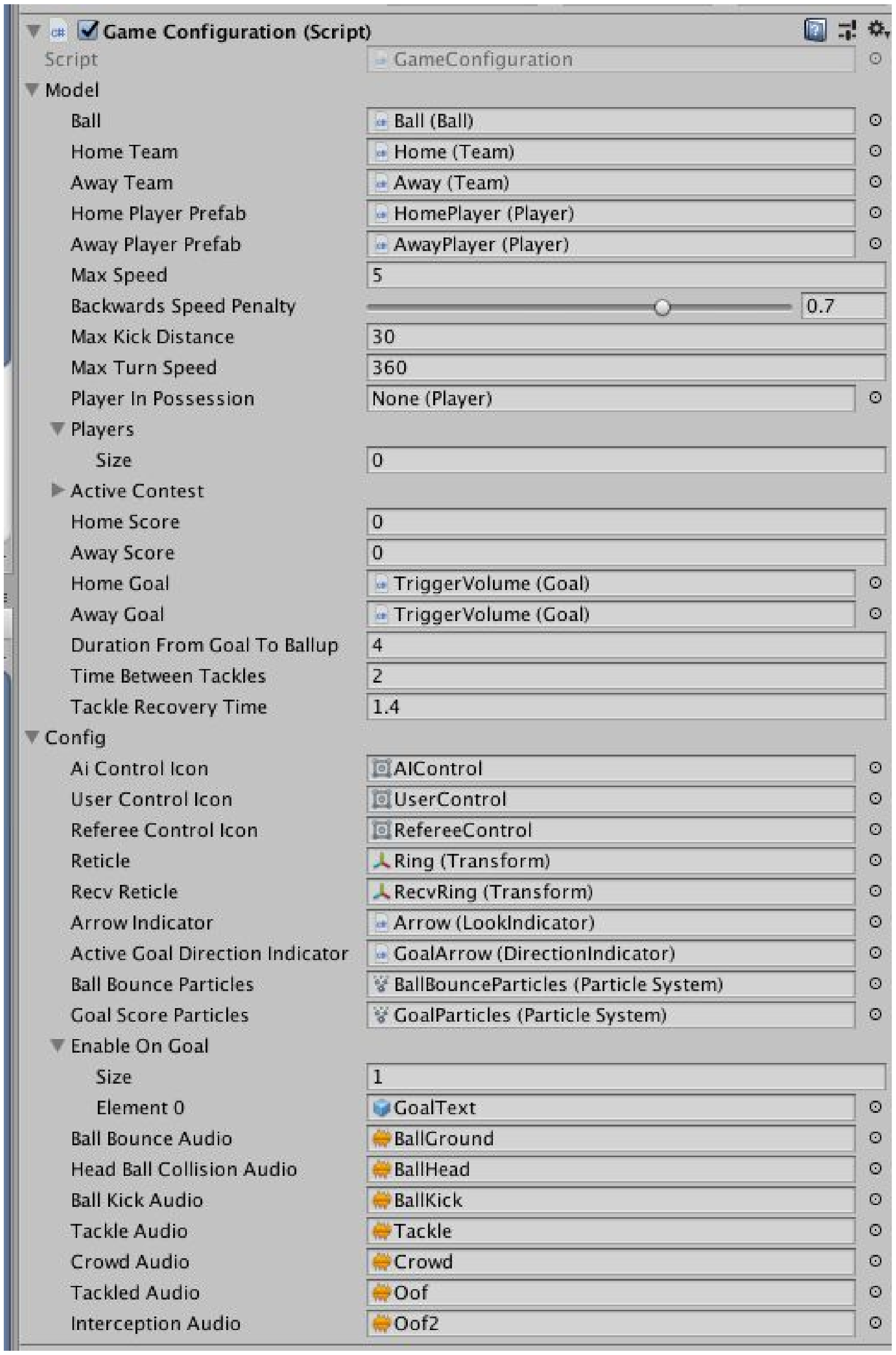
# TeamBallGame Template

Reference Documentation

This template is designed to allow creation of sports simulations and games which use a team of players, a ball, a field, and goals.

