AN ABSTRACT OF THE THESIS OF

Christoph Neumann for the degree of Master of Science in Computer Science presented on June 2, 2006.

Title: Interactive Football Playbook

Abstract approved: ___________________________ Ronald A. Metoyer

This thesis presents a domain specific visual language designed to allow coaches to create content that exhibits the complex 2D interactions observed in the game of American football. Coaches can visually program the content by using symbols and drawing primitives similar to those that they currently use to design static playbooks. However, the result is not a static play, but animated primitives that move according to the programmed rules. The symbols and primitives represent rules that can be applied to the 2D synthetic players. The user can specify rules and run the simulation. At runtime, the rules are unified as a set of vector constraints. The resultant vector is used to animate the motion of the 2D player. The combination of football primitives and parameterization of those primitives allows the user to program the simulation to achieve the desired performance. We present the language and user-centered design process. We discuss the language and visual components and conclude with a description of the feedback from interviews with football coaches.
Interactive Football Playbook

by
Christoph Neumann

A THESIS
submitted to
Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented June 2, 2006
Commencement June 2007
ACKNOWLEDGEMENTS

I would like to thank my adviser, Ron Metoyer, for his support and direction. I would like to thank Margaret Burnett for her assistance with the human factors and the end user programming aspects. I would like to thank the additional members of my committee for their feedback and support: Rajeev Pandey and Jimmy Yang. I would like to thank Coaches Mark Banker, Greg Newhouse, and Danny Langsdorf for their insightful discussions and access to their coaching sessions. I would like to thank John Pane, Alan Blackwell, and Mary Beth Rosson for their advice and feedback during the formative stages of research. Additionally, I would like to thank Joey Lawrance, Jason Dagit, and Sam Adams for their feedback and contributions. This work was supported in part by by NSF CCR-0237706.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Page</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
</tr>
<tr>
<td>3</td>
<td>User-Centered Design</td>
</tr>
<tr>
<td>3.1</td>
<td>Background on Cognitive Dimensions</td>
</tr>
<tr>
<td>3.2</td>
<td>Analysis Process</td>
</tr>
<tr>
<td>3.3</td>
<td>Insights and Design Maneuvers</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Visibility, Hard Mental Operations and Abstraction</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Hidden Dependencies, Premature Commitment, Error-proneness and Viscosity</td>
</tr>
<tr>
<td>3.3.3</td>
<td>Provisionality and Progressive Evaluation</td>
</tr>
<tr>
<td>3.3.4</td>
<td>Whoops, No Secondary Notation!</td>
</tr>
<tr>
<td>3.4</td>
<td>Evaluation Summary</td>
</tr>
<tr>
<td>4</td>
<td>The Interactive Football Playbook</td>
</tr>
<tr>
<td>4.1</td>
<td>Visual Language Environment</td>
</tr>
<tr>
<td>4.2</td>
<td>Language Definition</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Sequencing and Parallel Execution</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Players</td>
</tr>
<tr>
<td>4.2.3</td>
<td>Route</td>
</tr>
<tr>
<td>4.2.4</td>
<td>Block</td>
</tr>
<tr>
<td>4.2.5</td>
<td>Avoid</td>
</tr>
<tr>
<td>4.2.6</td>
<td>Guard</td>
</tr>
<tr>
<td>4.2.7</td>
<td>Pursue</td>
</tr>
<tr>
<td>4.3</td>
<td>Limitations of the Language</td>
</tr>
<tr>
<td>4.4</td>
<td>Implementation</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Internal Representation</td>
</tr>
<tr>
<td>4.4.2</td>
<td>Unification</td>
</tr>
<tr>
<td>4.4.3</td>
<td>Animation</td>
</tr>
<tr>
<td>4.4.4</td>
<td>Platform</td>
</tr>
<tr>
<td>5</td>
<td>Results</td>
</tr>
<tr>
<td>5.1</td>
<td>Demonstration Feedback</td>
</tr>
<tr>
<td>5.2</td>
<td>Identified Language Needs</td>
</tr>
<tr>
<td>6</td>
<td>Discussion</td>
</tr>
<tr>
<td>7</td>
<td>Conclusion</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>A screenshot of the Interactive Football Playbook interface.</td>
<td>2</td>
</tr>
<tr>
<td>3.1</td>
<td>An example of formations as drawn on a whiteboard during classroom instruction.</td>
<td>6</td>
</tr>
<tr>
<td>3.2</td>
<td>A portion of the paper prototype: the offensive player formation editing screen for the Interactive Football Playbook. The user would manipulate a player by dragging it from the toolbox, around the field, or off the field.</td>
<td>7</td>
</tr>
<tr>
<td>4.1</td>
<td>The formation editor. A player is being dragged onto the field from the icons on the left.</td>
<td>18</td>
</tr>
<tr>
<td>4.2</td>
<td>The scenario editor. Rules are associated with the players as well as spatial locations. The possible rule types are displayed on the left.</td>
<td>19</td>
</tr>
<tr>
<td>4.3</td>
<td>A fragment of a scenario. At the top left are the rules as specified by the user and the resulting animation is seen from left to right down the page.</td>
<td>20</td>
</tr>
<tr>
<td>4.4</td>
<td>Two examples of chaining.</td>
<td>21</td>
</tr>
<tr>
<td>4.5</td>
<td>Execution model of a scenario.</td>
<td>25</td>
</tr>
<tr>
<td>4.6</td>
<td>An example of the summation of vectors for the wide receiver (“WR”). The WR is attempting to avoid the CB while running a pass route and the CB is attempting to guard the receiver. The resultant unified vector is shown in black. (The vectors in the figure are approximations.)</td>
<td>27</td>
</tr>
<tr>
<td>4.7</td>
<td>Approximate resultant vectors for each player over time. The user specified rules are seen in grey.</td>
<td>28</td>
</tr>
</tbody>
</table>
LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>9</td>
</tr>
<tr>
<td>4.1</td>
<td>20</td>
</tr>
</tbody>
</table>

