해 Spring Bean으로 등록해 Singleton으로 구현하였고, 생성자에서 @Autowired를 이용해 Worker Bean을 주입받았다.

action method를 실행하면 Worker의 스레드 업데이트 주기를 500으로 설정하고 스레드를 실행한다.

4.4.3.9. [Controller] SlowAction class

SlowAction class는 Action interface를 구현한다.



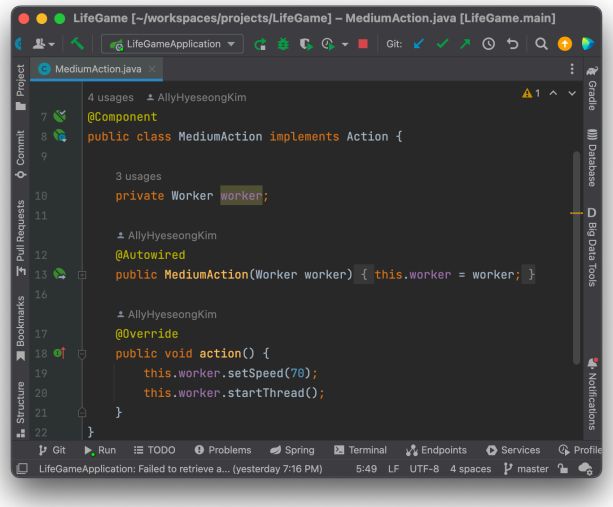
▲ [picture 55] SlowAction class

@Component Annnotation을 이용해 Spring Bean으로 등록해 Singleton으로 구현하였고, 생성자에서 @Autowired를 이용해 Worker Bean을 주입받았다.

action method를 실행하면 Worker의 스레드 업데이트 주기를 150으로 설정하고 스레드를 실행한다.

4.4.3.10. [Controller] MediumAction class

MediumAction class는 Action interface를 구현한다.



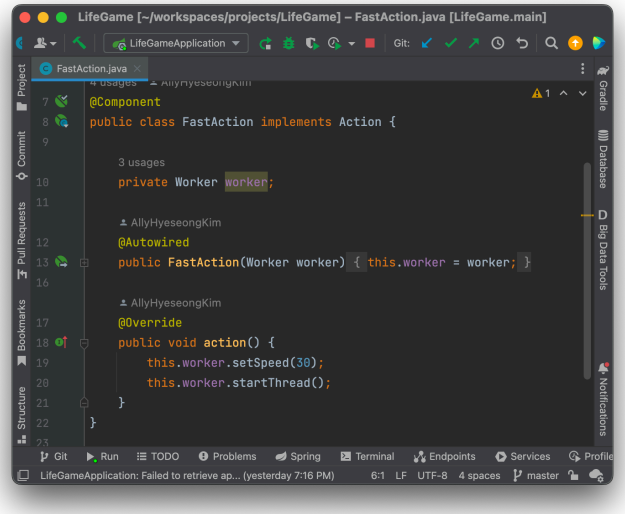
▲ [picture 56] MediumAction class

@Component Annnotation을 이용해 Spring Bean으로 등록해 Singleton으로 구현하였고, 생성자에서 @Autowired를 이용해 Worker Bean을 주입받았다.

action method를 실행하면 Worker의 스레드 업데이트 주기를 70으로 설정하고 스레드를 실행한다.

4.4.3.11. [Controller] FastAction class

FastAction class는 Action interface를 구현한다.



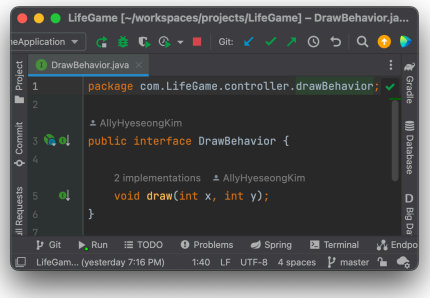
▲ [picture 57] FastAction class

@Component Annnotation을 이용해 Spring Bean으로 등록해 Singleton으로 구현하였고, 생성자에서 @Autowired를 이용해 Worker Bean을 주입받았다.

action method를 실행하면 Worker의 스레드 업데이트 주기를 30으로 설정하고 스레드를 실행한다.

4.4.3.12. [Controller] DrawBehavior interface

LifePanel에서 발생한 MouseEvent를 LifeController가 처리할 때 Strategy Pattern을 활용하여 기본 그리기와 패턴 그리기 중 선택하여 처리할 수 있게 개발하였다. DrawBehavior interface는 Strategy들의 프로토타입을 정의한 인터페이스이다.

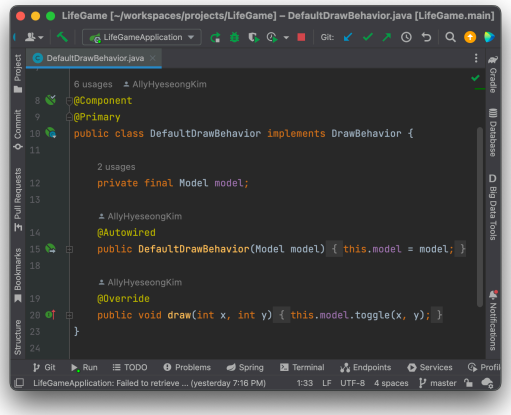


▲ [picture 58] DrawBehavior interface

draw method는 x, y(각각 Model의 map의 row, column)을 입력받아 draw 로직을 수행한다.

4.4.3.13. [Controller] DefaultDrawBehavior class

DefaultDrawBehavior class는 DrawBehavior interface를 구현한다.

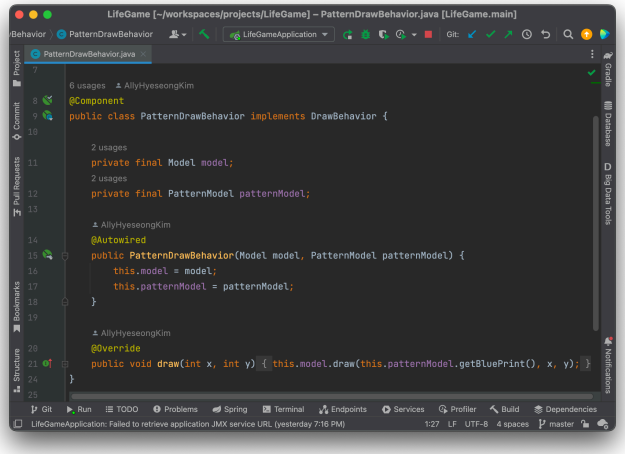


▲ [picture 59] DefaultDrawBehavior class

draw method를 실행하면 기존 동작하던 기능처럼 클릭한 위치의 칸을 toggle하는 기능을 수행한다.

4.4.3.14. [Controller] PatternDrawBehavior class

PatternDrawBehavior class는 DrawBehavior interface를 구현한다.

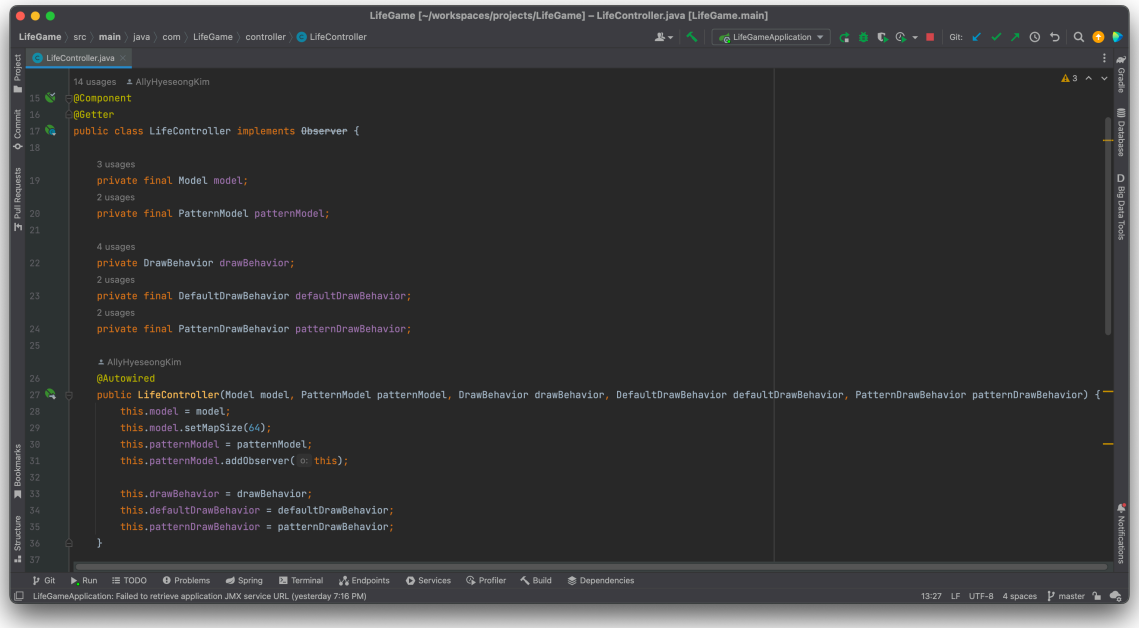


▲ [picture 60] PatternDrawBehavior class

draw method를 실행하면 Model의 x, y파라미터에 해당하는 위치에 patternModel에 등록된 pattern을 그리게 된다.

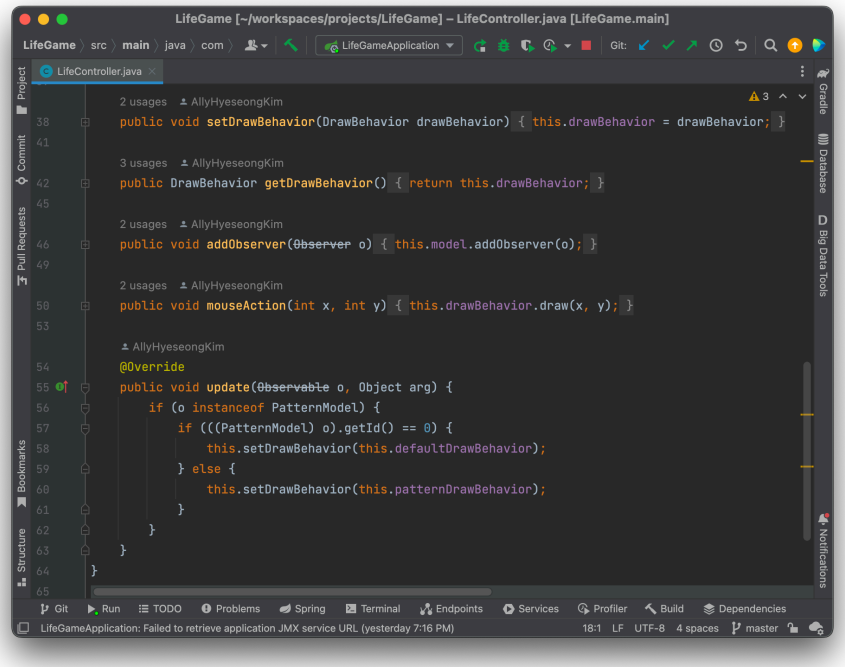
4.4.3.15. [Controller] LifeController class

LifeController class는 java.util.Observer를 구현한다. LifePanel에서 발생하는 이벤트를 핸들링하는 역할을 갖고 있으며, @Component Annotation을 이용해 Spring Bean으로 등록하고 Singleton으로 동작한다.



▲ **[picture 61] LifeController class** (1)

Model, PatternModel, DrawBehavior들의 의존성을 주입받는다.



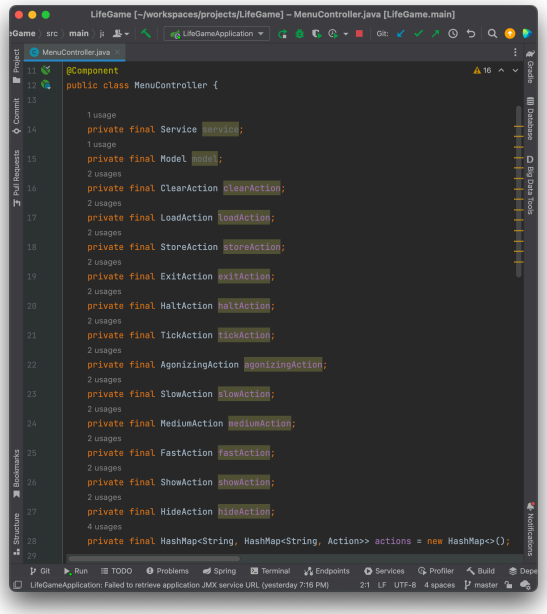
▲ **[picture 62] LifeController class** (2)

Strategy Pattern에서 Strategy를 설정하기 위한 setDrawBehavior method가 정의되어 있고, Model에 Observer를 추가하는 addObserver, mouseAction을 핸들링하여 적절한 Strategy를 실행하는 mouseAction method가 정의되어 있다.

PatternModel이 변할 때 마다 update method가 실행되고, 그 때마다 적절한 Strategy를 선택한다.

4.4.3.16. [Controller] MenuController class

MenuController는 Menu 관련 뷰에서 발생하는 모든 이벤트를 핸들링한다.

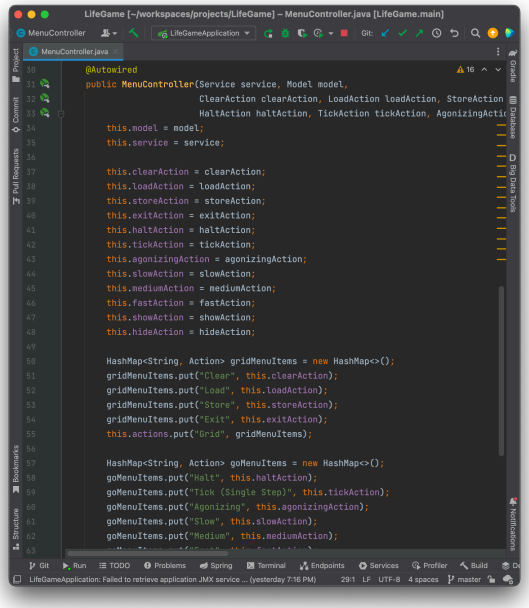


▲ **[picture 63] MenuController class** (1)

@Component Annnotation을 이용해 Spring Bean으로 등록해 Singleton으로 동작한다.

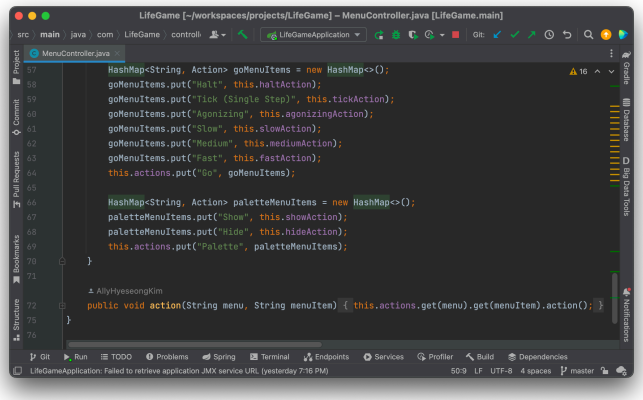
Service, Model 및 Action interface를 구현한 모든 Action들을 주입받는다.

메뉴 정보를 저장하는 actions HashMap을 갖고 있으며 Model의 역할도 일정 부분 함께 수행한다.



▲ **[picture 64] MenuController class** (2)

생성자에서 @Autowired Annotation을 이용해 의존성을 주입받고, HashMap에 Menu와 Action을 등록한다.

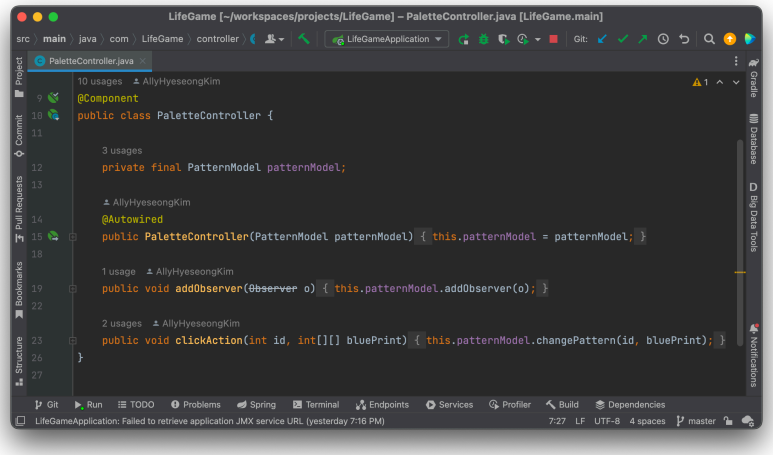


▲ **[picture 65] MenuController class** (3)

Menu에서 발생하는 모든 액션을 핸들링하는 action method를 갖고 있고, 파라미터로 넘겨받은 menu, menuItem 이름을 HashMap에 조회하여 적절한 Action을 실행한다.

4.4.3.17. [Controller] PaletteController class

PaletteController는 PatternPanel 뷰에서 발생하는 모든 이벤트를 핸들링한다.



▲ **[picture 66] PaletteController class**

@Component Annnotation을 이용해 Spring Bean으로 등록해 Singleton으로 동작한다.

생성자에서는 @Autowired Annotation을 이용해 PaletteModel 의존성을 주입받는다.

addObserver method를 이용해 view에서 PaletteModel에 옵저버로 등록할 수 있다.

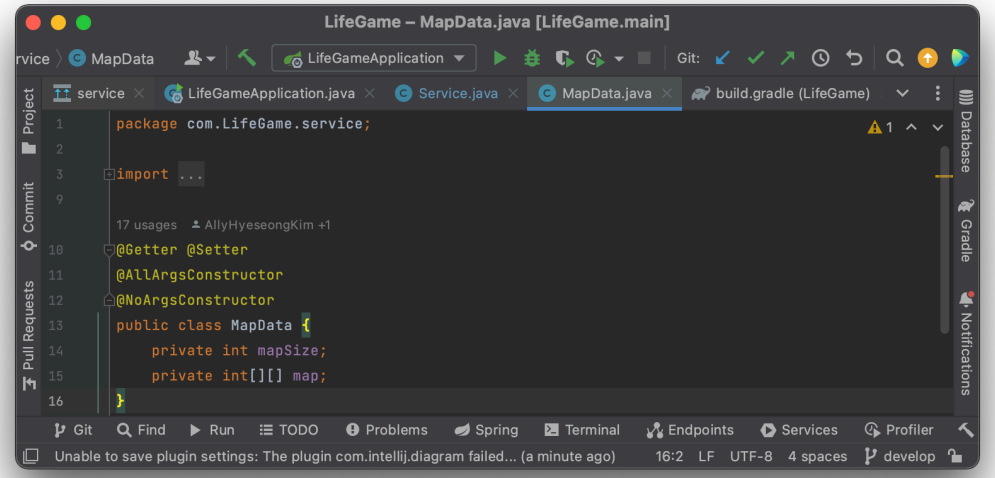
clickAction method는 PatternPanel에서 발생한 클릭 이벤트들을 핸들링하며, 파라미터로 전달받은 bluePrint를 모델에 적용한다.

4.5. Service

Controller가 view에서 넘어오는 매개변수를 이용하여 service 객체를 호출할 수 있다. Service 는 view 에 종속적인 코드가 없어 필요한 매개변수만 받으면 재사용이 가능하다. MapData 객체를 매개변수로 받아서 맵의 상태를 저장, 불러오기하는 기능을 service 에서 담당한다.

4.5.1. [Service] MapData class

전달받는 매개변수는 MapData 객체로 int 형의 mapSize 와 실제 toggle 된 좌표의 2차원 배열을 담고 있는 map 변수를 갖고 있다.

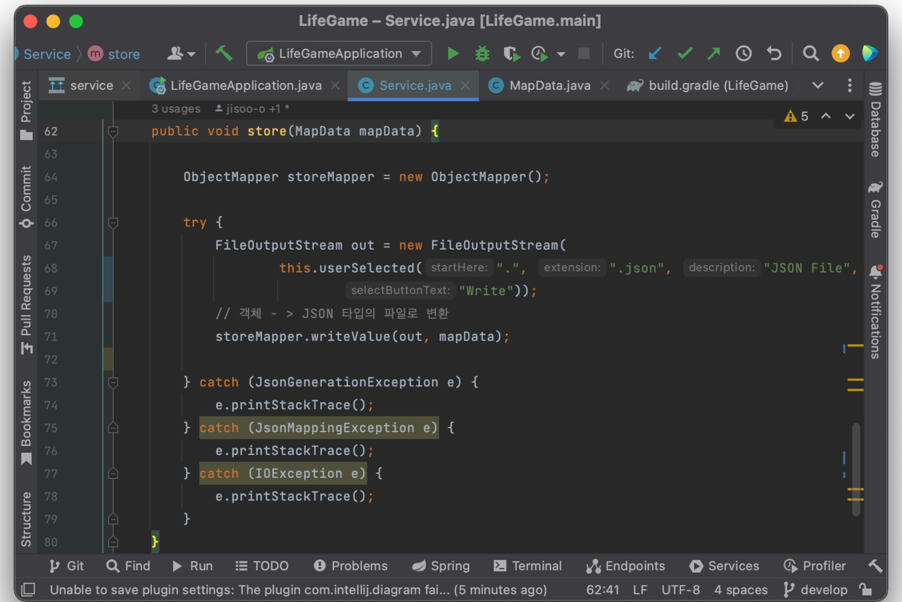


▲ **[picture 67] MapData class**

4.5.2. [Service] Service class

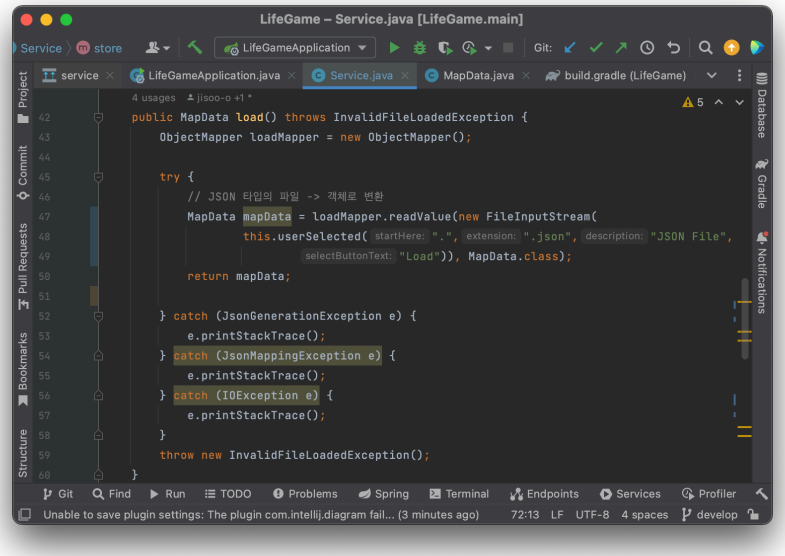
store, load 메소드를 구현하는 클래스로 store 는 MapData 객체를 전달 받아서 해당 객체를 Json 파일로 변환하여 저장한다. load 는 저장했던 Json 타입의 파일을 다시 MapData 객체로 변환하여 맵에 불러온다. 타입 변환은 ObjectMapper를 활용한다. 파일 선택 UI 에서 user 가 선택한 파일, 경로를 userSelected 로 받아 해당 파일을 불러오거나 경로에 저장한다.

▲ [picture 68] 변환되어 저장된 결과물



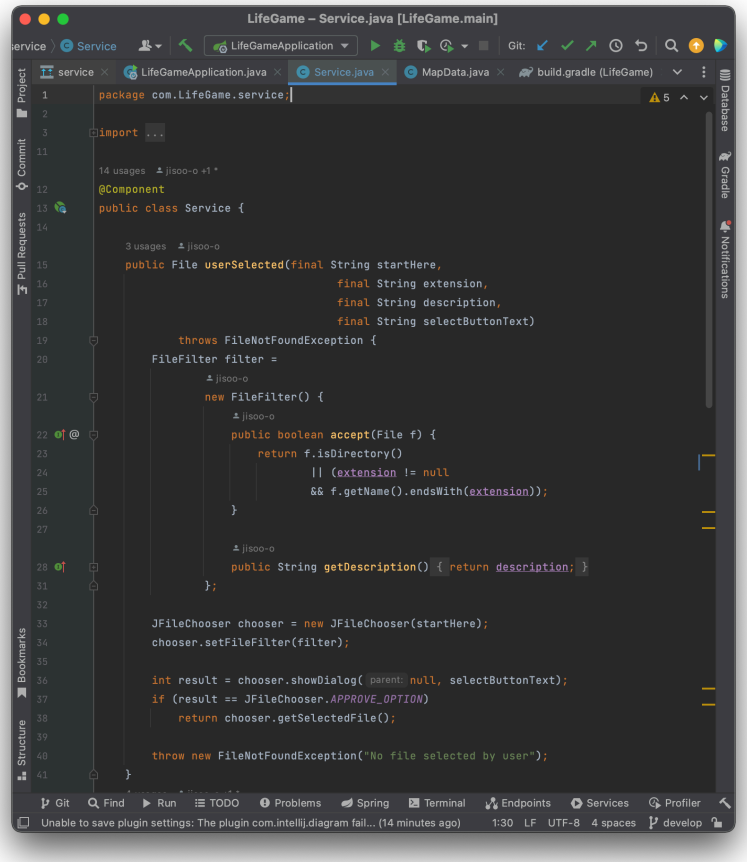
▲ [picture 69] Service class (1)

정보를 store할 파일의 경로를 filechooser UI를 통해 userSelected로 받아 FileOutputStream로 지정한다. ObjectMapper인 storeMapper의 writeValue 메소드를 이용하며 전달받은 mapData 객체를 Json 타입 파일로 변환하여 저장한다.



▲ [picture 70] Service class (2)

load하고자 하는 파일의 경로를 같은 방법으로 받아 FileInputStream으로 지정하고 loadMapper의 readValue 메소드를 이용하여 다시 MapData 객체로 변환시켜서 int mapSize 와 int[][] map 변수값을 받아온다.

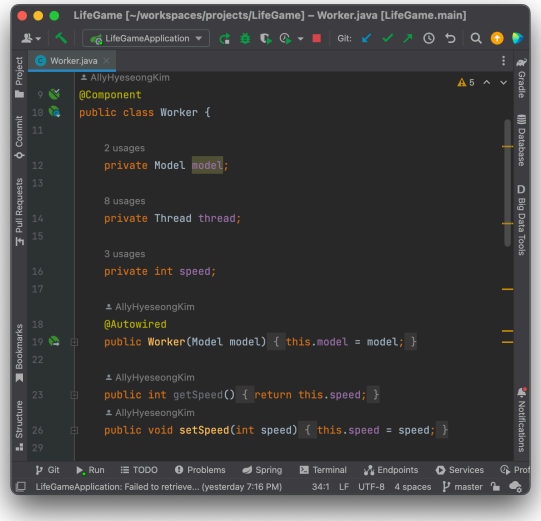


▲ [picture 71] Service class (3)

4.6. Worker

4.6.1. [Worker] Worker class

Worker class는 Model의 map을 설정한 주기로 업데이트하는 스레드를 관리하는 클래스이다.



▲ **[picture 72] Worker class (1)**

@Component Annnotation을 이용해 Spring Bean으로 등록해 Singleton으로 동작한다.

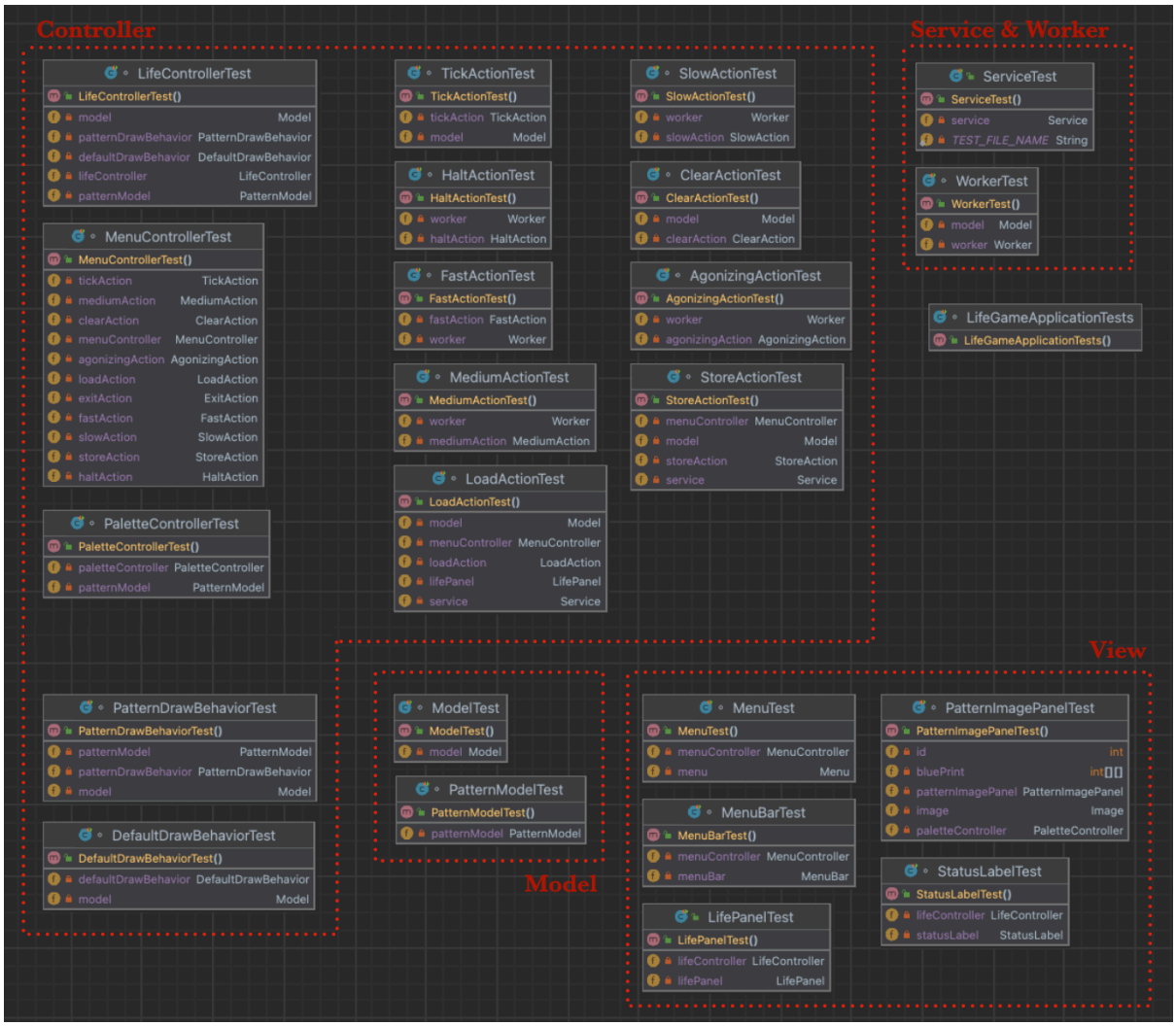
setSpeed method를 이용해 스레드가 모델을 업데이트하는 주기를 설정한다.

▲ **[picture 73] Worker class (2)**

startThread method를 이용해 설정한 주기마다 nextState를 호출하는 스레드를 동작시킨다. 스레드가 sleep하는 동안 클릭 이벤트에 의해 Model이 업데이트 되어야 하기 때문에 nextState method는 synchronized로 동작한다.

4.7. Test Cases

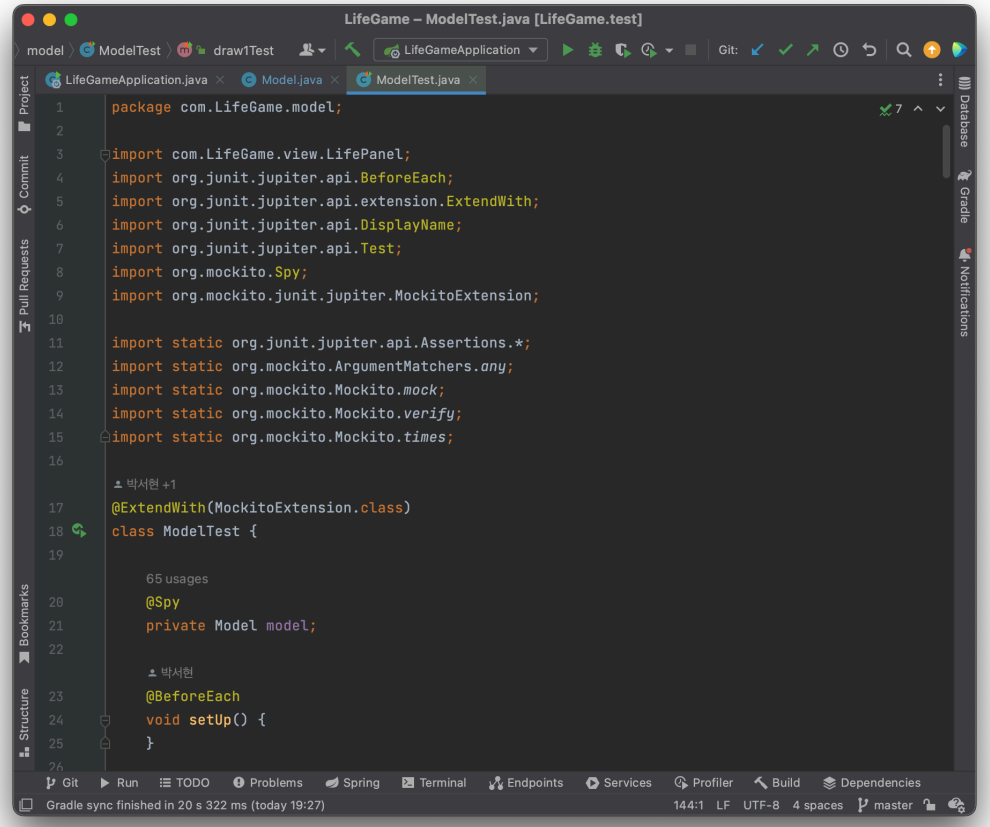
JUnit을 활용하여 모든 class에 대하여 테스트를 수행하였다.



▲ **[picture 74] classes of test cases**

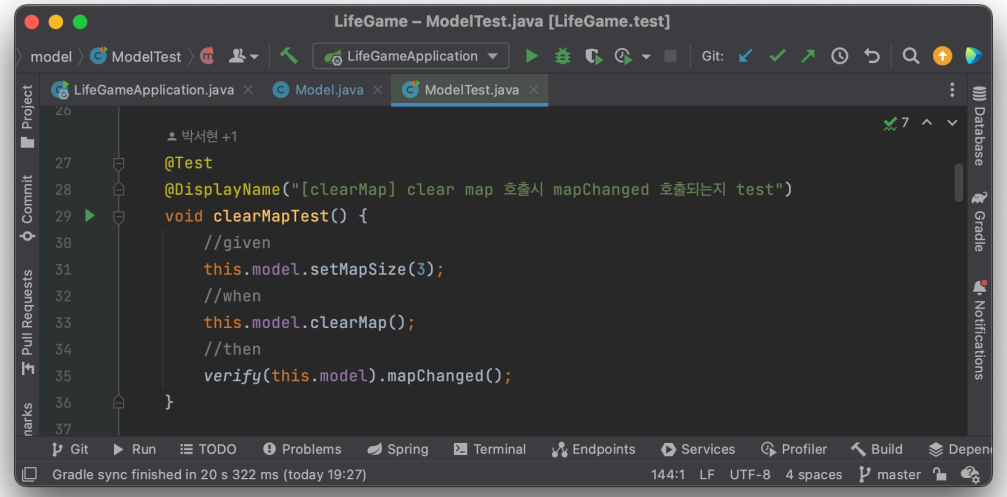
4.7.1. [Test] ModelTest

Model class를 테스트하는 class이다. UI와는 분리하여 Model class에 대한 기능만을 테스트한다.



