A Brief Introduction to the HAnS Tool

HAnS Overview

- HAnS Tool (IDE-Based Editing Support for Embedded Feature Annotations).

- Java-based plugin for recording feature traceability in code assets.

- Recording feature traceability explicitly eliminates the need for feature location.

What is a Feature?

- In software engineering, a feature refers to a specific quality or observable behaviour of a software system that users can experience. It is any functionality meaningful to an end-user (e.g., feature ``Calling'' for mobile phone users).

- Features are used as a means of describing the functionality and characteristics of the software.

- Developers often need to locate features in order to maintain them, enhance them, and even reuse them.

- Without explicit documentation, feature location can take significant time and efforts, especially in very large software systems.

HAnS Tool (IDE-Based Editing Support for Embedded Feature Annotations)

- Allows explicitly documenting features inside software assets.

- Simplifies the process of documenting and eliminates the need for locating features in software assets.

- Allows developers to write embedded feature annotations directly within their code assets, such as files, folders, code fragments, and individual lines of code.

HAnS Tool (IDE-Based Editing Support for Embedded Feature Annotations)

HAnS-text provides support for:

1. Embedded Feature Annotations
2. Mapping code fragments to features
3. Mapping files or directories to features
4. Completion aid when annotating
5. Feature Model View
6. Feature Referencing
7. Renaming features
8. Quick fixes
9. Live templates

Types of embedded annotations

- Feature model
- Feature to folder mapping
- Feature to file mapping
- Feature to code-block mapping
- Feature to line mapping

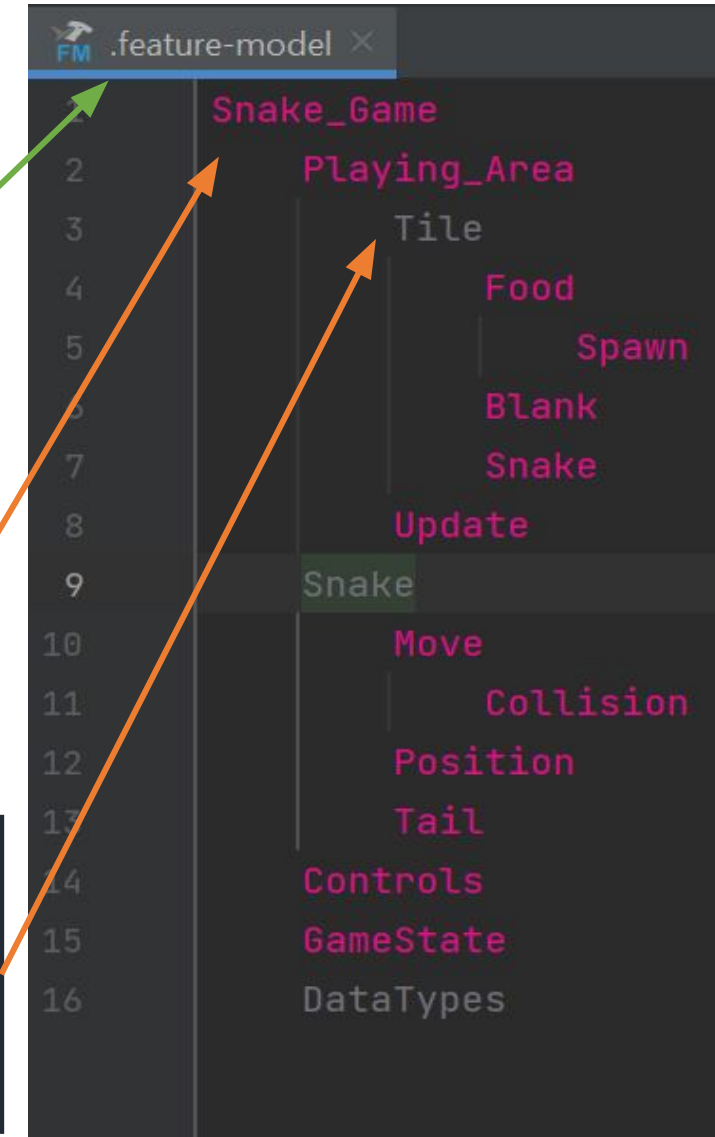# Embedded feature annotations

Feature Model

With the help of the HAns plugin, developers can create feature models, add and remove features, rename features, and add mappings between features and the assets that implement them (explained shortly)

