JUICE Game Architecture

This document describes the overall architecture and flow of games and game authoring within the JUICE web application.

**Table of Contents**

[File Structure](#_c2gjadcj4ata)

[Games](#_yvt4q5j5vypw)

[game.js](#_2aprj8pusrwf)

[view.js](#_75d7ajlnc5yb)

[schema.json](#_ktp4ap2lxy0y)

[physics.js](#_dx7inu2l1thz)

[Game Widgets](#_bbtgnl1n7tbk)

[[game\_name]\_widget.html](#_urp6nyd9unsg)

[[game\_name]\_widget.js](#_mzrempz9qm40)

[[game\_name]\_widget\_authoring.html](#_ldlbvhdkns64)

[[game\_name]\_widget\_authoring.js](#_iee99jy2m5d)

[.css + .txt files](#_yixrl8whh7b)

[Miscellaneous Files](#_24xmon2wd0lv)

[Game Authoring](#_e2rrz7fwm5dj)

[Architecture](#_e4u38ljxl6mr)

[Gameplay](#_4w7isw22g8zh)

[Game Modal Flow](#_cteeixe7pot7)

[Game Feedback Flow](#_8op00nbr22p5)

[Game Feedback Player Options](#_yf63ye4ou8hh)

# File Structure

The logic, view and physics layers for every game are located in the ***public/javascripts/game*** folder. The ***game*** folder contains some base shared files, shared folders (***images***, ***sounds*** and ***templates***) and a folder for every game.

## Games

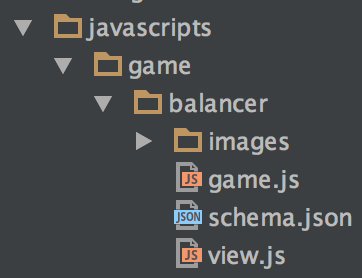
Each game folder consists of three to four files:

***game.js***

***view.js***

***schema.json***

***physics.js (puzzler and sorter only)***



*Example structure for the game “balancer”*

### game.js

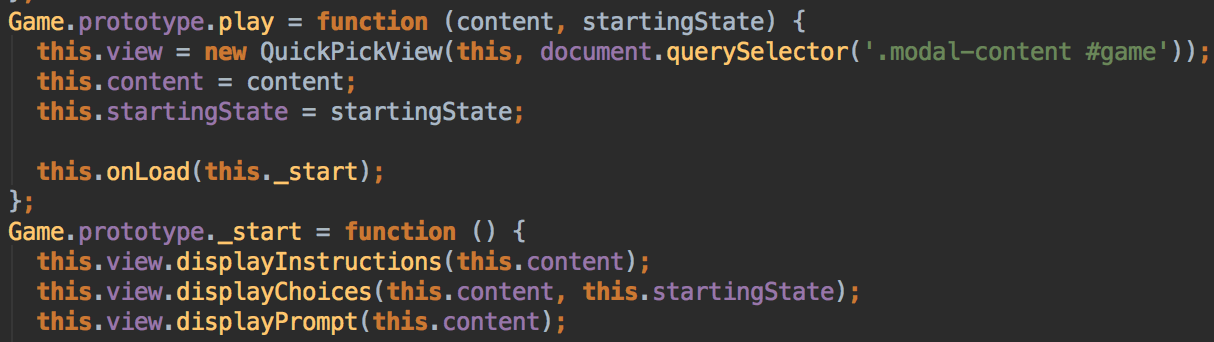
The ***game.js*** file is the main interface for each game, and is expected to register itself with the ***GameResolver***, validate itself with the ***GameMixin*** and control the general flow of gameplay. It is also handed the content of the current round being played.

It is responsible for instantiating and directing the view/physics layer, and for generating feedback based on user input. The ideal role of the ***game.js*** file is to act as the “controller” of the game. Some games have responsibilities mixed into other files as well, but centralizing in ***game.js*** is the goal.

The interface for the object specified here is defined and validated in the ***GameMixin*** object.

### view.js

The ***view.js*** file handles all display and events in your gameplay UI. There isn’t a required interface, but the general pattern is that there are a set of “display[Area]” methods which are called up front and output all of the main UI elements. For instance, this is a fairly typical example of the ***game.js*** file setting up the view and calling its initial “display” methods



*game.js instantiating and calling the View object for “quick\_pick”*

After the game and assets have loaded, we run through the view display methods “displayInstructions”, “displayChoices” and “displayPrompt”.

