RouteRender Language Specification

Tim Forth, Eric Gage (RouteRender)

0.1 Introduction

In American Football pre-designed plays are extremely important. If one player forgets the play or makes the wrong move, then the offense can be left vulnerable to defensive schemes. Therefore, it is imperative that players have the playbook memorized. For this to be accomplished, every play must be drawn out in an understandable way. However, this task is more daunting than it might seem. The average NFL playbook has 6,000 viable variations and almost infinite total variations. Therefore, we propose a programming language that would allow coaches to easily draw up plays for their playbook would bring efficiency to a currently inefficient task.

This problem requires a programming language because it simplifies the creation process. Every team runs a different offense and a different defense so there are numerous inputs that would be required to create a play (such as blocking schemes, routes, opposing defensive schemes, etc.). A programming language would allow coaches from different teams to create game-specific playbooks tailored to a specific team's defense.

0.2 Design Principles

In terms of design, a coach should be able to input very straightforward and simple inputs in order to create a play. It is also important that there is some way to save certain blocking schemes since there are a few schemes that are used frequently with small adjustments. Similarly, the language of football should be intertwined with the playbook language; the coach should be able to use keywords as inputs for schemes, routes, and formations (routes: go, basic, corner, dig. Schemes: power, counter, inside zone) as well as creating new ones. Simplicity is really the key factor in our design implementation.

0.3 Examples

1. dotnet run example1.rr

   Expected Output: ¡svg width="1500" height="1500" xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/199
   ¡circle cx="530" cy="425" r="30" stroke="black" stroke-width="3" fill="none"/¿

   ¡circle cx="610" cy="425" r="30" stroke="black" stroke-width="3" fill="none"/¿

   ¡rect x="660" y="395" width="60" height="60" stroke="black" stroke-width="3" fill="none"/¿

   ¡circle cx="770" cy="425" r="30" stroke="black" stroke-width="3" fill="none"/¿

   ¡circle cx="850" cy="425" r="30" stroke="black" stroke-width="3" fill="none"/¿

   ¡!–IZ Scheme against 3-4 Defense–¿ ¡!–Labeling Keys–¿ ¡text x="417" y="285" font-size="30" stroke="red" font-family="Arial, Helvetica, sans-serif"¿K¡/text¿ ¡text x="597" y="235" font-size="30" stroke="red" font-family="Arial, Helvetica, sans-serif"¿K¡/text¿ ¡!–Double to PSK–¿ ¡line x1="530" y1="395" x2="530" y2="380" stroke="black" stroke-width="3"/¿ ¡line x1="510" y1="380" x2="550" y2="380" stroke="black" stroke-width="3"/¿ ¡!–Double to PSK–¿ ¡line x1="610" y1="455" x2="610" y2="485" stroke="black" stroke-width="3"/¿ ¡line x1="610" y1="485" x2="870" y2="480" stroke="black" stroke-width="3"/¿ ¡line x1="870" y1="460" x2="870" y2="500" stroke="black" stroke-width="3"/¿ ¡!–A to BSK–¿ ¡line x1="690" y1="395" x2="560" y2="375" stroke="black" stroke-width="3"/¿ ¡line x1="555" y1="390" x2="565" y2="355" stroke="black" stroke-width="3"/¿ ¡!–A to BSK–¿ ¡line x1="770" y1="395" x2="730" y2="375" stroke="black" stroke-width="3"/¿ ¡line x1="725" y1="390" x2="735" y2="355" stroke="black" stroke-width="3"/¿ ¡!–C to -1–¿ ¡line x1="850" y1="395" x2="640" y2="290" stroke="black" stroke-width="3"/¿ ¡line x1="630" y1="305" x2="650" y2="275" stroke="black" stroke-width="3"/¿ ¡text x="670" y="385" font-size="60" font-family="Arial, Helvetica, sans-serif"¿N¡/text¿

   ¡text x="510" y="375" font-size="60" font-family="Arial, Helvetica, sans-serif"¿F¡/text¿

   ¡text x="670" y="375" font-size="60" font-family="Arial, Helvetica, sans-serif"¿N¡/text¿

   ¡text x="830" y="375" font-size="60" font-family="Arial, Helvetica, sans-serif"¿R¡/text¿

   ¡text x="590" y="275" font-size="60" font-family="Arial, Helvetica, sans-serif"¿B¡/text¿

¡text x=”750” y=”275” font-size=”60” font-family=”Arial, Helvetica, sans-serif”¿M¡/text¿

¡text x=”400” y=”325” font-size=”60” font-family=”Arial, Helvetica, sans-serif”¿W¡/text¿