Features are specified one per line, where the indentation shows hierarchy, i.e., features one tab later than their previous features are sub-features of those features (e.g., Tile is a sub-feature of Playing_Area)
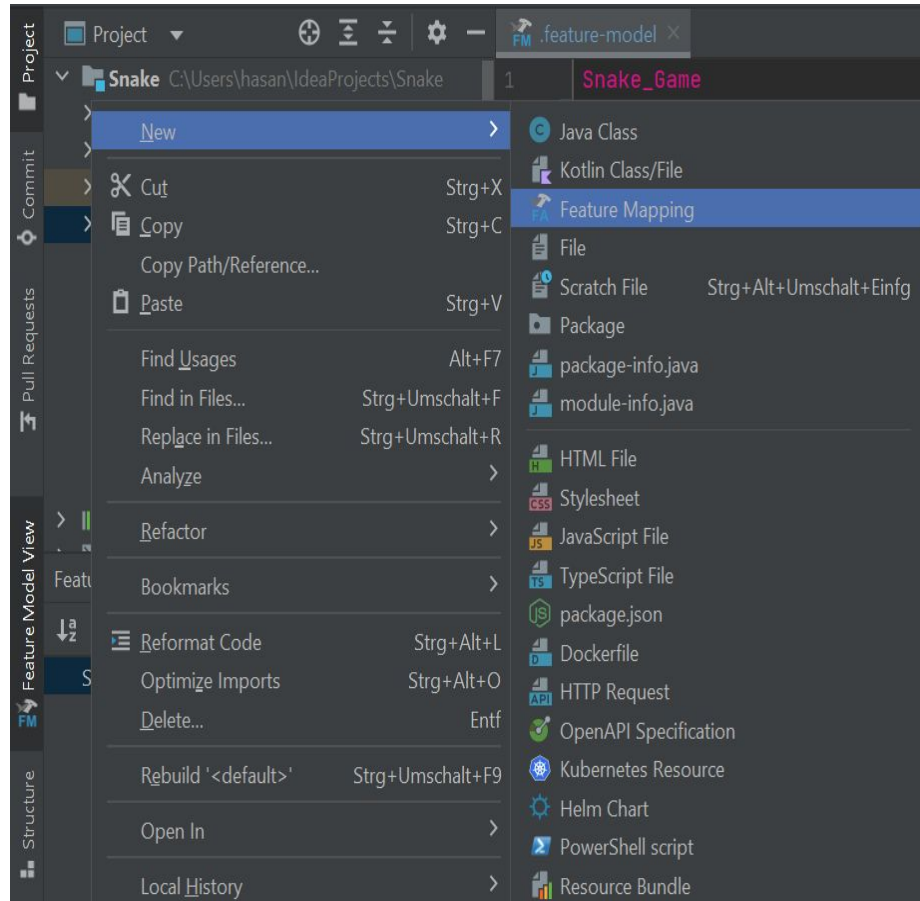
.feature-model

Feature models are used to describe the features, their relationships, and constraints between them.

Abstract features are grey in color (they do not have any mapped assets)


```
FM  .feature-model  ×

 1    Snake_Game
 2        Playing_Area
 3            Tile
 4                Food
 5                    Spawn
 6                Blank
 7                Snake
 8        Update
 9    Snake
10        Move
11            Collision
12        Position
13        Tail
14    Controls
15    GameState
16    DataTypes
```
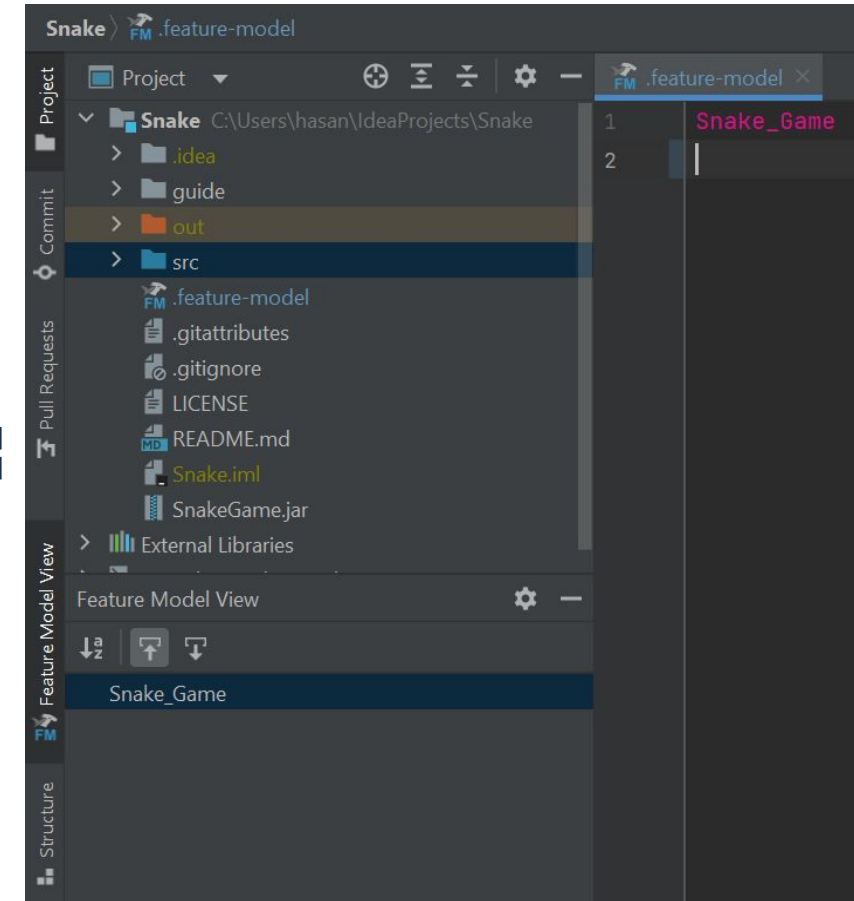
Adding **a feature model**