# GameObjects in SampleScene

## GameConfiguration



**Game Configuration (Script)**

| | |
|---|---|
| Script | GameConfiguration |
| Model | |
| Ball | Ball (Ball) |
| Home Team | Home (Team) |
| Away Team | Away (Team) |
| Home Player Prefab | HomePlayer (Player) |
| Away Player Prefab | AwayPlayer (Player) |
| Max Speed | 5 |
| Backwards Speed Penalty | 0.7 |
| Max Kick Distance | 30 |
| Max Turn Speed | 360 |
| Player In Possession | None (Player) |
| Players | |
| Size | 0 |
| Active Contest | |
| Home Score | 0 |
| Away Score | 0 |
| Home Goal | TriggerVolume (Goal) |
| Away Goal | TriggerVolume (Goal) |
| Duration From Goal To Ballup | 4 |
| Time Between Tackles | 2 |
| Tackle Recovery Time | 1.4 |
| Config | |
| Ai Control Icon | AIControl |
| User Control Icon | UserControl |
| Referee Control Icon | RefereeControl |
| Reticle | Ring (Transform) |
| Recv Reticle | RecvRing (Transform) |
| Arrow Indicator | Arrow (LookIndicator) |
| Active Goal Direction Indicator | GoalArrow (DirectionIndicator) |
| Ball Bounce Particles | BallBounceParticles (Particle System) |
| Goal Score Particles | GoalParticles (Particle System) |
| Enable On Goal | |
| Size | 1 |
| Element 0 | GoalText |
| Ball Bounce Audio | BallGround |
| Head Ball Collision Audio | BallHead |
| Ball Kick Audio | BallKick |
| Tackle Audio | Tackle |
| Crowd Audio | Crowd |
| Tackled Audio | Oof |
| Interception Audio | Oof2 |

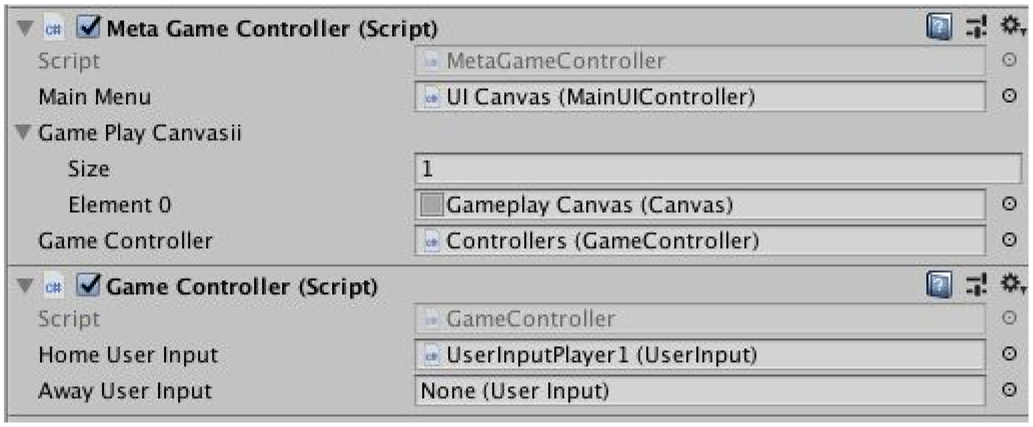
## Model

This field contains values and references used in the simulation of the game,
and values which modify the simulation and gameplay. You are able to access
this model anywhere in your own code via:
var model = Simulation.GetModel();

## Config

This field contains values and references used in non-gameplay parts of the game, such as audio clips and particle system prefabs. You are able to access this instance anywhere in your own code via:
var config = Simulation.GetModel();

## Controllers



Meta Game Controller (Script)

| | |
|---|---|
| Script | MetaGameController |
| Main Menu | UI Canvas (MainUIController) |
| Game Play Canvasii | |
| Size | 1 |
| Element 0 | Gameplay Canvas (Canvas) |
| Game Controller | Controllers (GameController) |

Game Controller (Script)

| | |
|---|---|
| Script | GameController |
| Home User Input | UserInputPlayer1 (UserInput) |
| Away User Input | None (User Input) |

## MetaGameController

This component controls the context switch between the menu and gameplay.

## GameController

This component runs the game simulation, initialising the field, reading input
and ticking the main simulation loop.

## UICanvas

This is the main menu canvas, built using the Unity UGUI system. It contains a number of switchable tabs which are placeholders for your own GUI screens.

## GameplayCanvas

This is the game canvas, it is displayed when the game simulation is running. It displays game score and a goal notification. You would add any other gameplay UI into this canvas.