¡text x=”940” y=”325” font-size=”60” font-family=”Arial, Helvetica, sans-serif”¿S¡/text¿ ¡text x=”380” y=”90” font-size=”60” font-family=”Arial, Helvetica, sans-serif”¿S¡/text¿

¡text x=”970” y=”90” font-size=”60” font-family=”Arial, Helvetica, sans-serif”¿FS¡/text¿

¡text x=”150” y=”375” font-size=”60” font-family=”Arial, Helvetica, sans-serif”¿C¡/text¿

¡text x=”1190” y=”375” font-size=”60” font-family=”Arial, Helvetica, sans-serif”¿C¡/text¿ ¡/svg¿

2. dotnet run example2.rr

   Expected Output: ¡svg width=”1500” height=”1500” xmlns=”http://www.w3.org/2000/svg” xmlns:xlink=”http://www.w3.org/199
   ¡circle cx=”530” cy=”425” r=”30” stroke=”black” stroke-width=”3” fill=”none”/¿

   ¡circle cx=”610” cy=”425” r=”30” stroke=”black” stroke-width=”3” fill=”none”/¿

   ¡rect x=”660” y=”395” width=”60” height=”60” stroke=”black” stroke-width=”3” fill=”none”/¿

   ¡circle cx=”770” cy=”425” r=”30” stroke=”black” stroke-width=”3” fill=”none”/¿

   ¡circle cx=”850” cy=”425” r=”30” stroke=”black” stroke-width=”3” fill=”none”/¿

   ¡!–Power Scheme against 3-4 Defense–¿ ¡!–Labeling Keys–¿ ¡text x=”417” y=”285” font-size=”30” stroke=”red” font-family=”Arial, Helvetica, sans-serif”¿K¡/text¿ ¡text x=”597” y=”235” font-size=”30” stroke=”red” font-family=”Arial, Helvetica, sans-serif”¿K¡/text¿ ¡!–Gap Hinge–¿ ¡line x1=”530” y1=”395” x2=”580” y2=”395” stroke=”black” stroke-width=”3”/¿ ¡line x1=”580” y1=”395” x2=”530” y2=”485” stroke=”black” stroke-width=”3”/¿ ¡line x1=”530” y1=”465” x2=”530” y2=”505” stroke=”black” stroke-width=”3”/¿ ¡!–Pull for -1–¿ ¡line x1=”610” y1=”455” x2=”610” y2=”495” stroke=”black” stroke-width=”3”/¿ ¡line x1=”610” y1=”495” x2=”790” y2=”495” stroke=”black” stroke-width=”3”/¿ ¡line x1=”790” y1=”495” x2=”790” y2=”300” stroke=”black” stroke-width=”3”/¿ ¡line x1=”770” y1=”300” x2=”810” y2=”300” stroke=”black” stroke-width=”3”/¿ ¡!–Down Block on Frank–¿ ¡line x1=”690” y1=”395” x2=”560” y2=”375” stroke=”black” stroke-width=”3”/¿ ¡line x1=”555” y1=”390” x2=”565” y2=”355” stroke=”black” stroke-width=”3”/¿ ¡!–Down Block on Nose–¿ ¡line x1=”770” y1=”395” x2=”730” y2=”375” stroke=”black” stroke-width=”3”/¿ ¡line x1=”725” y1=”390” x2=”735” y2=”355” stroke=”black” stroke-width=”3”/¿ ¡!–Block to Key–¿ ¡line x1=”850” y1=”395” x2=”640” y2=”290” stroke=”black” stroke-width=”3”/¿ ¡line x1=”630” y1=”305” x2=”650” y2=”275” stroke=”black” stroke-width=”3”/¿

   ¡text x=”510” y=”375” font-size=”60” font-family=”Arial, Helvetica, sans-serif”¿F¡/text¿

   ¡text x=”670” y=”375” font-size=”60” font-family=”Arial, Helvetica, sans-serif”¿N¡/text¿

   ¡text x=”830” y=”375” font-size=”60” font-family=”Arial, Helvetica, sans-serif”¿R¡/text¿

   ¡text x=”590” y=”275” font-size=”60” font-family=”Arial, Helvetica, sans-serif”¿B¡/text¿

   ¡text x=”750” y=”275” font-size=”60” font-family=”Arial, Helvetica, sans-serif”¿M¡/text¿

   ¡text x=”400” y=”325” font-size=”60” font-family=”Arial, Helvetica, sans-serif”¿W¡/text¿

   ¡text x=”940” y=”325” font-size=”60” font-family=”Arial, Helvetica, sans-serif”¿S¡/text¿ ¡text x=”380” y=”90” font-size=”60” font-family=”Arial, Helvetica, sans-serif”¿S¡/text¿

