





# Managing a rugby match

**Concept.** Building a toy model simulation of a rugby match whose outcome can be manipulated through correctly-timed player substitutions and game management decisions. The dextera state manipulation framework we have built around the stochadex can meet these requirements, and a dashboard can be created for user interaction. All this combines together to make a simple dashboard game, which we call: ‘trywizard’. For the mathematically-inclined, this chapter will motivate the construction of a specific modelling framework for rugby match simulation. For the programmers, the public Git repository for the code described in this chapter can be found here: <https://github.com/umbralcalc/trywizard>.

## 11.1 Designing the event simulation engine

Since the basic state manipulation framework and simulation engine will run using [dextera](#), the mathematical novelties in this project are all in the design of the rugby match model itself. And, as ever, we’re not especially keen on spending a lot of time doing detailed data analysis to come up with the most realistic values for the parameters that are dreamed up here. Even though this would also be interesting.<sup>1</sup>

Let’s begin by specifying an appropriate event space to live in when simulating a rugby match. It is important at this level that events are defined in quite broadly applicable terms, as it will define the state space available to our stochastic sampler and hence the simulated game will never be allowed to exist outside of it. It seems reasonable to characterise a rugby union match by the following set of events: Penalty, Free Kick (the punitive events); Penalty Goal, Drop Goal, Try (the scoring events); Kick Phase, Run Phase, Knock-on, Scrum, Lineout, Maul and Ruck (the general play events). Using this set of events, in Fig. 11.1 we have summarised our approach to match state transitions into a single event graph. In order to capture the fully detailed range of events that are possible in a real-world match, we’ve needed to be a little imaginative in how we define

---

<sup>1</sup>One could do this data analysis, for instance, by scraping player-level performance data from one of the excellent websites that collect live commentary data such as [rugbypass.com](http://rugbypass.com) or [espn.co.uk/rugby](http://espn.co.uk/rugby).

the kinds of state transitions which occur. In other words; it's fair to say that our simplified model here represents just a subset of events that a real rugby match could generate.

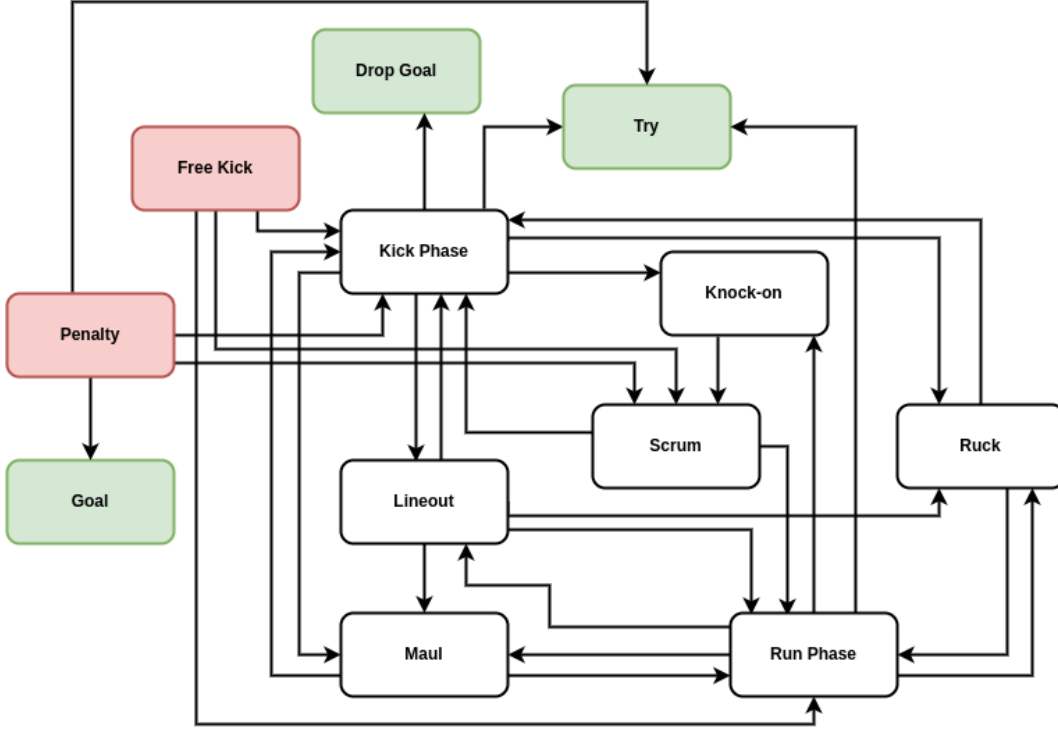


Figure 11.1: Simplified event graph of a rugby union match.

In addition to occupying some state in the event graph, the state of a rugby match must also include a binary ‘possession’ element which encodes which team has the ball at any moment. We should also include the 2-dimensional pitch location of the ball as an element of the match state in order to get a better sense of how likely some state transitions are, e.g., when playing on the edge of the pitch near the touchline it’s clearly more likely that a Run Phase is going to result in a lineout than if the state is currently in the centre of the pitch. To add even more detail, we will include states for each playing position on each side which encode: the playing abilities of each player, their fatigue status and their substitution status. The latter of these encodes whether or not the player has been injured or yellow/red-carded during the course of a game and should also enable management strategies to become more nuanced.

Since a rugby match exists in continuous time, it is natural to choose a continuous-time event-based simulation model for our game engine. As we have discussed in previous chapters already, this means we will be characterising transition probabilities of the event graph by ratios of event rates in time. Recalling our notation in previous chapters, if we consider the state history matrix of the match  $X$  and the timestep index  $t$ , we can start by assigning each transition  $a \rightarrow b$  on the event graph an associated rate of occurrence  $\lambda_{a \rightarrow b}(X', t)$  which is defined in units of continuous time. In addition to the transitions displayed on the graph, we can add a ‘possession change transition’;

where the possession of the ball in play moves to the opposing team. This transition may occur while the match is also in any of the white-coloured states on the graph in Fig. 11.1 and let's assign this a time and state-dependent expected rate of occurrence  $\lambda_{\text{pos}}(X', \mathbf{t})$ .

But how should we connect the match state history to specific values of the event rates? Let's now discuss this.

## **11.2 Linking event rates to match state**

### **11.3 Deciding on gameplay actions**

### **11.4 Writing the game itself**



# **Bibliography**