OR

Add a new feature model by right-clicking ``New`` in the Project/Folder and selecting ``Feature Mapping``
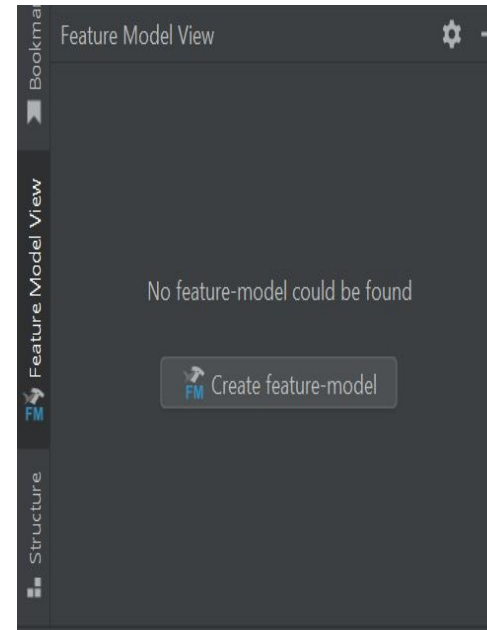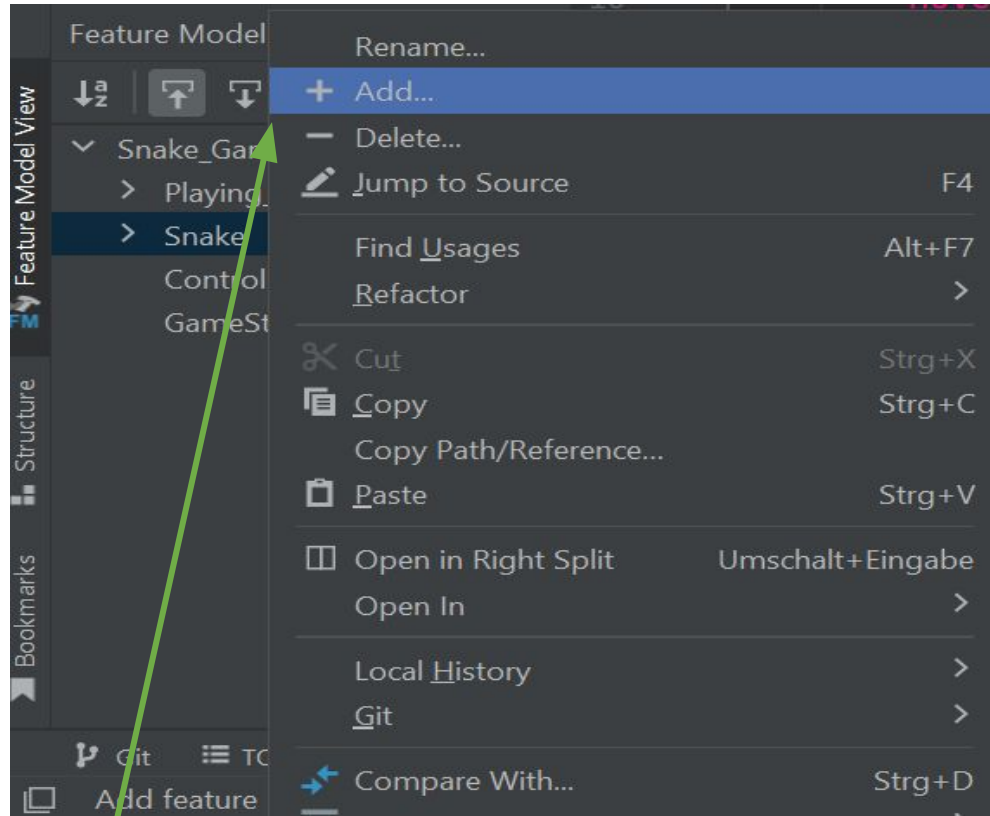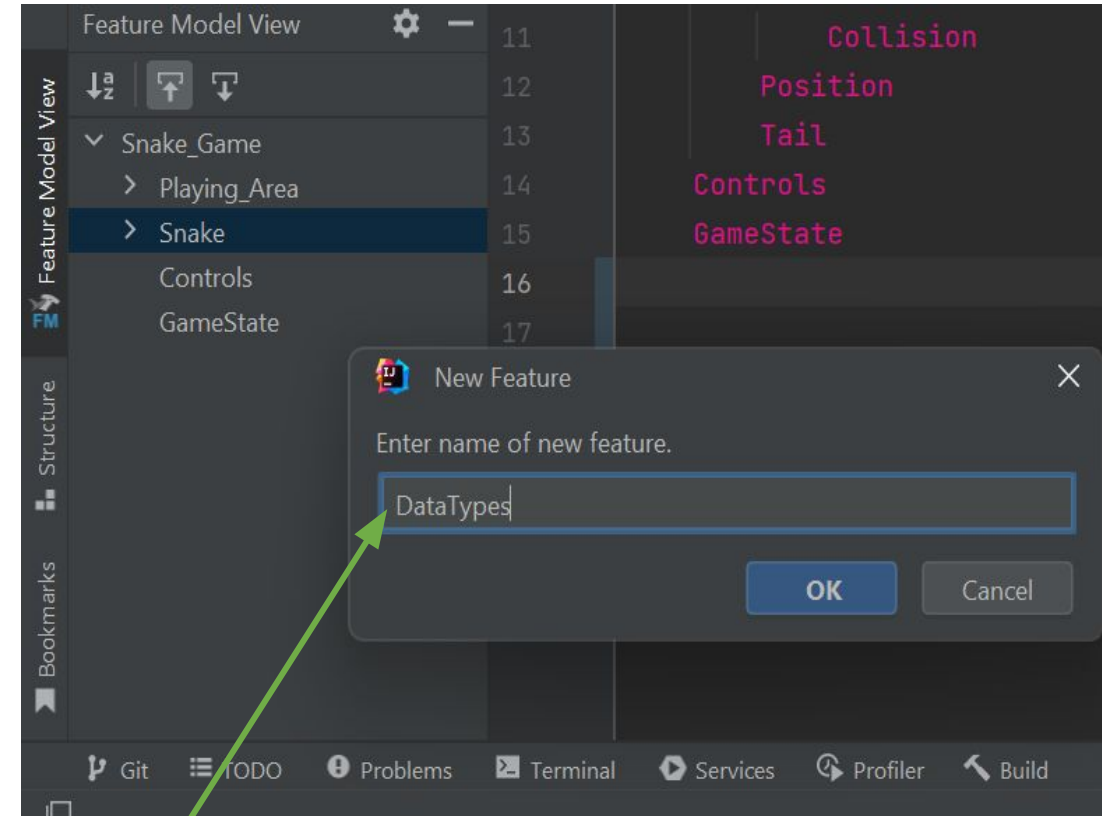
Click ``Create feature-model`` in the Feature Model View