3.1 The Cognitive Dimensions as summarized in [15].

DEDICATION

To my wife Lisa
For her support, patience and perseverance.
Chapter 1 – Introduction

Current collegiate football athletes are from the video game generation. College coaches believe that because of this experience, the athletes respond not only to visual content (such as 2D static plays), but to 3D animated visual content. In our interviews, the coaches communicated that despite white board drawings, overhead projections, printed playbooks with descriptions and annotations, and video footage, they find it difficult to help the athletes “visualize” a play executed properly. Furthermore, coaches often use several different video clips to illustrate the proper execution of a play because they do not have a single clip in which all of the athletes are doing precisely the right things at the right times. Additionally, in video, there may be only two or three static camera views of the field as the play progresses over time. The athlete is limited to viewing the play from these particular views.

Coaches hypothesize that if the athletes can see the play evolve properly, from their particular position on the field, then they will more quickly grasp the concepts of how they are to react in given situations. A 3D synthetic environment would allow an athlete to view the play from any 3D location, from any player’s position, and from any stadium location. There are essentially no limits on the camera view in 3D. The coaches expressed hope that such a 3D view could not only serve to augment the video footage but even replace it in some instances.

3D football content is not a new concept. Electronic Arts produces the Madden Football series every year [2]. The problem is that each collegiate team has their own content—their own offensive and defensive formations, their own offensive and defensive plays, and their own athletes. With authorable content the coach should be able to create the “perfect play” with the appropriate player personnel as well as create a play with precise imperfections.

For these reasons discussed above, the American football domain is both an interesting and challenging domain. The advantages of 3D content are clear: unlimited camera views, unlimited example plays, and familiarity with 3D animated media. However, this content must be quickly created by the coaches who must prepare for a new opponent.
Figure 1.1: A screenshot of the Interactive Football Playbook interface.

*each week.* It is also these two aspects: authorability and 3D visualization that excites coaches and that inspires our work. Although we will be presenting a 2D authoring language, the leap from 2D to 3D environments and character motion is not a difficult one, and can be addressed with existing technology from the graphics and animation field.

Fortunately, football coaches rely on a fairly standard symbolic language for specifying plays on static media such as paper or a whiteboard. Our solution, the Interactive Football Playbook (IFP), builds on this standardized symbolic language and augments it with primitives and language enhancements that incorporate the terminology that coaches use when specifying the attributes of a play. Our language consists of spatial constraints on the simulated players. The spatial constraints are unified resulting in a vector that is used to animate the simulated players (Figure 1.1).
Chapter 2 - Literature Review

Our work on the IFP builds on contributions from several research areas including computer graphics and visual programming and ideas from the commercial world. Our previously published research describes the needs and challenges for coaches authoring animated football content and our initial goals in trying to meet those needs [22, 23, 19]. We also build upon our previous explorations into interfaces for programming by demonstration [21, 20]. Although many commercial products exist for designing football plays, these products generally assist the coach in automating administrative tasks and building static plays [1] but do not allow coaches to create animated football simulations.

