Technical Design Document
Witch Ball

Table of Content

1. File & Folder Structure

Godot Project:

The File Structure in the Game-project itself should be descriptive and Object orientaded. So every Folder should describe what it is containing this includes its own resources and other Folders (objects).

Files:

Every file in the Project has to be named in camel_case. So everything is written in lower case and multiple words are seperated with underscores (_).
The names should be descriptiv and show what they contain and/or what they're part of. If there are exist multiple Variants or Versions of it the file name has to containan extra Capital Letter and/or digit that indicates what Version or Variant this is.
Examples:

background_V4.png
fire_direct.gif
witch_head_B_V2.xcf

2. Coding Style

Coding convention:

The Coding Convention is following the Godot Style Guide:
http://docs.godotengine.org/en/3.0/getting_started/scripting/gdscript/gdscript_styleguide.html
And therefor the PEP8 Style Guide:
https://www.python.org/dev/peps/pep-0008/

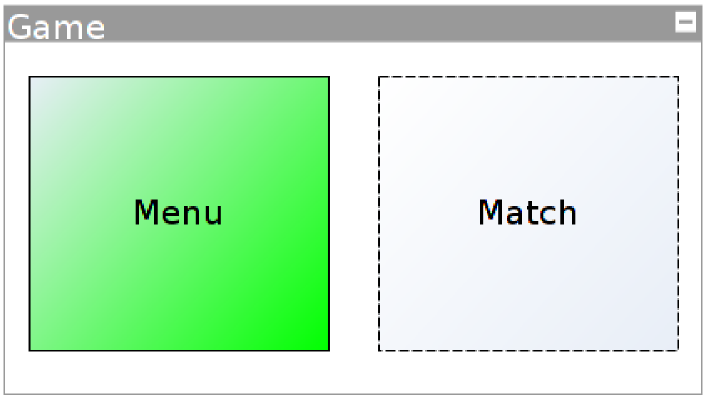
Coding structure:

Code should be written as Object-orientaded as possible.
Signals should be used for communication between scripts.
The whole structure should be designed to be frendly and appandable for new unplaned features.
Funktions in Godot Scripts should be ordered in following order:
Godot Funktions → Own Funktions → Funktions for signals

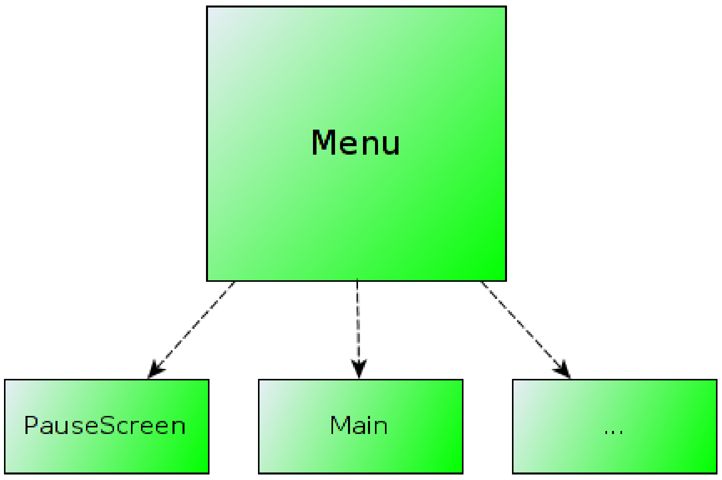
3. Project Structure

Rough Structure:

The following Picture describes the two main parts of the programm:

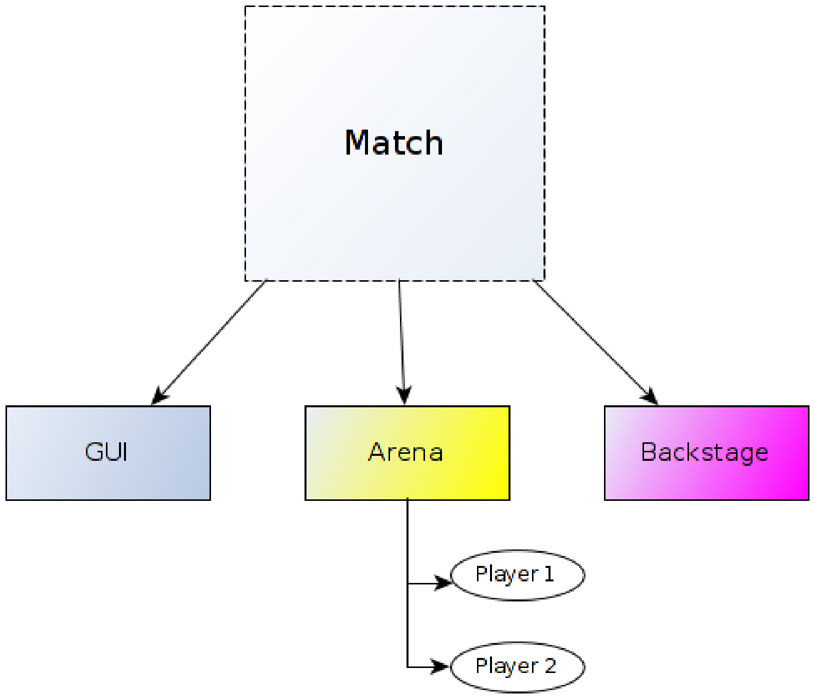


The Menu is a scripted Control Node which is alway existend in the Scene tree. It contains all scenes that have any menu context. These scenes get initialized when appiering and delete them selfs when requested.