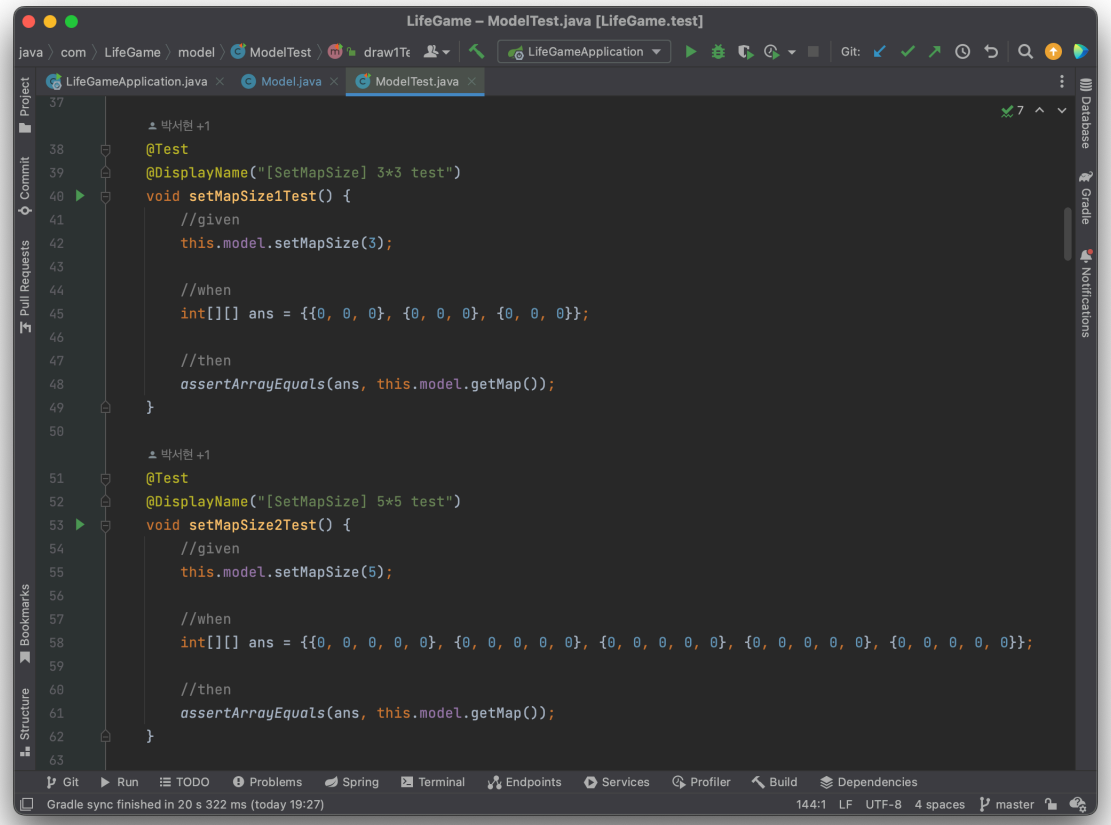
▲ [picture 75] Test cases of Model class (1)

필요한 부분에만 stub하여 사용하기 위해 model을 mock 대신 @Spy로 선언한다.



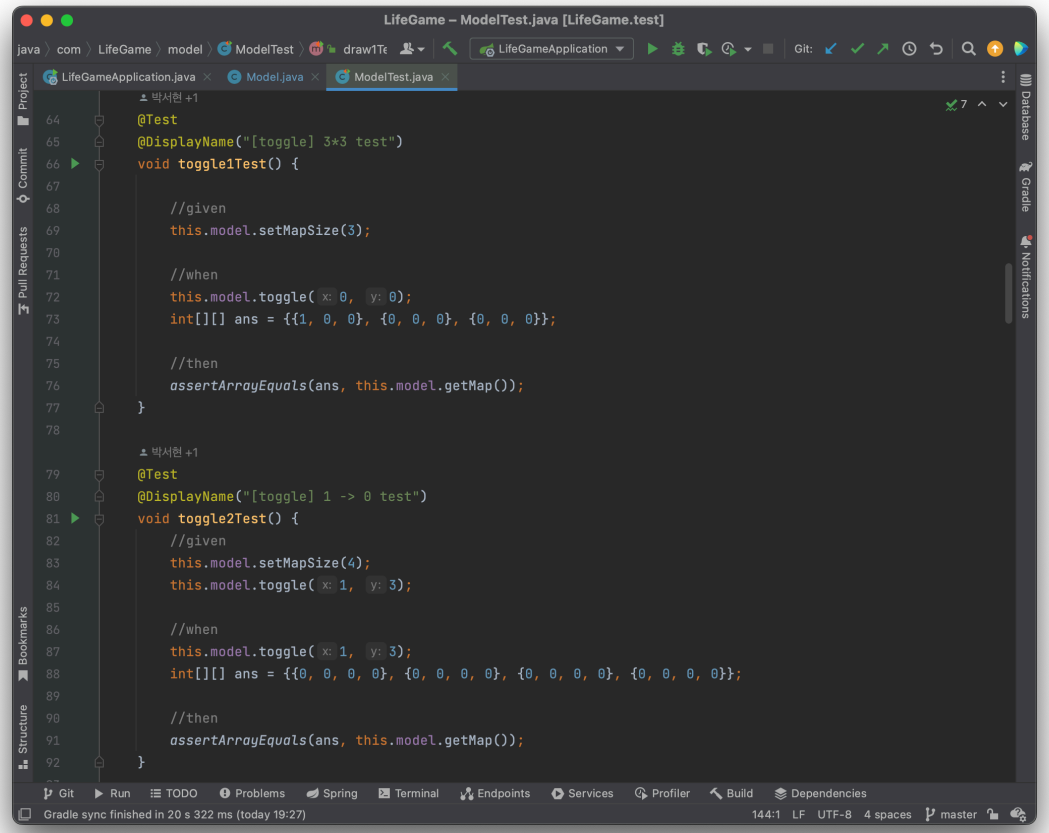
▲ [picture 76] Test cases of Model class (2)

clearMapTest로 clear map을 호출했을 때 mapChanged가 잘 호출되는지 verify를 활용하여 테스트한다.



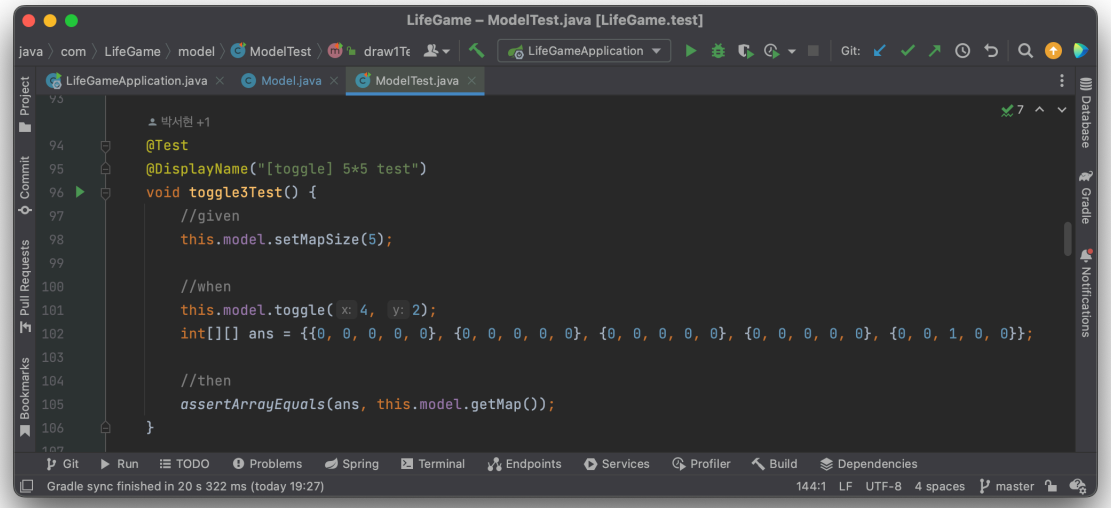
▲ **[picture 77] Test cases of Model class** (3)

setMapSize1Test와 setMapSize2Test로 변수를 각각 3, 5를 넣었을 때 해당하는 크기의 배열이 만들어지는지 assertArrayEquals로 테스트한다.



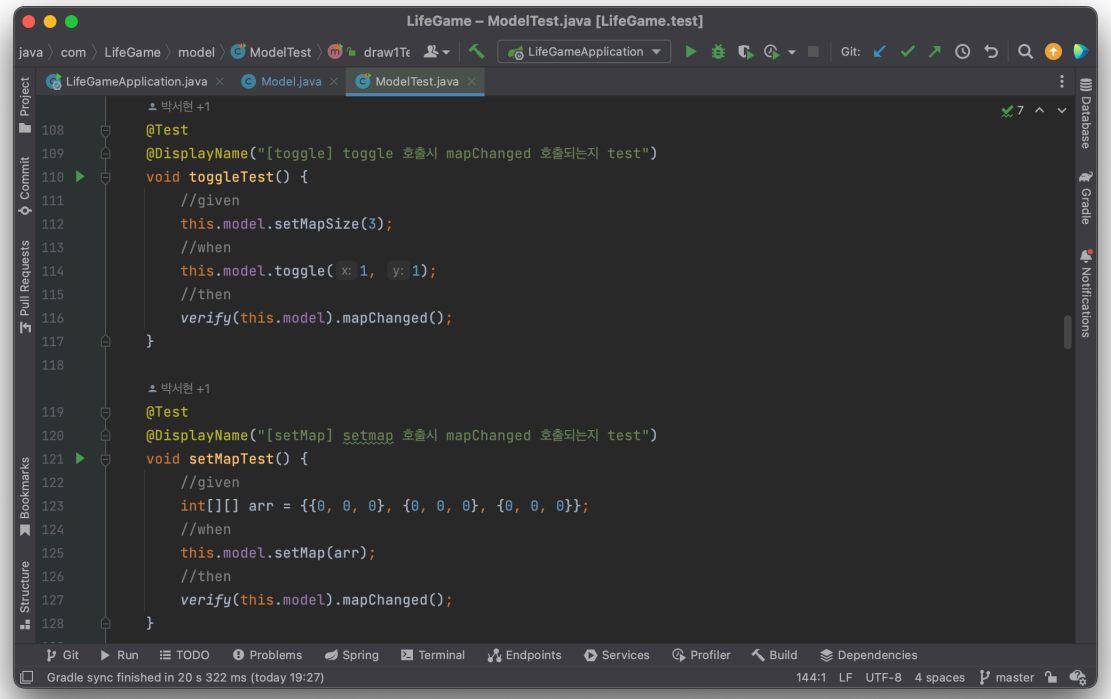
▲ **[picture 78] Test cases of Model class** (4)

toggle1Test로 3*3 크기의 map에서 좌표를 지정했을 때 값이 바뀌는지 확인하고, toggle2Test에서 4*4 크기의 map에서 값이 1인 인덱스가 0으로 바뀌는지 assertArrayEquals로 확인한다.



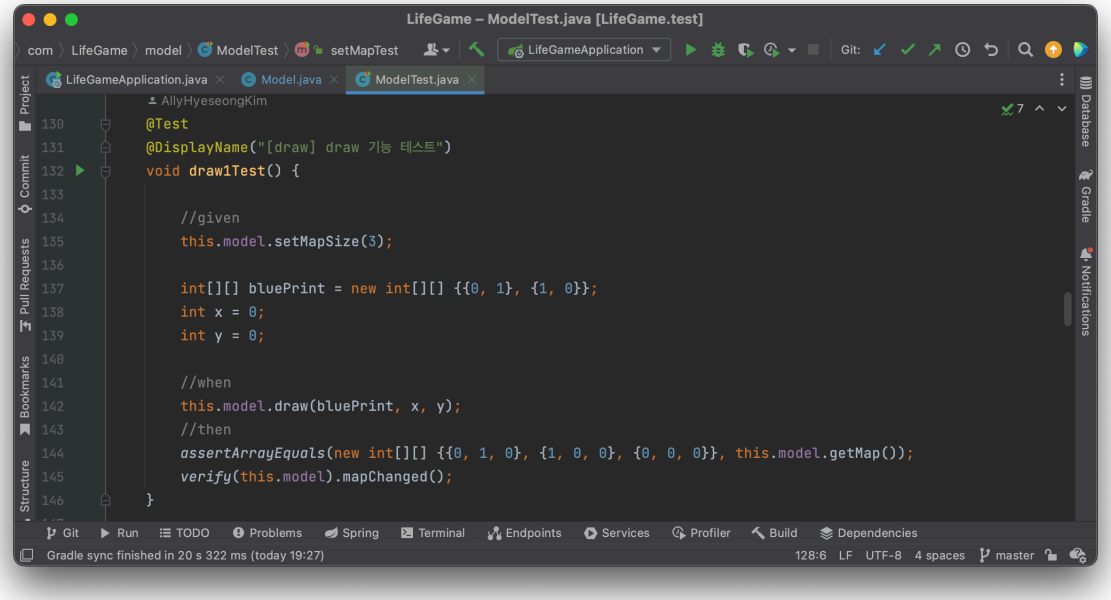
▲ **[picture 79] Test cases of Model class** (5)

toggle3Test로 5*5 크기의 map에서 지정한 좌표의 값이 바뀌는지 assertArrayEquals로 확인한다.



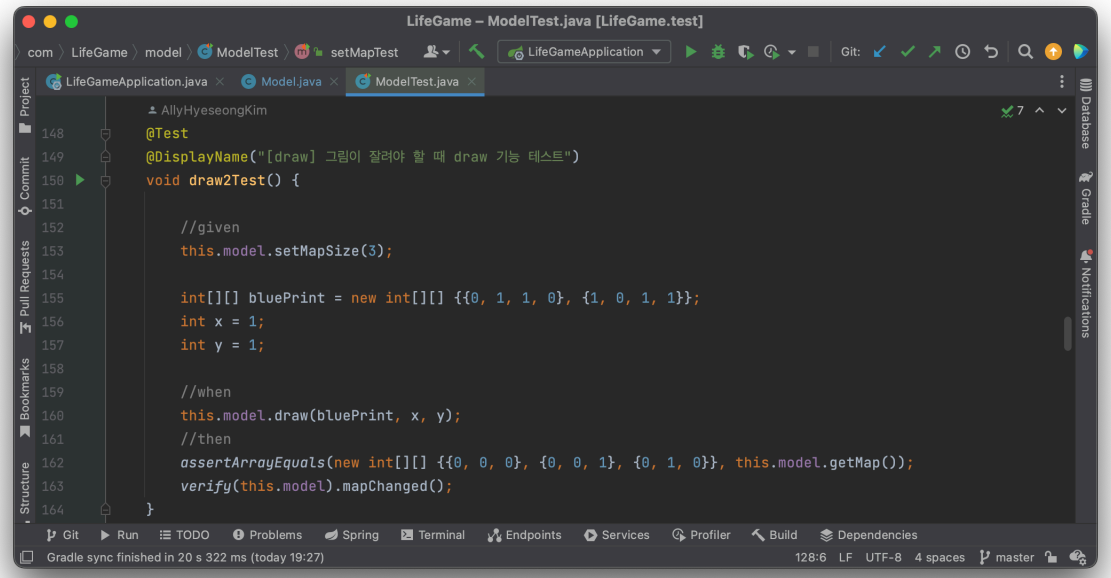
▲ **[picture 80] Test cases of Model class** (6)

toggleTest와 setMapTest로 toggle이 호출되면 내부의 mapChanged가 호출되는지 verify를 통해 확인한다.



▲ **[picture 81] Test cases of Model class** (7)

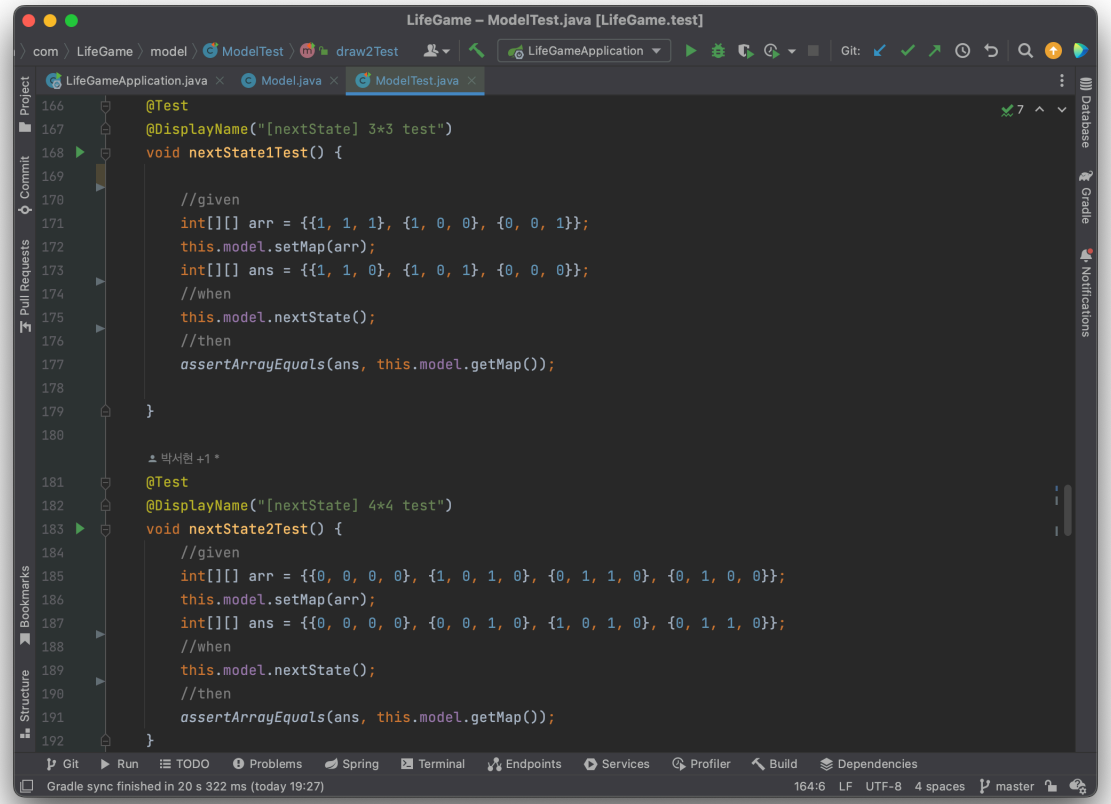
draw1Test로 draw 기능이 잘 동작하는지 assertArrayEquals로 확인하고, mapChanged가 호출되는지 verify로 확인한다.



▲ [picture 82] Test cases of Model class (8)

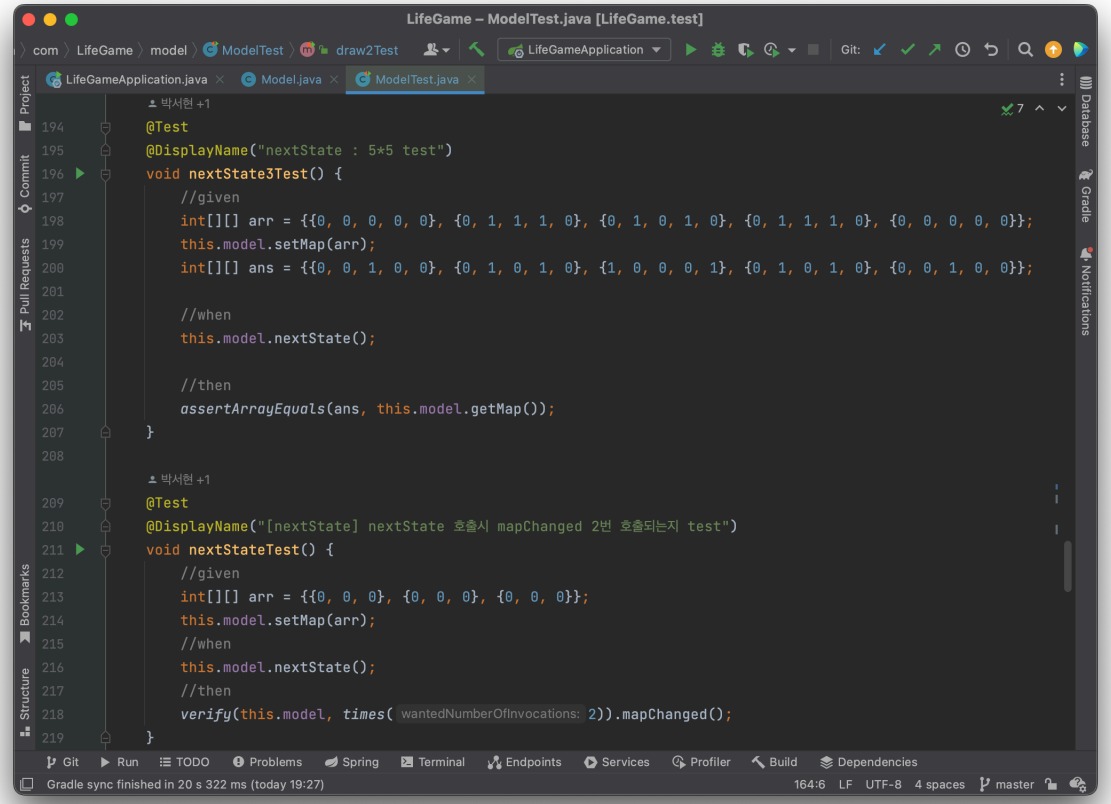
Model class 테스트하는 class이다.

draw2Test로 크기나 벽이 있어 패턴이 모두 들어가지 못할 때 draw 기능이 동작하는지 확인하기 위해 assertArrayEquals로 비교하고 verify로 mapChanged가 호출되는지 테스트한다.



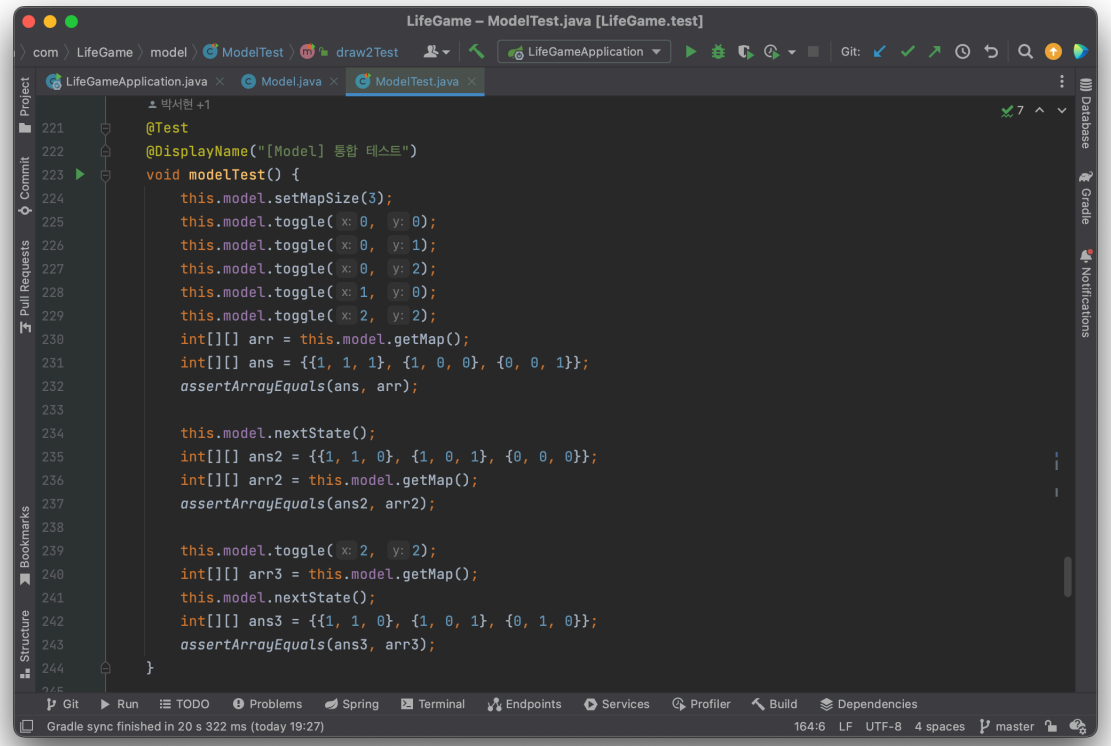
▲ [picture 83] Test cases of Model class (9)

nextState1Test와 nextState2Test로 현재 배열에서 다음 상태로 변한 값을 assertArrayEquals로 비교한다.



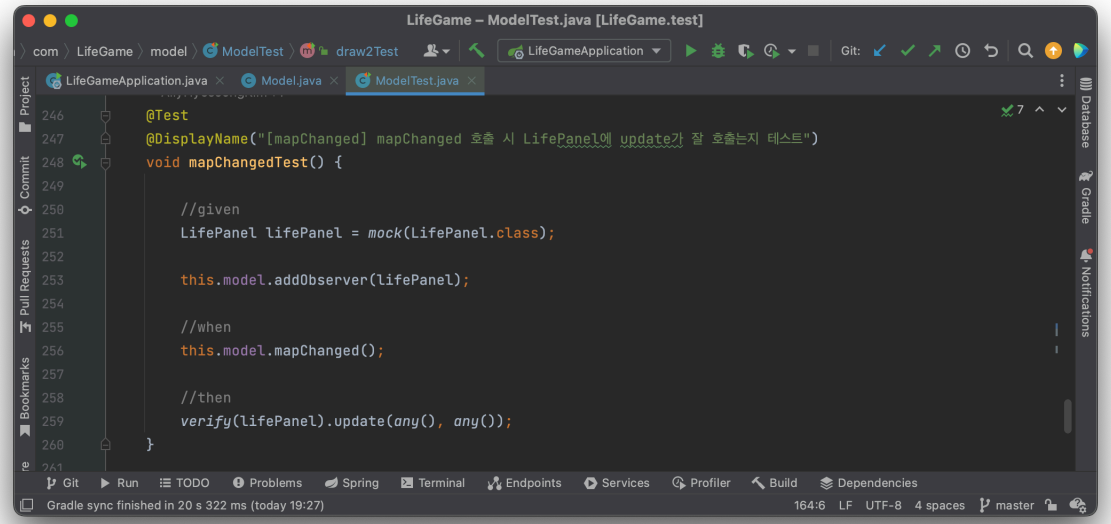
▲ **[picture 84] Test cases of Model class (10)**

nextState3Test로 다음 상태의 값을 assertArrayEquals로 비교하고, nextStateTest에서 mapChanged가 잘 호출되는지 verify로 확인한다.



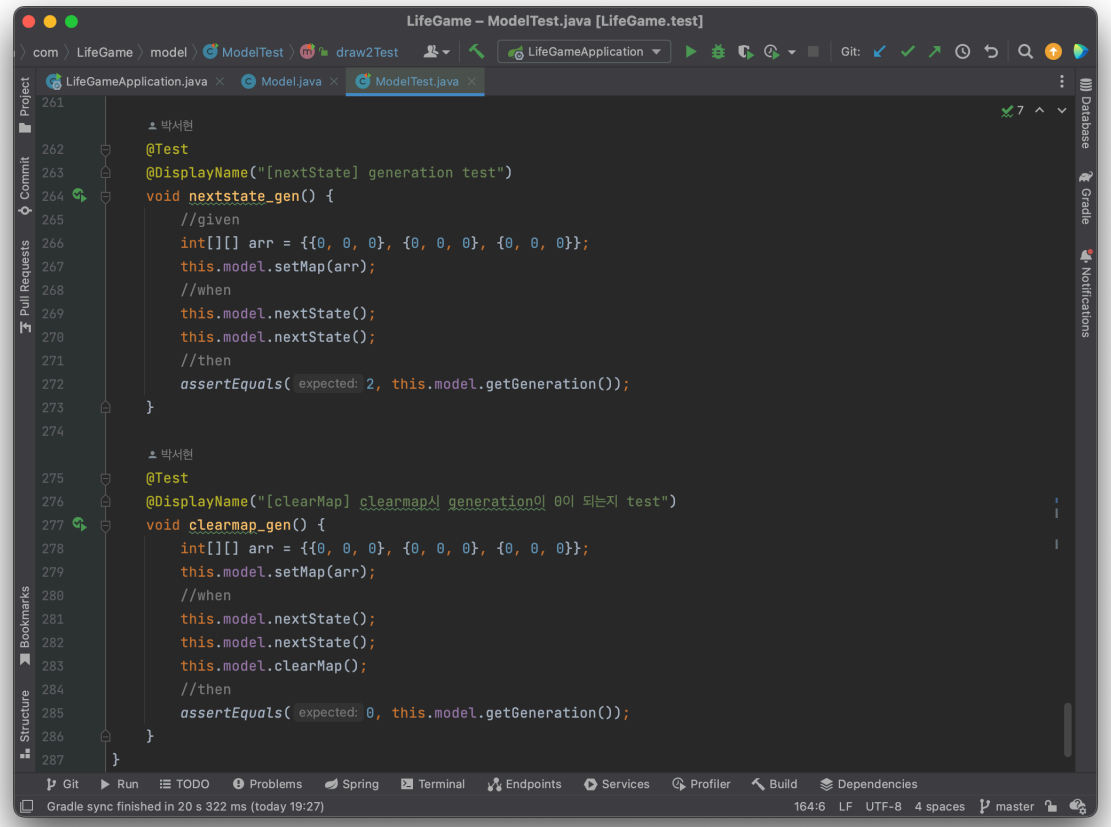
▲ [picture 85] Test cases of Model class (11)

modelTest로 setMapSize, toggle, getMap, nextState의 기능이 이어서 동작하는지 assertArrayEquals로 확인한다.



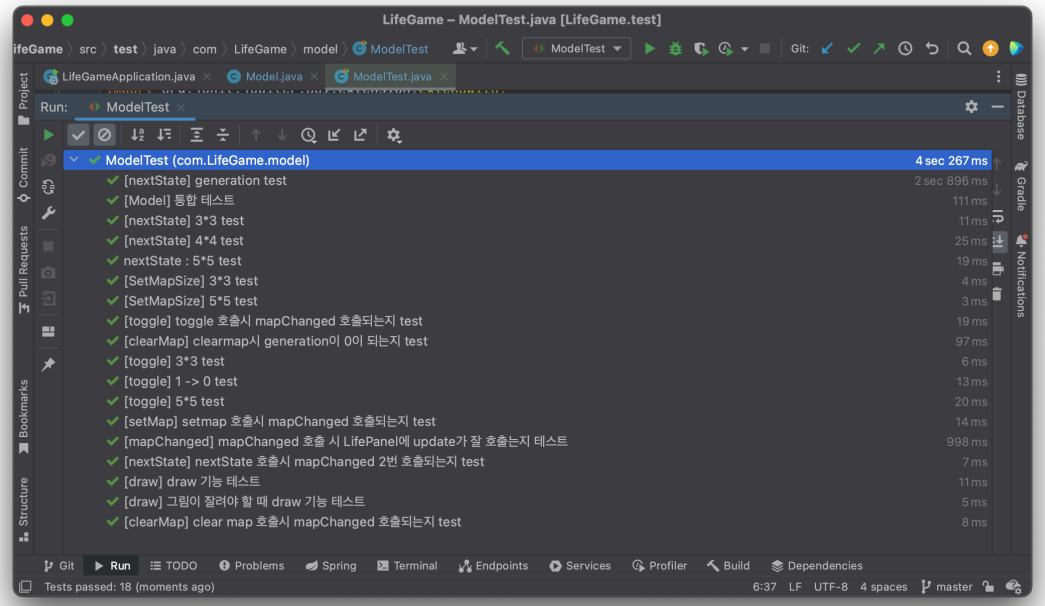
▲ [picture 86] Test cases of Model class (12)

mapChangedTest로 mock 객체를 만들고 함수가 호출되었을 때 update가 호출되는지 verify로 확인한다.



▲ [picture 87] Test cases of Model class (13)

nextstate_gen로 nextState가 호출되었을 때 generation이 증가하는지, clearmap_gen으로 clearMap이 호출되었을 때 generation이 초기화 되는지 assertEquals로 비교한다.

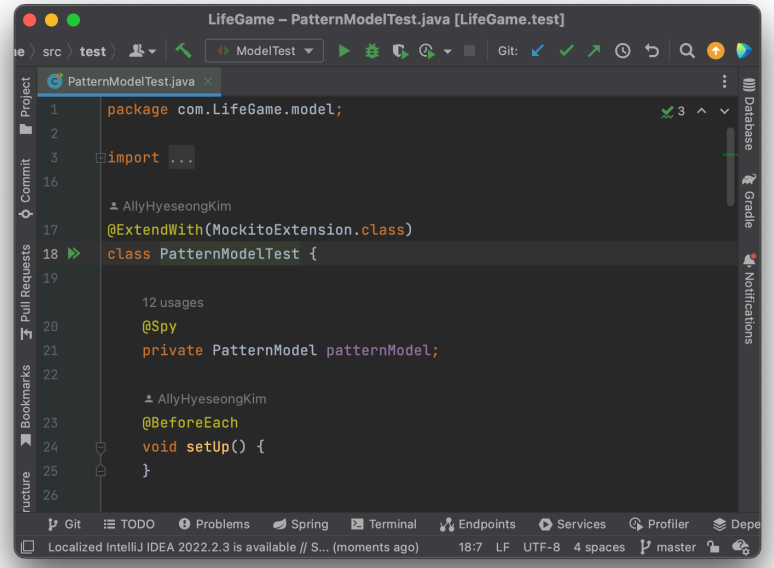


▲ [picture 88] ModelTest passed

모든 ModelTest가 통과한 것을 볼 수 있다.

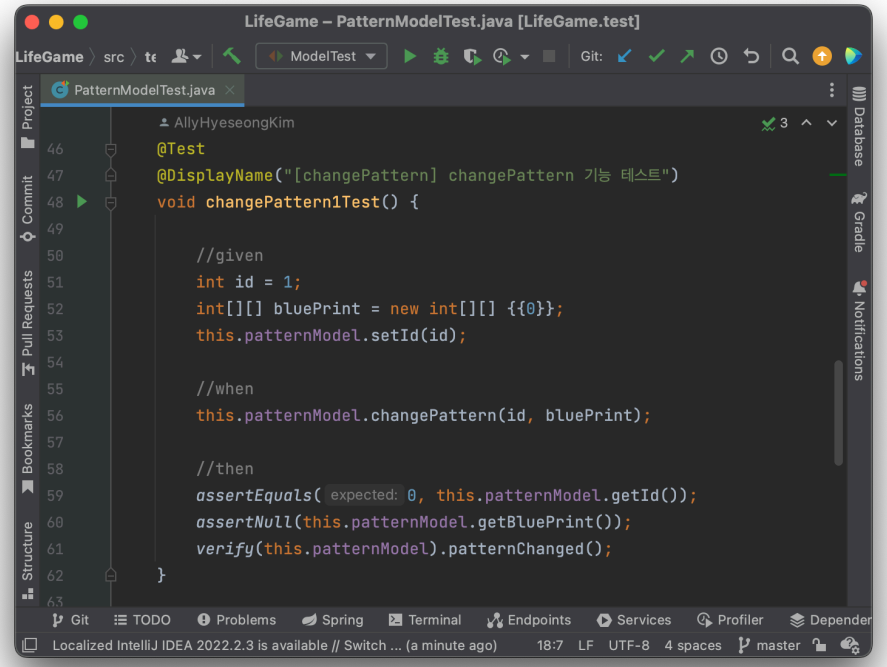
4.7.2. [Test] PatternModelTest

PatternModel class 테스트하는 class이다.