There is little published research pertaining to visual programming environments for the American football domain. The only research system related to ours is Pickering’s COACH system [26, 27]. The primary exploration of Pickering’s research was to develop an animated COACH playbook with an efficient and intuitive interface for inputting rules. Pickering focused on a gestural interface and created a set of distinguishable gestures that map to rule assignments for players. Since an animation language itself was not the subject of the research, no formal treatment of the language was performed. Rather, a large set of rules were developed to cover very specific behaviors. Additionally, the COACH system only permits a single rule to be assigned to a player. This approach requires all behavior to be encapsulated in predefined rules whereas the IFP focuses on creating a basic set of primitive rules that can be combined to describe more complex behavior. We seek to enable coaches to create sophisticated behaviors that were not and could not be specifically envisioned by us when creating the IFP (due to the coaches’ stronger understanding of the domain.) We use a simplified point-mass physics model to simulate the 2D mechanics and interactions of the moving players. Such models have been used for many purposes including the modeling of pedestrian motion as well as modeling flocks, herds, and schools [29, 16]. Pickering also used a similar model in COACH [26].
End-user programming environments that support animation are not new. Good examples of such environments include HANDS [24] and Agentsheets [28]. Although these languages have been designed with end-user programmers in mind, we chose to implement our own visual language environment to support our design goal of allowing the user to express constraints in a visual language familiar to football coaches. Despite not using the languages directly, there are a number of ways we drew from these languages.

“Human-centered Advances for Novice Development of Software” (HANDS) [24], though specifically designed for children, supports a variety of event-oriented 2D simulations. For our design approach for the IFP, we have used Pane’s user centered design process for programming systems which he outlined in his Ph.D. thesis and applied when designing HANDS [25]. These methods and ideas have also grown into the Natural Programming Project [3].

Agentsheets is an end-user, programming by demonstration system [28]. Users can develop their own simulations by creating a grid of interacting “agents”. This approach forces all visual primitives to be constrained to a grid—largely to support implicit spatial notation. Repenning notes the importance of implicit spatial relationships to increase the density of representation. We found the idea of implicit spatial notation intriguing and this lead us to identify spatial aspects of the coaches’ notation. We could not use Agentsheets directly because it lacks continuous representations for implicit spatial relationships, but we found this concept helpful.

In 2000, Cooper et al. presented the Alice programming environment which teaches programming by allowing users to program 3D characters [10]. Although Alice is a suitable tool for programming novices to construct a 3D football scene, it does not satisfy our design goal of allowing coaches express player behavior in a notation similar to the current playbook notation—Alice uses a textual programming language with a drag-and-drop approach to constructing code blocks. Secondly, teaching programming, a principle goal for Alice, is not one of our design goals.

El-Nasr and Smith explored creating football scenarios using the game editor for the commercially available Warcraft III game [12]. Programming in this environment requires that the user define player behavior by associating triggers with a 3D character. A trigger is a list of conditions that when met, cause specific actions to be run. These actions modify the state of the game with the intent of more triggers being fired—
advancing the user-programmed simulation forward in time. As the authors note, this approach requires extensive modification to the game's primitives since the game editor is intended for a military strategy game as opposed to a football scenario game. Though this allows one to make a specific scenario with football-like qualities, it does not advance our goal of allowing the football coaches to create animated scenarios using notation within the football domain.
Chapter 3 – User-Centered Design

We based our design approach on Pane’s user centered design process for programming systems [25]. The steps of Pane’s design process are as follows: 1) identify the target audience, 2) understand the target audience, 3) design the new system, and 4) evaluate the system.

From the outset, we decided to pursue a content authoring tool for football coaches, so our target audience has always been clear. We sought to understand the target audience through observing classroom instruction, interviewing the coaches, analyzing the static football play notation and familiarizing ourselves with the strategies used in football. (See Figure 3.1 for an example of the coaches’ notation.) In contrast to Pane’s approach, we did not perform empirical studies to further understand the domain.

![Example of football formations](image)

Figure 3.1: An example of formations as drawn on a whiteboard during classroom instruction.

We performed steps three and four twice: first with a paper prototype and then with a working implementation. We were highly motivated to find design problems
early in the design process. Our goal was to look for usability problems at a high level before committing the design into source code, so we applied a Cognitive Dimensions analysis [14] on our paper prototype using the CDs Questionnaire [6]. A portion of the early prototype can be seen in Figure 3.2.