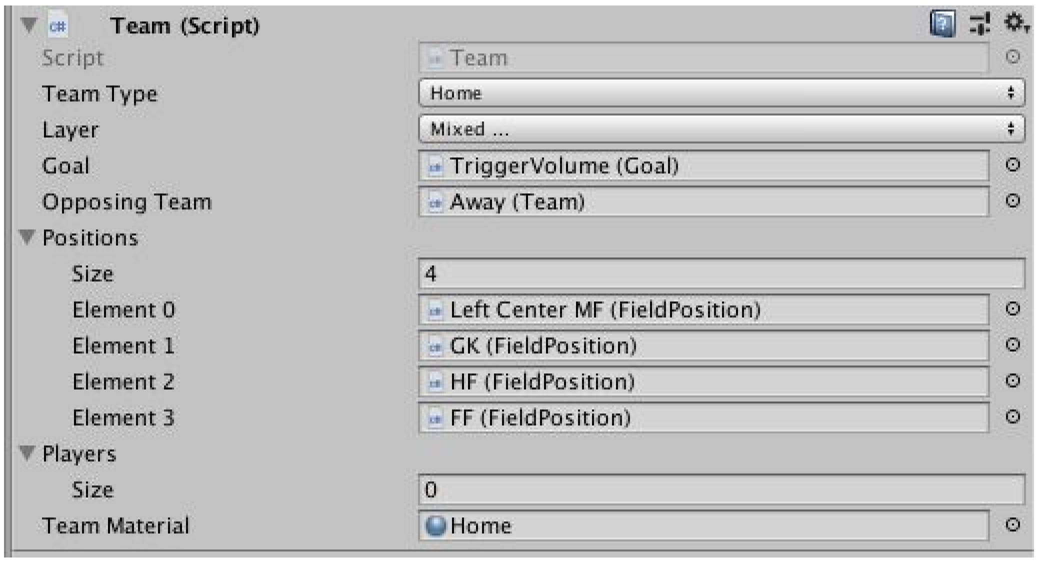
## Cameras

This is the container for the Cinemachine camera setup, including the dolly track used to move the camera along the length of the field.

## Teams: Home and Away

The team component on these game objects is used to configure the home team and away team. Each team must be assigned a goal game object from the scene, and a number of field positions. Field positions are points on the field where a player for the team will be instantiated when the game starts.

The players array is left empty, and is automatically populated with the players for that team when the game starts. The team components are available in the BallGameModel.

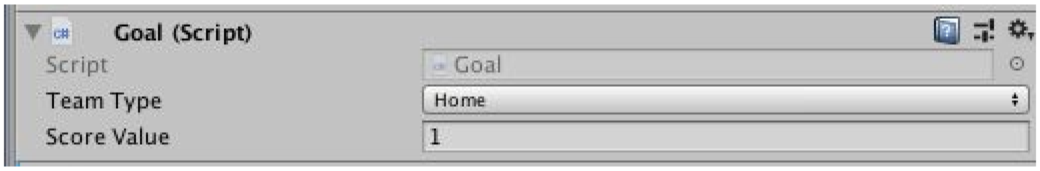


**Team (Script)**

| | |
|---|---|
| Script | Team |
| Team Type | Home |
| Layer | Mixed ... |
| Goal | TriggerVolume (Goal) |
| Opposing Team | Away (Team) |
| Positions | |
| Size | 4 |
| Element 0 | Left Center MF (FieldPosition) |
| Element 1 | GK (FieldPosition) |
| Element 2 | HF (FieldPosition) |
| Element 3 | FF (FieldPosition) |
| Players | |
| Size | 0 |
| Team Material | Home |

## Home Goal and Away Goal

These game objects contains the visual setup of the goal and physics colliders which surround the center trigger. This prevents entry into the trigger from all sides except the front of the goal. The trigger has a Goal component, which specified the team which owns the goal, and the point value of this goal.



| Goal (Script) | |
|---|---|
| Script | Goal |
| Team Type | Home |
| Score Value | 1 |

# Architecture

The code is split into four functional systems.

**Mechanics:**
The fundamental mechanics of the game are provided by the Mechanics namespace. This area holds all code required to create events which are used to derive gameplay. It can update the model and post events, but rarely subscribes to events. Game mechanics are generally implemented using regular Unity idioms.

**Gameplay:**
The Gameplay folder/namespace contains events that are triggered by the Mechanics. This is the area of code a game designer would modify to implement the rules of the game, creating gameplay. It can post events and modify the game model. Developers should try and avoid using Unity specific idioms in this code, just modify the model and post new events.

**Model:**
The Model folder/namespace contains classes which hold the data required for the game. Both the Mechanics and the Gameplay systems can access and modify the model. The model generally does not contain any code apart from data modification or query methods.

**Visual:**
The visual namespace typically provides functionality for modifying game feedback mechanisms, which are not related to mechanics or gameplay. Eg Visual Effects, Music or Animation code. Visual Systems can subscribe to events, but generally do not post events or modify the model.

## Customising Behaviour

There are two ways to create behaviour when an event is fired. The Execute method of the event class can be modified, or a delegate can be assigned to the OnExecute event delegate which is fired after the Execute method has run.