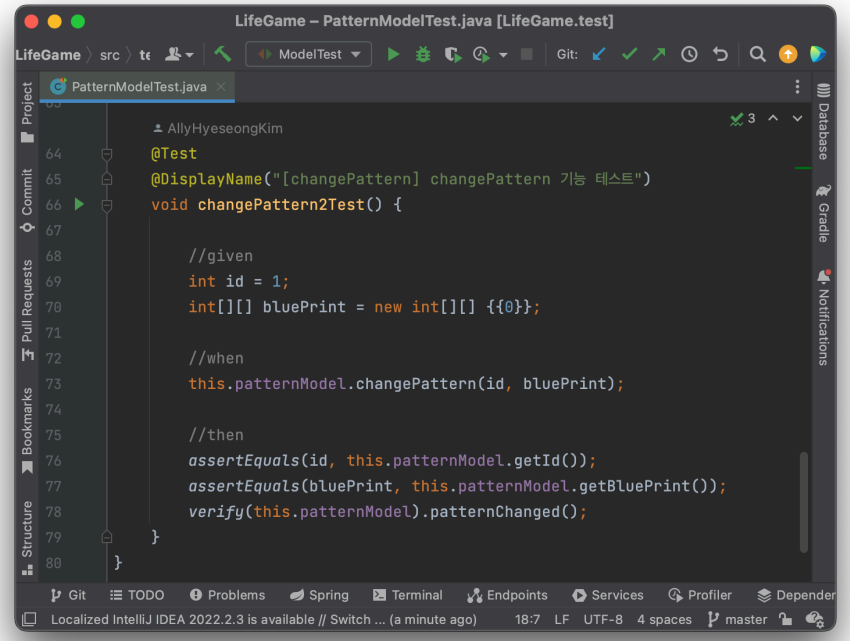
▲ **[picture 89] Test cases of PatternModel class (1)**

PatternModel에 @Spy Annotation을 이용해 verify와 when을 이용할 수 있도록 하였다.



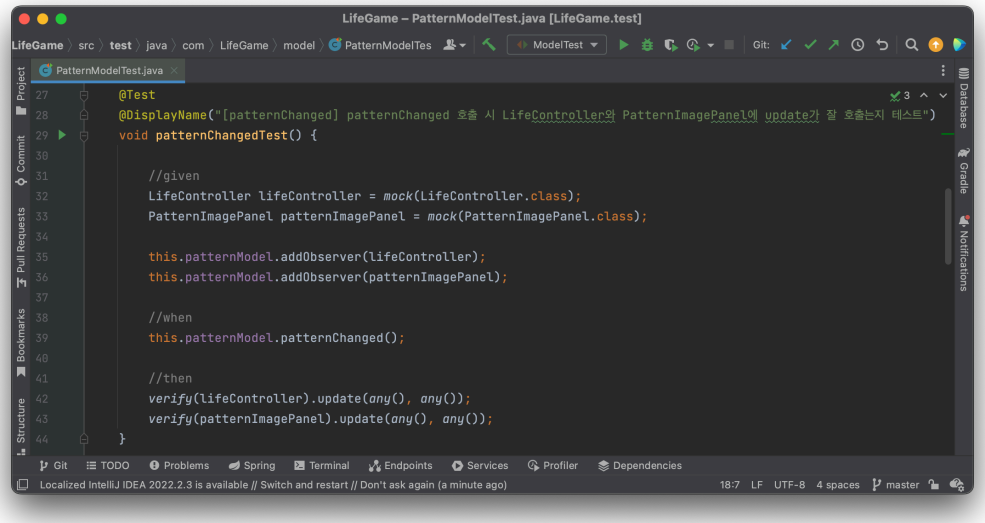
▲ **[picture 90] Test cases of PatternModel class (2)**

changePattern1Test는 changePattern method를 실행했을 때 이미 설정된 id로 패턴이 설정되는 경우 bluePrint를 Null로 설정하고 id를 0으로 잘 설정하는지 테스트한다.



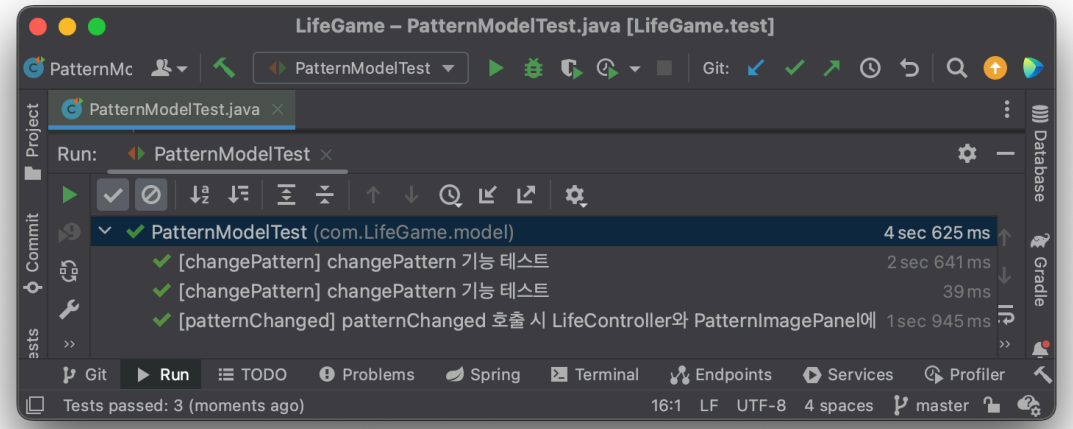
▲ **[picture 91] Test cases of Pattern Model class** (3)

changePattern2Test는 기존 id와 다른 id를 설정할 때 설정한 id 및 bluePrint가 잘 설정 되는지, patternChanged가 잘 호출되는지 테스트한다.



▲ **[picture 92] Test cases of PatternModel class** (4)

patternChangedTest는 patternChanged를 호출하는 경우 Observer들의 Update가 잘 호출되는지 테스트한다.

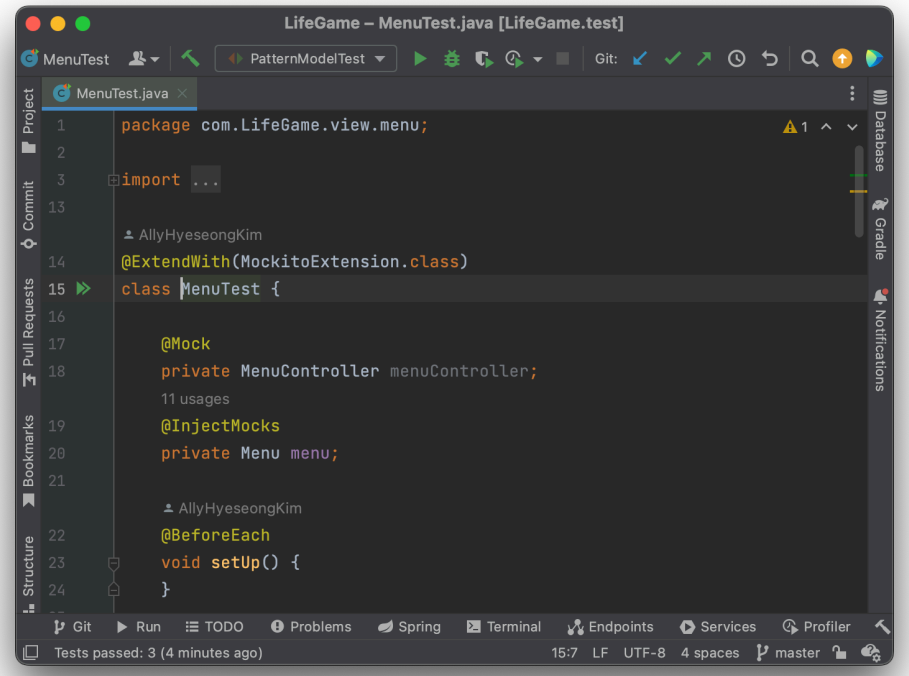


▲ [picture 93] PatternModelTest Passed

Test가 모두 통과하였음을 확인할 수 있다.

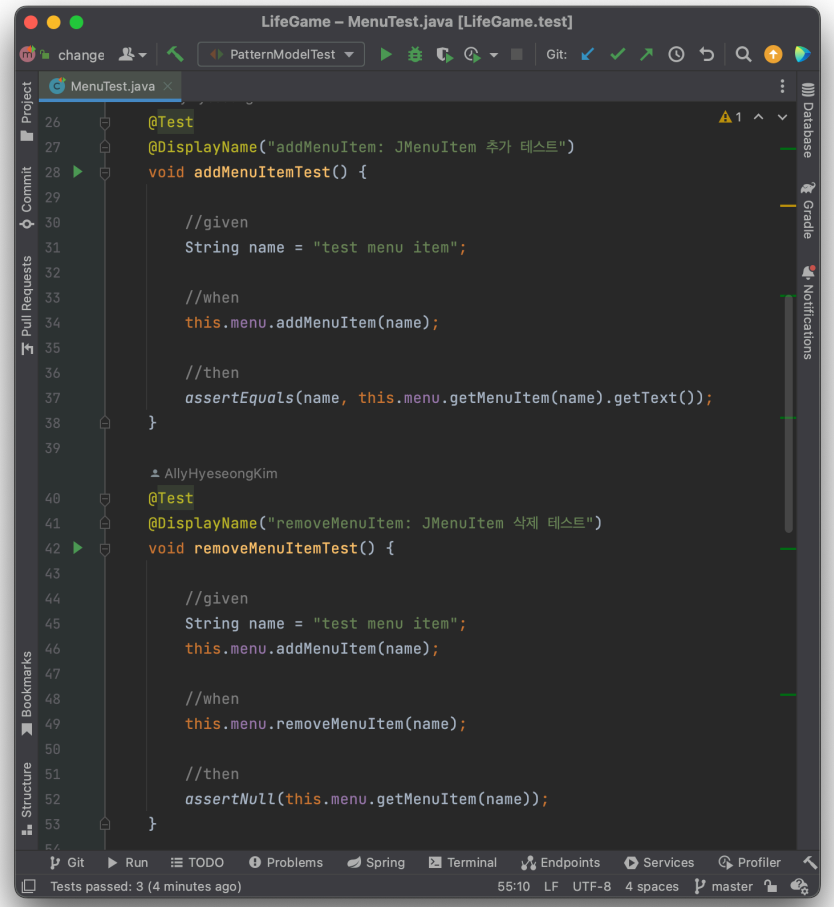
4.7.3. [Test] MenuTest

Menu class 테스트하는 class이다.



▲ [picture 94] Test cases of Menu class (1)

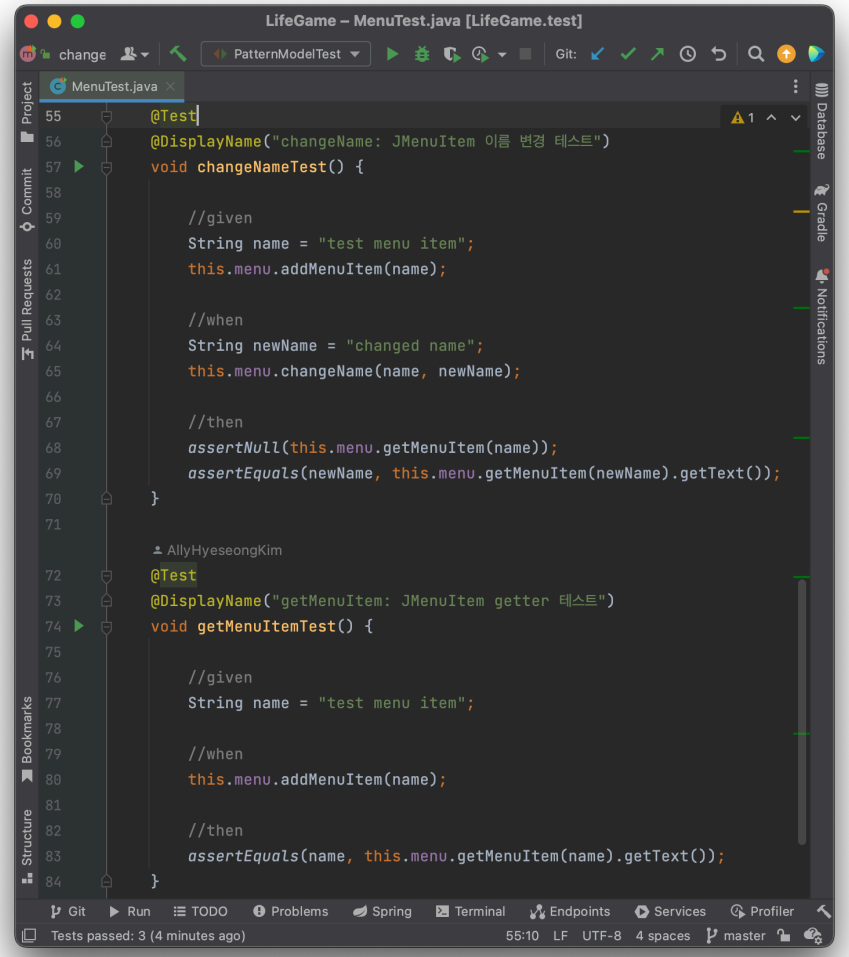
Menu를 독립적으로 테스트하기 위해 Mockito의 @Mock을 이용해 MenuController를 Mocking해 Menu에 주입하였다.



▲ **[picture 95] Test cases of Menu class** (2)

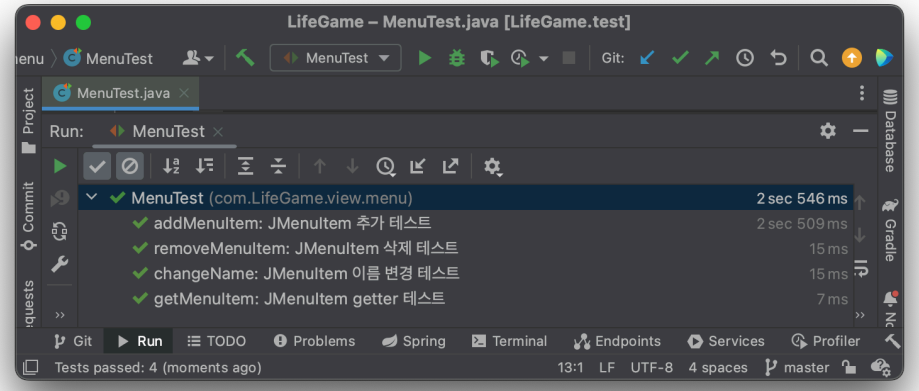
addMenuItemTest는 addMenuItem을 호출했을 때 MenuItem이 잘 생성되었는지 Text를 비교하여 검증하는 테스트이다.

removeMenuItemTest는 removeItem을 호출했을 때 MenuItem을 잘 삭제하는지 테스트한다. getMenuItem을 호출했을 때 Null이 반환되는지 확인한다.



▲ [picture 96] Test cases of Menu class (3)

changeNameTest는 changeName을 호출했을 때 MenuItem의 이름이 잘 변하는지 테스트한다. changeName을 호출하고 기존 이름의 MenuItem을 get 했을 때 Null인지, 새로운 이름으로 get 했을 때 설정한 Text가 적용되었는지 검증한다.

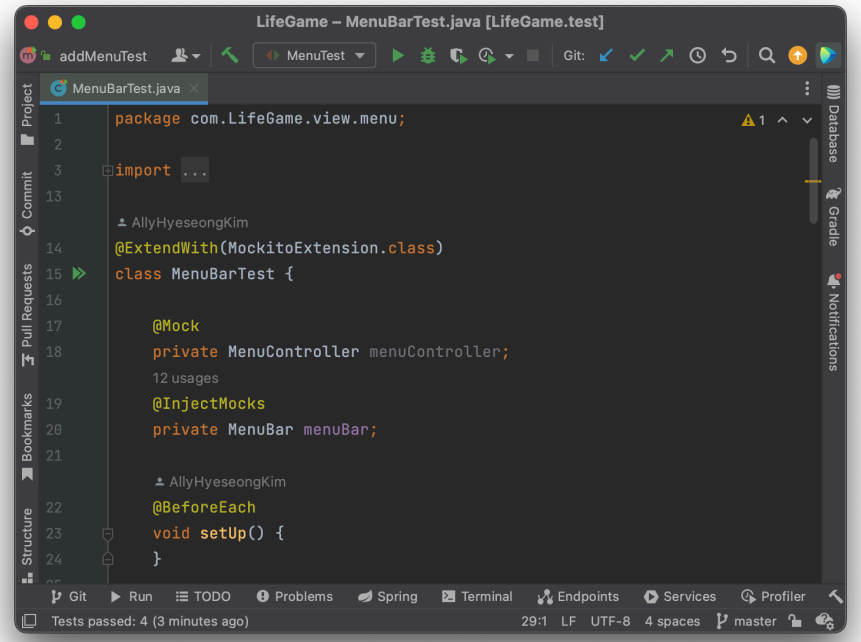


▲ [picture 97] MenuTest Passed

모든 Test가 통과하였음을 알 수 있다.

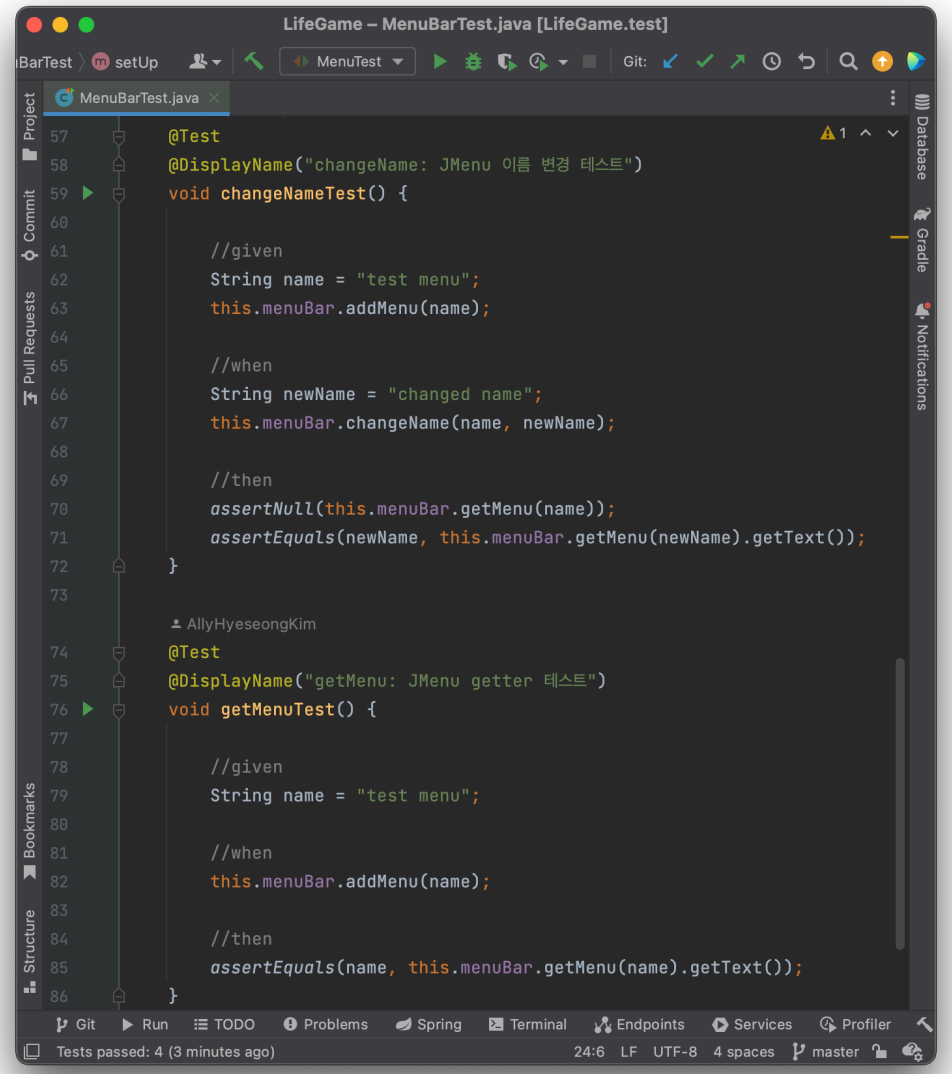
4.7.4. [Test] MenuBarTest

MenuBar class 테스트하는 class이다.



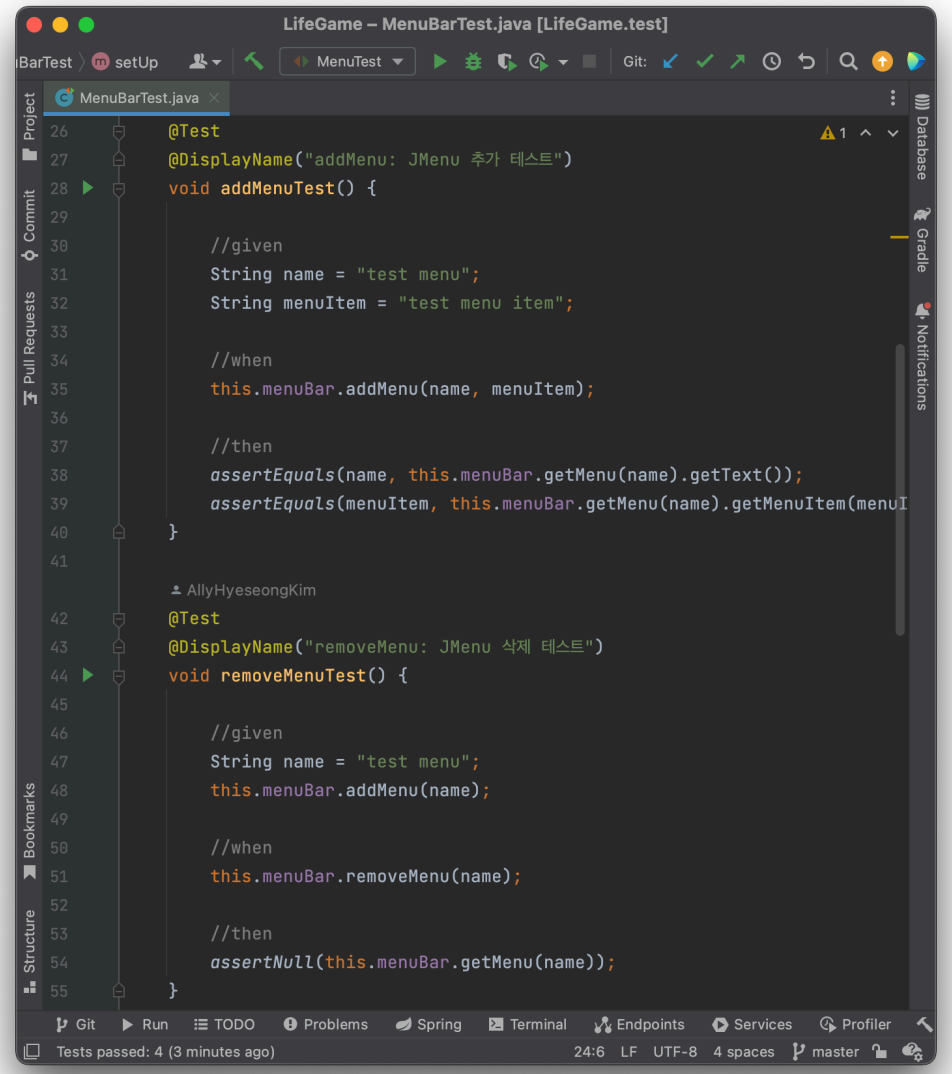
▲ **[picture 98] Test cases of MenuBar class (1)**

MenuBar를 독립적으로 테스트하기 위해 Mockito @Mock을 이용해 MenuController를 Mocking하고 MenuBar에 주입하여 테스트 하였다.



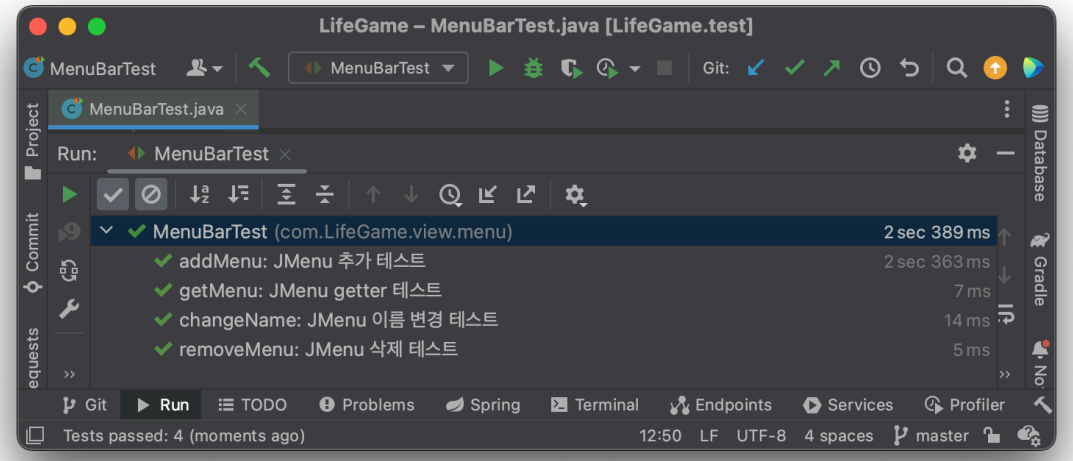
▲ **[picture 99] Test cases of MenuBar class** (2)

changeNameTest는 changeName method를 실행하였을 때 기존 name의 메뉴가 null인지, 바뀐 이름의 메뉴가 존재하는지 테스트한다.
getMEnuTest는 getMenu method를 실행하였을 때 올바른 Menu가 반환되는지 테스트한다.



▲ **[picture 100] Test cases of MenuBar class** (3)

addMenuTest는 addMenu를 호출하였을 때 설정한 name과 menuItem 인스턴스가 설정되었는지 assertEquals를 이용해 검증한다.
removeMenuTest는 removeMenu를 호출하였을 때 name에 해당하는 menu가 삭제되어 get했을 때 null이 반환되는지 assertNull로 체크한다.

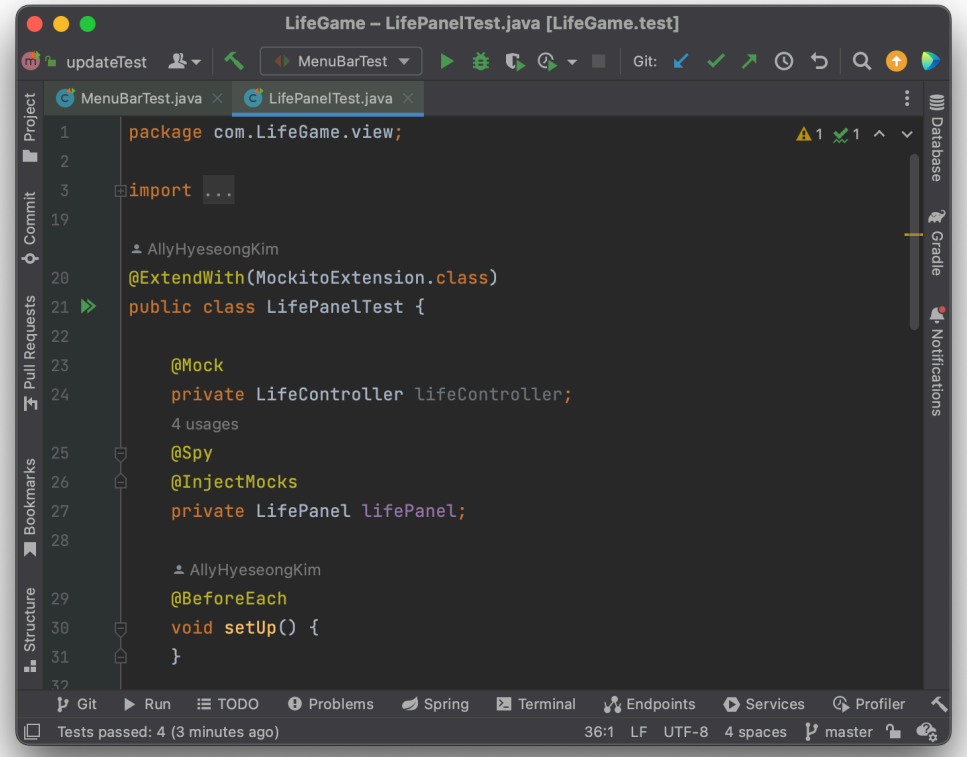


▲ [picture 101] MenuBarTest Passed

MenuBarTest의 모든 Test가 통과하는 것을 확인할 수 있다.

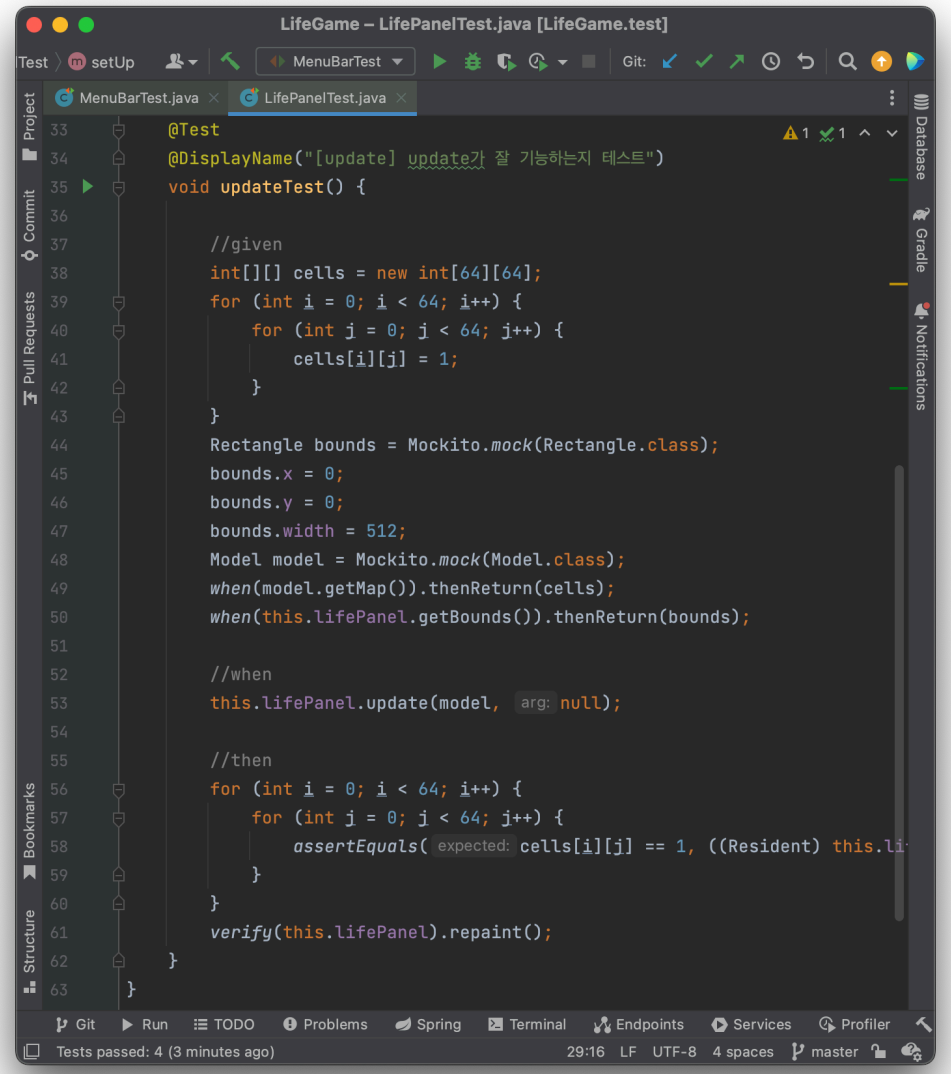
4.7.5. [Test] LifePanelTest

LifePanel class 테스트하는 class이다



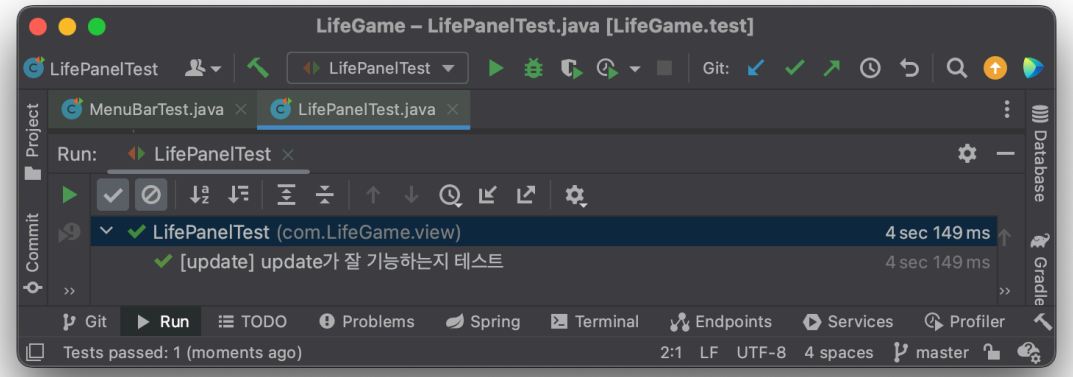
▲ [picture 102] Test cases of LifePanel class (1)

LifePanel을 독립적으로 테스트하기 위해 LifeController를 Mocking 하여 주입하였다.



▲ **[picture 103] Test cases of LifePanel class** (2)

updateTest는 update가 호출되었을 때 올바른 Resident의 amAlive가 1로 설정 되었는지, repaint가 한번만 잘 실행되었는지 검증한다.