Aside from these types of methods, the view API should give access to whatever areas of the game the ***game.js*** file needs to handle user interaction and feedback.

### schema.json

This file describes the content of your game in JSON Schema format. This is the format that your game authoring tool produces, and the format of the data handed to your ***game.js*** file.

### physics.js

Two of the games - puzzler and sorter - use a full physics engine (box2d) to simulate their interactions. In the case of a game where this is in play, that logic should live in the ***physics.js*** file.

The reason it is better to separate since into its own file is that typically a physics engine is running a simulation alongside your game. On each frame it is providing information about how the different physics bodies are reacting to their environment (their position, velocity, etc). This runs alongside your gameplay frames, and you utilize that information to place your gameplay UI elements.

## Game Widgets

Each game widget folder consists of eight files:

***[game\_name]\_widget.html***

***[game\_name]\_widget.css***

***[game\_name]\_widget.js***

***[game\_name]\_widget.txt***

***[game\_name]\_widget\_authoring.html***

***[game\_name]\_widget\_authoring.css***

***[game\_name]\_widget\_authoring.js***

***[game\_name]\_widget\_authoring.txt***

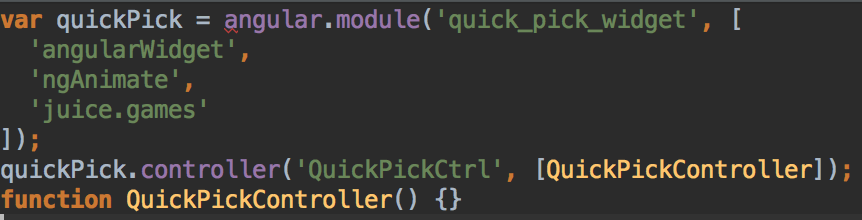
### [game\_name]\_widget.html

All games have a widget html file, which is used to load the game files described earlier (game.js, view.js, etc). They all use the ***play-game*** directive, and look nearly identical.



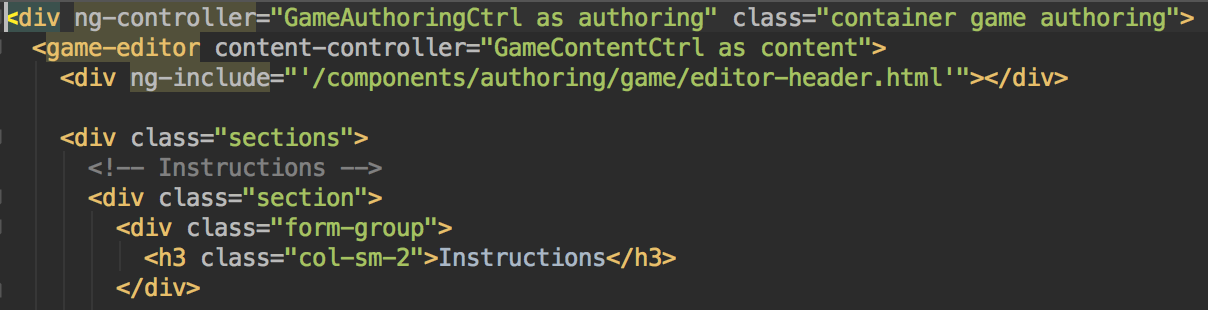
### [game\_name]\_widget.js

All games have a widget js file, which provides a simple controller for your widget.html, if one is needed. They all define their widget module, and look nearly identical.



### [game\_name]\_widget\_authoring.html

These files define the HTML layout and angular controllers for authoring specific to that game. There are various helpers and include files used to streamline this process as much as possible.



Here, the ***game-editor*** directive sets up the shell of the authoring interface.

***content-controller*** is a property of the directive that specifies which controller to use to manage the content.

***editor-header.html*** contains common UI pieces (like number of rounds, controls for copying/adding rounds, etc) used in the editor.

Each section of the editor is specified using the ***.sections*** and ***.section*** classes.

### [game\_name]\_widget\_authoring.js

These files define the javascript controller, which is specified as the ***content-controller*** of the ***game-editor*** directive in ***[game\_name]\_widget\_authoring.html***.



Here we see the content controller definition. There are a variety of services used to help (notifications, file handling, accessing game data), as well as a mixin for some common functionality (***GameContentAuthoringMixin***).

### .css + .txt files

The .css and .txt files could be used to extend your widget with specific styling or content, but I haven’t used them at all within the game widgets.