![Figure 3.2: A portion of the paper prototype: the offensive player formation editing screen for the Interactive Football Playbook. The user would manipulate a player by dragging it from the toolbox, around the field, or off the field.](image)

Our rapid prototype evaluation process is similar to Cloyd’s process in [9]. Cloyd uses a paper prototype and a cognitive walkthrough [30] to look for task-related usability issues early in the design process. In a similar vein, we use Cognitive Dimensions to look for structural issues with the interface early in the design process.
3.1 Background on Cognitive Dimensions

Cognitive Dimensions is an intellectual usability framework designed for non-HCI experts for evaluating cognitive factors of usability for “information artifacts” of a programming system (or any static or dynamic notational system such as a map or interactive user interface.) Cognitive Dimensions were first formulated by Green in [13] and then expanded and applied to visual programming environments by Green and Petre in [14]. The framework is further detailed and explained in the Cognitive Dimensions tutorial [15].

Cognitive Dimensions (CDs) allows one to consider the usability of a system from a high level. Unlike other HCI techniques which focus on the low-level interactions between the user and the system, CDs represent a vocabulary of terms which can be used to discuss the system under scrutiny at a structural level. Many HCI techniques, such as GOMS [7] and its derivatives, are limited to interactive systems and are used later in the design process, but due to CDs’ focus on structure, CDs may be used with static notations and early in the design process.

The dimensions, which are listed in Table 3.1, are mutually orthogonal concepts with which one may characterize a system (or notation). So, CDs serve as a shared vocabulary for discussing and evaluating a system. Additionally, the CDs can serve as a checklist of usability factors to consider when designing a system. One may even characterize end users needs in terms of the dimensions and create a “preferred profile” across the dimensions—as originally suggested in the 1996 paper [14] and expanded on in Clarke’s work on personas [8]. This profile can serve as a design guide when evaluating tradeoffs that affect the end users.

When using CDs, it is important to remember their role as a broad-brush evaluation and shared vocabulary. CDs are useful for identifying and discussing tradeoffs pertaining to cognitive factors affecting usability. CDs cannot prove the absence of HCI problems, nor do they serve as a replacement for user studies. These pitfalls, among others, are discussed in Dagit et al. [11].
Abstraction types and availability of abstraction mechanisms
Closeness of mapping closeness of representation to domain
Consistency similar semantics are expressed in similar syntactic forms
Diffuseness verbosity of language
Error-proneness notation invites mistakes
Hard mental operations high demand on cognitive resources
Hidden dependencies important links between entities are not visible
Premature commitment constraints on the order of doing things
Progressive evaluation work-to-date can be checked at any time
 Provisionality degree of commitment to actions or marks
Role-expressiveness the purpose of a component is readily inferred
Secondary notation extra information in means other than formal syntax
Visibility resistance to change
Visibility ability to view components easily

Table 3.1: The Cognitive Dimensions as summarized in [15].

3.2 Analysis Process

Both the broad, structural approach and the ability to be used early motivated our decision to use Cognitive Dimensions for our early evaluation mechanism. We created a mockup for the offensive formation editing screen (see Figure 3.2.) A brief written description accompanied the mockup to describe how one interacts with the interface. For example: the players are placed and moved through dragging, the interface is zoomed by dragging the slider, etc. We then used the CDs Questionnaire [6, 5] to perform a formal CDs analysis on the mockup to find problems with the design.

The Cognitive Dimensions Questionnaire [6, 5] is a general-purpose tool to enable end users (as well as designers) to apply the CDs framework to systems. The CDs framework alone is not easily applied by end users since one must be familiar with the framework to apply its concepts and vocabulary. By framing the concepts of CDs into questions, the questionnaire is intended to make the framework accessible to end users for use in evaluating a system.

Although the CDs Questionnaire targets end users, we (as system designers) turned to the questionnaire because it provided us with the advantage of concrete questions to illuminate the abstract descriptions of the dimensions in [14]. Because the questions provide a perspective on Cognitive Dimensions, the questionnaire caused us to consider
aspects of the IFP interface beyond our own interpretation of CDs. This served to increase the breadth of our analysis.

We performed our analysis with the assumption that American football coaches will be our end users. We consider it a problem when the environment imposes an unnecessary or inappropriate cognitive burden on this particular audience. The CDs framework allowed us to analyze the cognitive aspect of our environment in an abstract way (independent of specific users), but we achieved our strongest results when we took into account the tasks and characteristics of American football coaches. Our results are detailed below.