The feature model is created and added under the "src" folder of the Snake project
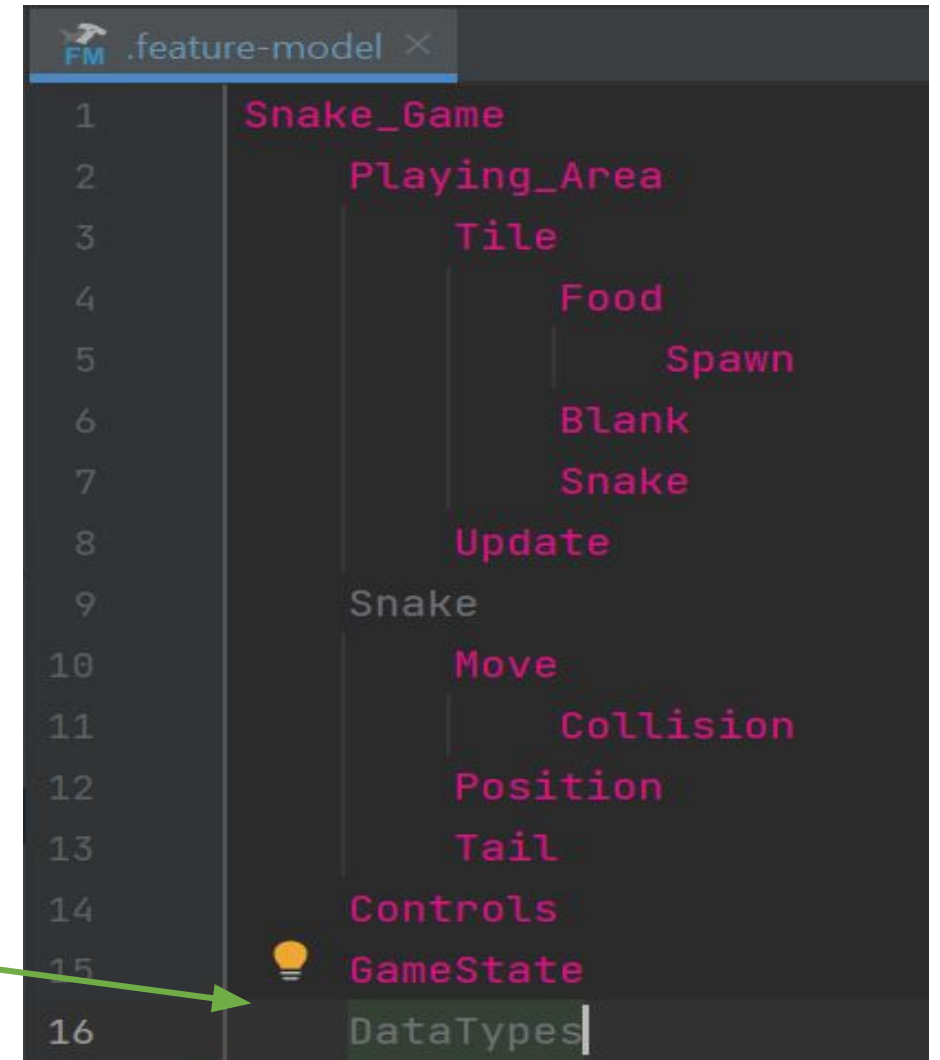
Adding **a feature: Method 1**