## Miscellaneous Files

***javascripts/game/directives.js*** contains all the gameplay directives, as well as some miscellaneous helper directives.

***javascripts/game/services.js*** contains a variety of services used throughout the games including the ***GameResolver*** and ***GameMixin*** as well as DOM helpers and animations.

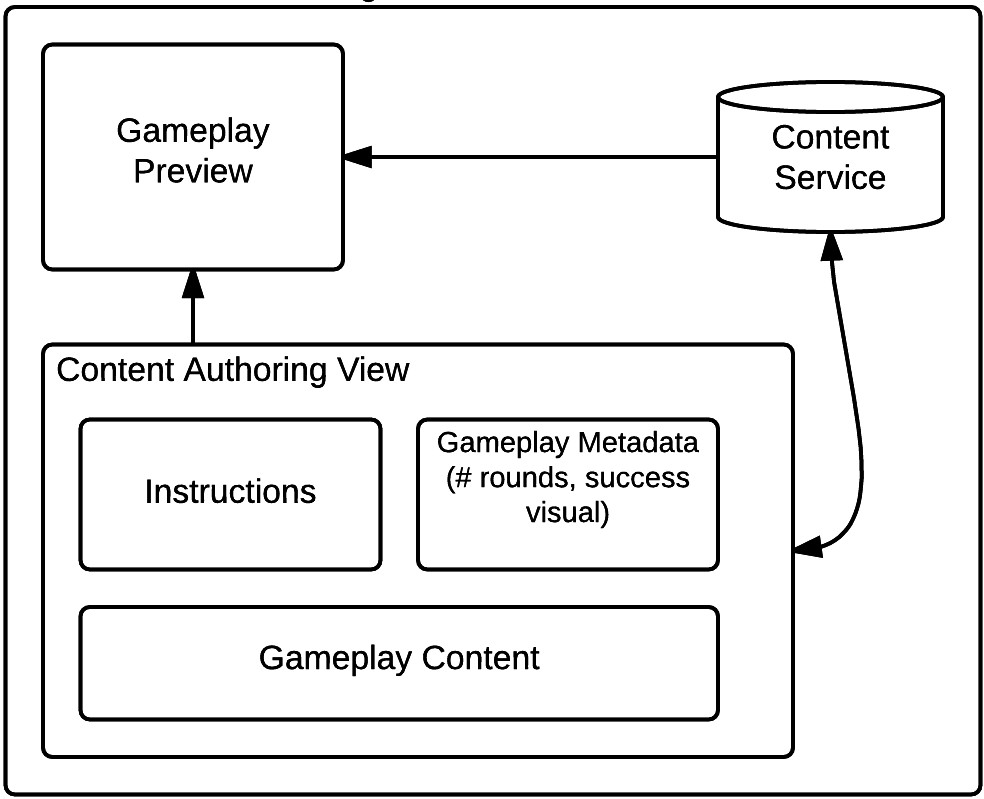
***javascripts/game/framework.js***defines all of the game modules and their dependencies

***javascripts/game/controllers.js*** contains the main gameplay modal controllers.

***components/authoring/game/authoring.js***containsauthoring controllers, directives, services and helpers

***components/services/services.js*** general services that were larger in scope than just games. Includes text parsing, id generation, file handling, sounds and a grab bag of utilities.

# Game Authoring

****

Game authoring works the same way for every game. The user visits the ***/juice/author/[widget\_name]/[widget\_file]*** path, which loads the appropriate widget. The files associated with authoring are defined in the [game widgets](#_bbtgnl1n7tbk) section.

Game authoring has two sections: ***Instructions*** and ***Content***.