   ¡text x=”970” y=”90” font-size=”60” font-family=”Arial, Helvetica, sans-serif”¿FS¡/text¿

   ¡text x=”150” y=”375” font-size=”60” font-family=”Arial, Helvetica, sans-serif”¿C¡/text¿

   ¡text x=”1190” y=”375” font-size=”60” font-family=”Arial, Helvetica, sans-serif”¿C¡/text¿ ¡/svg¿

3. dotnet run example3.rr

   Expected Output: ¡svg width=”1500” height=”1500” xmlns=”http://www.w3.org/2000/svg” xmlns:xlink=”http://www.w3.org/199
   ¡circle cx=”530” cy=”425” r=”30” stroke=”black” stroke-width=”3” fill=”none”/¿

   ¡circle cx=”610” cy=”425” r=”30” stroke=”black” stroke-width=”3” fill=”none”/¿

   ¡rect x=”660” y=”395” width=”60” height=”60” stroke=”black” stroke-width=”3” fill=”none”/¿

   ¡circle cx=”770” cy=”425” r=”30” stroke=”black” stroke-width=”3” fill=”none”/¿

¡circle cx=”850” cy=”425” r=”30” stroke=”black” stroke-width=”3” fill=”none”/¿

¡!–Counter Scheme against 4-3 Defense–¿ ¡!–Labeling Keys–¿ ¡text x=”560” y=”235” font-size=”30” stroke=”red” font-family=”Arial, Helvetica, sans-serif”¿K¡/text¿ ¡!–Pull for -1–¿ ¡line x1=”530” y1=”455” x2=”545” y2=”495” stroke=”black” stroke-width=”3”/¿ ¡line x1=”545” y1=”495” x2=”790” y2=”495” stroke=”black” stroke-width=”3”/¿ ¡line x1=”790” y1=”495” x2=”790” y2=”300” stroke=”black” stroke-width=”3”/¿ ¡line x1=”770” y1=”300” x2=”810” y2=”300” stroke=”black” stroke-width=”3”/¿ ¡!–Kickout–¿ ¡line x1=”610” y1=”455” x2=”610” y2=”485” stroke=”black” stroke-width=”3”/¿ ¡line x1=”610” y1=”485” x2=”870” y2=”480” stroke=”black” stroke-width=”3”/¿ ¡line x1=”870” y1=”460” x2=”870” y2=”500” stroke=”black” stroke-width=”3”/¿ ¡!–Down Block on Frank–¿ ¡line x1=”690” y1=”395” x2=”560” y2=”375” stroke=”black” stroke-width=”3”/¿ ¡line x1=”555” y1=”390” x2=”565” y2=”355” stroke=”black” stroke-width=”3”/¿ ¡!–Down Block on Nose–¿ ¡line x1=”770” y1=”395” x2=”730” y2=”375” stroke=”black” stroke-width=”3”/¿ ¡line x1=”725” y1=”390” x2=”735” y2=”355” stroke=”black” stroke-width=”3”/¿ ¡!–Block to BS Key–¿ ¡line x1=”850” y1=”395” x2=”640” y2=”290” stroke=”black” stroke-width=”3”/¿ ¡line x1=”630” y1=”305” x2=”650” y2=”275” stroke=”black” stroke-width=”3”/¿

¡text x=”480” y=”385” font-size=”60” font-family=”Arial, Helvetica, sans-serif”¿F¡/text¿ ¡text x=”590” y=”385” font-size=”60” font-family=”Arial, Helvetica, sans-serif”¿E¡/text¿ ¡text x=”750” y=”385” font-size=”60” font-family=”Arial, Helvetica, sans-serif”¿N¡/text¿ ¡text x=”870” y=”385” font-size=”60” font-family=”Arial, Helvetica, sans-serif”¿R¡/text¿ ¡text x=”670” y=”285” font-size=”60” font-family=”Arial, Helvetica, sans-serif”¿M¡/text¿ ¡text x=”800” y=”285” font-size=”60” font-family=”Arial, Helvetica, sans-serif”¿S¡/text¿ ¡text x=”540” y=”285” font-size=”60” font-family=”Arial, Helvetica, sans-serif”¿W¡/text¿

¡text x=”940” y=”325” font-size=”60” font-family=”Arial, Helvetica, sans-serif”¿S¡/text¿ ¡text x=”380” y=”90” font-size=”60” font-family=”Arial, Helvetica, sans-serif”¿S¡/text¿

¡text x=”970” y=”90” font-size=”60” font-family=”Arial, Helvetica, sans-serif”¿FS¡/text¿

¡text x=”150” y=”375” font-size=”60” font-family=”Arial, Helvetica, sans-serif”¿C¡/text¿