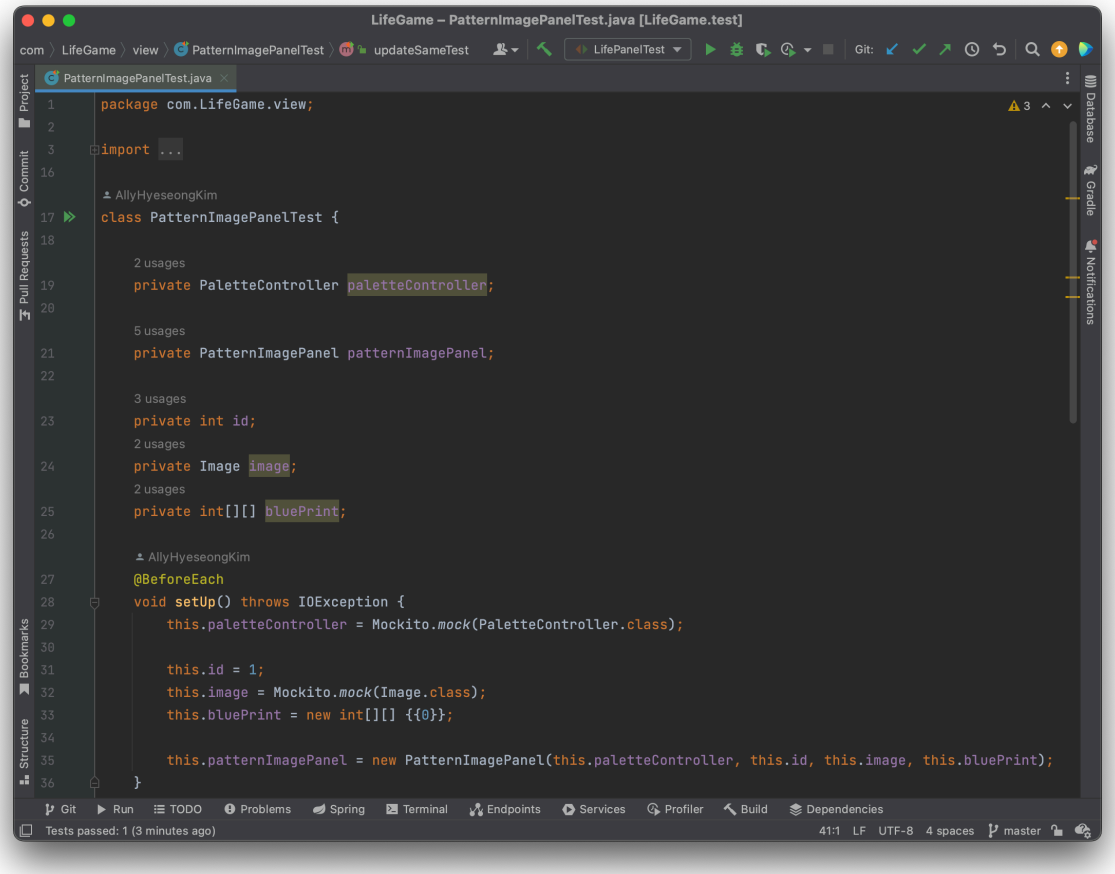


▲ [picture 104] LifePanelTest Passed

모든 LifePanelTest가 통과하였음을 확인할 수 있다.

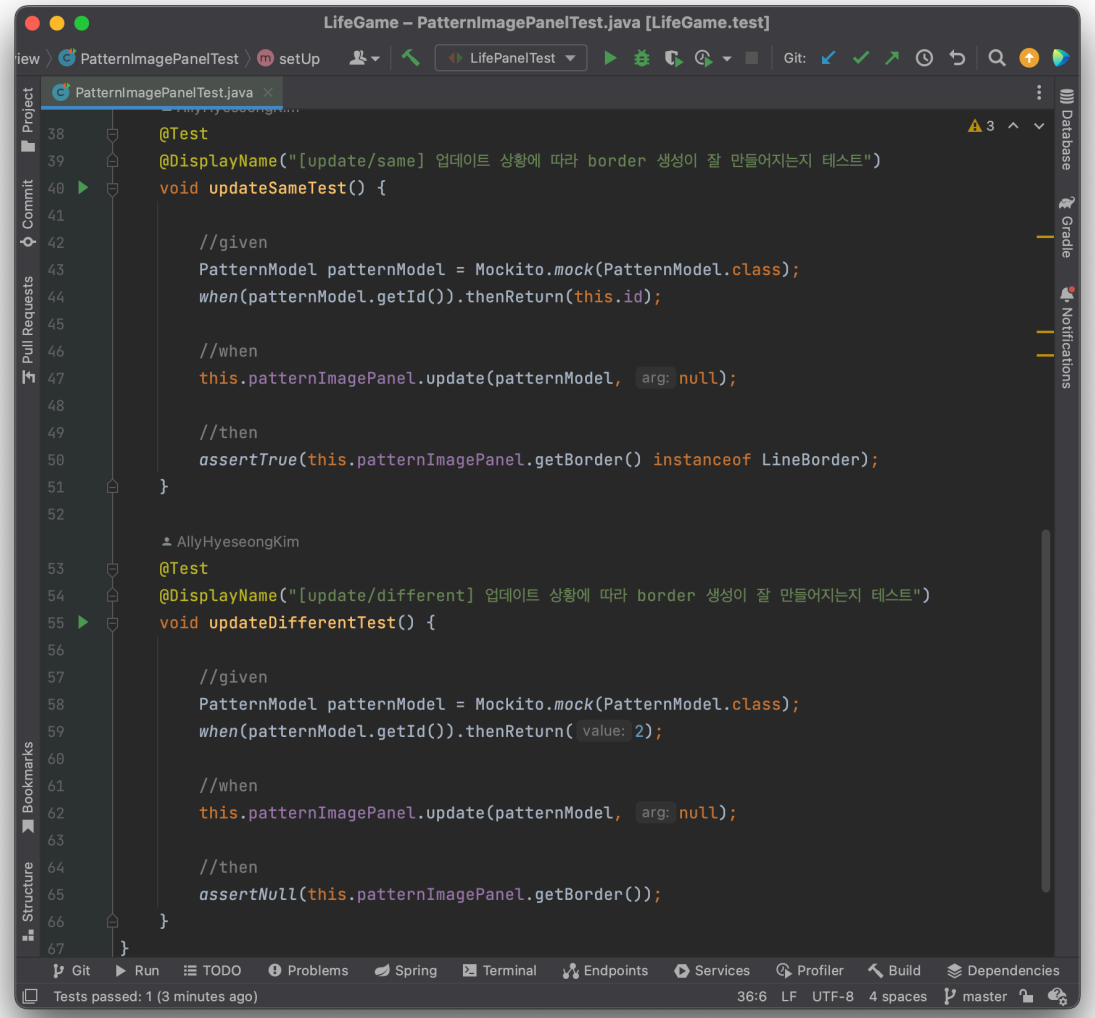
4.7.6. [Test] PatternImagePanelTest

PatternImagePanel class 테스트하는 class이다.



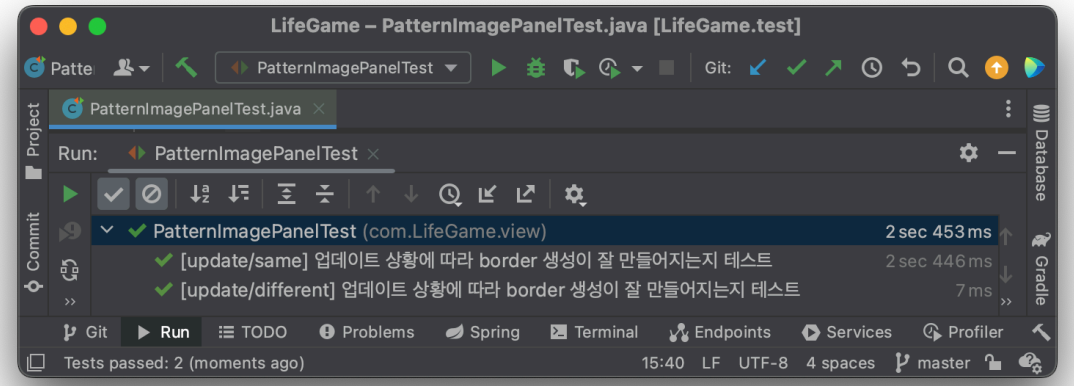
▲ [picture 105] Test cases of PatternImagePanel class (1)

PaletteController, Image를 Mockito로 Mocking하고, 테스트에 필요한 PatternImagePanel을 @BeforeEach Annotation이 붙은 메소드에 적용하여, 각 테스트 이전에 초기화하도록 하였다. PatternImagePanel은 Spring Bean이 아니기 때문에 생성자로 생성하였다.



▲ **[picture 106] Test cases of PatternImagePanel class** (2)

updateSameTest는 update를 호출했을 때 PatternModel의 id값이 PatternImagePanel과 같은 경우, Border가 LineBorder로 잘 설정되는지 검증한다.
updateDifferentTest는 update를 호출했을 때 PatternModel의 id값이 PatternImagePanel과 다른 경우, Border가 Null로 잘 설정되는지 검증한다.

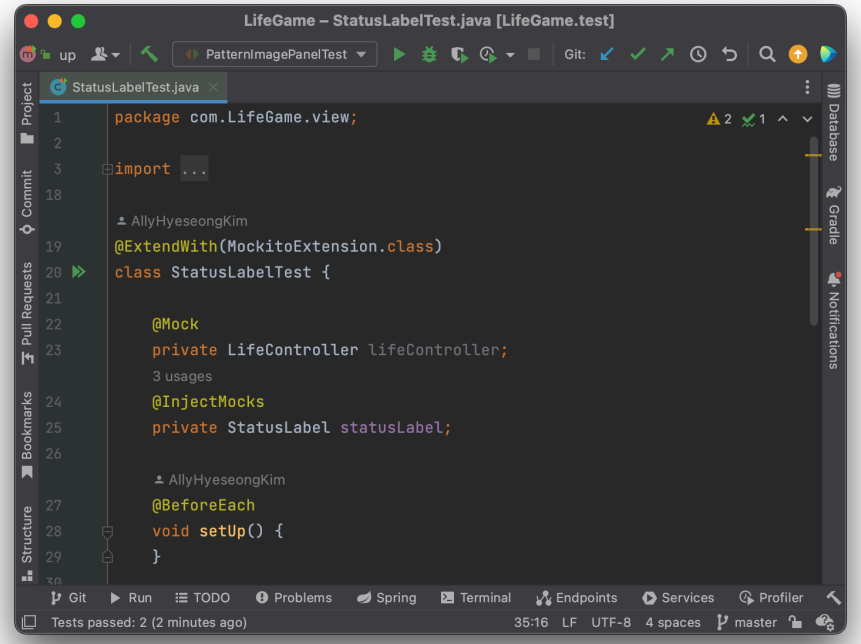


▲ [picture 107] PatternImagePanelTest Passed

PatternImagePanelTest가 모두 잘 통과하였음을 확인할 수 있다.

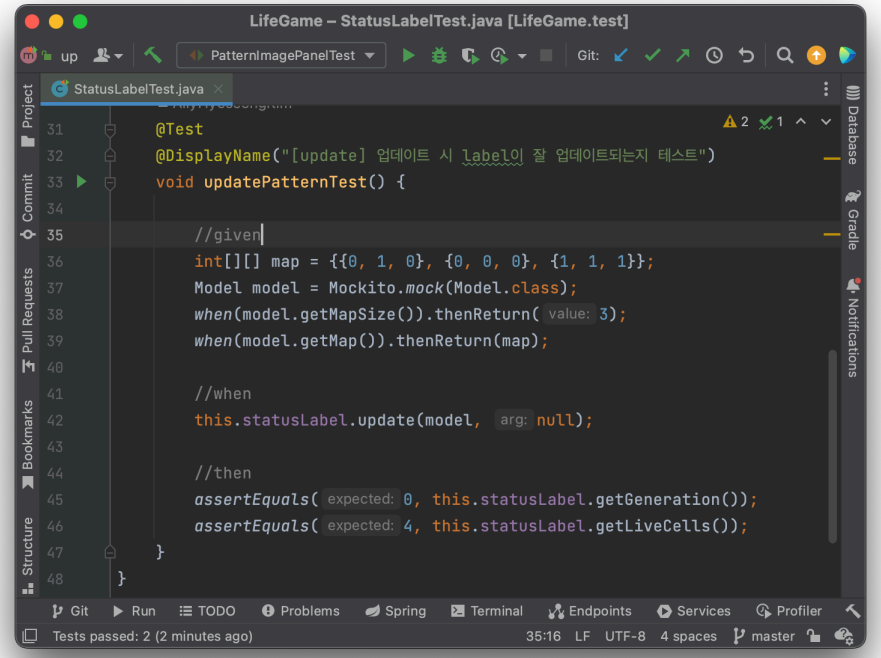
4.7.7. [Test] StatusLabelTest

StatusLabel class 테스트하는 class이다.



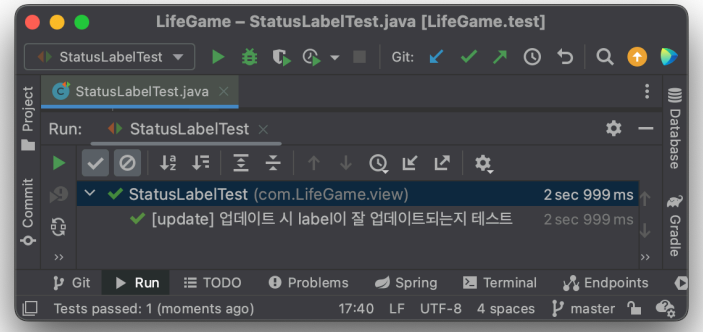
▲ [picture 108] Test cases of StatusLabel class (1)

LifeController를 Mocking하여 주입하여 독립적으로 테스트하였다.



▲ [picture 109] Test cases of StatusLabel class (2)

updatePatternTest는 update가 호출되었을 때 generation과 liveCell이 잘 설정되었는지 검증한다.

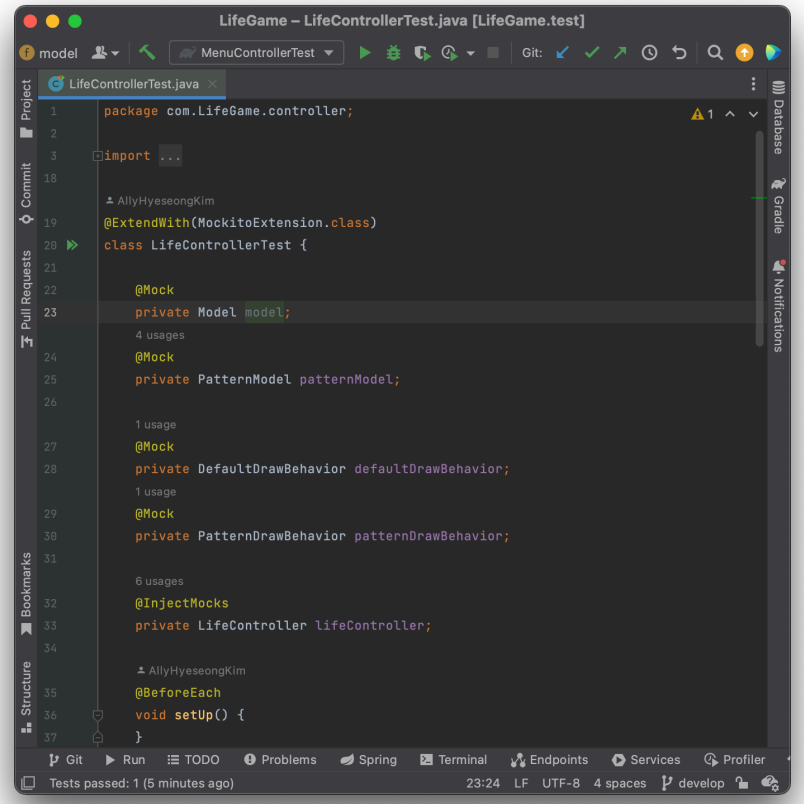


▲ [picture 110] StatusLabelTest Passed

모든 StatusLabelTest가 통과되었음을 확인할 수 있다.

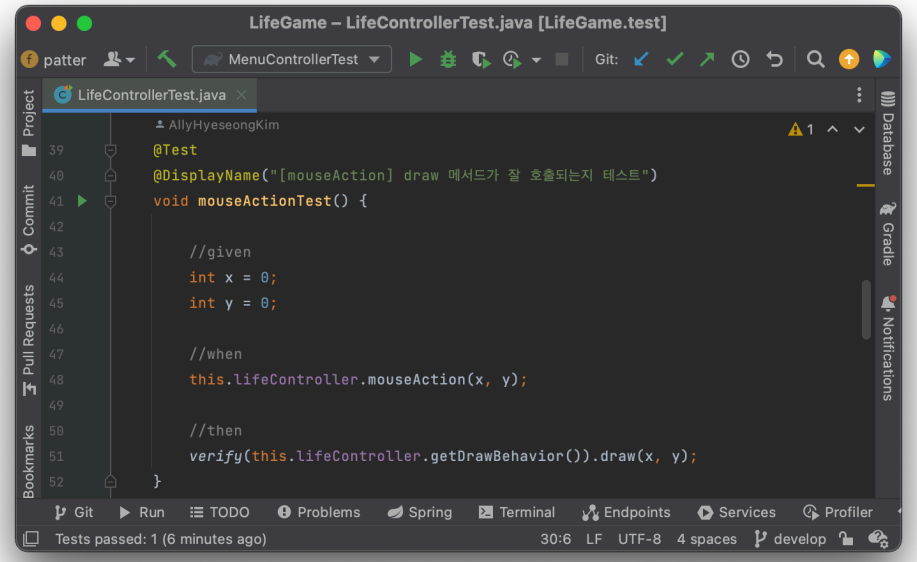
4.7.8. [Test] LifeControllerTest

LifeController class 테스트하는 class이다.



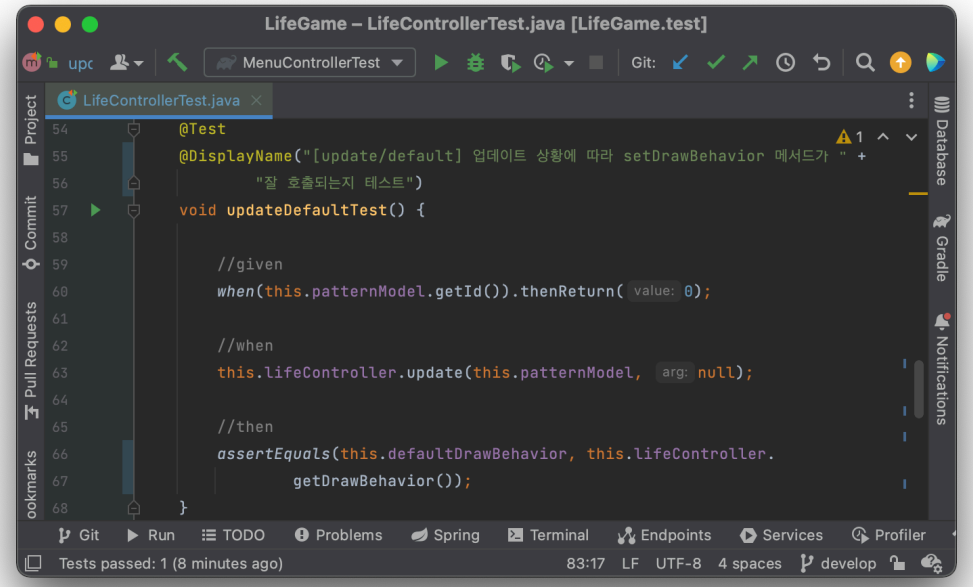
▲ **[picture 111] test cases of LifeController class** (1)

Model, PatternModel, DefaultDrawBehavior, PatternDrawBehavior를 Mocking하여 LifeController에 주입하였다.



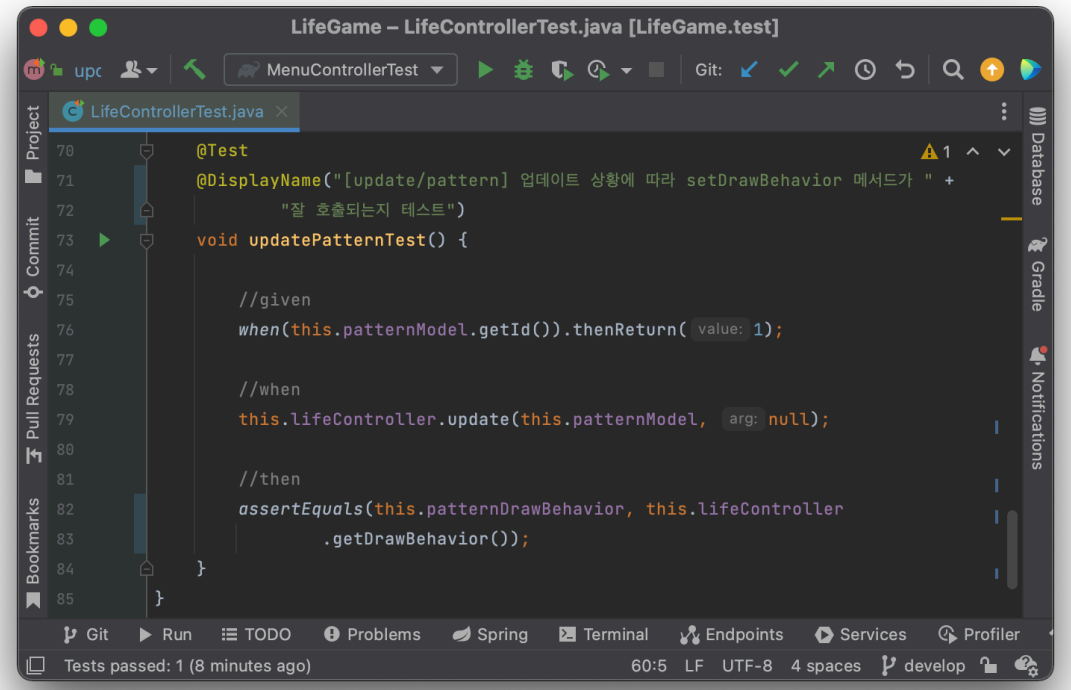
▲ **[picture 112] test cases of LifeController class** (2)

mouseActionTest는 mouseAction이 호출되었을 때 drawBehavior을 이용해 draw가 호출되었는지 검증한다.



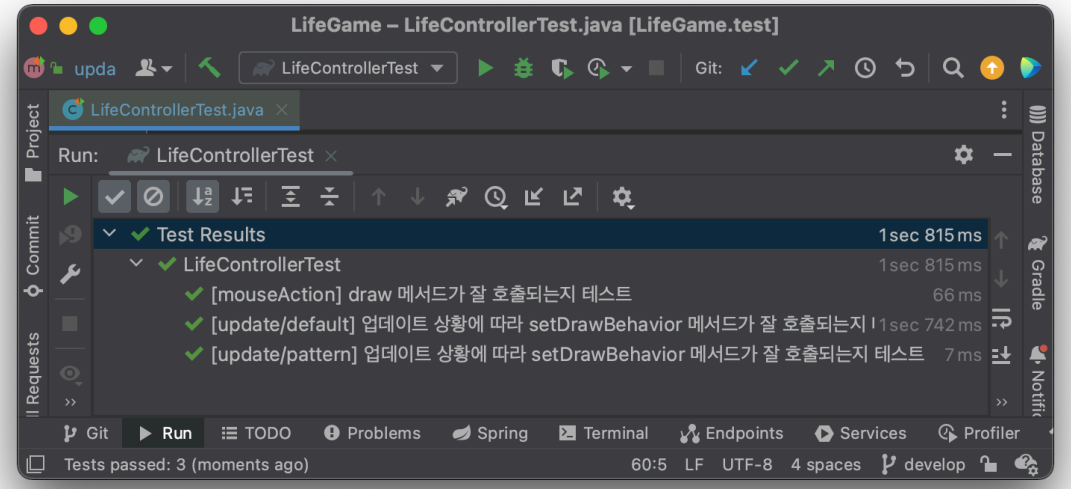
▲ **[picture 113] test cases of LifeController class** (3)

updateDefaultTest는 PatternModel의 id가 0일 때 drawBehavior이 0으로 잘 업데이트 되었는지 확인하는 테스트이다.



▲ [picture 114] test cases of LifeController class (4)

updatePatternTest는 lifeController의 update가 PatternModel에 의해 호출되었을 때 id가 0이 아닌 경우 drawBehavior Strategy가 patternDrawBehavior로 잘 설정되는지 검증한다.

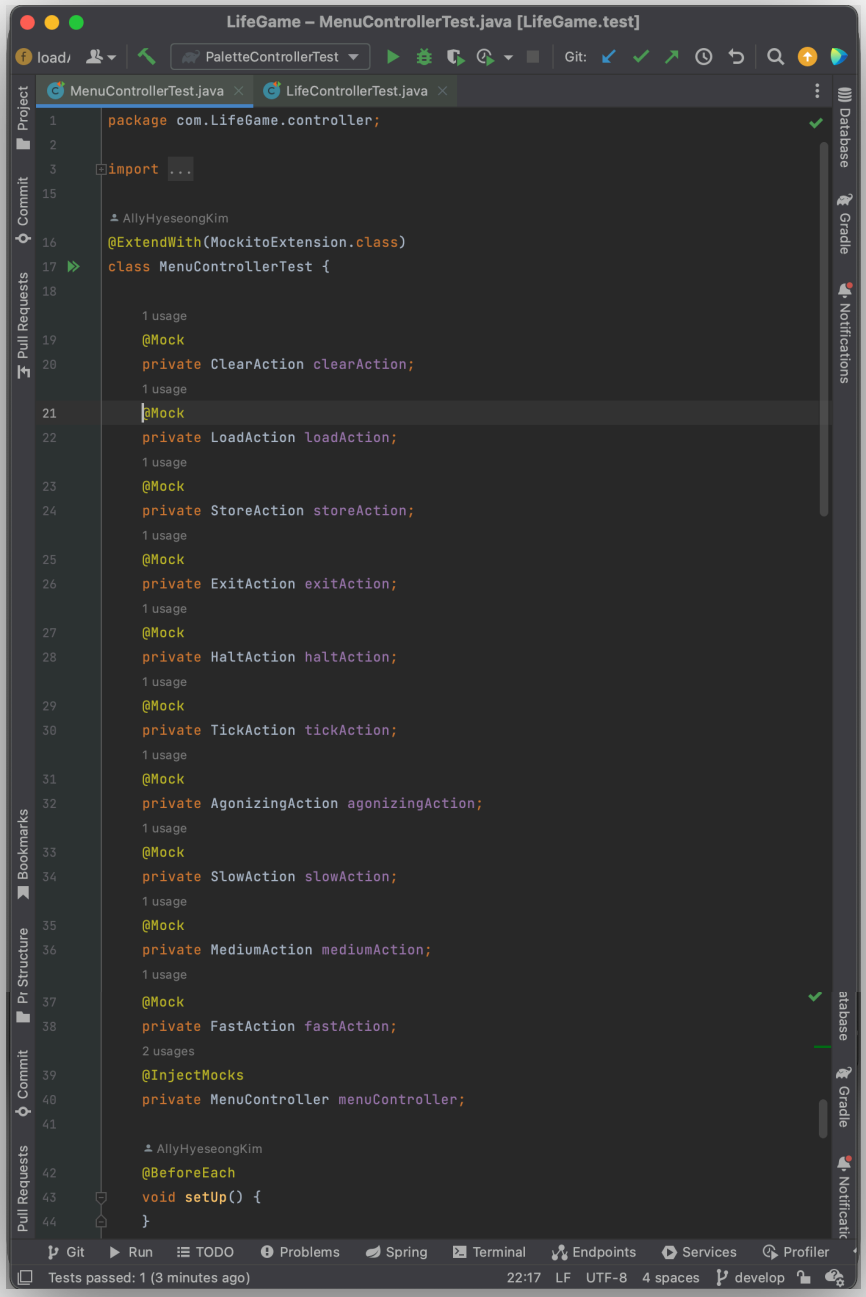


▲ [picture 115] LifeControllerTest passed

LifeControllerTest 모두가 통과되었음을 확인할 수 있다.

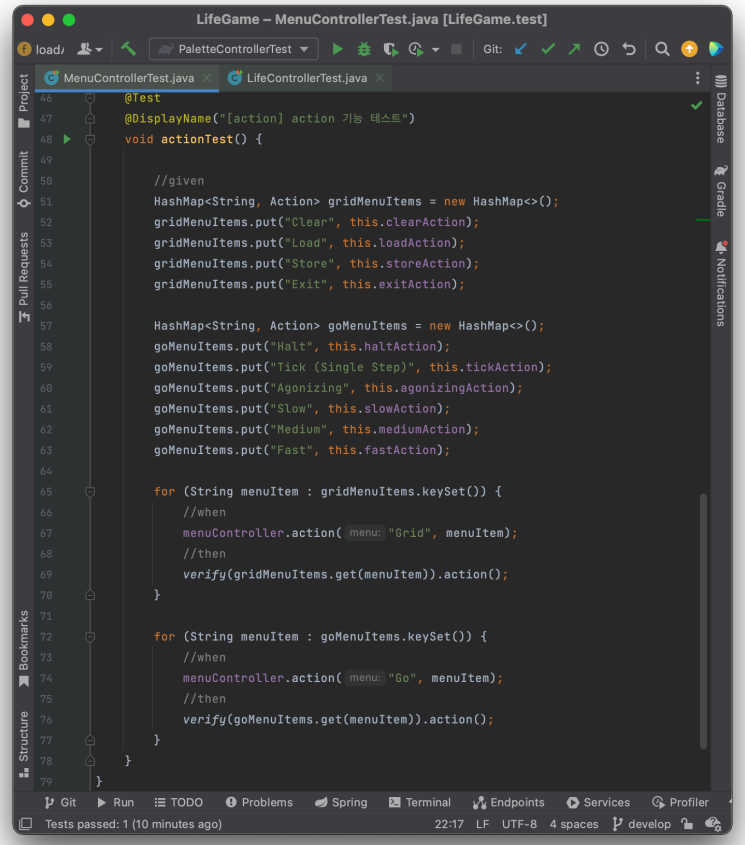
4.7.9. [Test] MenuControllerTest

MenuController class 테스트하는 class이다.



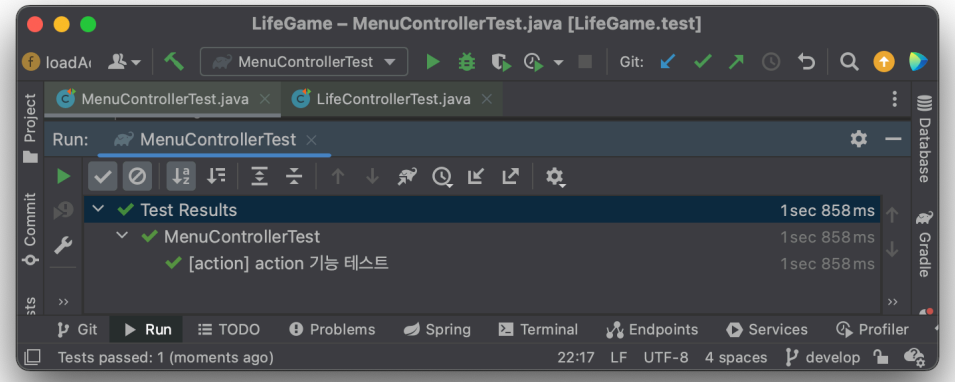
▲ [picture 116] test cases of MenuController class (1)

각 Action들을 모두 Mocking하여 주입하였다.



▲ [picture 117] test cases of MenuController class (2)

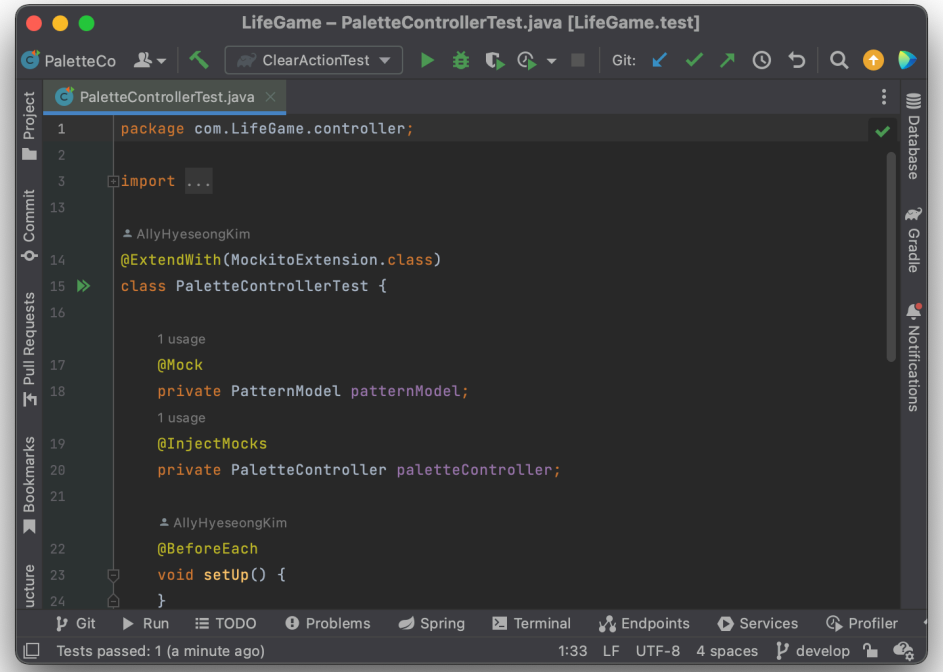
actionTest는 각 menuItem들의 action을 호출했을 때 올바른 Action이 실행되는지 검증한다.



▲ [picture 118] MenuControllerTest passed

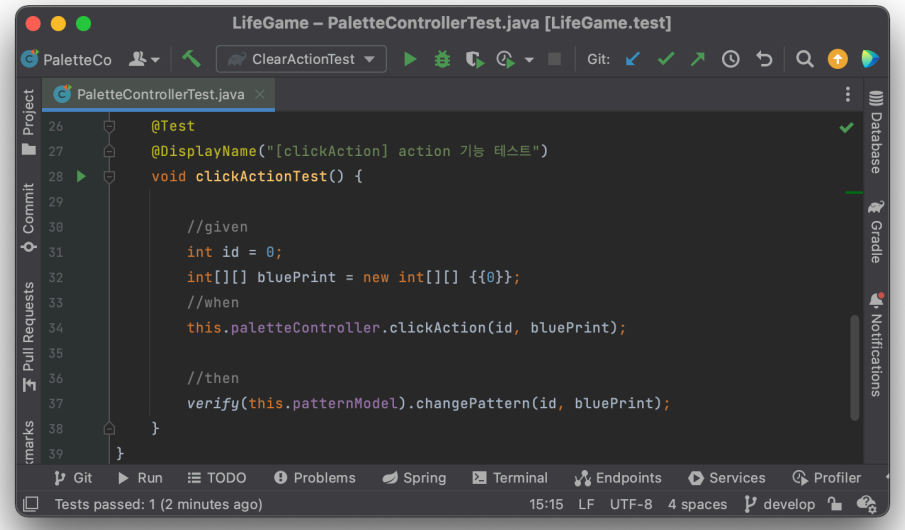
모든 MenuControllerTest가 통과하였음을 확인할 수 있다.

4.7.10. [Test] PaletteControllerTest
PaletteController class 테스트하는 class이다.