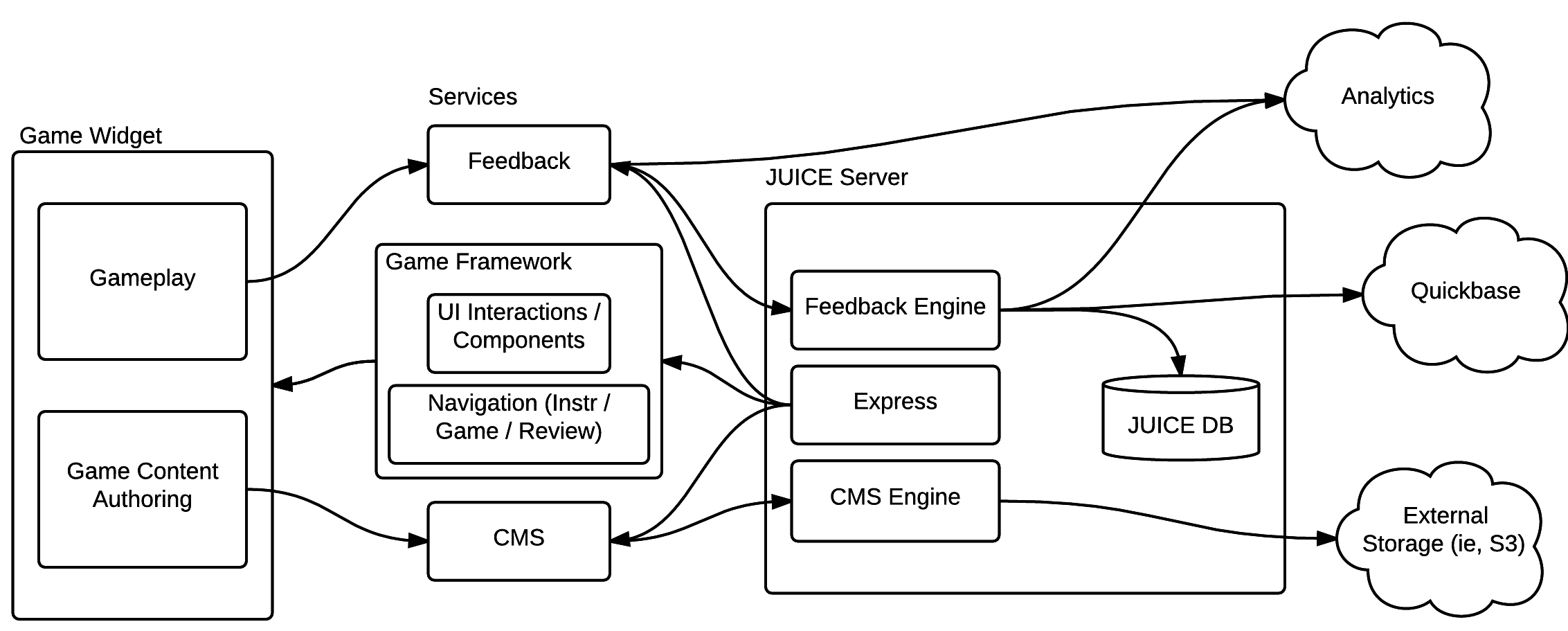
***Instructions*** Instructions, a tip and an image

***Content*** Every game has a consistent interface for number of rounds, which round is being edited, and options for Adding/Deleting/Copying/Previewing rounds. Actual round content can vary but typically includes text, dropdowns, and some images.

All of them have a feedback area, which can also vary between games in terms of how they work (simple feedback, feedback tied to specific answers, feedback tied to specific set of answers).

Success animations are the same for all games and are randomly determined from the set "fireworks" ,"balloons" and "confetti"

# Architecture



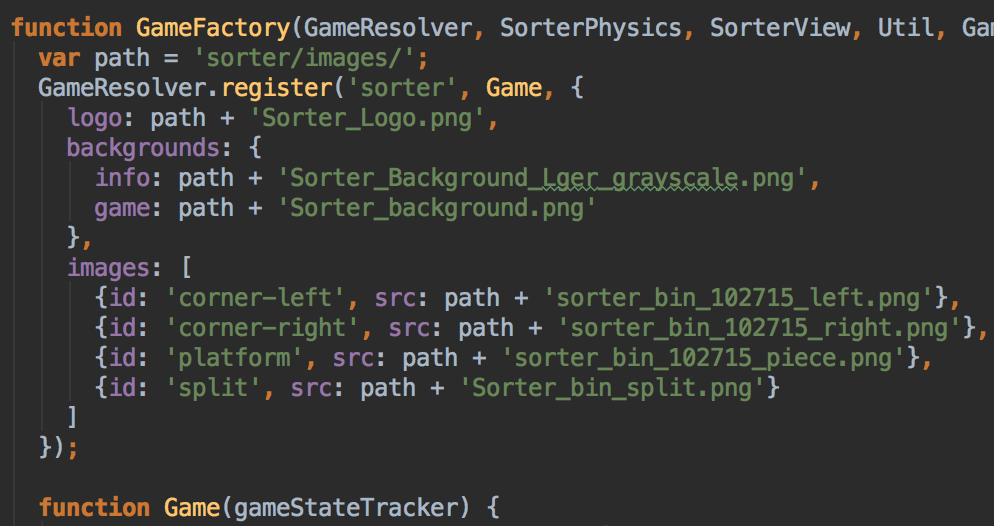
The game architecture consists of widgets, services and controllers, interacting with the overall JUICE application.