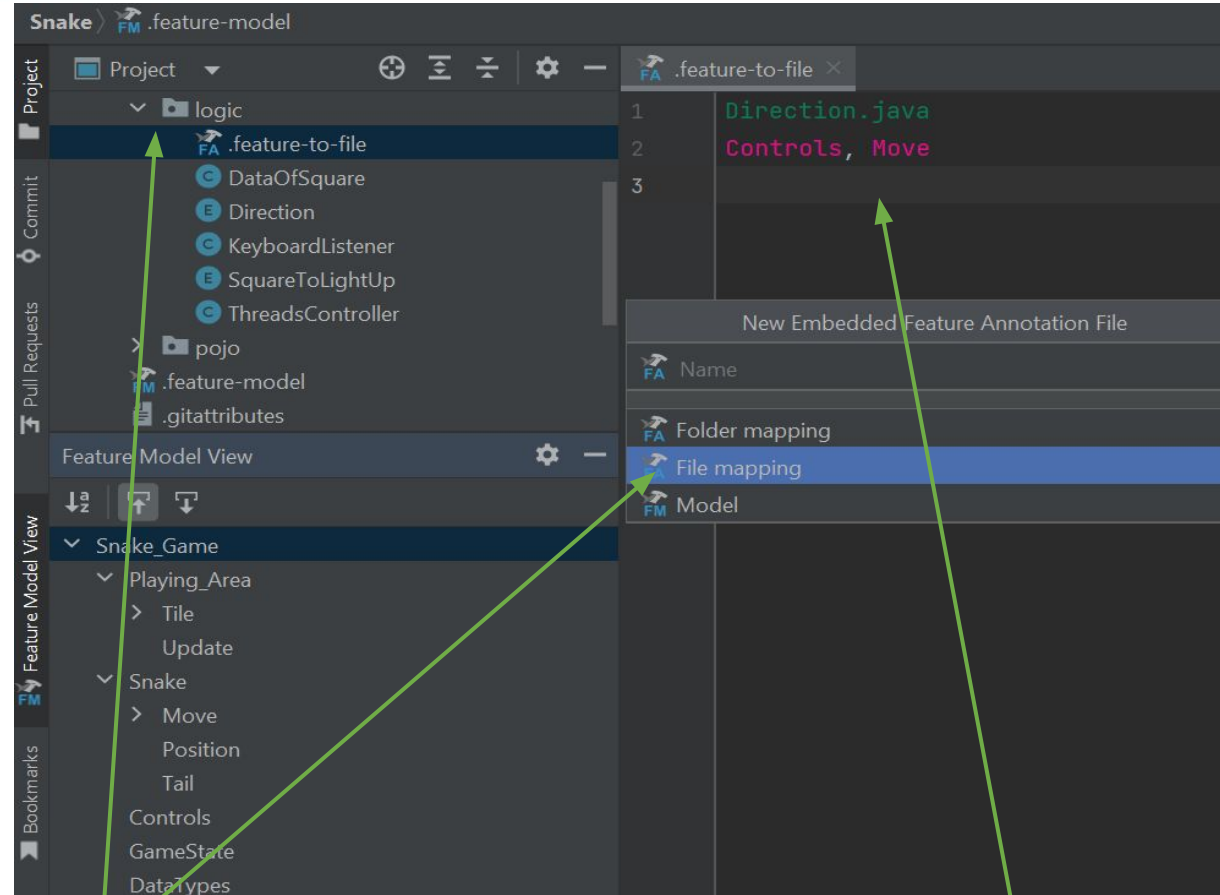By right-clicking New the "Feature Model View" and clicking "+ Add "

After Clicking "+ Add " you will get a new window then you can write the name of the Feature as below " DataTypes"

Adding **a feature: Method 2**

By clicking the ".feature-model" and writing the New feature under the ".feature-model" as below : ,
" DataTypes"

```
1    Snake_Game
2        Playing_Area
3            Tile
4                Food
5                    Spawn
6            Blank
7            Snake
8        Update
9    Snake
10        Move
11            Collision
12        Position
13        Tail
14    Controls
15    GameState
16    DataTypes
```

