SoPra - **Project description**
- Single phase -

October 8, 2021

Minh Khue Pham **2579036**

Functionality

The implementation of new feature (crossroad) is divided into 3 phases:

- **Reading JSON-File, creating crossroads and consequences and adding them to the game**:
  We have 2 factories for crossroads and consequences (like abilities and cards) and 2 abstract classes are created corresponding to them. The crossroad is divided into 5 sub-classes based on their trigger condition. The consequence is divided also into 4 sub-classes based on their modification to the game (choice consequence is not counted as a consequence in this case). Crossroads are then added to `Model`, `ModelBuilderImpl` and `Player`. Consequence is a part of `Crossroad`.
- **Checking for trigger conditions**:
  Condition checking is added to every executing phase of each command. If the crossroad is triggered after the command, `crossroadActivate` in `Player` will be set to `true`. In `PlayerPhaseState`, while sending `ActNow` and handling commands, a condition is also be added there to constantly checking for interrupt from `crossroadActivate`.
- **Suming up the vote and executing consequences**
  If there is an interrupt, we will handle that in private method `processCrossroad` in `PlayerPhaseState`: If no vote needed, we create a `ConsequenceVisitor` and our consequence will accept that. Otherwise, we sum up the votes in `Model`, then call the voted consequence to accept visitor. The visitor will handle all modification and send events to player. Then we return the `ActNow` loop of current player.

Design decision

We decide to have 1 *Visitor* pattern here.

There are 4 separate consequences and each consequence modifies the game differently. We need to know exactly what consequence is executed, so that the consequence can be handled correctly. Using visitor here help us typecast these 4 consequences and avoid `instanceof`.

Challenge

The biggest challenge in this project is how to split crossroads and consequences, so that we can react correctly to each type. Enumeration (*BARRICADE, EQUIP, MOVED, SEARCHED, WASTECHANGED*) is treated as`CrossroadType` in this case. For example when we execute barricadeCommand, if current crossroad has type *BARRICADE* and all other requirements are fullfilled, crossroad is triggered. Visitor separates the consequences as above description.