A ***widget*** consists of the HTML/CSS/JS used to load and invoke [game components](#_urp6nyd9unsg).

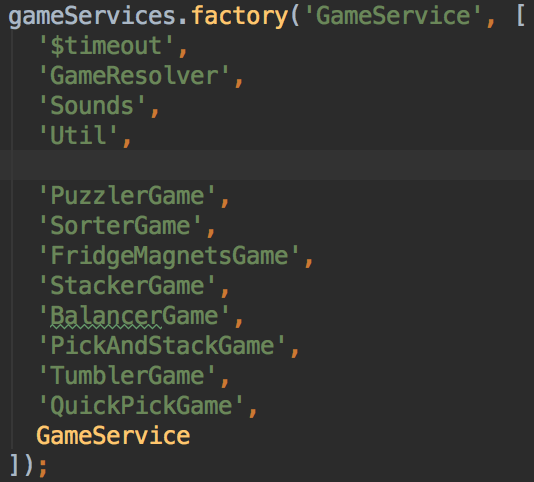
An ***authoring widget*** consists of the HTML/CSS/JS used to load and invoke [game authoring components](#_ldlbvhdkns64).

Services and controller are used to load and display gameplay content. For a [game](#_yvt4q5j5vypw) to load, the ***GameResolver*** and ***GameService*** are required.

Each game must register with the ***GameResolver***so that it can be dynamically loaded by name later on. ***GameResolver*** is a factory which knows how to instantiate games, how to look them up by an id, and how to load game assets.



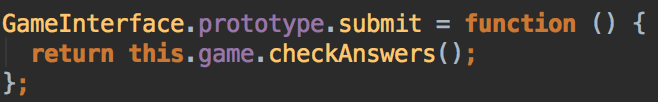
*game.js file for “sorter”, registering itself and its assets with the GameResolver*

The ***GameService*** handles game flow. It uses the ***GameResolver*** to instantiate the game, and then utilizes the game API to control gameplay.

*Each game is defined as a dependency of the GameService, which forces the games to be resolved.*



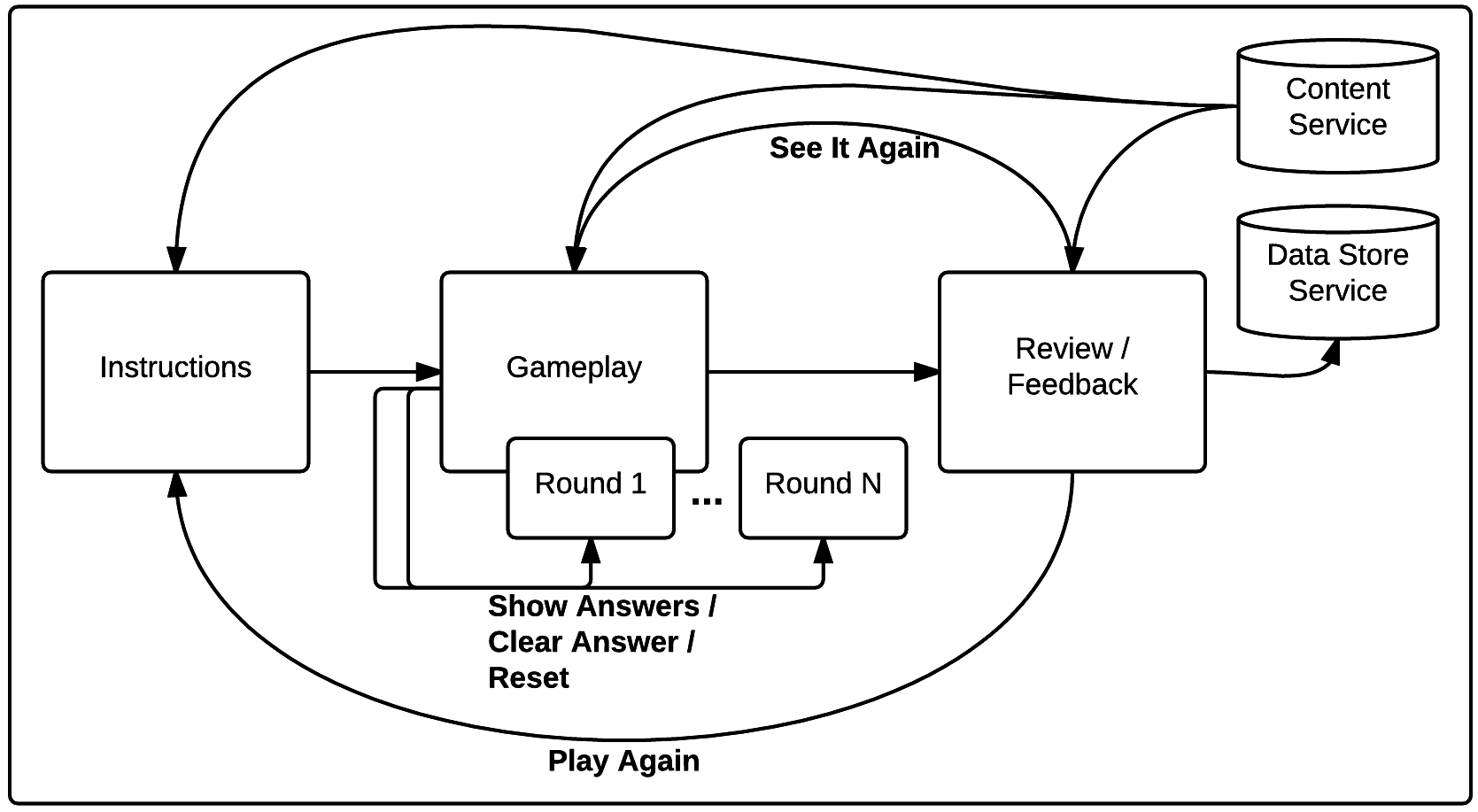
*GameService sets up the resolver and instantiates the game.*