Adding a .feature-to-file mapping

By right-clicking New e.g. under the "logic" and selcting the file mapping
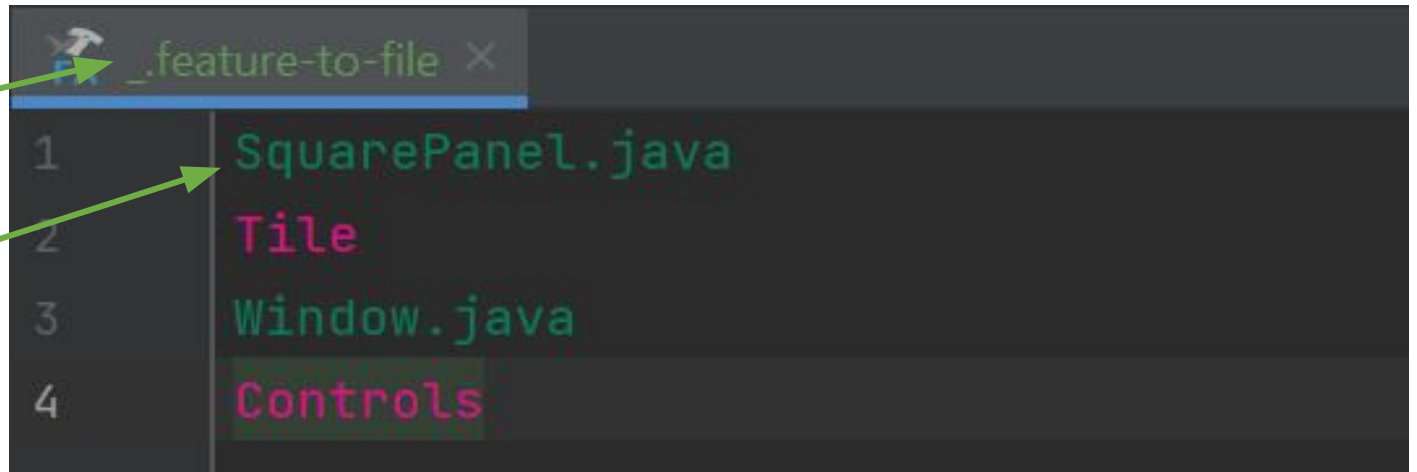
Mapping a .feature-to-file in this case the Direction.java

Adding a **.feature-to-file** mapping

Feature-to-file mapping is the mapping of entire files to one or many features. This allows specific features to be associated with the entire files. The mapping of features to files is stored in a dedicated file identified by the **.feature-to-file** extension.

