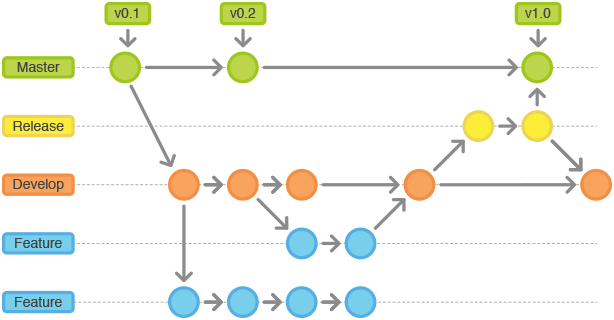
## Git style



New features and content get developed in seperate branches, branching out from the development branch. When a larger/external test is to be conducted, a build from the development branch gets merged to the release branch, once it has been tested and approved it can be merged into the master branch. When a feature or content is completed they are to be tested, primarily by QA, and moved into the development branch, after they have been tested, if major testing is needed, otherwise things can be merged into development if they are minor.

* feature branch names are to be prefixed with f\_
* content branch names are to be prefixed with c\_
* adjustment branches are prefixed with a\_
* bug-fixing branches are prefixed with b\_.

When merging, NO FAST\_FORWARD. (if possible)

Commits to the development branch are to be named with a development id at the beginning like: “D#5 something something”.  
Commits to the release branch are to be named so it references a development build in a similar manner: “R#2,D#5 something something”.

Commits to the master branch uses the same name as the commit to the release branch.

## Code style

Clean code. Give descriptive names to fields, methods, classes, etc. Rather a long and descriptive name, than a short and confusing name.  
Please don’t write enormously long methods, instead split it into several smaller methods and call them in sequence from the original method. In some cases long methods might make sense though, but as a rule of thumb, short methods.

If explicit states are needed for a GameObject (e.g. a door that interacts in different ways depending on it being open or closed), define them as an Enum- type at the beginning of the script-file, between the import statements, and the class definition. Name the enum: State\_[GameObjectName].

Write extension methods in the Extensions.cs file.

When working you can write all the public fields you want, but before merging to the development branch refactor. Private for the fields that are not to be referenced by other scripts (attach the [SerializeField] attribute if it is needed in the inspector), and make a property for fields that other scripts need access to (if the value is needed in the inspector, make an explicit private backing field).

## Unity Editor style

Version of Unity used: 2019.4.8f1

Asset folder contains the \_MY\_ASSETS folder, which is where we put most of our own stuff.

Folders we need to access are:

\_ASSETS

Scripts

\_Scenes

* Playgrounds

Prefabs  
 Materials

Resources

Audio

Sprites

Textures

Use of resource folder is normally something one should avoid, by there are a few usecases that apply to use “The ease of the Resources folder makes it an excellent system to rapidly prototype.” (unity, <https://learn.unity.com/tutorial/assets-resources-and-assetbundles#5c7f8528edbc2a002053b5a7>, section 3.2 ) It is bad for fine-garined memory control, but that is unlikely needed by us in this project.

More subfolders will be created as needed. In the Scenes/Playgrounds folder will be a scene for each member named: MemberName\_Playground. Each playground is meant to be the person’s working space, and they can do whatever they want inside it, experiment, and develop new features. New scenes for content and feature development can also be created if we feel the need.

Move featurers into development scene before testing with QA.