****

*The GameService delegates methods to the actual Game interface*

# Gameplay

## Game Modal Flow

****

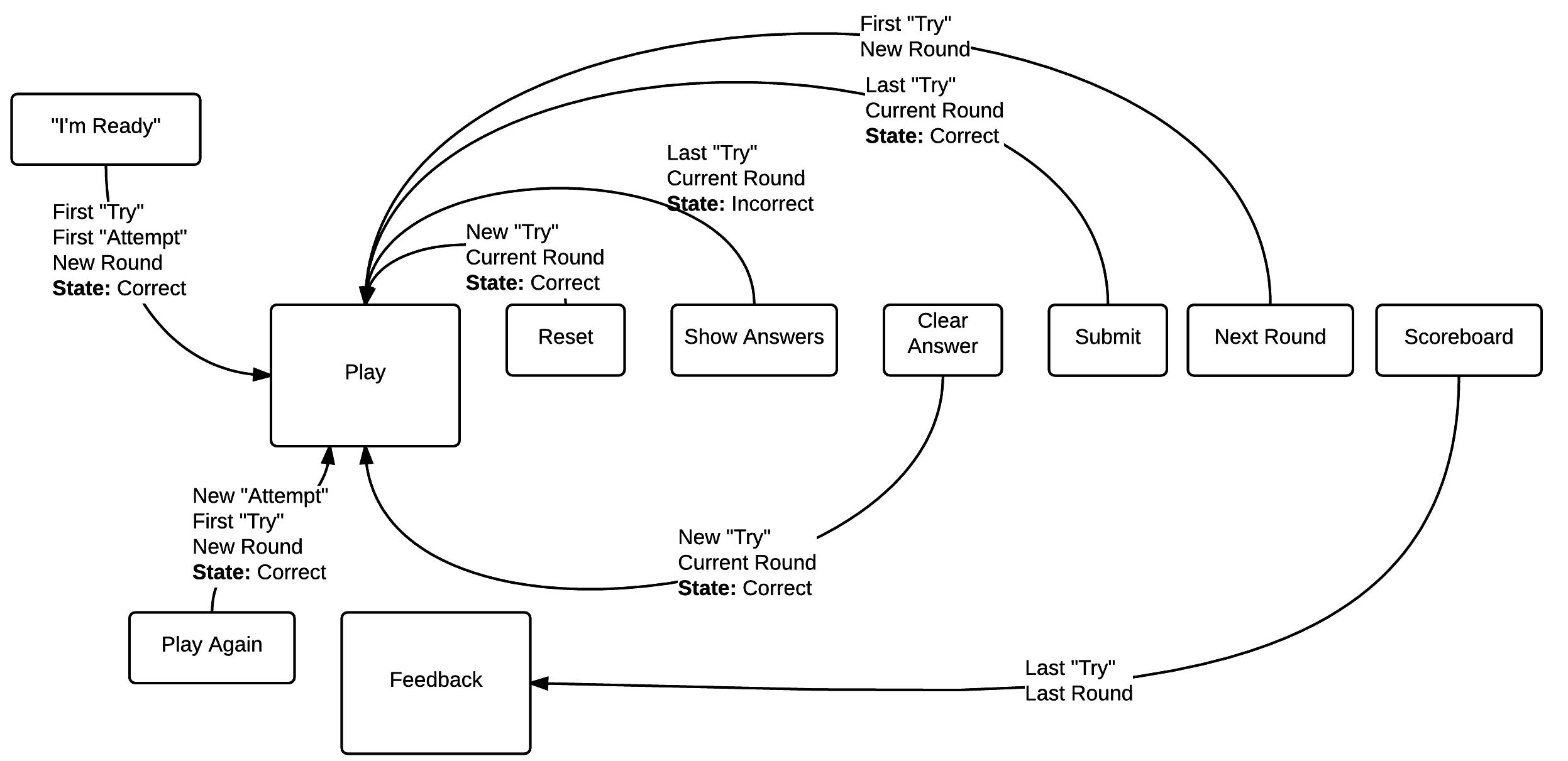
The Game Modal is broken up into three main sections: ***Instructions***, ***Gameplay*** and ***Scoreboard***.