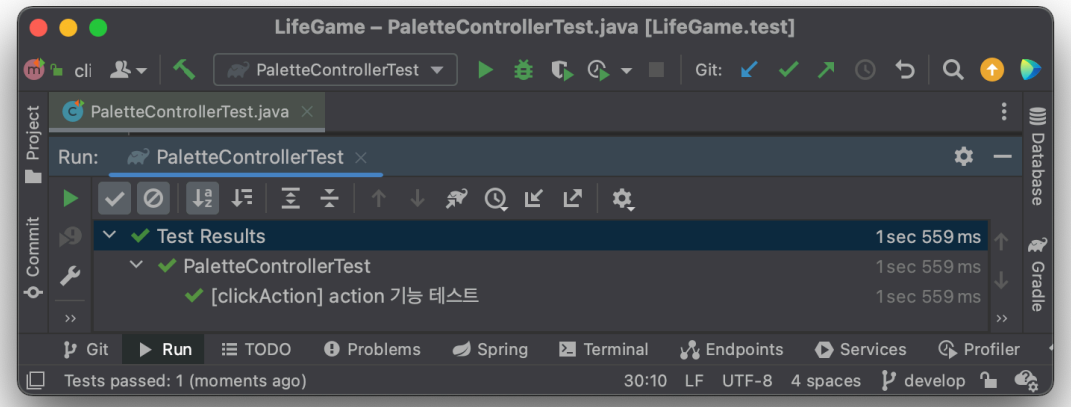
▲ **[picture 119] test cases of PaletteControl class** (1)

PatternModel을 Mocking하여 테스트하였다.



▲ **[picture 120] test cases of PaletteControl class** (2)

clickActionTest는 clickAction이 발생했을 때 PatternModel의 changePattern이 적절하게 호출되었는지 테스트한다.

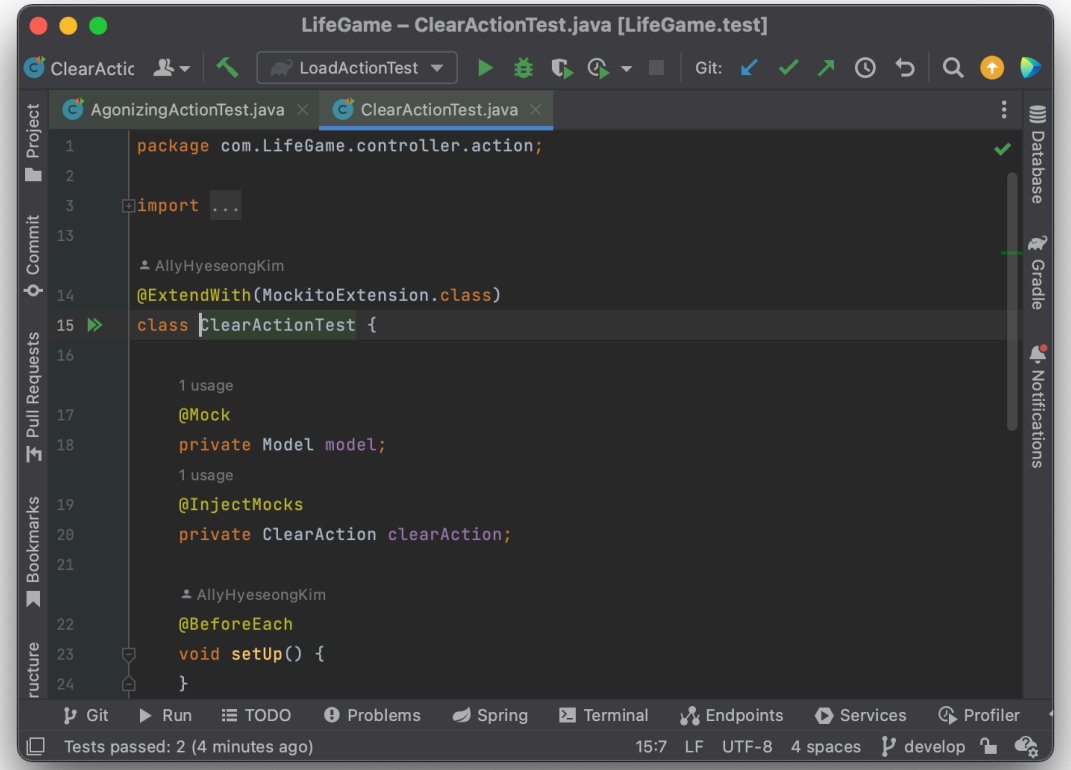


▲ [picture 121] PaletteControlTeset passed

모든 PaletteControlTest가 통과함을 확인할 수 있다.

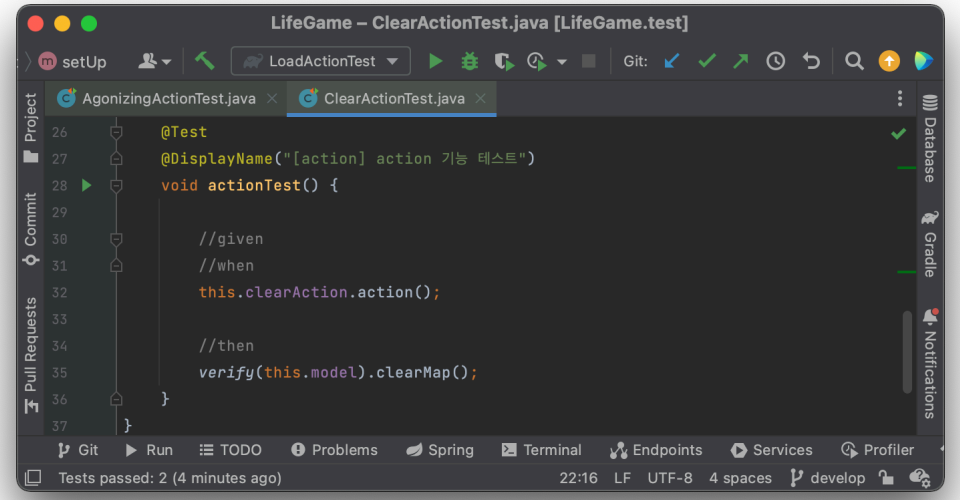
4.7.11. [Test] ClearActionTest

ClearAction class 테스트하는 class이다.



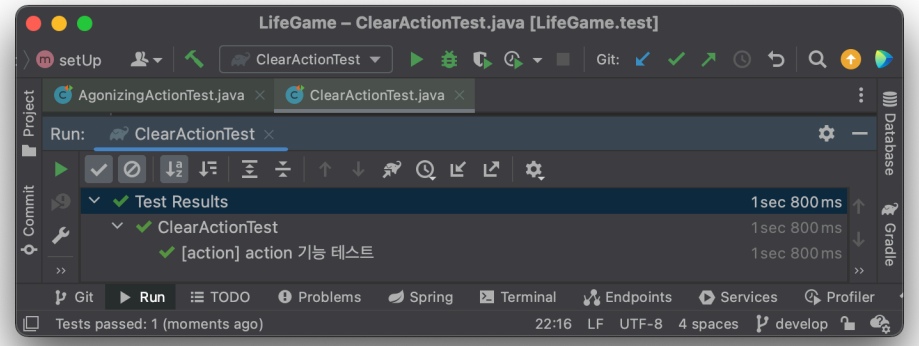
▲ [picture 122] test cases of ClearAction class (1)

Model을 Mocking하여 주입하였다.



▲ **[picture 123] test cases of ClearAction class** (2)

ClearAction을 실행했을 때 Map이 잘 초기화되는지 테스트한다.

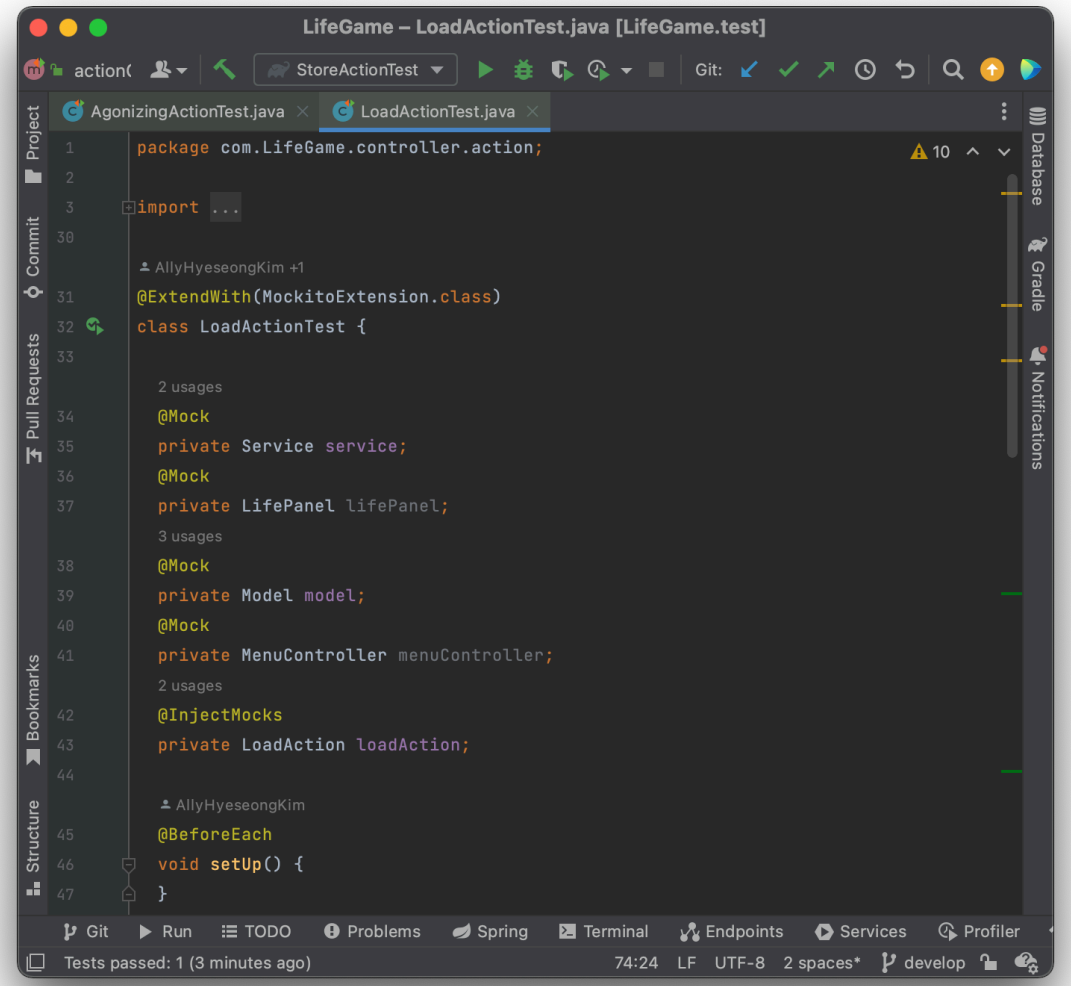


▲ **[picture 124] ClearActionTest passed**

모든 ClearActionTest가 통과하였음을 확인할 수 있다.

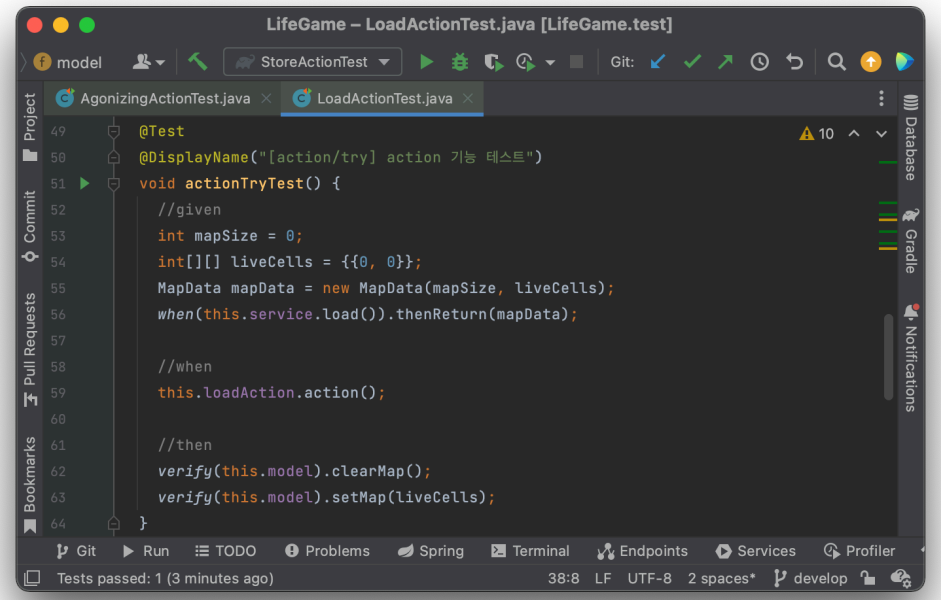
4.7.12. [Test] LoadActionTest

LoadAction class 테스트하는 class이다.



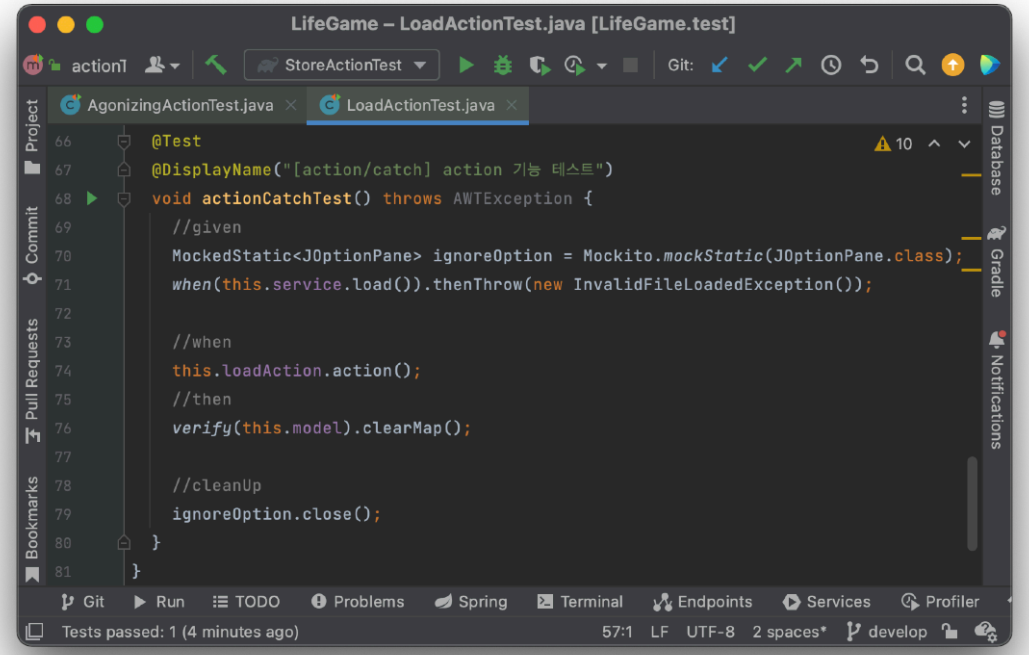
▲ **[picture 125] test cases of LoadAction class** (1)

Service, LifePanel, Model, MenuController를 Mocking하여 주입하였다.



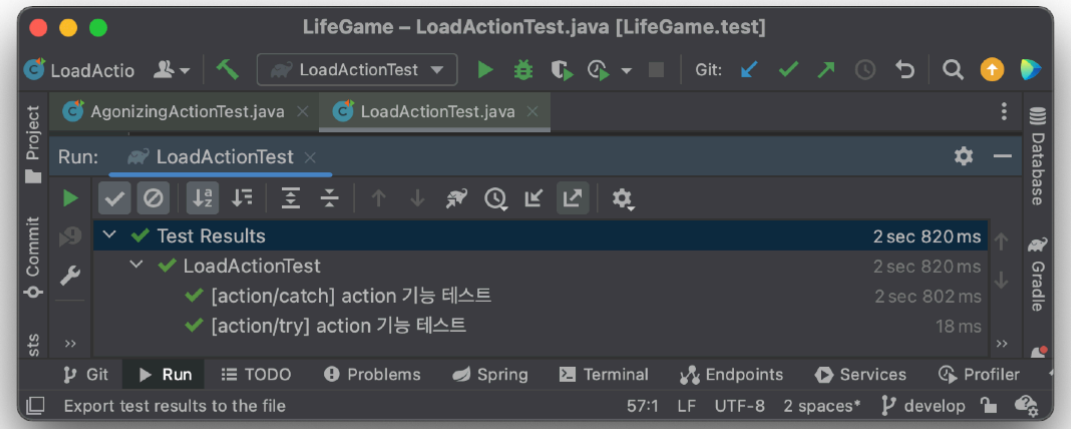
▲ **[picture 126] test cases of LoadAction class** (2)

LoadAction이 동작할 때 Mocking된 Service에서 받아온 값을 Model에 잘 반영하는지 테스트한다.



▲ **[picture 127] test cases of LoadAction class** (3)

actionCatchTest는 잘못된 파일을 읽으려 할 때 model이 잘 클리어 되는지 테스트한다.

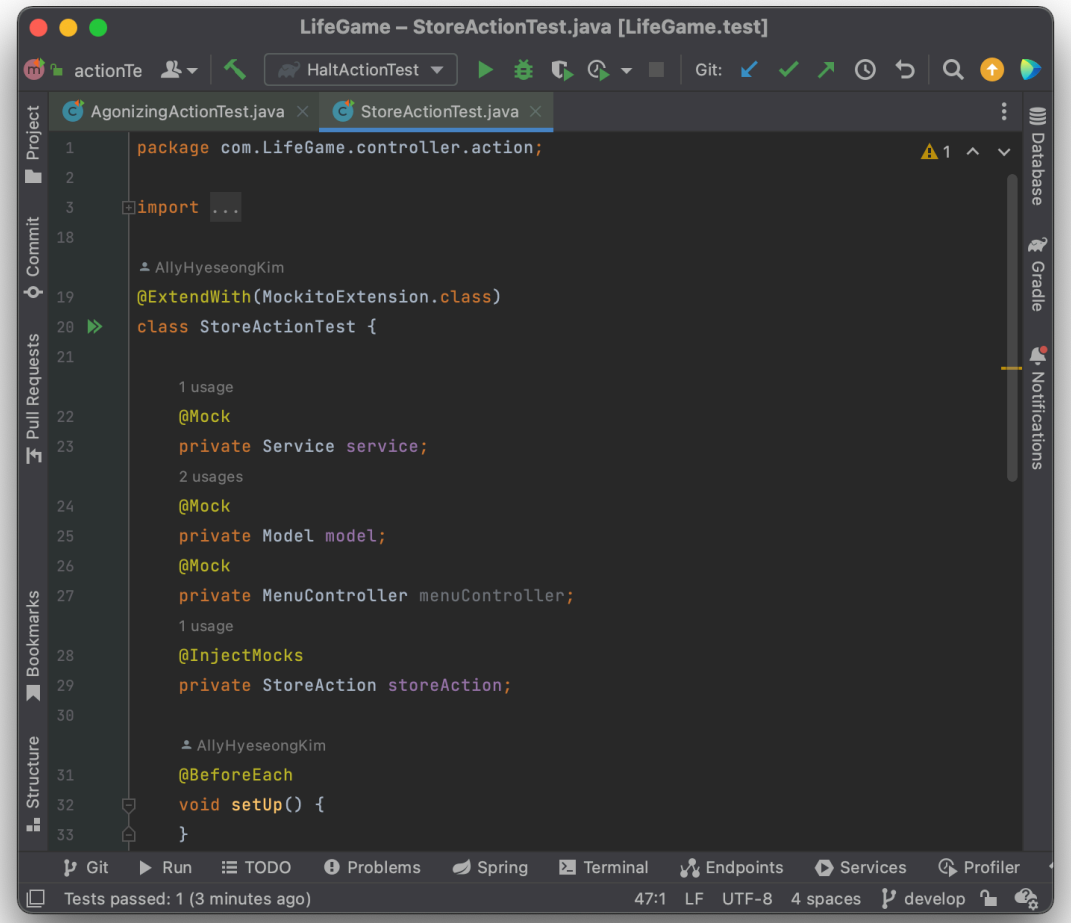


▲ [picture 128] LoadActionTest passed

모든 LoadActionTest가 통과하였음을 확인할 수 있다.

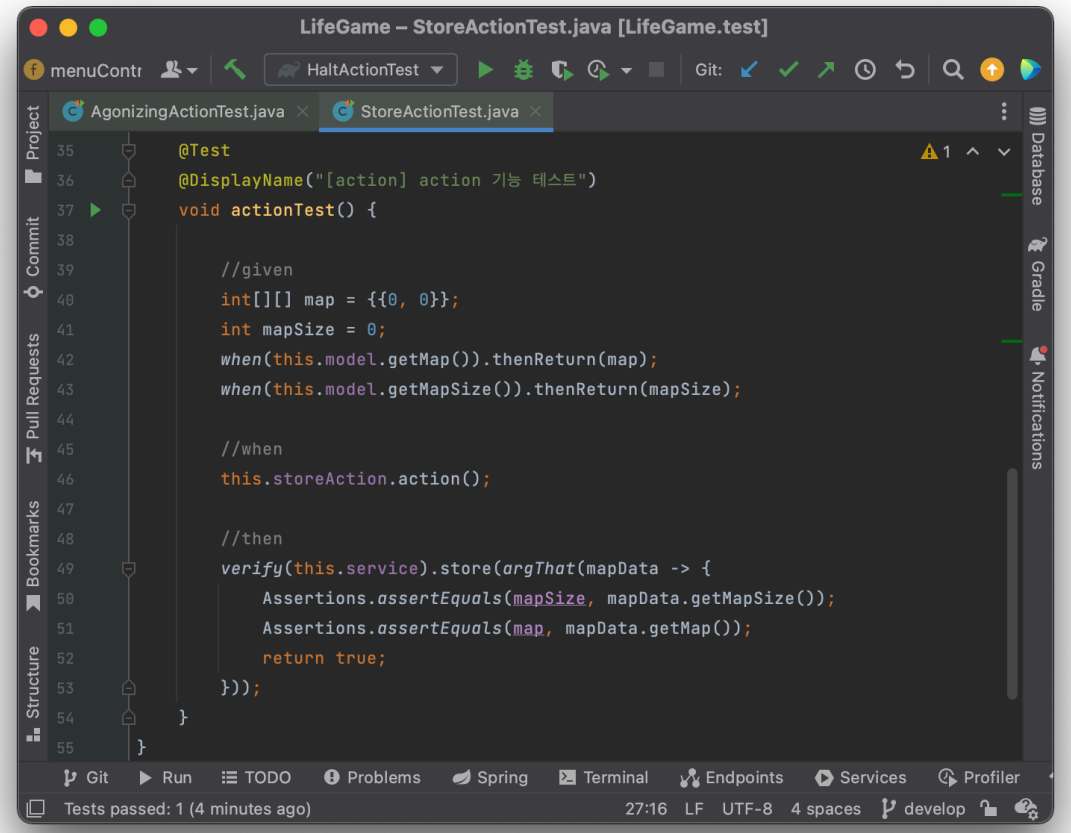
4.7.13. [Test] StoreActionTest

StoreAction class 테스트하는 class이다.



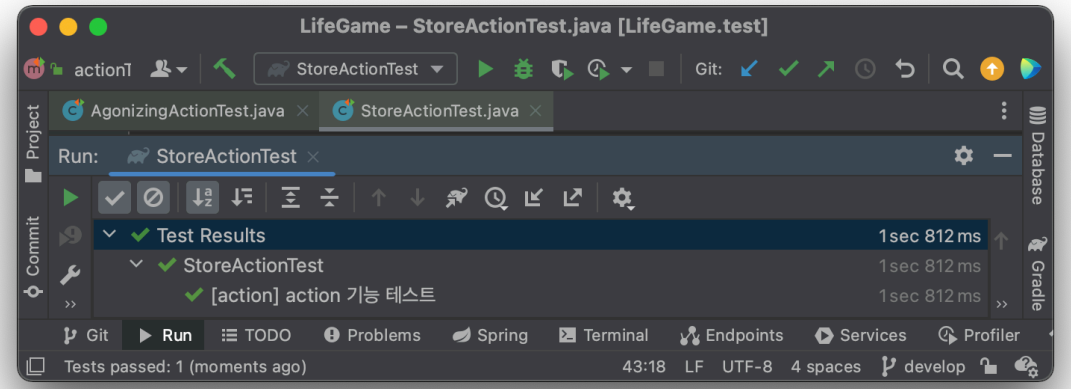
▲ **[picture 129] test cases of StoreAction class (1)**

Service, Model, MeuController를 Mocking하였다.



▲ **[picture 130] test cases of StoreAction class** (2)

StoreAction이 실행될 때 적절한 데이터를 Service에 요청하는지 테스트한다.

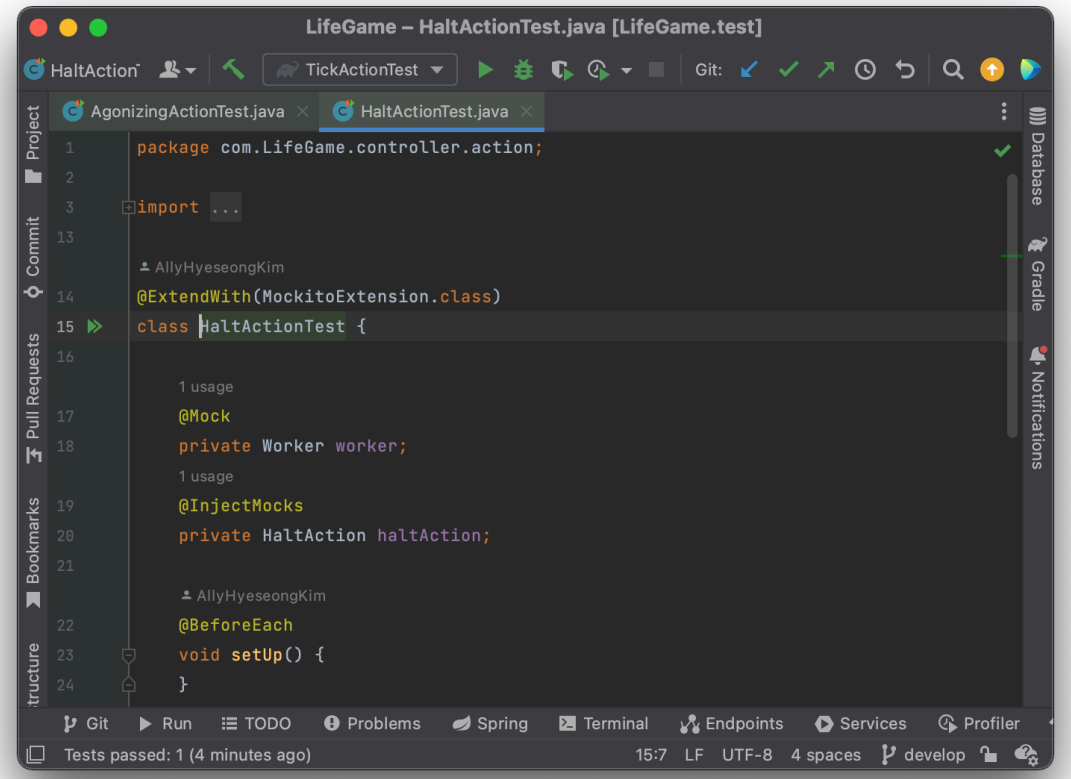


▲ **[picture 131] StoreActionTest passed**

StoreActionTest의 모든 Test가 통과하는 것을 확인할 수 있다.

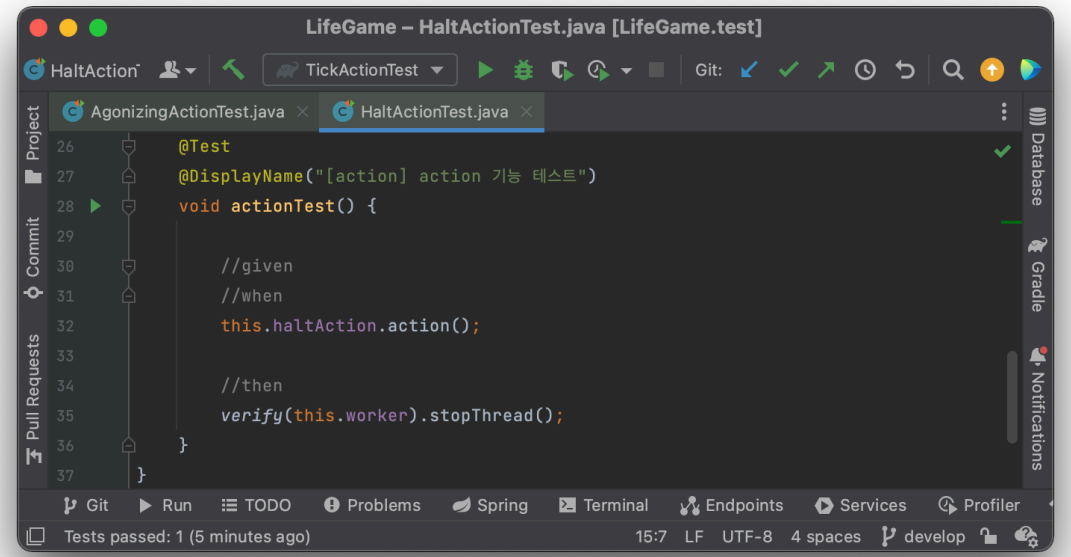
4.7.14. [Test] HaltActionTest

HaltAction class 테스트하는 class이다.



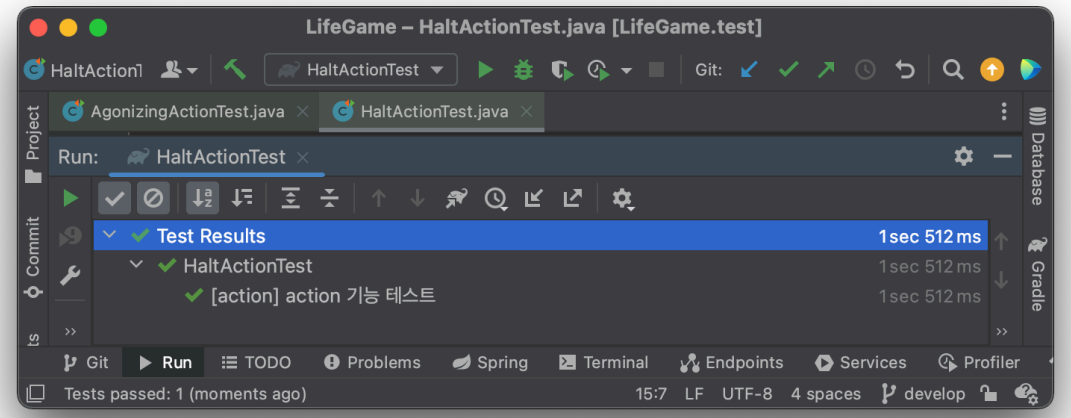
▲ [picture 132] test cases of HaltAction class (1)

Worker를 Mocking하여 주입하여 테스트를 진행하였다.



▲ [picture 133] test cases of HaltAction class (2)

actionTest는 HaltAction이 실행될 때 worker에 stopThread를 잘 호출하는지 테스트한다.

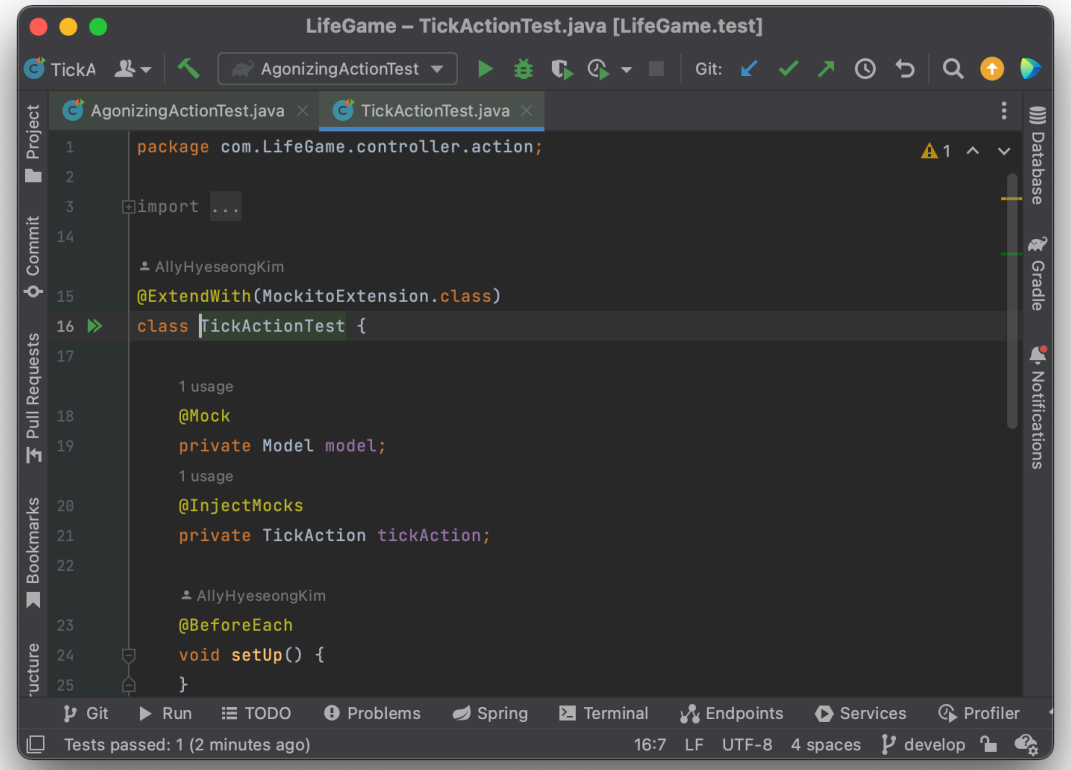


▲ [picture 134] HaltActionTest passed

HaltActionTest의 모든 Test가 통과하는 것을 확인할 수 있다.

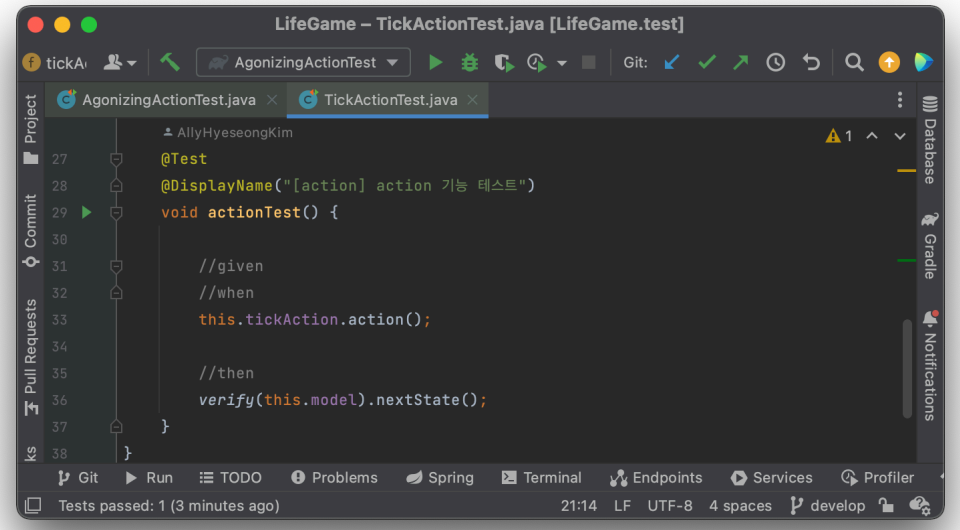
4.7.15. [Test] TickActionTest

TickAction class 테스트하는 class이다.

