Frisbee Play Language Specification

Skylar Yarter   Jocelyn Bliven

May 13, 2024

Introduction

This language helps a user decide what ultimate frisbee play to call in a given situation. When you are in the heat of the game, it can be hard to figure out what play to call. No longer! This programming language will help you decide what plays to call at any given moment based on where the players, disc, and the defensive force are.

Ultimate frisbee requires "reading" the field to figure out what play is best to call and this reading of the field requires evaluating multiple parameters. A language that allows the user to read the field more easily and without human emotion and bias swaying the result would allow more appropriate plays to be called. Many of the evaluations being made by the players or coaches when choosing a play are quantitative (ie. where are the players and disc on the field, which way is the defense forcing). These quantitative parameters fit well within the scope of a programming language and can be compared better by a computer than a person standing on the field with a skewed perspective.

Design Principles

We aim to create an easily readable, quick-to-type programming language so it can be used by frisbee coaches and players, who may not be familiar with general programming practices. Therefore, we prioritize simplicity in the syntax of our language so that it is easy to learn and to read. We do not want to overcomplicate the language by adding too many elements or rules, as that would hinder our user base. The language need not be object-oriented or follow a complex model as it is fairly basic, which allows for the easy-to-learn syntax.

Examples

(Our syntax for our examples changed but for the most part, the underlying goal of the examples was accomplished)

Example 1: Basic.txt
Offense = (120,390), true; (140,220), false; (160,220), false; (180,220), false; (200,220), false; (220,220), false; (240,220), false;
Defense = (120,370); (140,200); (160,200); (180,200); (200,200); (220,200); (240,200);
Defense = Manual
Force = Home

Example 2: Automatic Person Defense (auto.txt)
Offense = (220,390), true; (220,220), false; (300,220), false; (320,220), false; (340,220), false; (360,220), false; (380,220), false;
Defense = Automatic
Force = Away

Example 3: Endzone
Offense = (220,390), true; (220,270), false; (220,140), false; (400,350), false; (400,275), false; (400,175), false; (400,100), false;
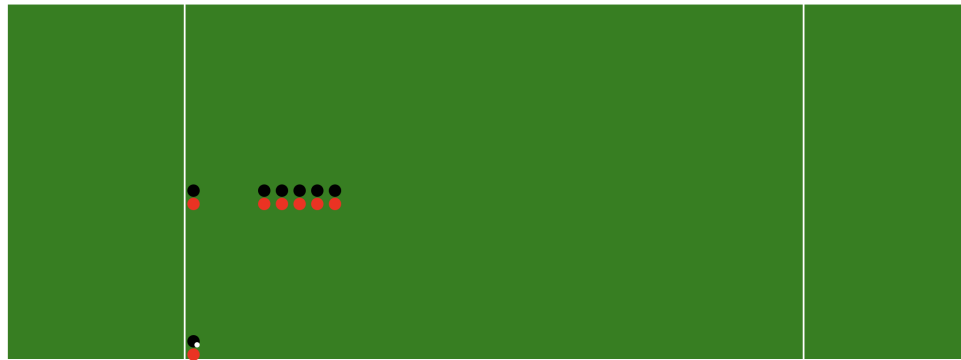Defense = Automatic
Force = Home
Play = if disc in range 100 to 300 and force is Force = Home
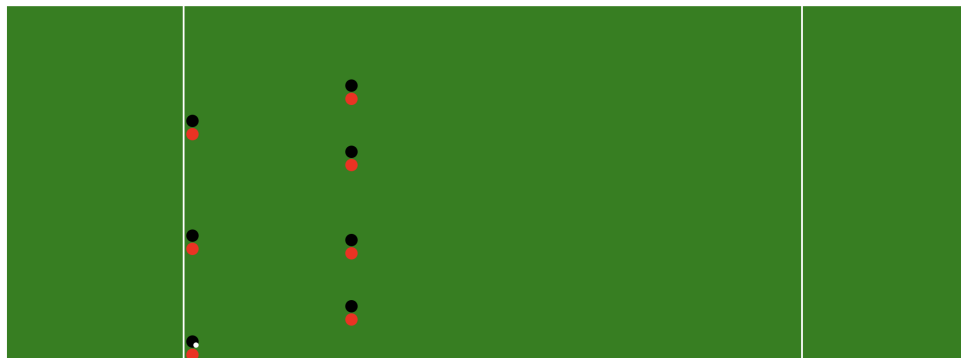
Example 1 Output:

Recommended Play: Color
User Input Play: N/A

Example 2 Output:

Recommended Play: Angled Stack
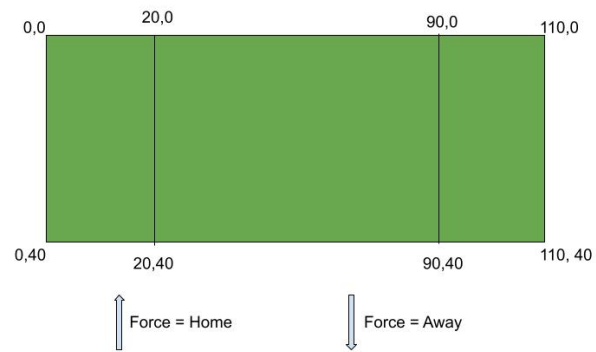User Input Play: N/A

Example 3 Output:

Recommended Play: Color
User Input Play: run entered play

Language Concepts

A user needs to understand that each offensive and defensive player is a unique object which is essentially just a coordinate. The primitives of this language are coordinates which ultimately determine where the players and the disc are. The multiple players are combined into a team. Other primitives are the defensive force which is basically just a string. Adding the force and where the disc is allows the user to fine-tune what play they should call. The field is a combining form which takes all the teams, disc, and force and places them in the correct position. Additionally, the user should understand how the field is set up and how to visualize it.

Field Visualization:

Formal Syntax

Current syntax:
<expr> ::=<offense><defense>
<offense> ::= <oplayer><offense> | <oplayer>
<defense> ::= <dplayer><defense> | <dplayer>
<oplayer> ::= <coordinate><disc>
<dplayer> ::= <coordinate>
<disc> ::= true | false
<coordinate>::= (<digit>,<digit>)
<digit> ::= <digit>n | n where n is any positive integer

Future syntax:
<expr> ::= <field><plays> | <field>
<field> ::=<offense><defense><force>
<offense> ::= <oplayer><offense> | <oplayer>
<defense> ::= <dplayer><defense> | <dplayer> | zone | person
<disc> ::= true | false
<oplayer> ::= <coordinate><disc>
<dplayer> ::= <coordinate>
<force> ::= home | away | flat
<plays> ::= <play> | <play><plays> | Library
<play> ::= string
<coordinate>::= (<digit>,<digit>)
<digit> ::= <digit>n | n where n is any positive integer

Semantics

The primitives in this language are coordinates, defense types, booleans, and the force. Plays are programs passed in or accessed in a Library as strings.

The combining forms in this language are players, which are made from coordinates and booleans determining if they have the disc; teams, which are made of lists of players; and a field, which is made from combining teams and the force on the field.

The programs in our language can optionally read in a file containing a team's plays. The plays would be written in with booleans to indicate when to use them (e.g., "if force=home, then angled stack"). If the user does not provide a file of plays, the program will use the default plays that we provide.

In its simplest state, the output of a program will be a frisbee field with the players and disc at the correct coordinates. For our final version of the language, the output of a program will be a picture of the field with a list of suggested plays, either from the library or from the user's inputted plays.

Syntax	Abstract Syntax	Prec./Assoc.	Meaning
(<n>,<n>) where n is an integer	Coordinate of x: int; y: int	n/a	x and y are integers. A coordinate is a primitive that represents where on the field a player is.
<coordinate> , bool	Offense of Coordinate * HasDisc	n/a	An offensive player is made up of a coordinate and a boolean value of whether or not they have the disc. The player is drawn on the field in the appropriate location with/without the disc based on this information.
(<offense>, <offensive>) | e	Offensive of Offense list	n/a	Offensive (which symbolizes an offensive team) is a list of offensive players. This is a combining form of offensive players.