3.3 Insights and Design Maneuvers

Our Cognitive Dimensions analysis identified a number of problems and design considerations. When using CDs, attempting to improve in one dimension often affects other dimensions. These changes are referred to as design maneuvers, and a good discussion of them is found in [15]—including a nice graph of common relationships for tradeoffs in the dimensions. Below, we will summarize our results and identify possible design maneuvers and their implications. Evaluating these tradeoffs is done by thinking carefully about the design goals, in our case, the design goals for the IFP with respect to football coaches (our target audience).

3.3.1 Visibility, Hard Mental Operations and Abstraction

A design goal in the IFP is to provide football coaches with high visibility for all parts of a formation, because in the classroom football coaches draw and discuss football plays in their totality. We supported this design goal by allowing the coach to work with an overview of the formation and then zoom into parts of the formation. Even with this in mind from the beginning, we still found problems with our design by using the CDs Questionnaire.

For example, below are the questions with our answers from the Visibility/Juxtaposition section of the CDs Questionnaire:

**Question 1:** How easy is it to see or find the various parts of the notation while it is being created or changed? Why?
Answer: The user can find all the parts of the notation while being changed because all the players are on the screen simultaneously. A problem occurs when zooming in: players can go off the screen and there is no indication that they are “out there.”

Question 2: What kind of things are more difficult to see or find?
Answer: It is a little difficult to find types of players since there are so many.

Question 3: If you need to compare or combine different parts, can you see them at the same time? If not, why not?
Answer: The user cannot compare two formations at the same time. The user can toggle between the two easily, but cannot have them on-screen at the same time. Also, the user cannot view an offensive formation in relation to a defensive formation.

Likewise, we found problems in the other dimensions by answering the questions for the other dimensions in a similar manner.

Answering the questions served to identify problems, but not until we started to develop remedies did we begin to encounter tradeoffs. For example, the zooming problem, identified from question one, requires the user’s working memory to retain which players have been placed but are not visible (a hard mental operation). One remedy might be to incorporate a bird’s eye view into the UI so the user can see where the zoomed area is in relation to other players. This would increase the visibility of the system and reduce the mental operations, however it introduces new notations which constitute the interface for the bird’s eye view. These notations, although they do not increase the abstraction gradient of the main notation, are abstractions which the user must learn and understand. However, the learning curve may not be significant if the interface is consistent with notation the user already understands.

The problem identified from question three is that the user cannot compare two formations at the same time. The user may toggle between two formations easily, but cannot have more than one formation on-screen at the same time. It became clear in our classroom observations that football coaches need to be able to compare two different formations of the same kind as well as pair an offensive formation with a defensive formation to see the alignment. This creates another hard mental operation
by taxing the user’s working memory since the user must compare a visible formation with another formation stored in their working memory.

In order to support viewing arbitrary formations at the same time, we could allow a user to float a formation in its own window. By floating two formations, they could be compared and edited simultaneously. Though this change allows for juxtapositioning of formations, the change introduces a tradeoff: coaches must learn a new concept, floating a window, and its associated interface (an abstraction representing that concept). Likewise, allowing coaches to see the alignment of two opposing formations could be supported by allowing a defensive formation to be displayed on the layout field while the offensive formation is being edited. Once again, this would require the user to learn the mechanism (another abstraction) for displaying the opposing formation.

### 3.3.2 Hidden Dependencies, Premature Commitment, Error-proneness and Viscosity

Formations are the basis of scenarios, and as such, are abstractions themselves. This creates a more powerful notation than having to create formations of players in each scenario since formations can be defined once and reused. Our design, while seeking to increase expressive power, falls victim to hidden dependencies, premature commitment, error-proneness and high viscosity.

A formation may be modified once and affect all scenarios built on that formation. Since formations are edited separately from scenarios, when a formation changes, there is no indication scenarios will be changed and of the extent of the change; this creates a hidden dependency for the user. If there are many scenarios depending on a certain formation, then the effect of the change can be very large. This also contributes to error-proneness within the IFP since the system leads the user to make changes that could introduce errors. Because formations are the basis of scenarios, the user is forced to think first in terms of formations and then in terms of combining formations into scenarios. This approach can cause a problem if the coaches are developing a new formation in response to a certain scenario. This creates a premature commitment since the coach must decide on an arrangement of players before the scenario is created and may not move players when editing the scenario. Allowing the user to view an opposing formation while editing partially solves this problem since it provides an alignment
mechanism, but that solution still does not allow the coaches to adjust a formation within a scenario.