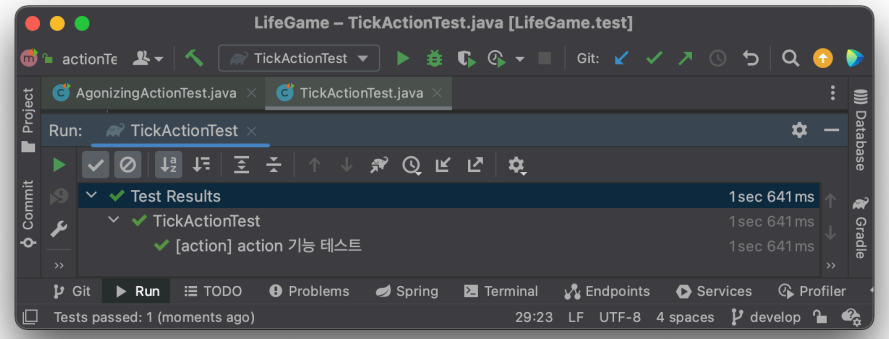


▲ [picture 135] test cases of TickAction class (1)

Model을 Mocking하여 주입하여 테스트를 진행하였다.



▲ **[picture 136] test cases of TickAction class** (2)

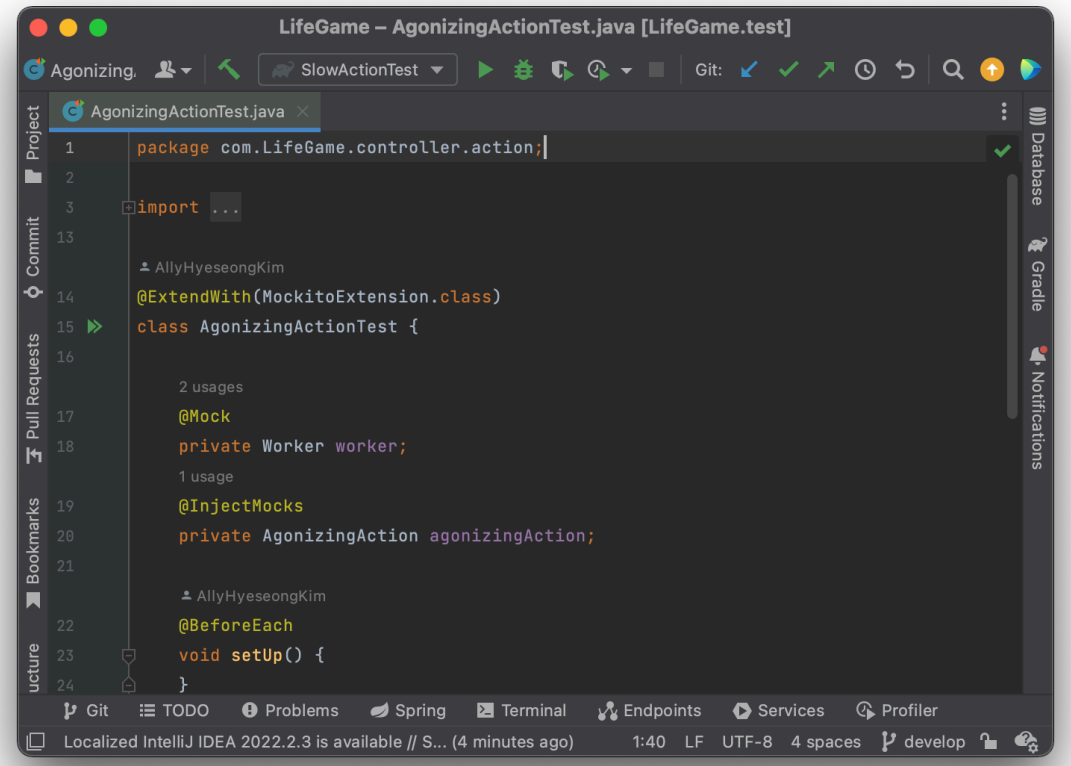


▲ **[picture 137] TickActionTest passed**

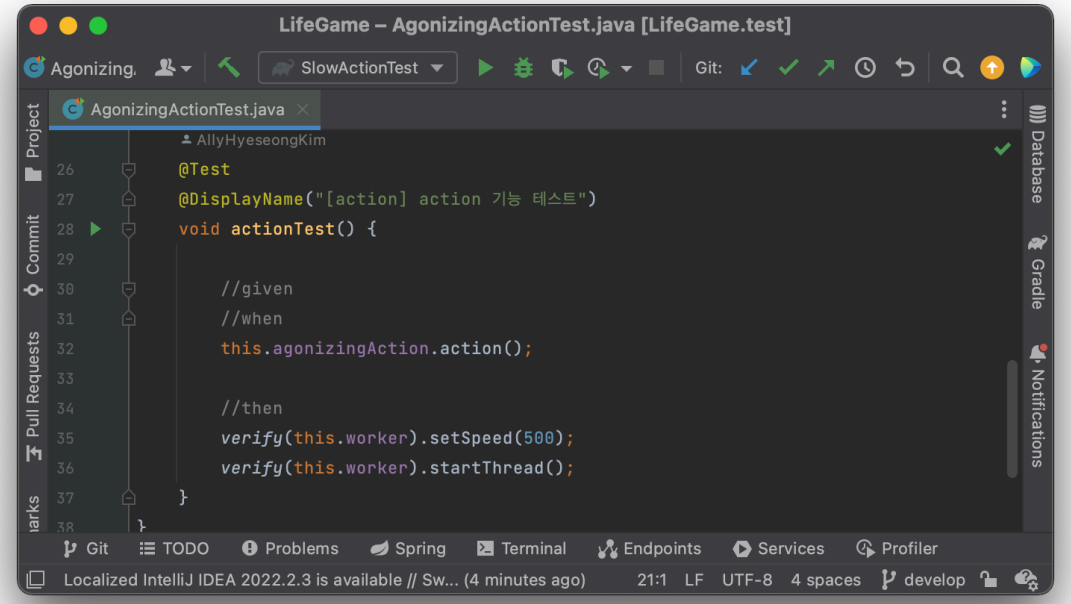
TickActionTest의 모든 Test가 통과하는 것을 확인할 수 있다.

4.7.16. [Test] AgonizingActionTest

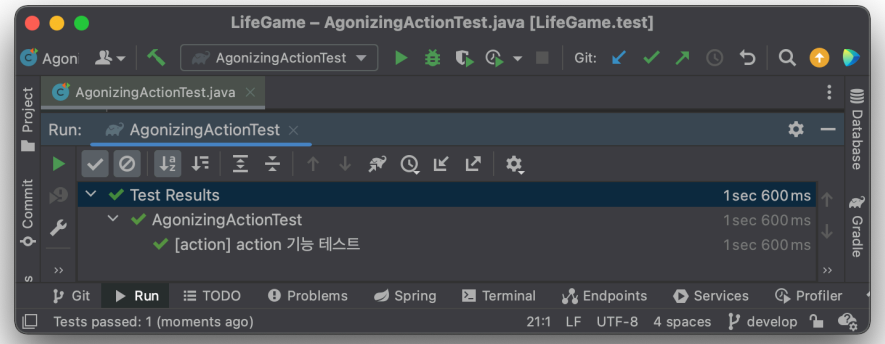
AgonizingAction class 테스트하는 class이다.



▲ [picture 138] test cases of AgonizingAction class (1)



▲ [picture 139] test cases of AgonizingAction class (2)

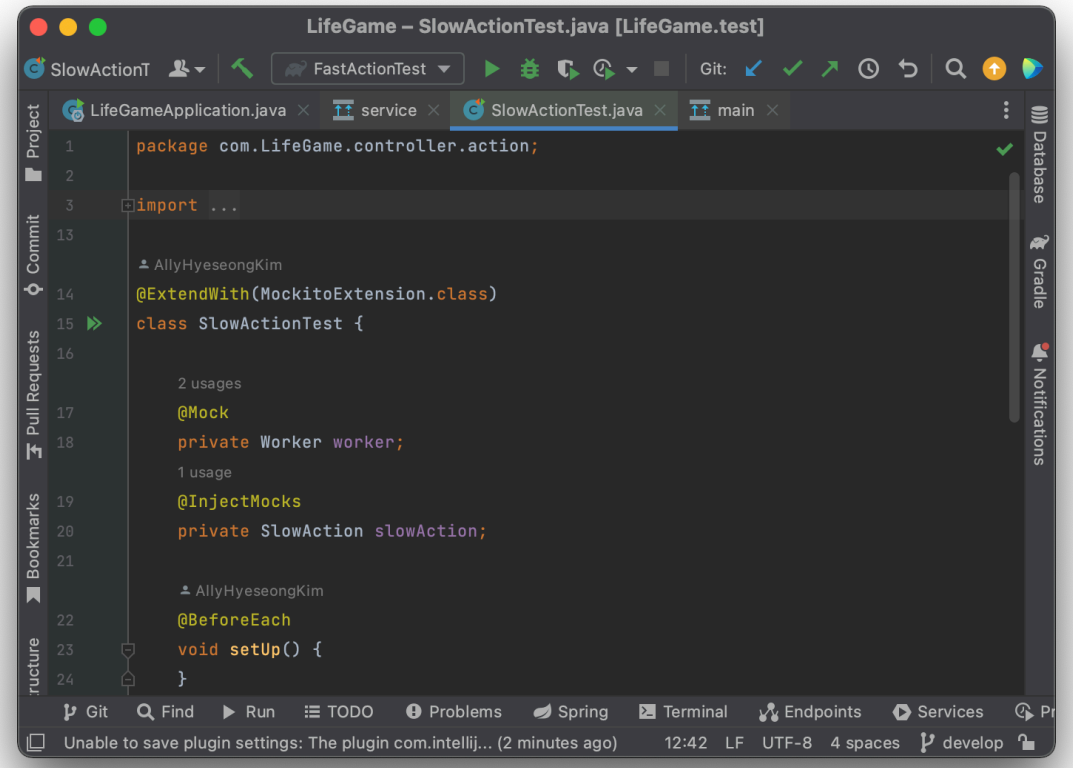


▲ [picture 140] AgonizingActionTest passed

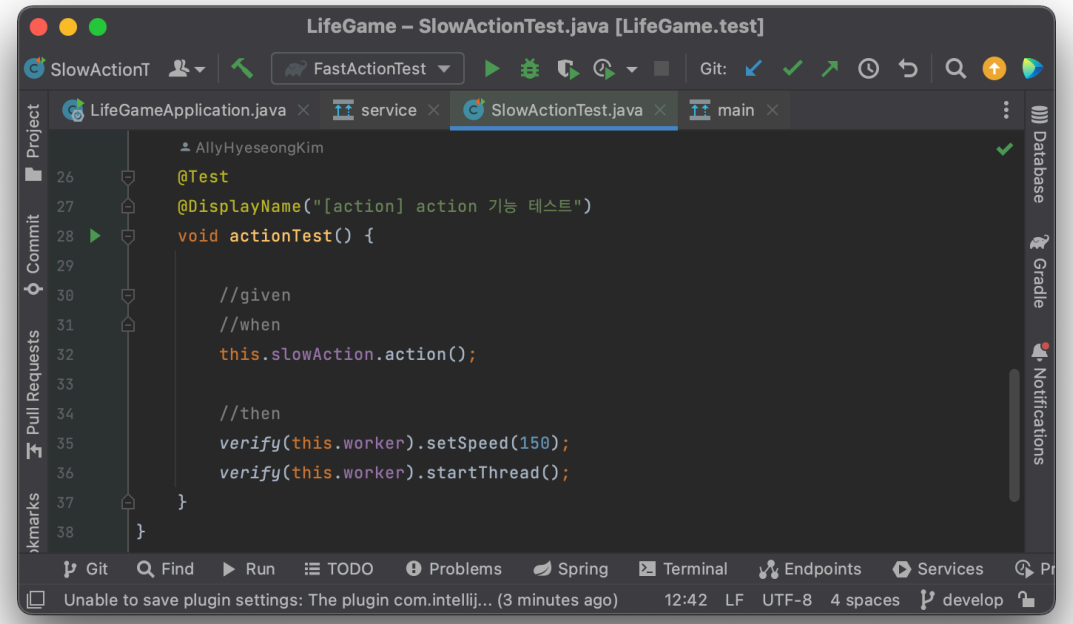
AgonizingActionTest의 모든 Test가 통과하는 것을 확인할 수 있다.

4.7.17. [Test] SlowActionTest

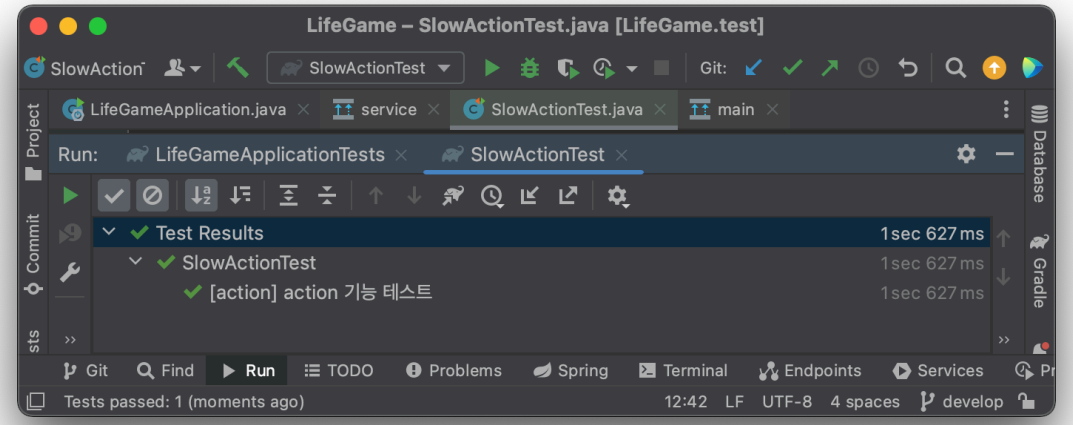
SlowAction class 테스트하는 class이다.



▲ [picture 141] test cases of SlowAction class (1)



▲ [picture 142] test cases of SlowAction class (2)

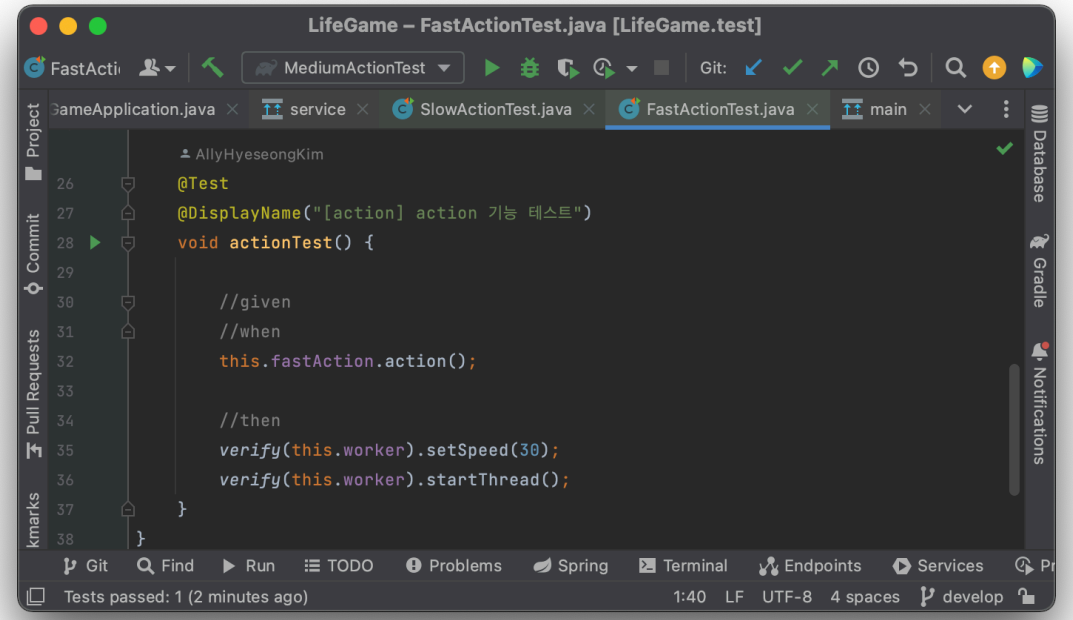


▲ [picture 143] SlowActionTest passed

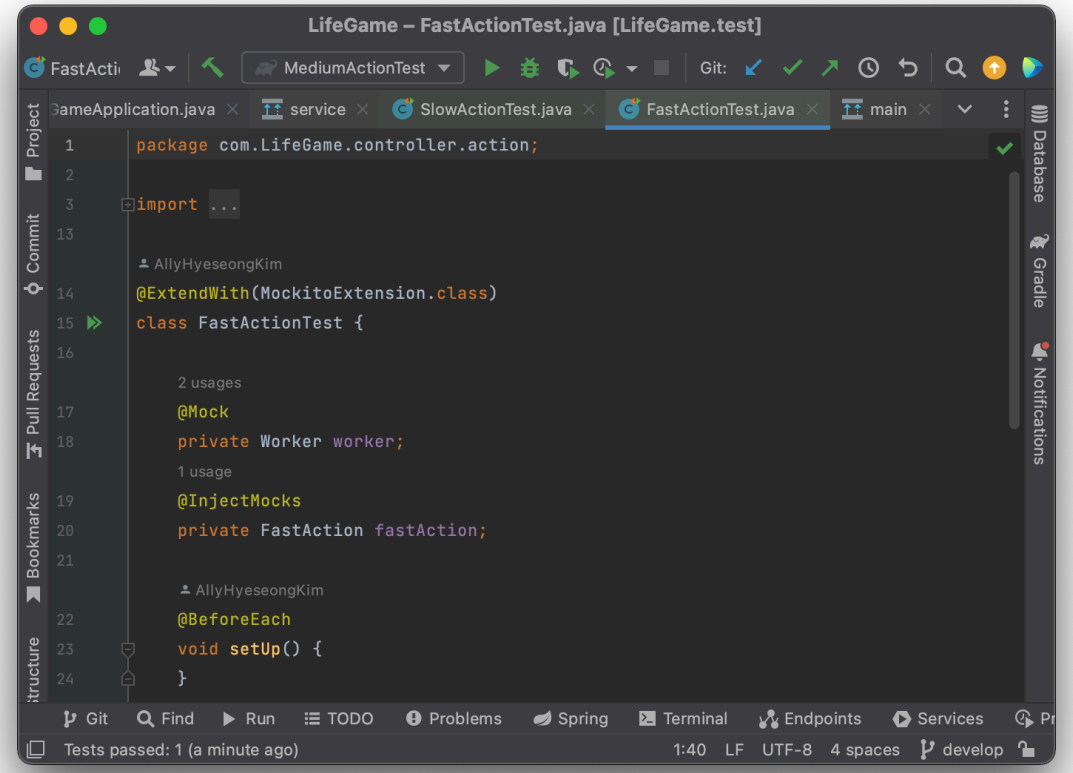
SlowTest의 모든 Test가 통과하는 것을 확인할 수 있다.

4.7.18. [Test] FastActionTest

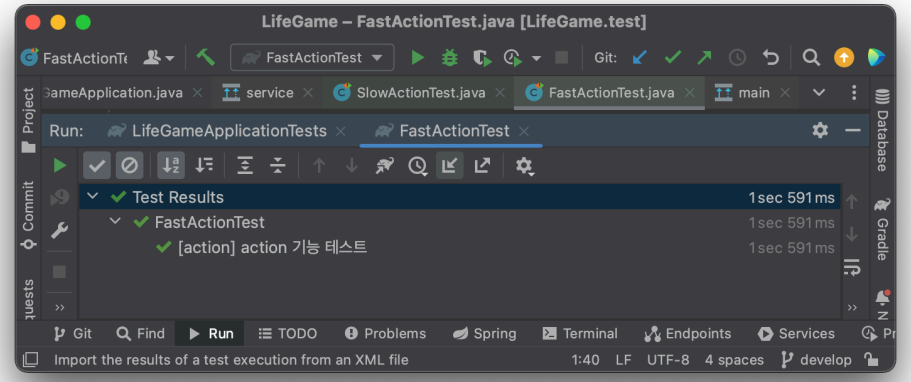
FastAction class 테스트하는 class이다.



▲ [picture 144] test cases of FastAction class (1)



▲ [picture 145] test cases of FastAction class (2)

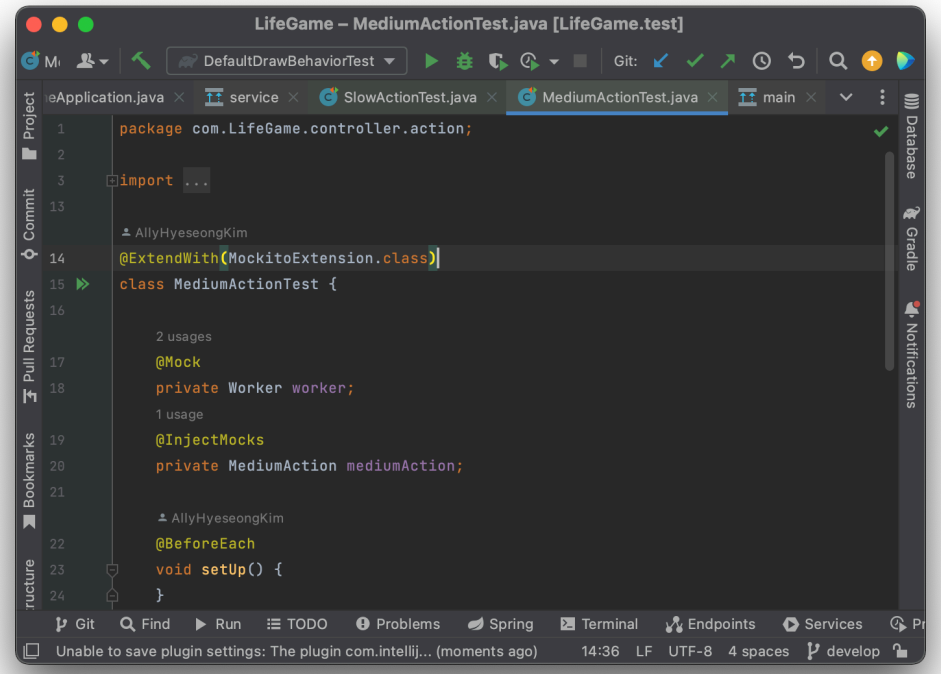


▲ [picture 146] FastActionTest passed

FastActionTest의 모든 Test가 통과하는 것을 확인할 수 있다.

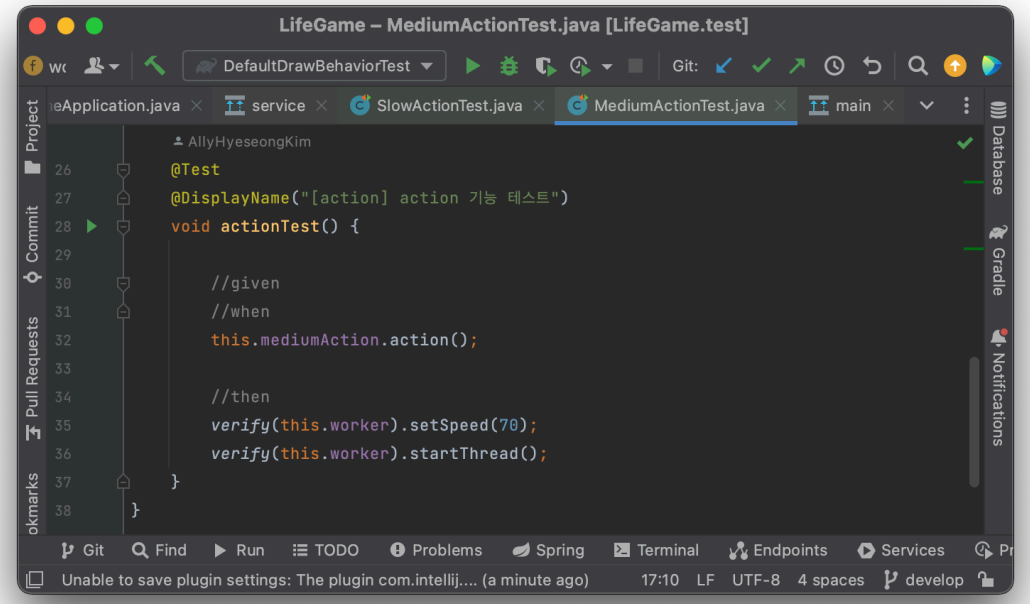
4.7.19. [Test] MediumActionTest

MediumAction class 테스트하는 class이다.



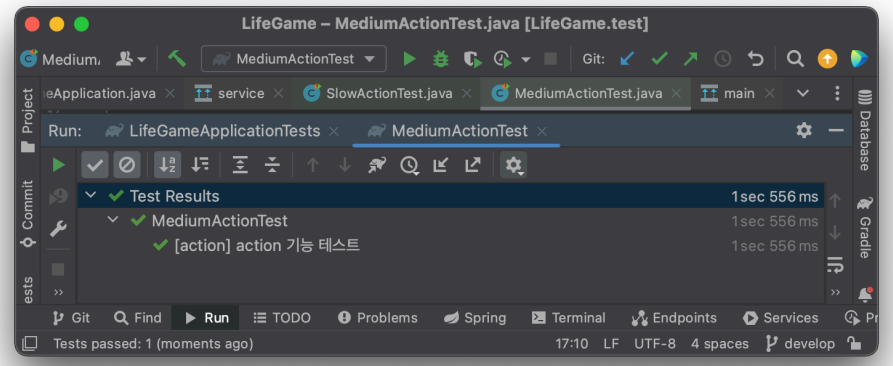
▲ [picture 147] test cases of MediumAction class (1)

Worker를 Mockito로 Mocking하여 MediumAction에 주입하였다.



▲ **[picture 148] test cases of MediumAction class** (2)

actionTest는 MediumAction이 실행될 때 speed가 70으로 잘 설정되고 스레드가 동작하는지 검증한다.

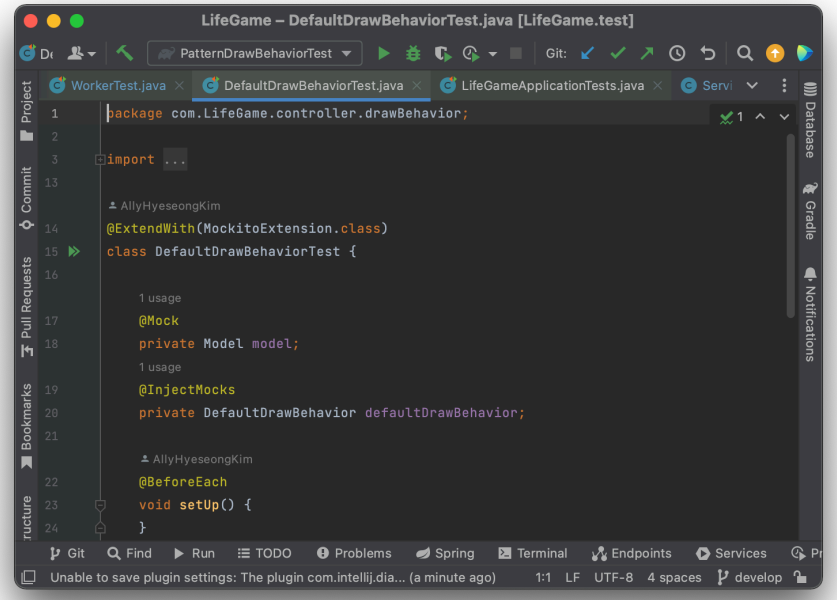


▲ **[picture 149] MediumActionTest passed**

MediumActionTest의 모든 Test가 통과하는 것을 확인할 수 있다.

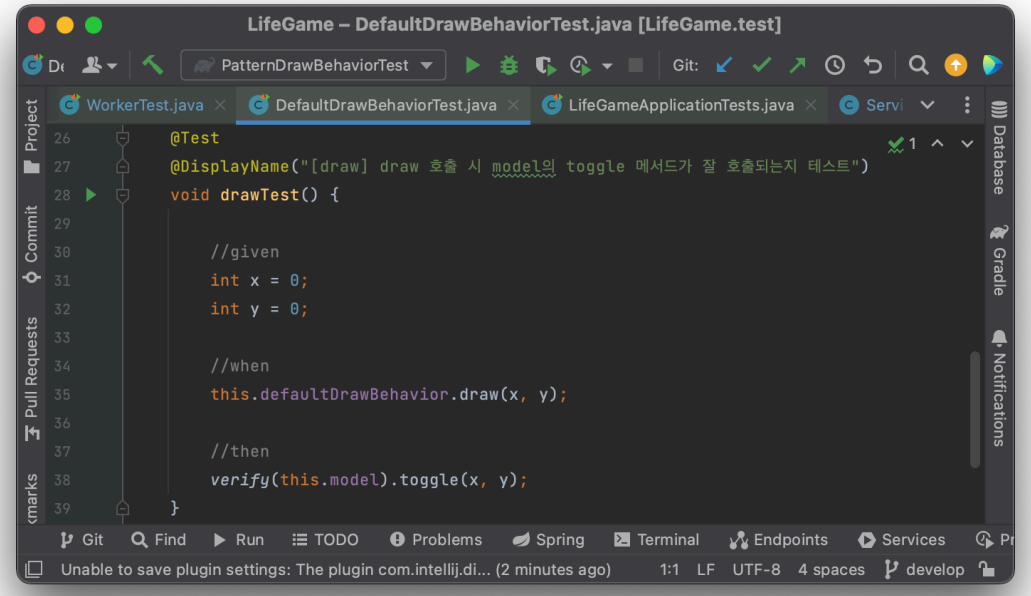
4.7.20. [Test] DefaultDrawBehaviorTest

DefaultDrawBehavior class 테스트하는 class이다.



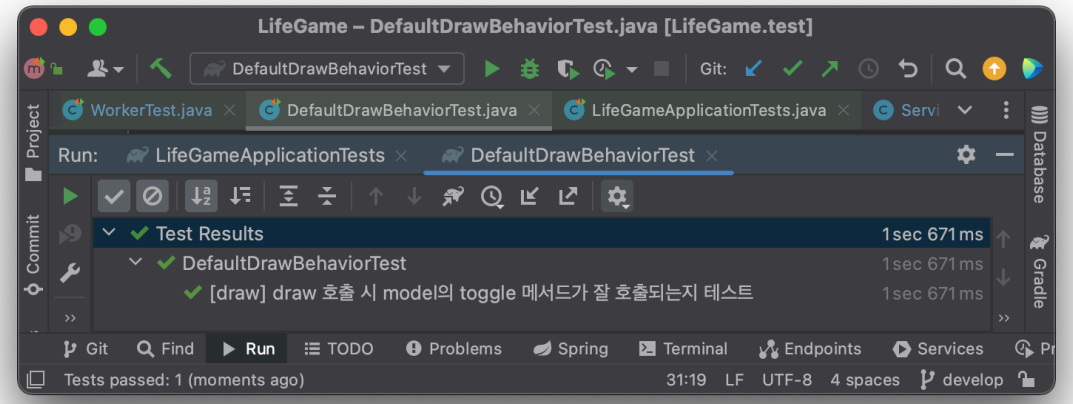
▲ [picture 150] test cases of DefaultDrawBehavior class (1)

Model을 Mocking하여 주입하였다



▲ [picture 151] test cases of DefaultDrawBehavior class (2)

drawTest는 Model에 올바른 좌표값을 넘겨 toggle을 실행하는지 테스트한다.

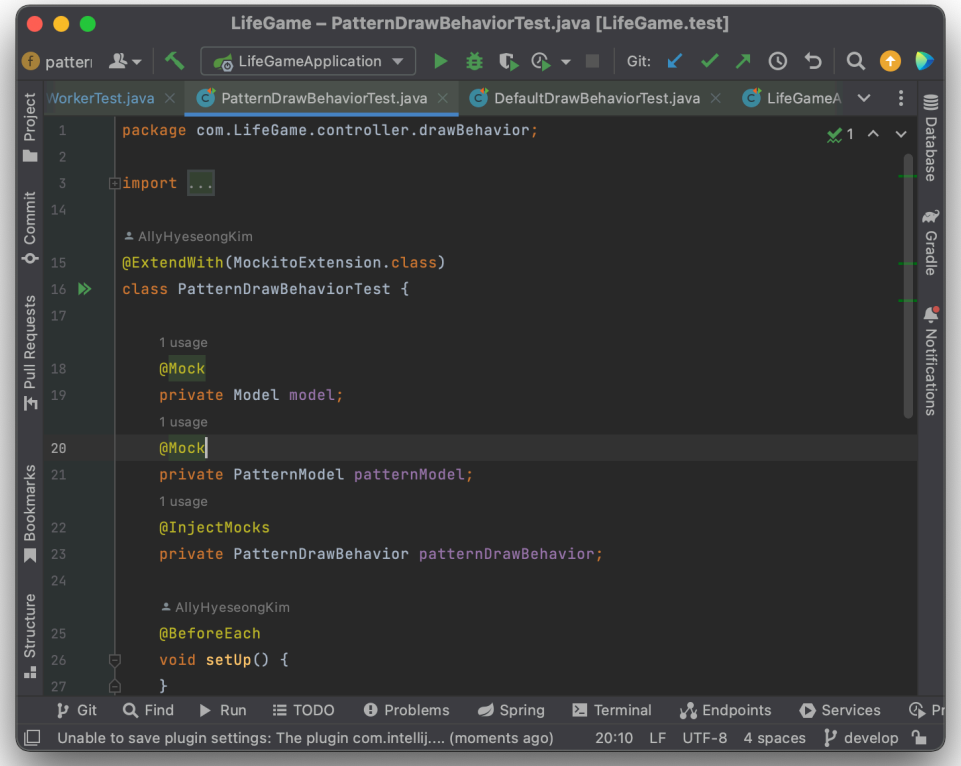


▲ [picture 152] DeafultDrawBehaviorTest passed

DefaultDrawBehaviorTest의 모든 Test가 통과하는 것을 확인할 수 있다.

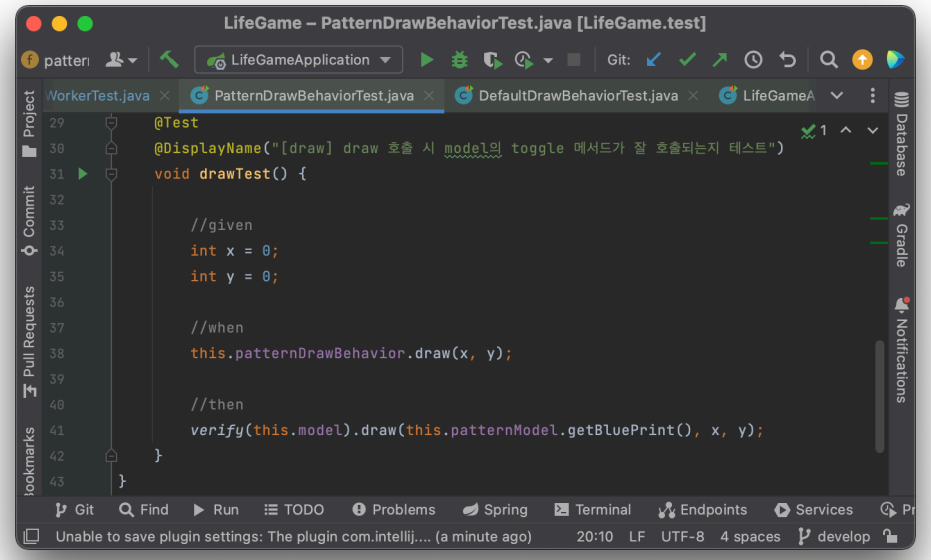
4.7.21. [Test] PatternDrawBehaviorTest

PatternDrawBehavior class 테스트하는 class이다.