Feature-to-file mappings imply that the entire file is a complete or partial implementation of a feature. File names and features they map to are written in alternate lines (e.g., SquarePanel.java contains the implementation of feature ``Tile'')
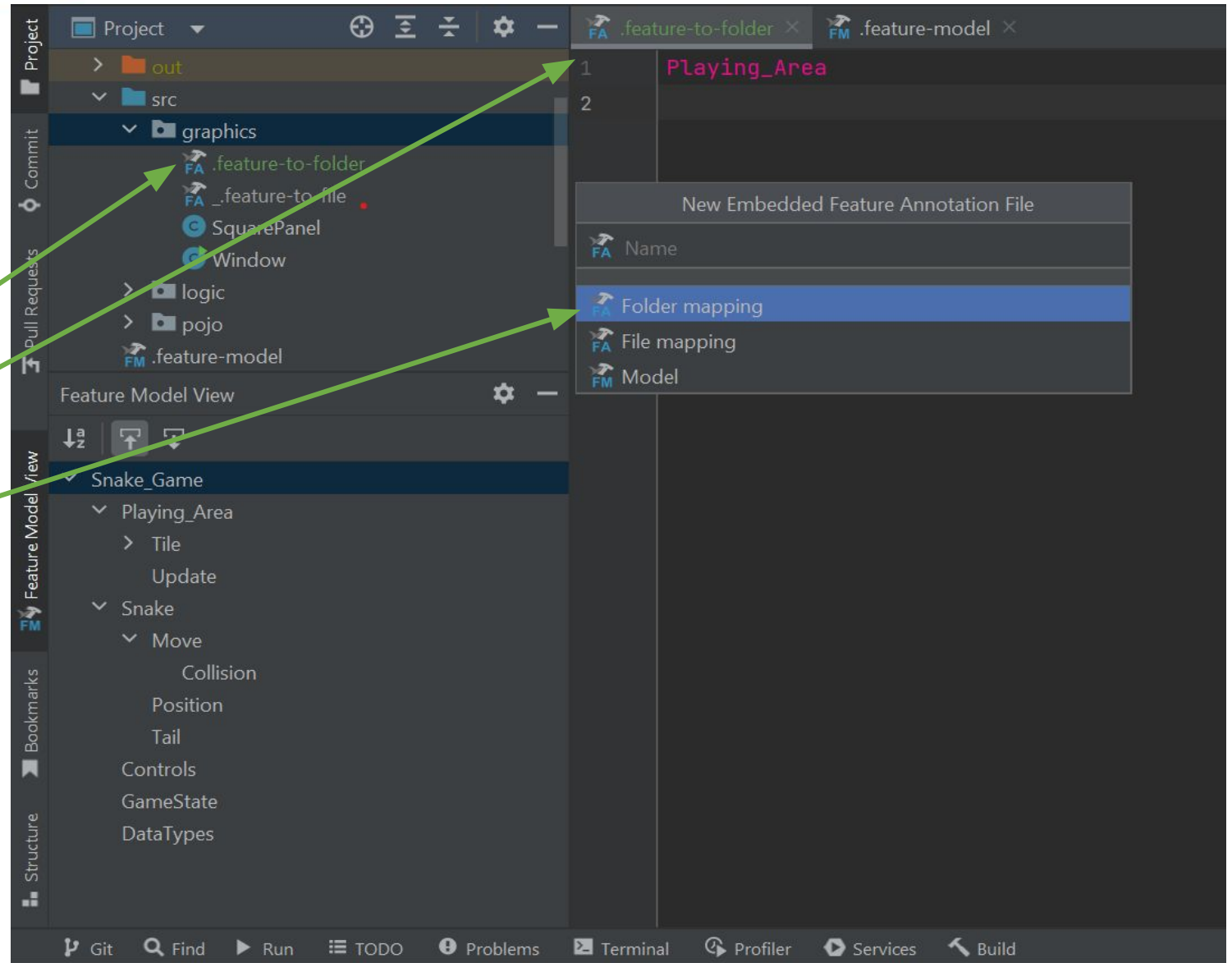
Should look like the following

```
.feature-to-file  ×
1    SquarePanel.java
2    Tile
3    Window.java
4    Controls
```
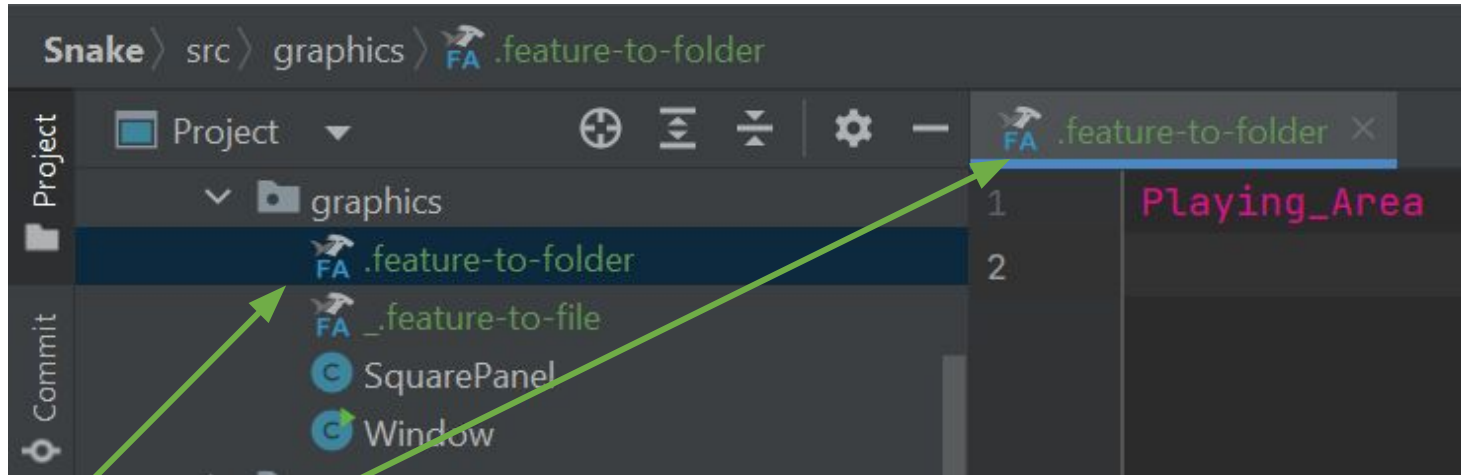
Adding a **.feature-to-folder** mapping

By right-clicking New under the "graphics" and selecting the Folder mapping

Adding a **.feature-to-folder** mapping

The mapping of features to folder is stored in a dedicated file identified by the **.feature-to-folder** extension.

Project
Commit

Project ▾   ⊕ ⊵ ≑ | ⚙ —   FA .feature-to-folder ✕

∨ 📁 graphics
    FA .feature-to-folder
    FA _.feature-to-file
    C SquarePanel
    C Window

1   Playing_Area
2

Should look like the following

Feature-to-folder mapping is the mapping of entire folders and their contents to one or more features. This allows specific features to be associated with the entire folder, including the containing files and sub-folders.

Adding **a feature to code mapping**

```
32
33      // &begin[Food]
34      foodPosition = new Tuple( x: Window.getWindowHeight() - 1,  y: Window.getWindowWidth() - 1);
35      spawnFood(foodPosition);//&line[Spawn]
36      // &end[Food]
37    }
38
```

A feature to code mapping is used to map specfic blocks of the code to one or more features. This part of code should be encapsulated with **// &begin** and **// &end** **annotations.** In our example as we see the feature **[Food]** is mapped to the code block by **begin-end** annotations.