¡text x=”1190” y=”375” font-size=”60” font-family=”Arial, Helvetica, sans-serif”¿C¡/text¿ ¡/svg¿

0.4 Language Concepts

In order to create plays, users need a firm grasp of the core concepts: primitives and combining forms. Primitives encompass player positions, routes, offensive and defensive formations, schemes, and defensive coverages. Coaches put the primitives together in numerous different variations to create game plans against specific defenses, which they also implement. Understanding the primitives involves knowing how routes complement specific coverages or how run schemes exploit defensive weaknesses based on formations.
The combining forms, offensive and defensive plays, are the combination of these primitives. To create effective offensive plays, users must comprehend how different route concepts and offensive formations interact with defensive coverages and formations to exploit vulnerabilities. While understanding the primitives is more simple, such as a specific route against man or zone coverage, understanding the combining forms of the language are much more complex. It involves, for example, knowing how the combination of a 3x2 formation and 5 specific routes can pull apart a defense leaving some receiver open.

0.5 Formal Syntax

$\langle program \rangle ::= \langle play \rangle$
$\langle play \rangle ::= \langle defense \rangle \langle scheme \rangle \langle routes \rangle \langle formation \rangle$
$\langle defense \rangle ::= (\langle box \rangle \,, \langle coverage \rangle)$
$\langle box \rangle ::= 34 \;\mid 43$
$\langle coverage \rangle ::= man \;\mid cover1 \;\mid cover2 \;\mid cover3 \;\mid cover4 \;\mid cover6$
$\langle formation \rangle ::= (\langle unit \rangle \,, \langle receivers \rangle)$
$\langle unit \rangle ::= iformation \;\mid empty \;\mid singleback \;\mid shotgun$
$\langle receivers \rangle ::= (axb);\; a+b<6;\; a,\; b\, are\, integers\, greater\, than\; 0 \;\mid \_$
$\langle scheme \rangle ::= power \;\mid counter \;\mid insidezone \;\mid outsidezone$
$\langle routes \rangle ::= \langle route \rangle * list$ | length of routes $= a + b$ from receivers
$\langle route \rangle ::= (player,\; movement,\; read)$

$\langle player \rangle ::= x \ | \ y \ | \ z \ | \ h \ | \ a$
$\langle movement \rangle ::= Go \ | \ Slant \ | \ Out \ | \ In \ | \ Post \ | \ Corner \ | \ Curl \ | \ Dig \ | \ Hitch \ | \ Comeback \ | \ Block \ | \ Post - Corner \ | \ Fade \ | \ Screen$
$\langle read \rangle ::= 1 \ | \ 2 \ | \ 3 \ | \ 4 \ | \ 5$

0.6 Semantics

Syntax	Abstract Syntax	Type	Meaning
43	Box of string	string	tells evaluator what the front 7 looks like i.e. how it should be represented in the svg file; stored as an F# primitive string data type
cover1	Coverage of string	string	tells evaluator what the coverage of the defense is i.e. how it should be represented in the svg file; stored as an F# primitive string data type
(43, cover1)	Defense of Box * Coverage	string * string	two-tuple consisting of a box and coverage that tells eval how to draw defense
power	Scheme of string	string	tells evaluator what scheme the offense should be in
1	Read of int	int	tells a Quarterback what read a receiver is
x	Player of char	char	identifier for which player is running a route
post	Movement of string	string	identifies what movement a player is doing
(x, post, 1)	Route of Player ∗ Movement ∗ Read	char ∗ string ∗ int	player runs the movement specified with the read specified for the Quarterback
[(x, post, 1), (y, corner, 2), (z, dig, 3)]	Routes of Route list	List⟨ Route ⟩	list of Routes
iformation	Unit of string	string	describes the unit of the offense
(2x1)	Receivers of int ∗ int	int ∗ int	describes the alignment of the receivers
(iformation, (2x1))	Formation of Unit ∗ Receivers	string ∗ (int ∗ int)	describes a full formation for the offense including the scheme and where the receivers are
(43, cover1)power[(x, post, 1), (y, corner, 2), (z, dig, 3)](iformation, (2x1));	Play of Defense ∗ Formation ∗ Scheme ∗ Routes	(string ∗ string) ∗ (string ∗ (int ∗ int)) ∗ string ∗ List ⟨ Route ⟩	describes a full play with all needed information for the svg evaluator to draw it
;	n/a	char	tells the parser that a play is done

Table 1: Semantics of data types.

0.7 Remaining Work

We still need to implement the route data type, and figure out how to draw routes dynamically. We also need to implement an operator for creating more than one play at a time. Additionally, we need to update the Abstract Data

Types to represent our current thinking.