***Instructions*** are the players first interaction with the game and include a description of what to do, a possible “Tip”, and an image showing the gameplay (often an animated gif).

***Gameplay*** is the users actual interaction with the game content and mechanics. Each game is functionality different, but operate in one of three modes: ***Immediate***, ***Check*** or ***Mixed***. ***Immediate*** mode means they are given feedback as soon as they interact with a game element. ***Check*** mode means they are given feedback after clicking the “Check It” button. ***Mixed*** mode means the game consists of an ***Immediate*** phase and a ***Check*** phase (as of this writing, only Pick + Stack follows this format).

***Scoreboard*** is where the user is shown what rounds were correct or incorrect, and are given sound (applause) and visual (fireworks, confetti, or ballons) feedback based on how well they did. From here they can click “Play Again” to start over with a new set of rounds, or “See It Again” on a specific round to see what their answers were.

## Game Feedback Flow

****

As a player progresses through a game, information about their progress is being stored in three layers: ***Attempts***, ***Rounds*** and ***Tries***.

An ***Attempt*** is a play session consisting of a set of ***Rounds***.

A ***Round*** is the player’s interaction with a game’s content, and can have multiple ***Tries***

A ***Try*** is the outcome of a specific player interaction. There can be multiples ***Tries*** per ***Round***, as the player can do things like reset their answers, or clear out what they’ve answer incorrectly. Each time they do this, they are ***Try***ing again.

Each layer of information is individually tracked and timestamped with “startedAt” and “endedAt” timestamps. The number of correct and incorrect ***Tries***the player receives while playing the game determines the intensity of feedback when they reach the ***Scoreboard***.

The ***Data Store API*** takes this gameplay data and sends it to the JUICE server.

## Game Feedback Player Options

Each game has several options for the player to choose. They are: ***Check It***, ***Show Answers***, ***Clear Answer***, ***Reset***, ***Play Again***, and ***Next Round***.

***Check It*** is for ***Check*** or ***Mixed*** mode games, and checks to see what answers the player has gotten right or wrong. If the player has correctly answered the questions, this finishes the current ***Try*** as correct.

***Show Answers*** can be used in ***Immediate*** and ***Check*** mode games, and it removes all incorrect answers and displays all correct answers. This finishes the current ***Try*** as incorrect.

***Clear Answer*** clears out any wrong answers, and resets some (or all) of the answer visual cues in the game. This triggers a new ***Try***.

***Reset*** clears out all answers (right or wrong) in the current round, “resetting” the round back to its original state. This triggers a new ***Try***.

***Next Round*** moves the game forward to the next round. This triggers a new ***Round***.

***Play Again*** starts the user over in the game with a new set of rounds. This triggers a new ***Attempt***.