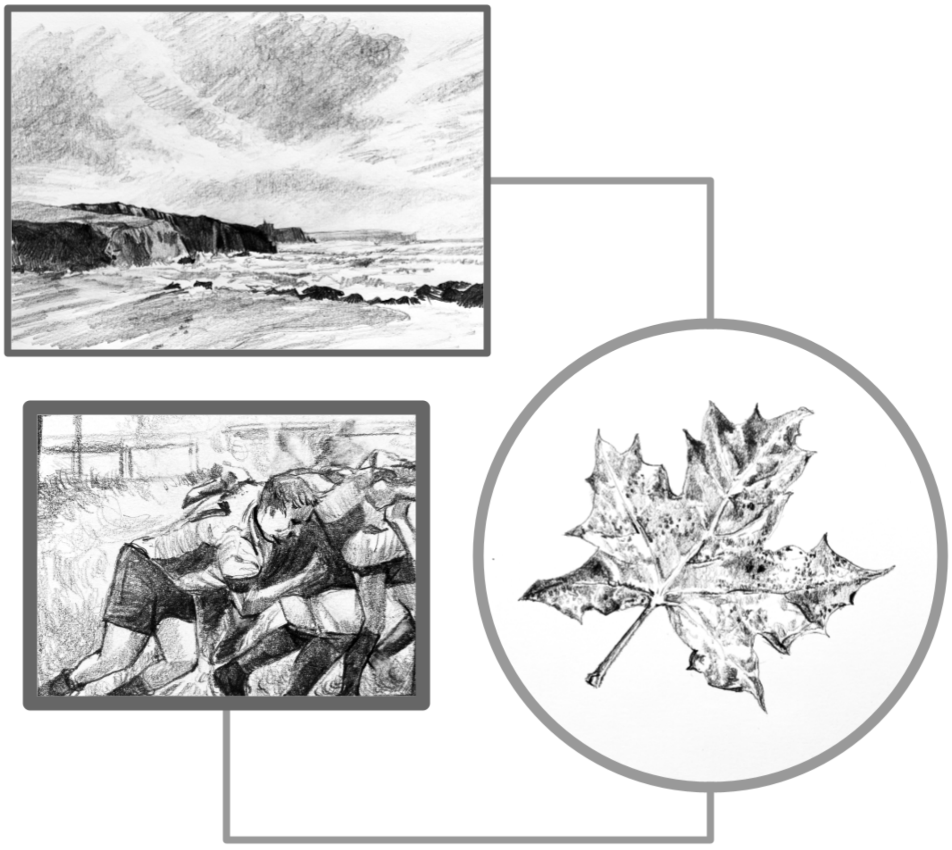

11

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 dexetera 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 dexetera, 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.[1]

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 the kinds

[1]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 or espn.co.uk/rugby.

of event 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.

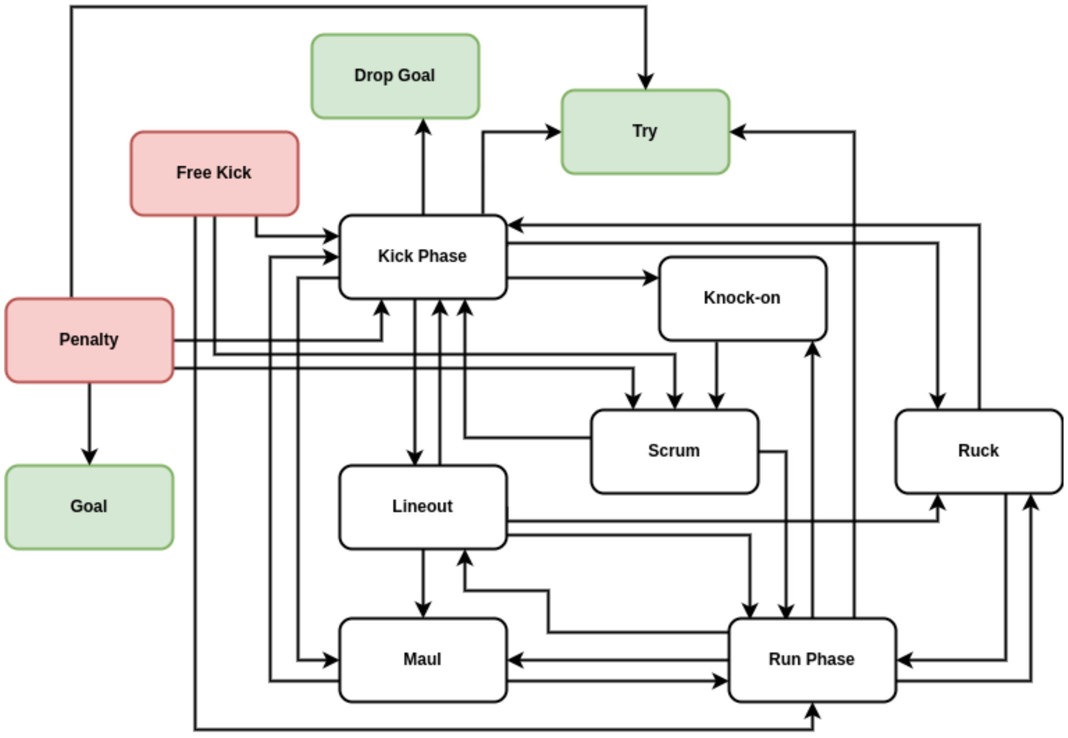

Figure 11.1: Simplified event graph of a rugby union match.

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. But what are the event transitions we should be considering and how should we model them? Let's now go through these in turn with our motivations in each case.

We begin with what we'll call the 'possession change transition'.

11.2 Linking to player attributes

11.3 Deciding on gameplay actions

11.4 Writing the game itself

Bibliography