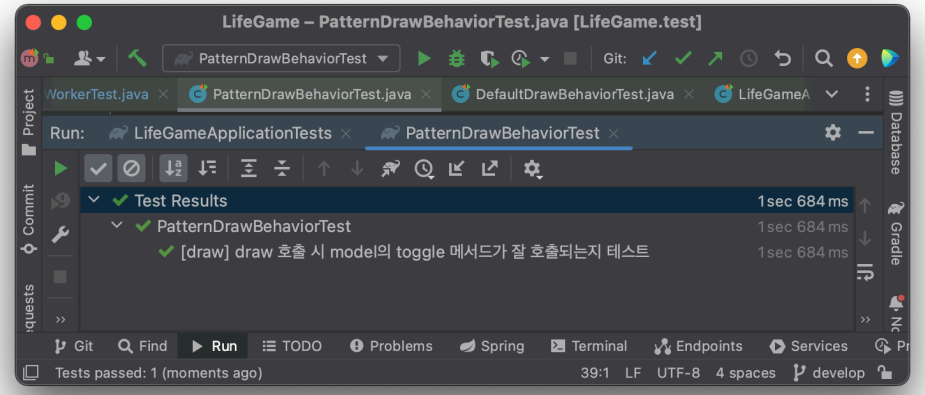
▲ [picture 153] test cases of PatternDrawBehavior class (1)

Model, PatternModel을 Mocking한다.



▲ **[picture 154] test cases of PatternDrawBehavior class** (2)

drawTest는 Model의 draw에 올바른 bluePrint와 x, y 좌표를 설정해 실행하는지 테스트한다.

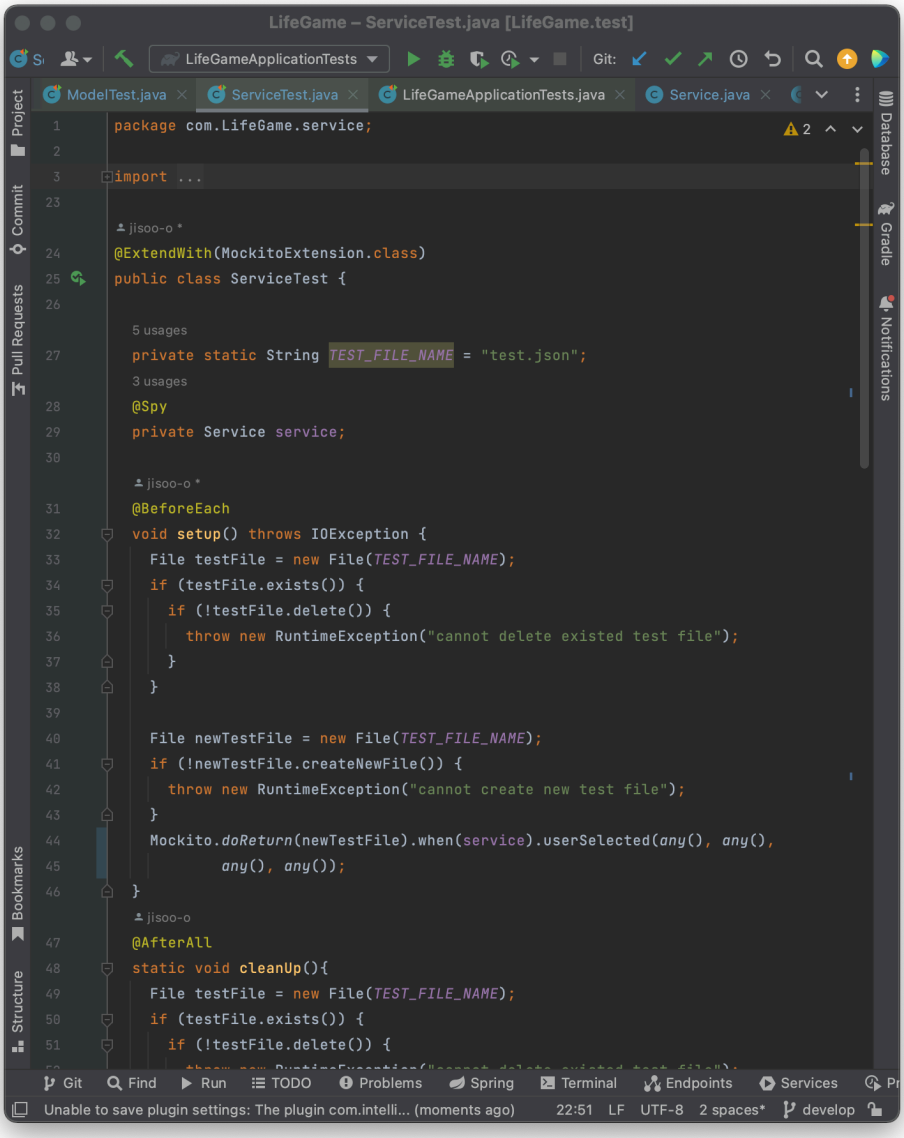


▲ **[picture 155] PatternDrawBehaviorTest passed**

PatternDrawBehaviorTest의 모든 Test가 통과하는 것을 확인할 수 있다.

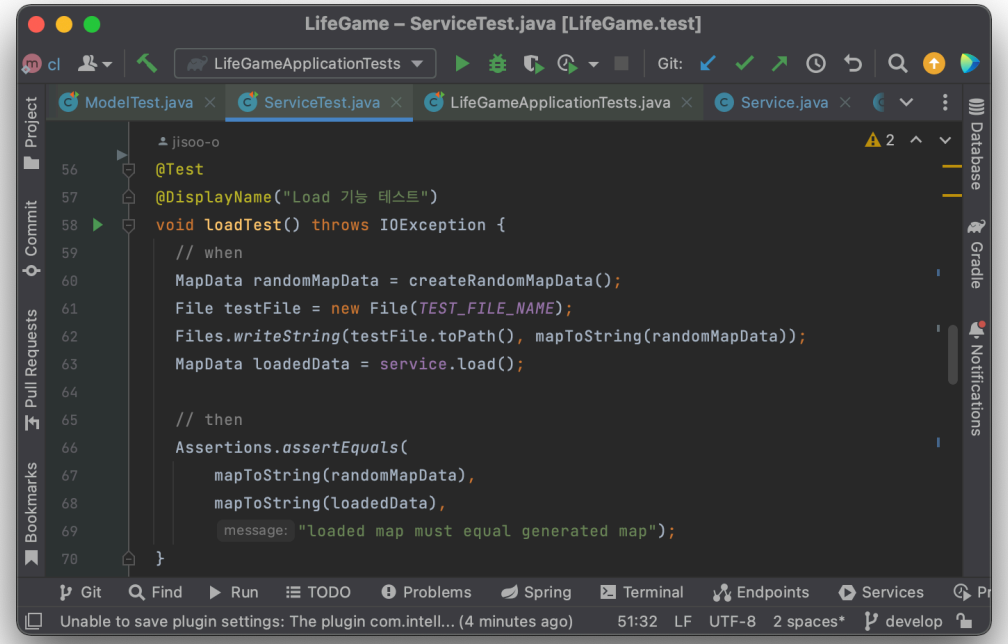
4.7.22. [Test] ServiceTest

Service class 테스트하는 class이다.



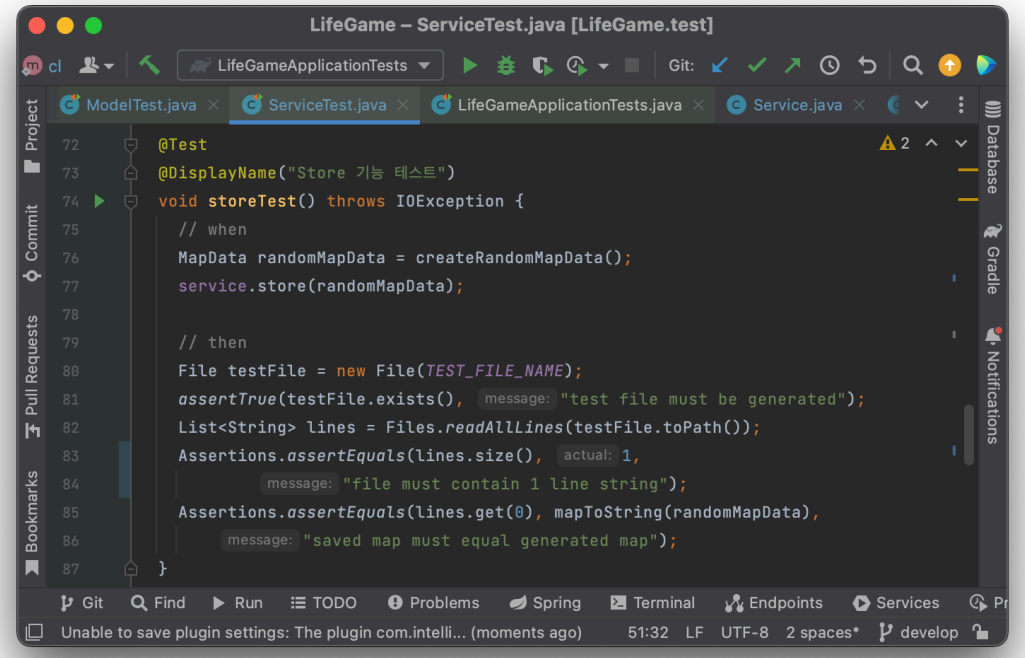
▲ [picture 156] test cases of Service class (1)

UI와 분리하여 테스트 하기 위해 @BeforeEach setUp( )에서 mock 테스트 파일을 선언하고 Mockito를 사용하여 userSelected를 처리한다. @AfterAll cleanUp( )에서 테스트에서 만든 mock 파일을 삭제한다.



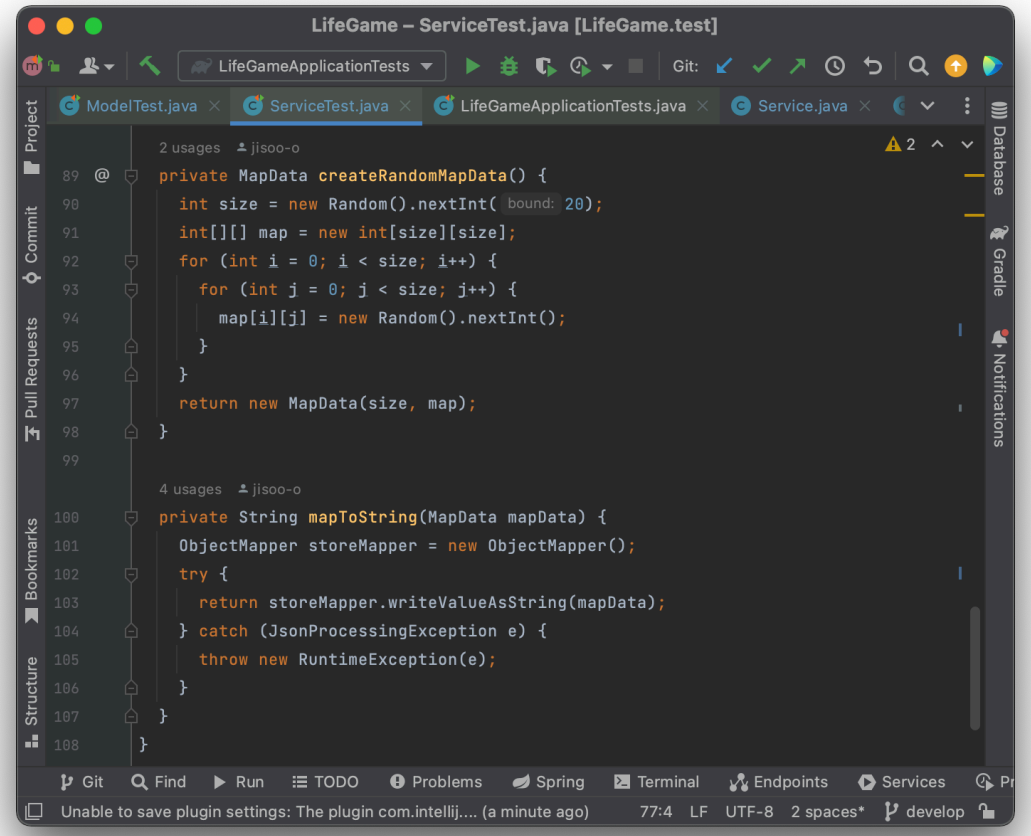
▲ [picture 157] test cases of Service class (2)

createRandomMapData()에서 만든 randomMapData에 대한 store, load 테스트를 진행한다. 테스트 파일의 값과 불러온 값이 일치하는지 assertEquals로 확인한다.



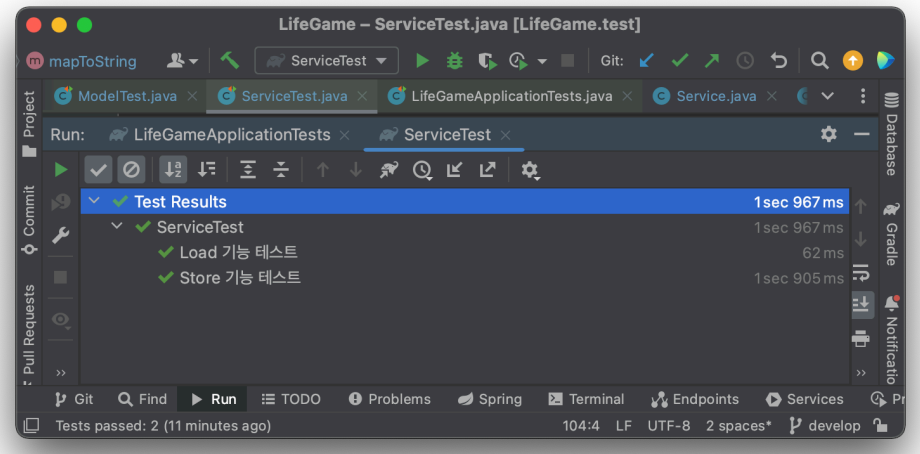
▲ [picture 158] test cases of Service class (3)

storeTest는 Service의 store method가 호출되었을 때 올바른 파일명에 올바른 파일 내용으로 덤프 파일이 생성되었는지 검증하는 테스트이다.



▲ [picture 159] test cases of Service class

storeTest와 loadTest에 사용되는 랜덤한 맵 데이터를 생성하는 createRandomMapData 메소드와 storeMapper를 이용하여 객체를 string 타입으로 변환하는 mapToString 메소드이다.

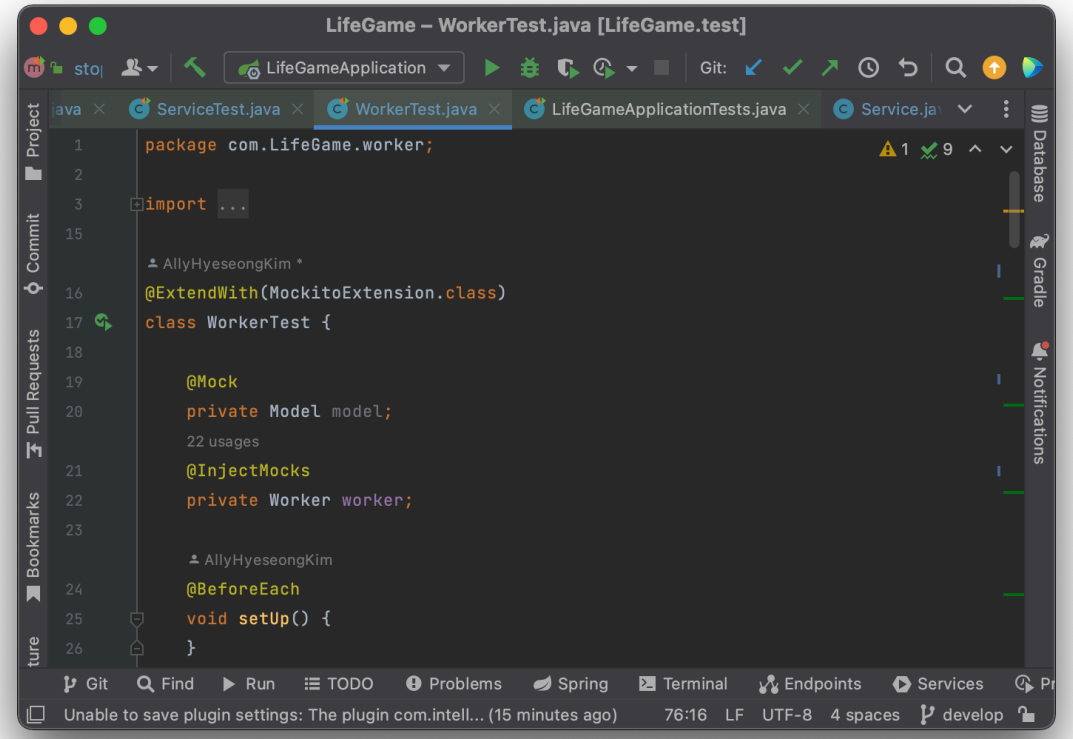


▲ [picture 160] ServiceTest passed

ServiceTest의 모든 Test가 통과하는 것을 확인할 수 있다.

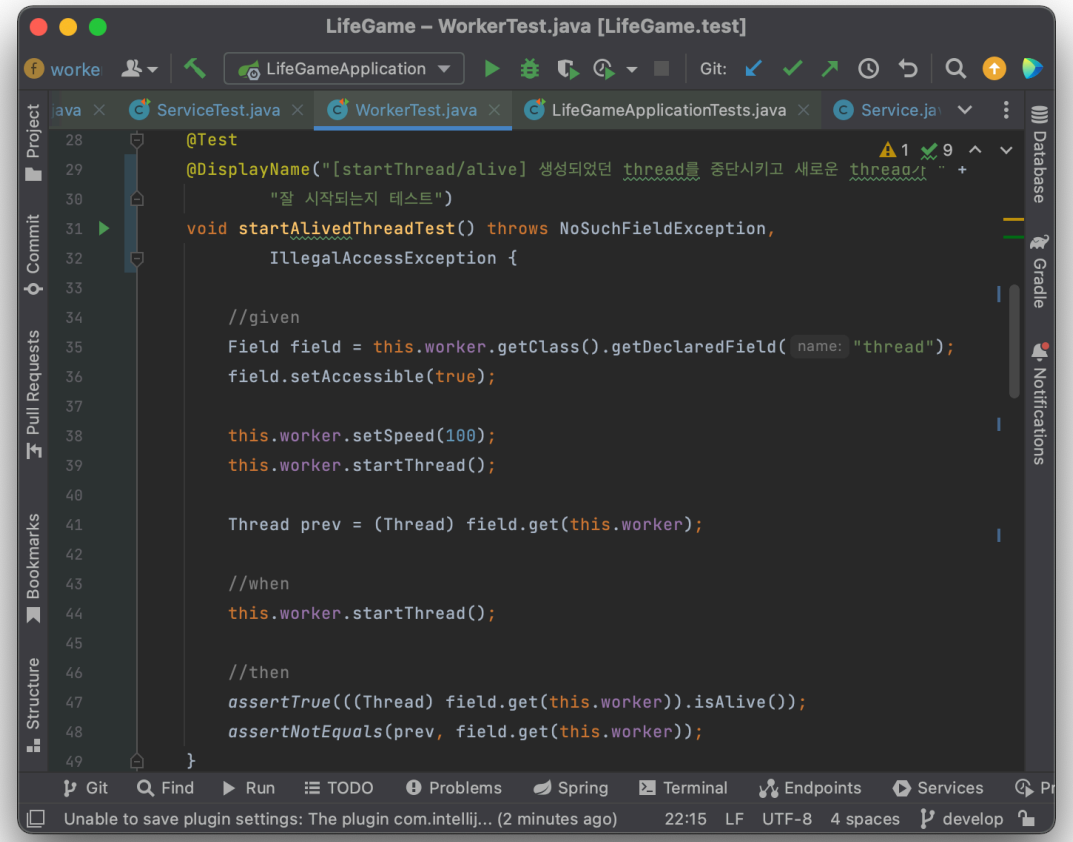
4.7.23. [Test] WorkerTest

Worker class 테스트하는 class이다.



▲ **[picture 161] test cases of Worker class (1)**

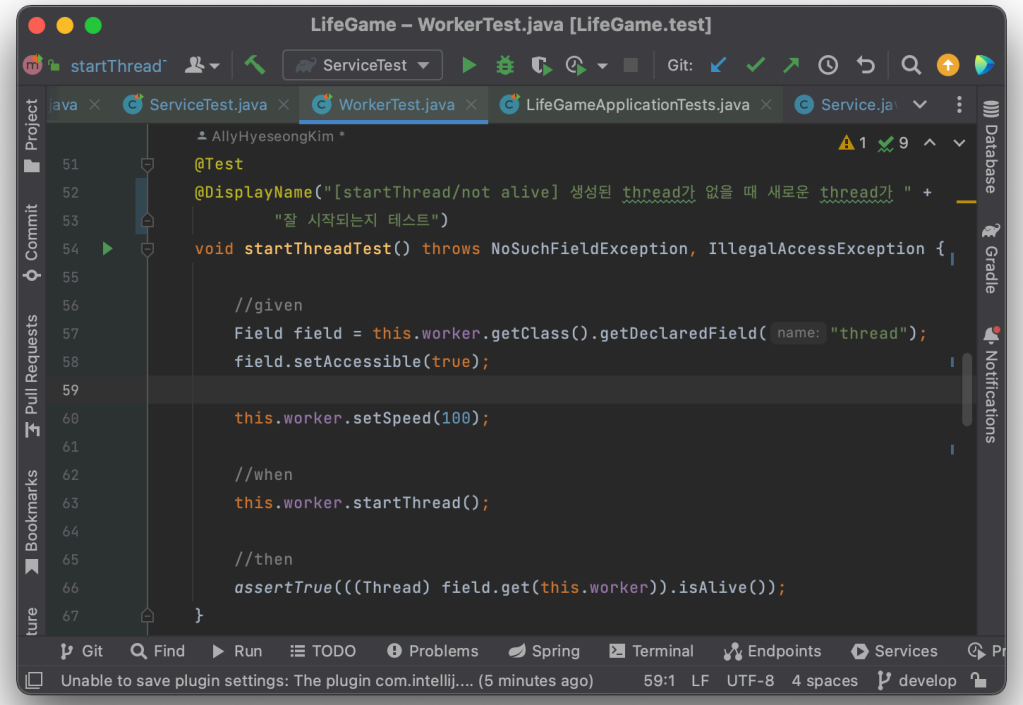
Model을 Mocking하여 Worker에 주입하여 테스트한다.



▲ **[picture 162] test cases of Worker class** (2)

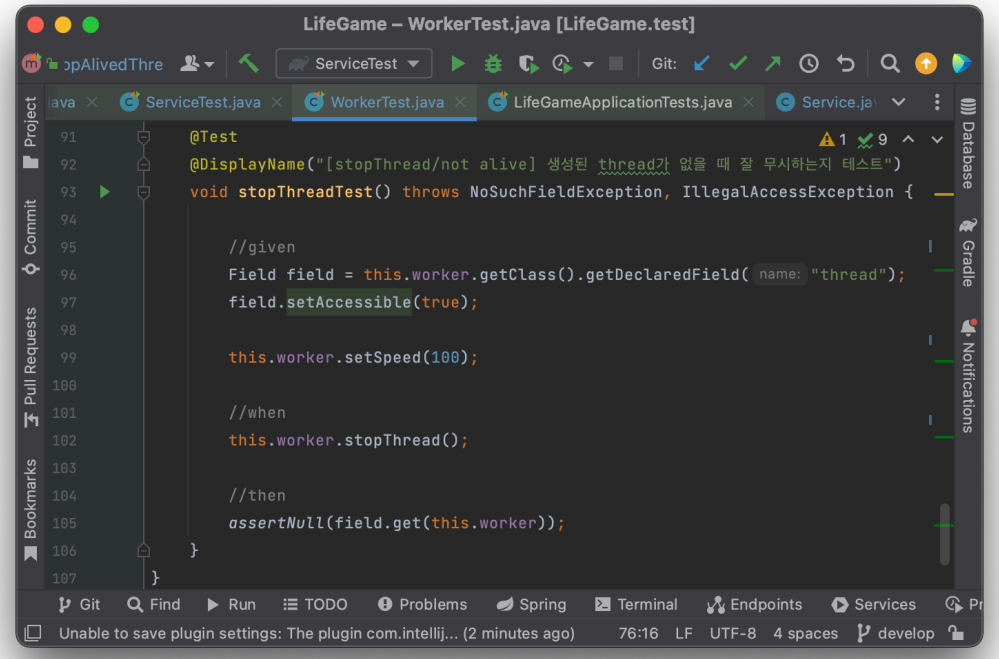
startAlivedThreadTest는 기존에 존재하는 스레드를 중단하고 새로운 스레드를 잘 실행하는지 테스트한다.

현재 worker의 스레드가 alive인지, 예전 스레드와 인스턴스가 다른지를 assert한다.



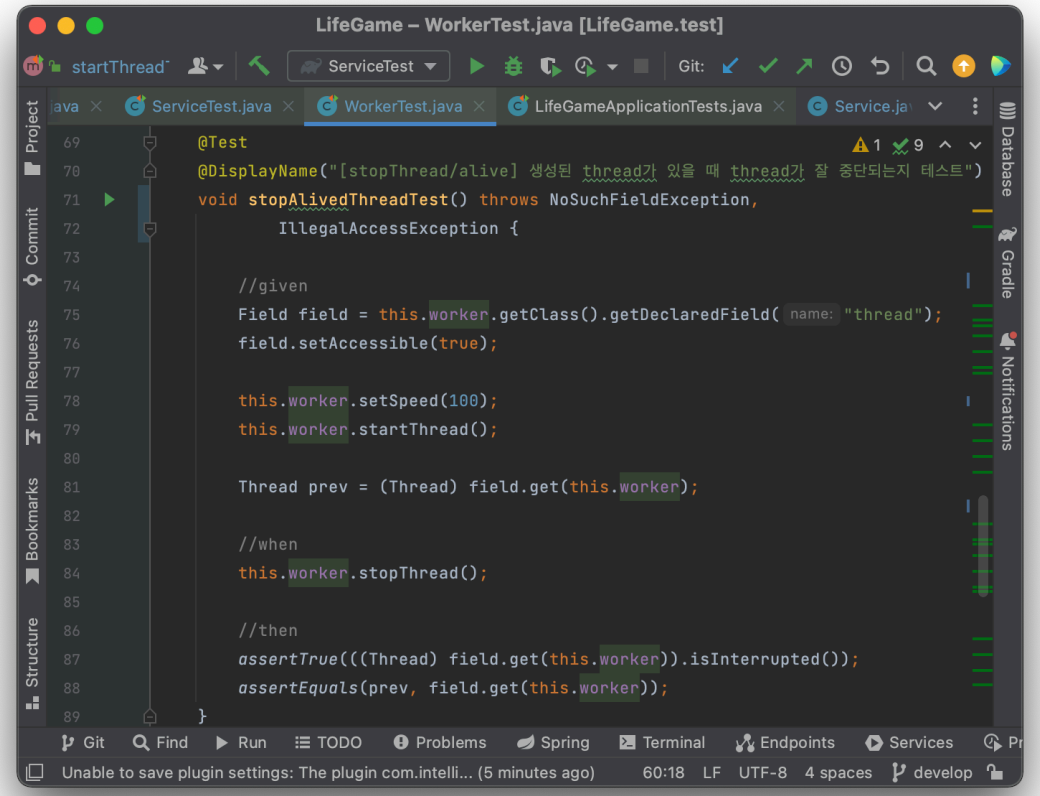
▲ [picture 163] test cases of Worker class (3)

startThreadTest는 기존에 스레드가 없을 때 새로운 스레드가 올바른 speed 설정값으로잘 생성되는지 테스트한다.



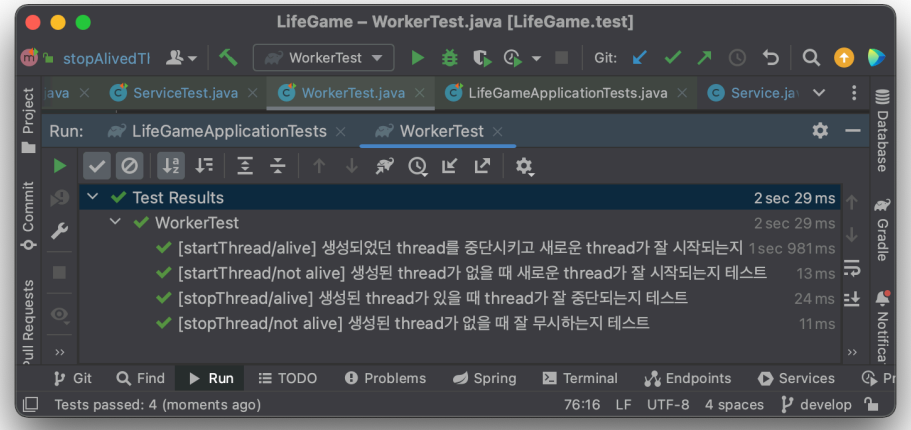
▲ [picture 164] test cases of Worker class (4)

stopThreadTest는 생성된 스레드가 없을 때 stopThread가 호출되어도 잘 무시되는지 테스트한다.



▲ **[picture 165] test cases of Worker class** (5)

stopAlivedThreadTest는 실행중인 스레드가 있을 때 stopThread를 호출하면 스레드가 잘 정지하는지 테스트한다.



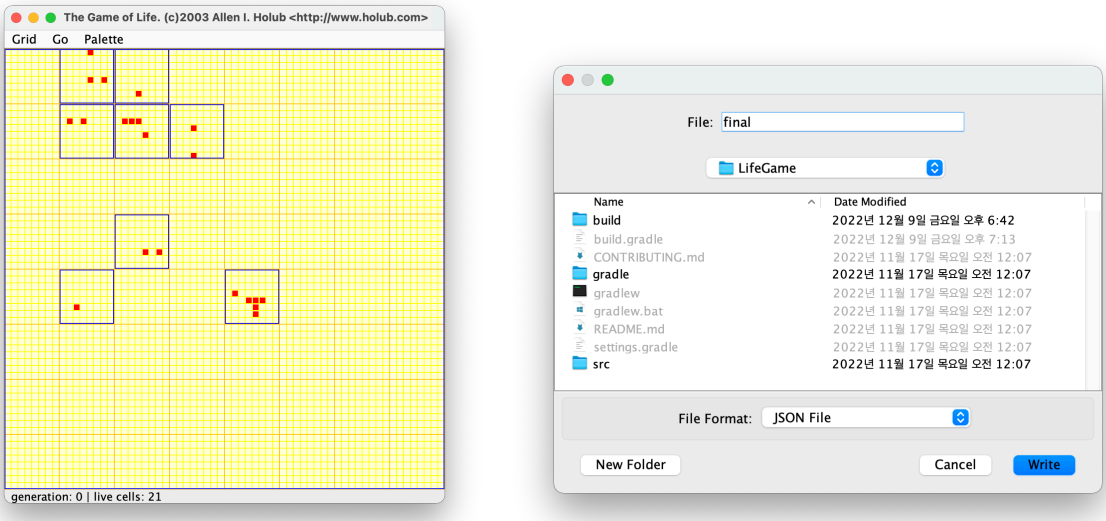
▲ **[picture 166] WorkerTest passed**

WorkerTest의 모든 Test가 통과하는 것을 확인할 수 있다.

5. 결과

5.1. 기존 Life Game 프로젝트 지원 기능

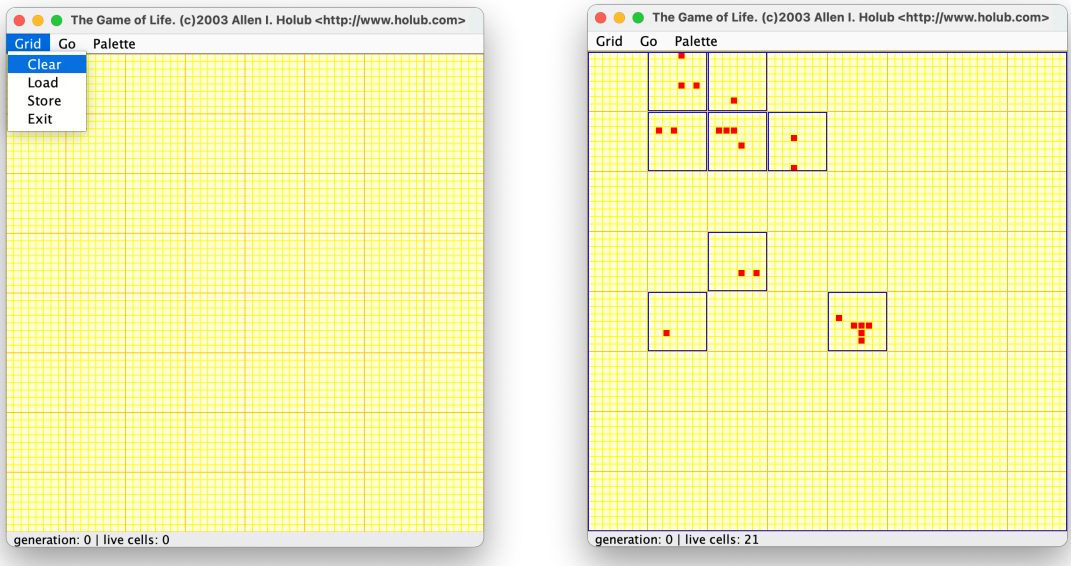
- Toggle , Store



▲ **[picture 167] Toggle, Store** 동작

빈 cell을 눌러 live 하거나 dead 상태로 toggle 하는 기능과 현재 상태 store 하는 기능이 정상적으로 동작한다.