Line annotations are used to map specfic lines of code to one or more features. The code lines to be mapped should be ended by **//&line[ ]** annotations. In our example, we see the feature **[Spawn]** is mapped to the Line 35 using the **//&line[ ]** annotation.

Feature Model View

•The **Feature Model View of HAnS Plugin** offers a special perspective on the software system.

•It displays a visual representation of the feature model, showing the relationships, hierarchies, and dependencies among different features.

The **Feature Model View of HAnS Plugin** looks like this in the **latest version**
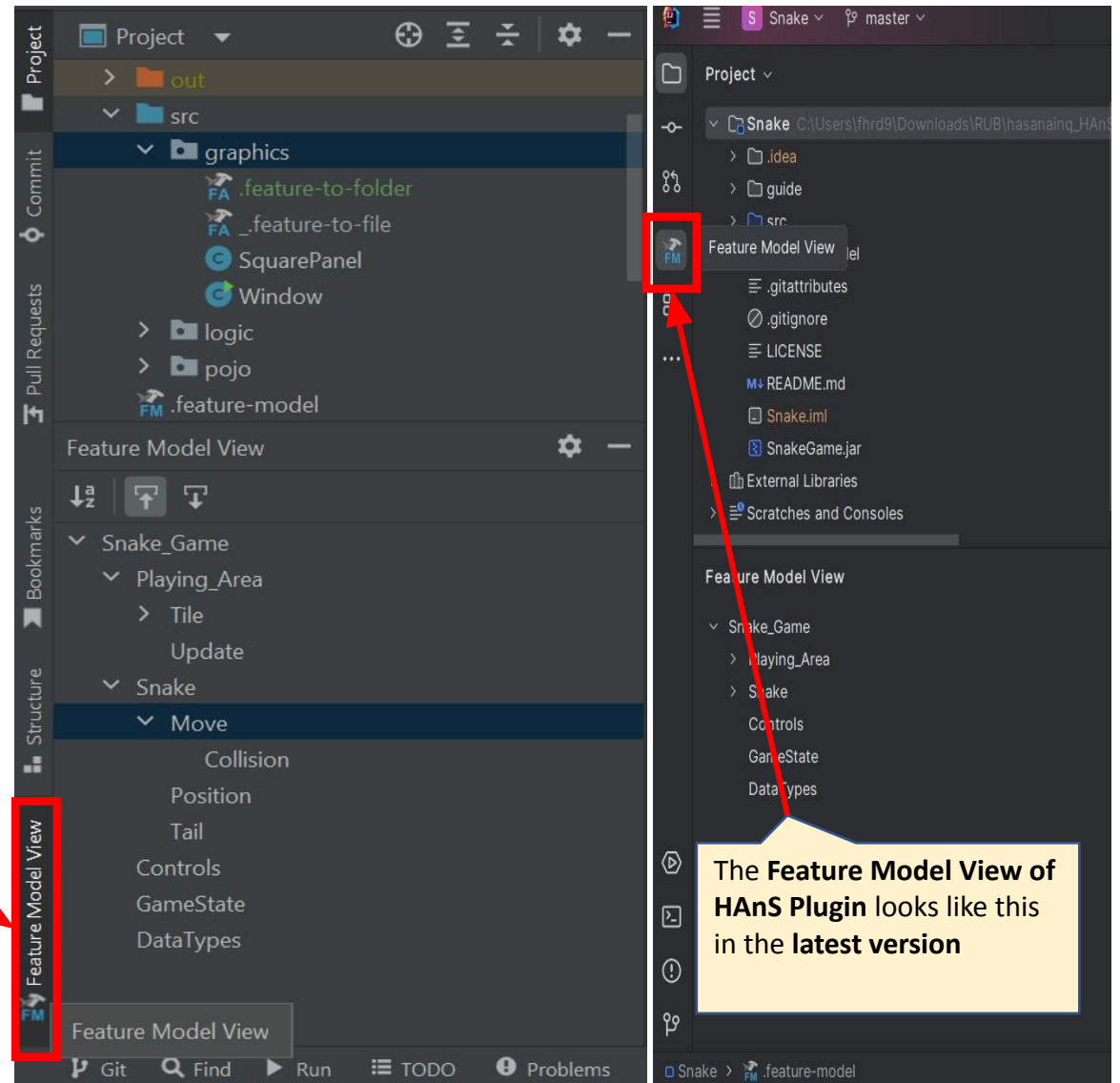
Navigating Feature Models

- In the **Feature Model View window** at the bottom left, as you can see, by right-clicking on any feature, you can view a list of functionality.

- **HAnS** provides powerful support for **adding**, **deleting**, and **renaming** through (**refactoring**) features.

- Developers can also use the "**Find Usages**" button to see exactly where these features have been annotated in the source code.