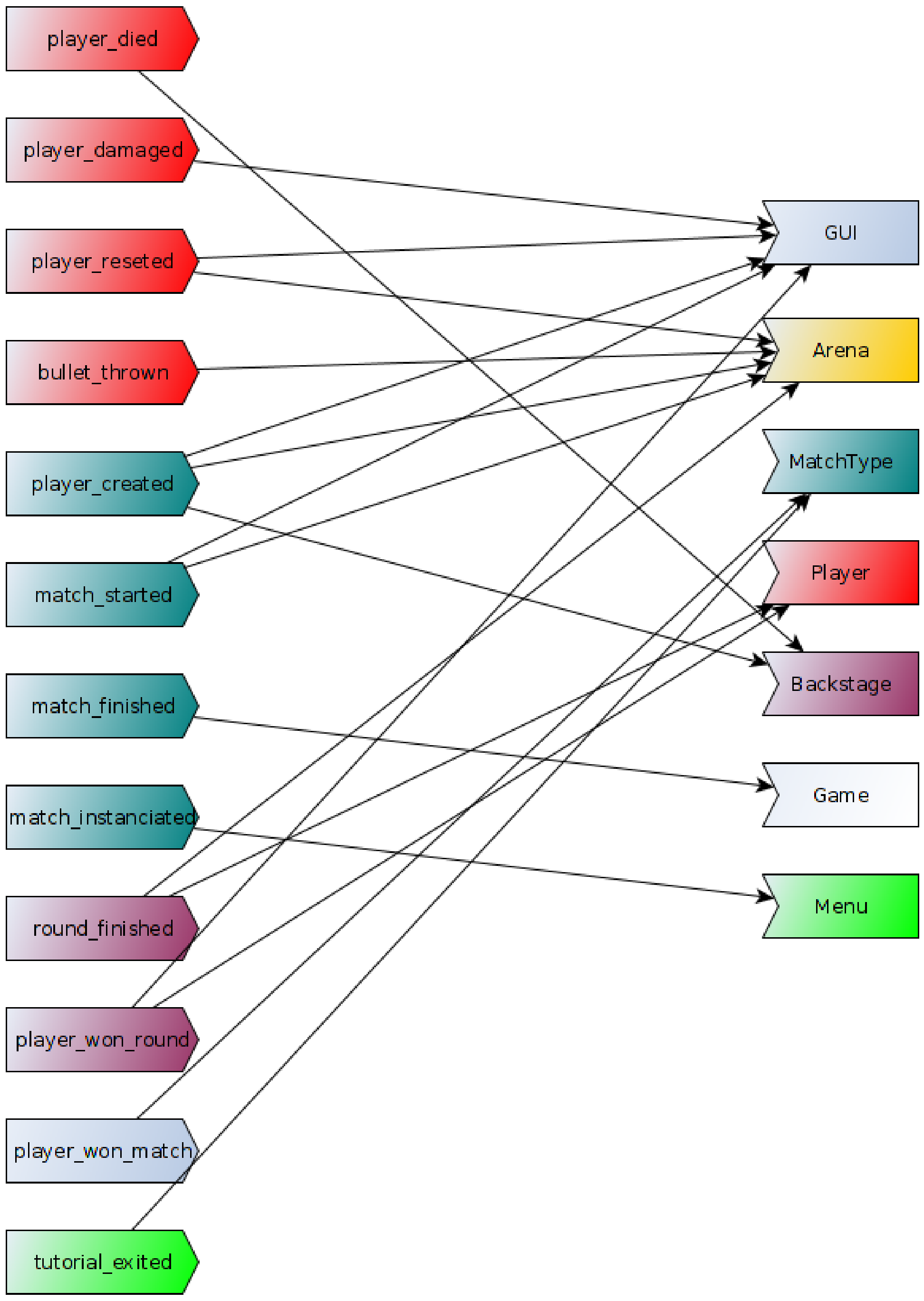
The Match Scene gets initialized when requested through the menu. It consists of three parts:

- Arena:
  The Arena Scene contains the Arena Sprite, anything that has to do with it (like colliders and Spawn positions) and anything that is placed in the Game „World“ (like both players and PickUps).
- GUI:
  Contains importand information for the players
- Backstage:
  The Backstage manages transitions and events while the Match is running


Match
GUI
Arena
Backstage
Player 1
Player 2

Signal Structure:

The Following Picture shows all in Code written signals and where they‘re going:



<u>4.Mercurial Guidlines</u>

Generel Workflow:

While working with the Repository changes should be commited and pushed as often as possible but in a logical range (making a commit for every new written letter would be to much). Also the local project should be syncronized and updated at least befor every commit.

Also it should be comunicated on what Feature wich Person is currently working on and when a commit is about to be pushed to the server.

Branches:

The default branch contains the latest working state of the game. If there are just minor changes or new assets they should be comitted and pushed directly on the default-branch. When working on a new Feature, a new branch for this Feature should be created which should be named after following template:

Feature_[Definition]

Commits and Commit-Messages:

As mentioned in **Gernerl Workflow** commits should be done as often as possible.
The commit is a short description of what is changed in this commit.
A Commit message consists of one sentence long summery of what was done in this commit and multiple additional enumerations starting with a „-“ followed by a keyword like these: ***added, fixed, changed, removed, ...***
Blocks of text should not be used.

Merge:

After a Feature or a worstep is complete the related branch should merged back into the default-branch. This is done by following workflow:
After the last changes are commited and pushed (!) the local client gets updated to the last default-branch commit. After that the last commit on the Feature-branch is merged witch local state. This is done by selecting the last Feature-branch commit in the project-view, right clicking on it and choose „merge with local“. Then the branges get merged and a merged commit is created which also has to get pushed (!).

If a new default commit gets pushed to the public while working on a Feature branch, the default commit should merged to the own Feature branch to avoid merge errors and problems on the default branch. This done the same way like mergeing a Feature branch into default, just with default default and Feature branch switched.