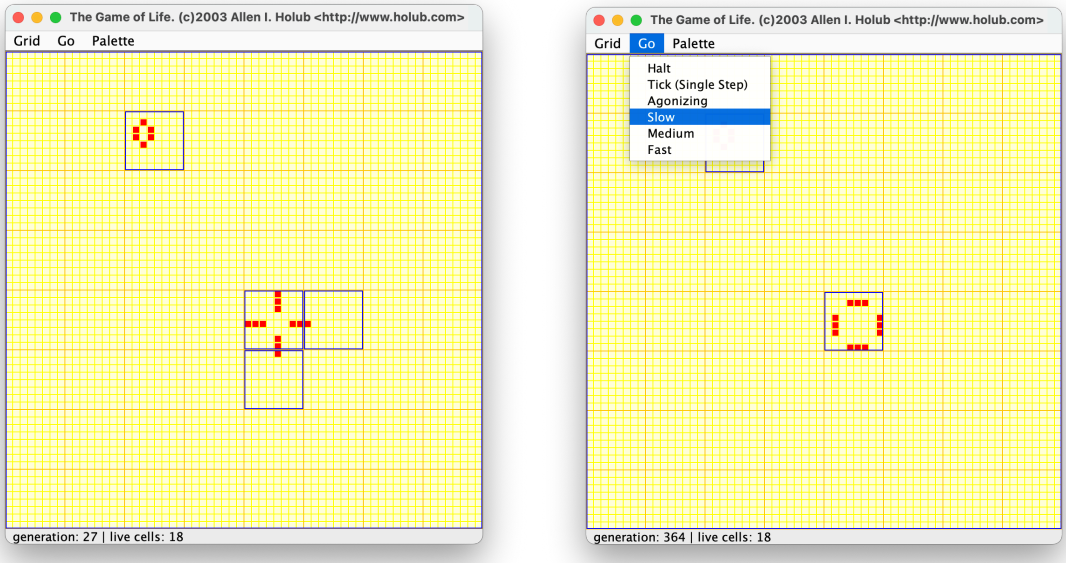
- Clear, Load



▲ **[picture 168] Clear, Load** 작동

Map을 한번에 지울 수 있는 clear 기능과 저장해둔 상태를 다시 불러오는 load 기능이 정상적으로 동작한다.

• Halt, Tick, Agonizing, Slow, Medium, Fast

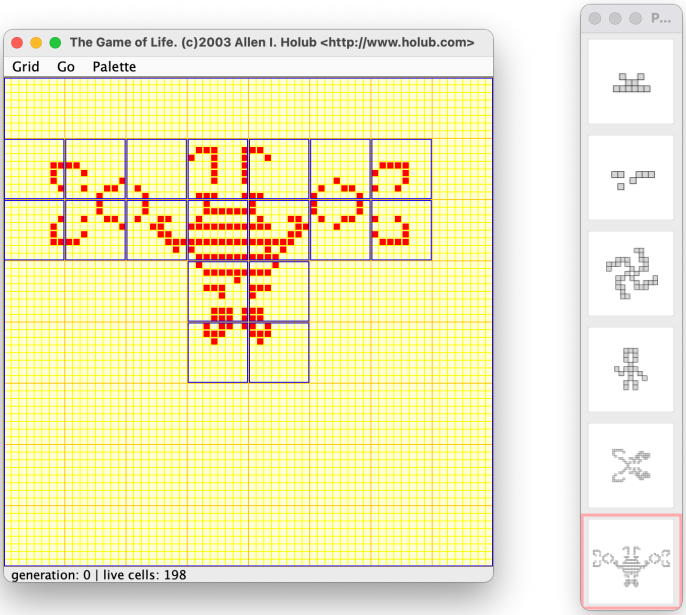


▲ **[picture 169] Go menu** 작동

진행을 멈추는 halt, 한 generation만 진행하는 tick, 속도 별로 진행하는 slow, medium, fast 기능도 모두 정상적으로 작동한다.

5.2. 확장 기능

• Palette



▲ **[picture 170] Palette** 작동

메뉴바의 Palette의 show, hide를 통해 팔레트를 열고 닫는다. 팔레트에 저장되어 있는 흥미로운 패턴을 선택하여 원하는 위치에 붙일 수 있다.

• 현재 generation 수, live cells 개수

▲ **[picture 171] Status bar** 작동

Status bar에서 현재 진행 상태인 generation 수와 live cells 수의 변화를 확인할 수 있다.

6. GitHub

6.1. Repository URL

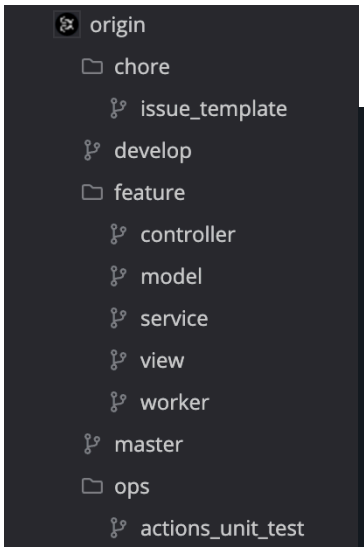
https://github.com/CAU-DesignPattern/LifeGame

해당 레포에서 각자 개인 레포에 fork를 떠서 작업하였다.

6.2. Commit history

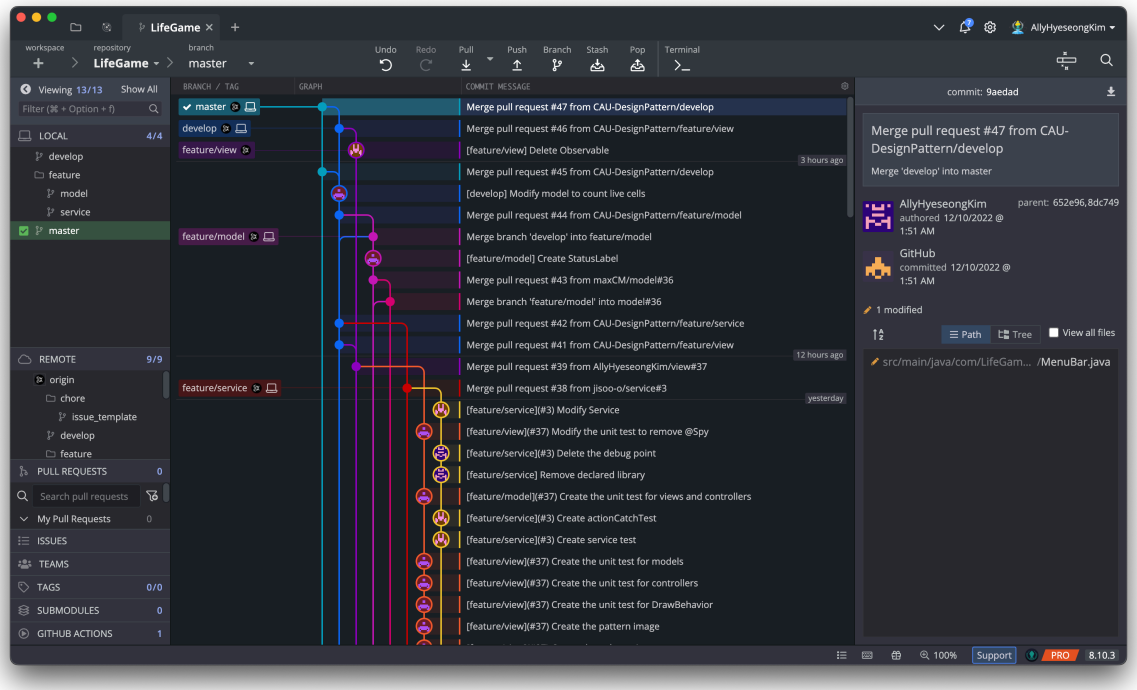
https://github.com/CAU-DesignPattern/LifeGame/commits/master

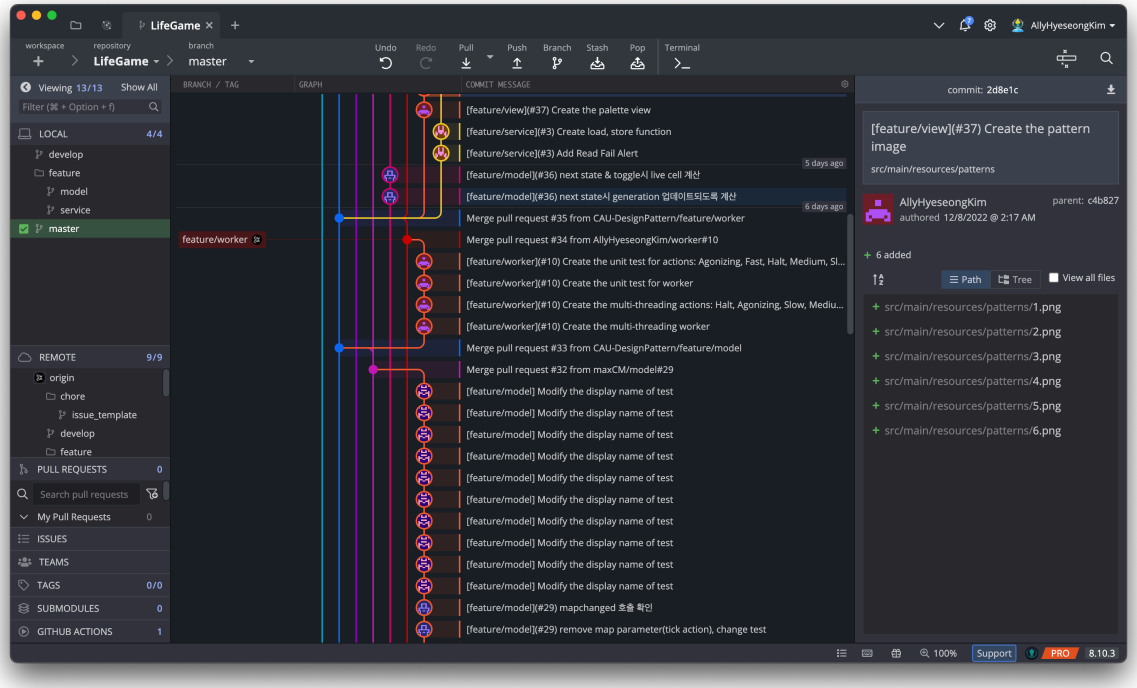
https://github.com/CAU-DesignPattern/LifeGame/blob/master/CONTRIBUTING.md

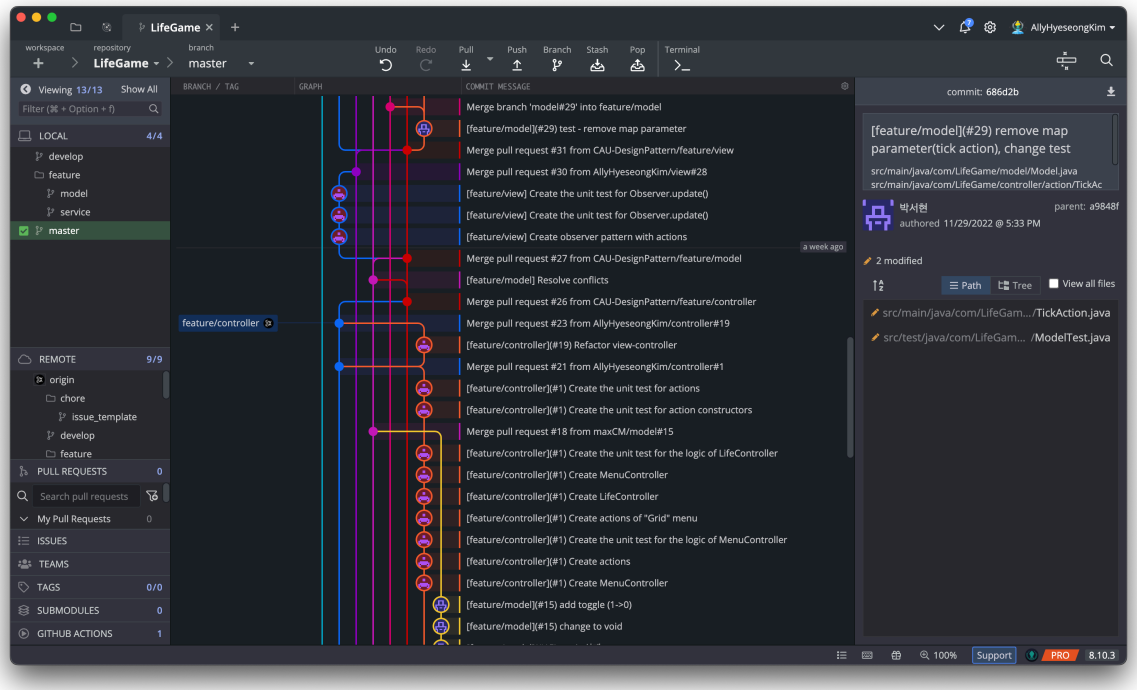


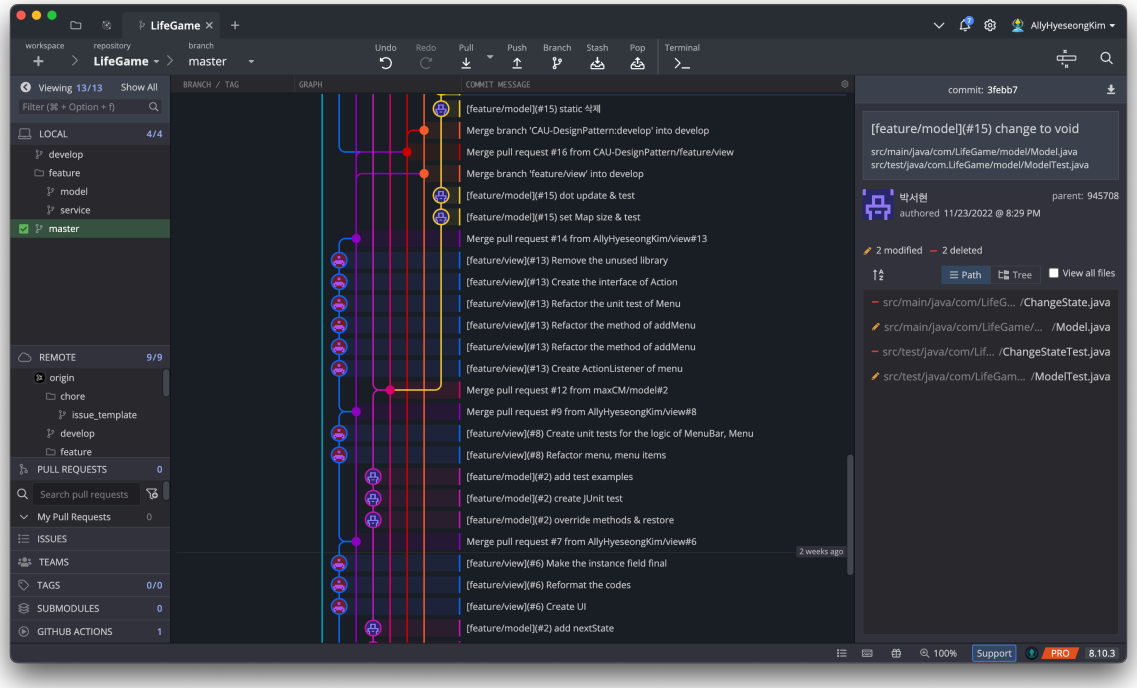

```
master: 제품으로 출시될 수 있는 브랜치
develop: 다음 출시 버전을 개발하는 브랜치
feature: 기능을 개발하는 브랜치
* feature/{기능}
* issue가 있는 경우, [feature/{기능}](#{이슈번호}) message
ops: 프로젝트 운영에 쓰이는 브랜치
chore: 문서 등 기타 단순 수정하는 브랜치
* [chore/{한 일}]
fix: 이슈를 반영하여 수정하는 브랜치
* [fix/{한 일}]
* [fix/{한 일}](#{이슈번호}) message
release: 이번 출시 버전을 준비하는 브랜치
hotfix: 출시 버전에서 발생한 버그를 수정하는 브랜치
```

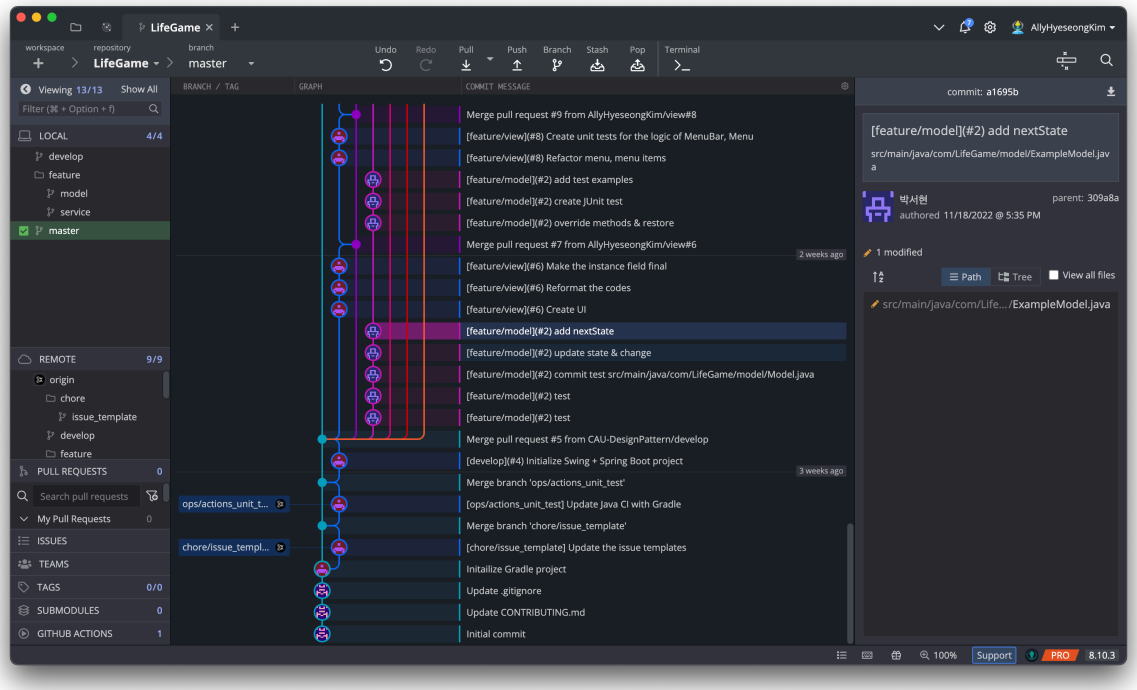
해당하는 각 브랜치에 {브랜치}#{이슈번호} 브랜치로 Pull Request를 날리는 방식으로 작업하였다.


LifeGame
AllyHyeseongKim
workspace
repository
branch
LifeGame
master
Undo
Redo
Pull
Push
Branch
Stash
Pop
Terminal
Viewing 13/13
Show All
Filter (⌘ + Option + f)
LOCAL 4/4
develop
feature
model
service
master
REMOTE 9/9
origin
chore
issue_template
develop
feature
PULL REQUESTS 0
Search pull requests
My Pull Requests 0
ISSUES
TEAMS
TAGS 0/0
SUBMODULES 0
GITHUB ACTIONS 1
BRANCH / TAG
GRAPH
COMMIT MESSAGE
master
develop
feature/view
feature/model
feature/service
Merge pull request #47 from CAU-DesignPattern/develop
Merge pull request #46 from CAU-DesignPattern/feature/view
[feature/view] Delete Observable
3 hours ago
Merge pull request #45 from CAU-DesignPattern/develop
[develop] Modify model to count live cells
Merge pull request #44 from CAU-DesignPattern/feature/model
Merge branch 'develop' into feature/model
[feature/model] Create StatusLabel
Merge pull request #43 from maxCM/model#36
Merge branch 'feature/model' into model#36
Merge pull request #42 from CAU-DesignPattern/feature/service
Merge pull request #41 from CAU-DesignPattern/feature/view
12 hours ago
Merge pull request #39 from AllyHyeseongKim/view#37
Merge pull request #38 from jisoo-o/service#3
yesterday
[feature/service](#3) Modify Service
[feature/view](#37) Modify the unit test to remove @Spy
[feature/service](#3) Delete the debug point
[feature/service] Remove declared library
[feature/model](#37) Create the unit test for views and controllers
[feature/service](#3) Create actionCatchTest
[feature/service](#3) Create service test
[feature/view](#37) Create the unit test for models
[feature/view](#37) Create the unit test for controllers
[feature/view](#37) Create the unit test for DrawBehavior
[feature/view](#37) Create the pattern image
commit: 9aedad
Merge pull request #47 from CAU-DesignPattern/develop
Merge 'develop' into master
AllyHyeseongKim
parent: 652e96, 8dc749
authored 12/10/2022 @ 1:51 AM
GitHub
committed 12/10/2022 @ 1:51 AM
1 modified
Path
Tree
View all files
src/main/java/com/LifeGam... /MenuBar.java
100%
Support
PRO
8.10.3


LifeGame
AllyHyeseongKim
workspace
repository
branch
LifeGame
master
Undo
Redo
Pull
Push
Branch
Stash
Pop
Terminal
Viewing 13/13
Show All
Filter (⌘ + Option + f)
LOCAL 4/4
develop
feature
model
service
master
REMOTE 9/9
origin
chore
issue_template
develop
feature
PULL REQUESTS 0
Search pull requests
My Pull Requests 0
ISSUES
TEAMS
TAGS 0/0
SUBMODULES 0
GITHUB ACTIONS 1
BRANCH / TAG
GRAPH
COMMIT MESSAGE
feature/worker
[feature/view](#37) Create the palette view
[feature/service](#3) Create load, store function
[feature/service](#3) Add Read Fail Alert
5 days ago
[feature/model](#36) next state & toggle시 live cell 계산
[feature/model](#36) next state시 generation 업데이트되도록 계산
6 days ago
Merge pull request #35 from CAU-DesignPattern/feature/worker
Merge pull request #34 from AllyHyeseongKim/worker#10
[feature/worker](#10) Create the unit test for actions: Agonizing, Fast, Halt, Medium, Sl...
[feature/worker](#10) Create the unit test for worker
[feature/worker](#10) Create the multi-threading actions: Halt, Agonizing, Slow, Mediu...
[feature/worker](#10) Create the multi-threading worker
Merge pull request #33 from CAU-DesignPattern/feature/model
Merge pull request #32 from maxCM/model#29
[feature/model] Modify the display name of test
[feature/model] Modify the display name of test
[feature/model] Modify the display name of test
[feature/model] Modify the display name of test
[feature/model] Modify the display name of test
[feature/model] Modify the display name of test
[feature/model] Modify the display name of test
[feature/model] Modify the display name of test
[feature/model] Modify the display name of test
[feature/model] Modify the display name of test
[feature/model](#29) mapchanged 호출 확인
[feature/model](#29) remove map parameter(tick action), change test
commit: 2d8e1c
[feature/view](#37) Create the pattern image
src/main/resources/patterns
AllyHyeseongKim
parent: c4b827
authored 12/8/2022 @ 2:17 AM
6 added
Path
Tree
View all files
+ src/main/resources/patterns/1.png
+ src/main/resources/patterns/2.png
+ src/main/resources/patterns/3.png
+ src/main/resources/patterns/4.png
+ src/main/resources/patterns/5.png
+ src/main/resources/patterns/6.png
100%
Support
PRO
8.10.3


LifeGame
AllyHyeseongKim
workspace
repository
LifeGame
branch
master
Undo
Redo
Pull
Push
Branch
Stash
Pop
Terminal
Viewing 13/13
Show All
Filter (⌘ + Option + f)
LOCAL 4/4
develop
feature
model
service
master
REMOTE 9/9
origin
chore
issue_template
develop
feature
PULL REQUESTS 0
Search pull requests
My Pull Requests 0
ISSUES
TEAMS
TAGS 0/0
SUBMODULES 0
GITHUB ACTIONS 1
BRANCH / TAG
GRAPH
COMMIT MESSAGE
Merge branch 'model#29' into feature/model
[feature/model](#29) test - remove map parameter
Merge pull request #31 from CAU-DesignPattern/feature/view
Merge pull request #30 from AllyHyeseongKim/view#28
[feature/view] Create the unit test for Observer.update()
[feature/view] Create the unit test for Observer.update()
[feature/view] Create observer pattern with actions
a week ago
Merge pull request #27 from CAU-DesignPattern/feature/model
[feature/model] Resolve conflicts
Merge pull request #26 from CAU-DesignPattern/feature/controller
feature/controller
Merge pull request #23 from AllyHyeseongKim/controller#19
[feature/controller](#19) Refactor view-controller
Merge pull request #21 from AllyHyeseongKim/controller#1
[feature/controller](#1) Create the unit test for actions
[feature/controller](#1) Create the unit test for action constructors
Merge pull request #18 from maxCM/model#15
[feature/controller](#1) Create the unit test for the logic of LifeController
[feature/controller](#1) Create MenuController
[feature/controller](#1) Create LifeController
[feature/controller](#1) Create actions of "Grid" menu
[feature/controller](#1) Create the unit test for the logic of MenuController
[feature/controller](#1) Create actions
[feature/controller](#1) Create MenuController
[feature/model](#15) add toggle (1->0)
[feature/model](#15) change to void
commit: 686d2b
[feature/model](#29) remove map parameter(tick action), change test
src/main/java/com/LifeGame/model/Model.java
src/main/java/com/LifeGame/controller/action/TickAc
박서현
parent: a9848f
authored 11/29/2022 @ 5:33 PM
2 modified
Path
Tree
View all files
src/main/java/com/LifeGam.../TickAction.java
src/test/java/com/LifeGam... /ModelTest.java
100%
Support
PRO
8.10.3


LifeGame
AllyHyeseongKim
workspace
repository
LifeGame
branch
master
Undo
Redo
Pull
Push
Branch
Stash
Pop
Terminal
Viewing 13/13
Show All
Filter (⌘ + Option + f)
LOCAL 4/4
develop
feature
model
service
master
REMOTE 9/9
origin
chore
issue_template
develop
feature
PULL REQUESTS 0
Search pull requests
My Pull Requests 0
ISSUES
TEAMS
TAGS 0/0
SUBMODULES 0
GITHUB ACTIONS 1
BRANCH / TAG
GRAPH
COMMIT MESSAGE
[feature/model](#15) static 삭제
Merge branch 'CAU-DesignPattern:develop' into develop
Merge pull request #16 from CAU-DesignPattern/feature/view
Merge branch 'feature/view' into develop
[feature/model](#15) dot update & test
[feature/model](#15) set Map size & test
Merge pull request #14 from AllyHyeseongKim/view#13
[feature/view](#13) Remove the unused library
[feature/view](#13) Create the interface of Action
[feature/view](#13) Refactor the unit test of Menu
[feature/view](#13) Refactor the method of addMenu
[feature/view](#13) Refactor the method of addMenu
[feature/view](#13) Create ActionListener of menu
Merge pull request #12 from maxCM/model#2
Merge pull request #9 from AllyHyeseongKim/view#8
[feature/view](#8) Create unit tests for the logic of MenuBar, Menu
[feature/view](#8) Refactor menu, menu items
[feature/model](#2) add test examples
[feature/model](#2) create JUnit test
[feature/model](#2) override methods & restore
Merge pull request #7 from AllyHyeseongKim/view#6
2 weeks ago
[feature/view](#6) Make the instance field final
[feature/view](#6) Reformat the codes
[feature/view](#6) Create UI
[feature/model](#2) add nextState
commit: 3febb7
[feature/model](#15) change to void
src/main/java/com/LifeGame/model/Model.java
src/test/java/com.LifeGame/model/ModelTest.java
박서현
parent: 945708
authored 11/23/2022 @ 8:29 PM
2 modified
2 deleted
Path
Tree
View all files
src/main/java/com/LifeG... /ChangeState.java
src/main/java/com/LifeGame/... /Model.java
src/test/java/com/Lif... /ChangeStateTest.java
src/test/java/com/LifeGam... /ModelTest.java
100%
Support
PRO
8.10.3



▲ **[picture 172] Commit history**

6.3. Issues (역할분배)

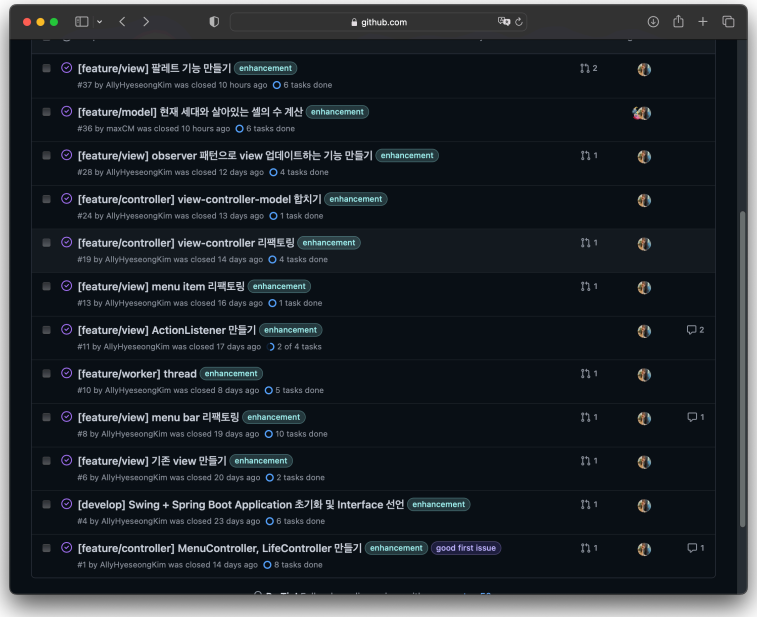
https://github.com/CAU-DesignPattern/LifeGame/tree/master/.github/ISSUE_TEMPLATE

https://github.com/CAU-DesignPattern/LifeGame/issues?q=is:issue+is:closed

Issue template을 만들어서 모든 작업은 issue로 등록하여 issue 단위로 각 역할을 분배하여 작업하였다.

6.3.1. 김혜성 작업 issue

https://github.com/CAU-DesignPattern/LifeGame/issues?q=is:issue+is:closed+assignee:AllyHyeseongKim

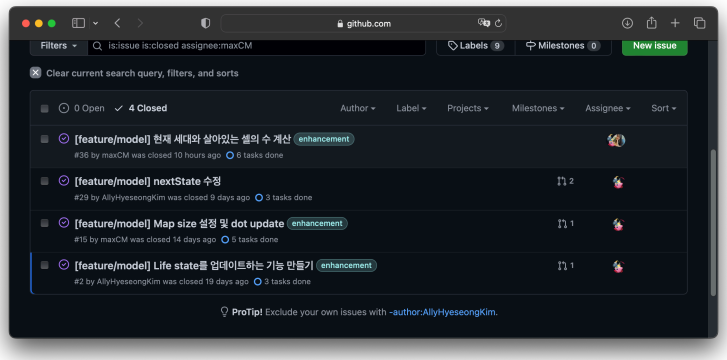


▲ **[picture 173]** 김혜성 작업 **issue**

• 기존 기능 view, controller 개발
• 기존 기능 worker (thread를 사용하여 state 업데이트) 개발
• Observer pattern 적용하여 view 업데이트
• Spring 적용
• 확장 기능 Palette(model, view, controller) 개발
• 확장 기능 현재 세대, 셀 수 계산 개발

6.3.2. 박서현 작업 issue

https://github.com/CAU-DesignPattern/LifeGame/issues?q=is:issue+is:closed+assignee:maxCM

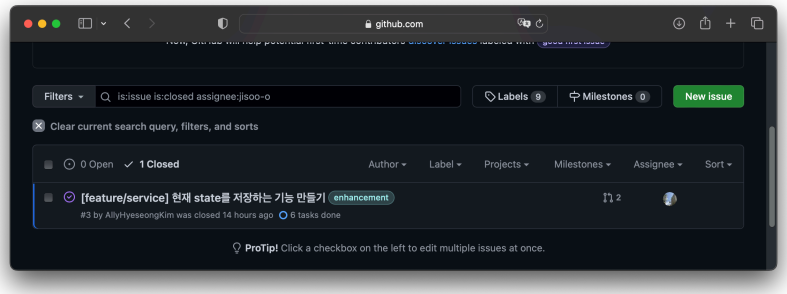


▲ **[picture 174]** 박서현 작업 **issue**

• 기존 기능 model 개발
• 확장 기능 현재 세대, 셀 수 계산 개발

6.3.3. 박지수 작업 issue

https://github.com/CAU-DesignPattern/LifeGame/issues?q=is:issue+is:closed+assignee:jisoo-o



▲ **[picture 175]** 박지수 작업 **issue**

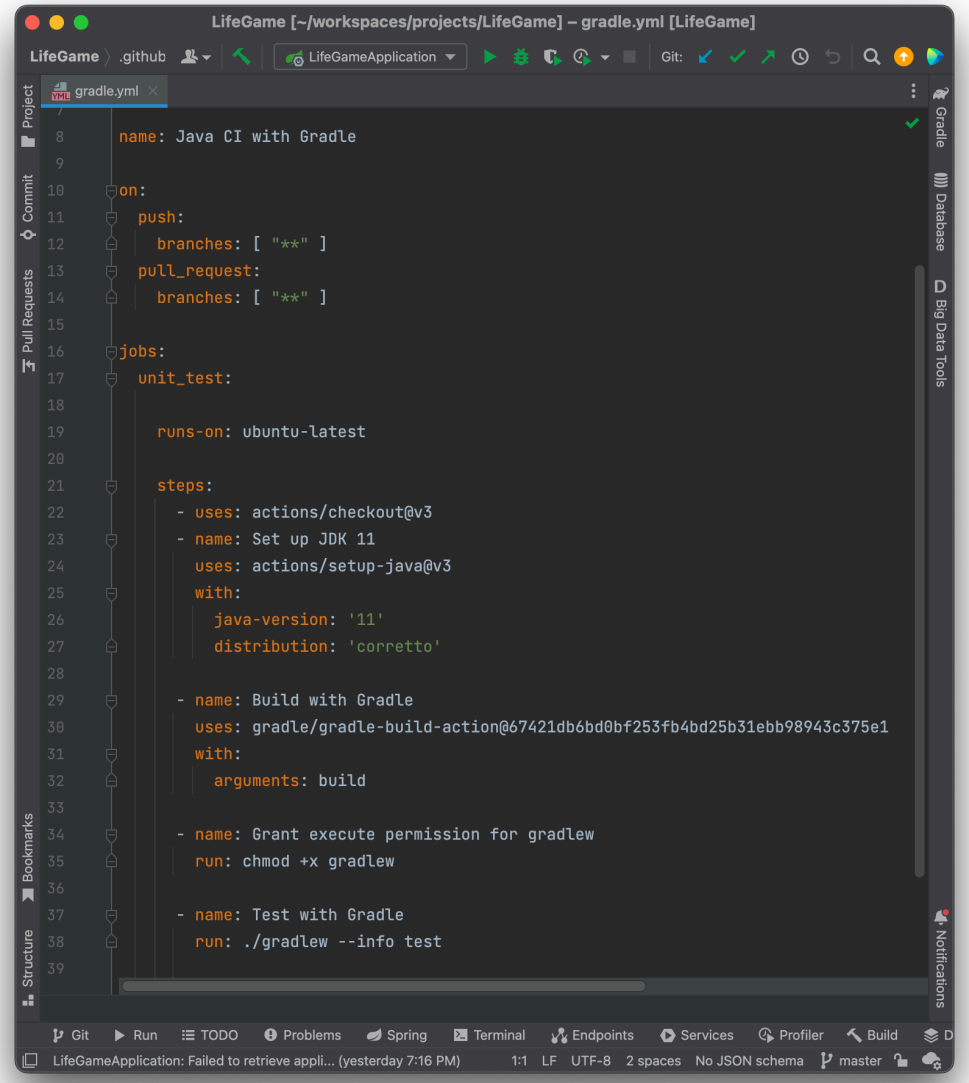
• 기존 기능 service 개발

6.4. Actions

https://github.com/CAU-DesignPattern/LifeGame/blob/master/.github/workflows/gradle.yml

https://github.com/CAU-DesignPattern/LifeGame/actions

Java gradle unit test CI를 만들어서 test를 통과하지 않으면 Pull Request를 merge하지 않도록 하였다.



▲ [picture 176] Java CI with Gradle