Our choice to abstract formations from scenarios also creates viscosity problems. The formation editor itself demonstrates low viscosity—one simply drags a player around to modify the formation—but the inability to change formations within scenarios creates a more viscous notation for scenarios. We could attempt to lower viscosity by allowing the user to move players within the scenario editing environment, but that raises the question of the effect of such a change. When a formation is modified within a scenario, it could either update the formation in all scenarios (globally) or only in the current scenario (locally). A global change creates an especially dangerous hidden dependency because the environment seems to suggest changes to one scenario will not affect others (there is no visible indication of change), but a local change weakens the power of the abstraction—editing a formation via the formation editor may not update all the places where that formation is used.

Our CD’s analysis brought out these design tradeoffs: (1) abstracting out formations gives the notation more power, but creates a hidden dependency; (2) requiring the user to create a formation before a scenario creates premature commitment, but removing this requirement prevents us from abstracting out formations (3) allowing a user to modify a formation within the scenario editor makes the notation less viscous, but creates a hidden dependency; and (4) allowing the user to customize a formation within the scenario editor makes the notation less viscous but weakens the power of the abstraction. We resolved (1) and (2) by considering our design goals, but (3) and (4) will require more observations or interviews to find out to what extent the coaches will create or adjust new formations within a scenario.

3.3.3 Provisionality and Progressive Evaluation

The formation editor exhibits high provisionality in the sense that the user may edit the scenario in an ad-hoc manner without having to commit to a player arrangement until the user determines the formation to be complete. The user may “play” with the formation to sketch out an idea. The user can re-arrange players at any time and see the result. However, the formation editor does not allow the user to make formations less precise. To support less precision, the environment could incorporate notation to
indicate that a player is positioned in an approximate location instead of an exact one. Once again, this increases the abstraction gradient by introducing a new notation. By making the less-precise notation optional, the user would not be forced to use (or even learn) the new notation.

There is a sense of progressive evaluation as a formation is constructed since there is clear visual progress as the players are positioned in the formation. However, the user may not incorporate partial formations into a scenario. So, while the user may create a partial set of rules within a scenario and evaluate them, the user cannot extend this idea by having partial sets of players. Removing this barrier would allow coaches to progressively develop formations within an associated context of a scenario. That change would also serve to increase the provisionality of the system since the user would not have to create formations first and then scenarios.

Extending provisionality and progressive evaluation in the IFP may allow for more opportunistic development, but it could result in hidden dependencies between the formations and scenarios. From our interviews and classroom observations, it is not clear if coaches need to develop formations in an ad-hoc manner. In the classroom, we have primarily observed coaches working from a canonical list of formations, but the CDs analysis pointed out to us that we need to observe the coaches doing new formation development as well, to obtain the necessary insights into the formation development process.

3.3.4 Whoops, No Secondary Notation!

The notation in the IFP is modeled after the notation coaches already use in their playbooks, providing good closeness of mapping. This source also leads to good role-expressiveness, since the formation in the IFP looks like a formation in the coach’s printed playbook. This representation also exhibits low diffuseness (is compact) since there is one symbol for one player and each kind of player takes the same amount of space to represent. And, although there are different kinds of players, the notation maintains consistency by having players retain the same shape and use different labels. Thus, these particular CDs make the design look very good.

An important advantage in CD analysis is that CDs provide a specific list of concepts to consider, so that important concepts do not slip by without being considered. In our
case, this is exactly what happened. The CDs pointed out to us that there is a complete absence of secondary notation within the system. We do not provide an escape from formalism: a way for one to make notes to oneself or add informal information within the environment. In real life, coaches write notes all throughout their printed playbooks, so the environment should support some form of commenting or augmentation attached to different players, formations, and scenarios. For example, this would allow a coach to include the goals and tasks for a player in a particular formation.

3.4 Evaluation Summary

The CDs framework enabled us to identify a number of problems and tradeoffs very early in the design process for the IFP. Identifying the tradeoffs early on helped guide our design process and steer us away from problems that would inhibit our design goals. We used the CDs questionnaire to provide us with a concrete list of things to check and consider about our design. This approach allowed us to find things we had failed to consider at all. In particular, it was our experience that the questionnaire achieved its intended purpose of being accessible: even those in our group who had low levels of familiarity with CDs were able to identify problems with the IFP by answering the questionnaire.
After evaluating a paper prototype, we designed and programmed a working prototype of the Interactive Football Playbook (IFP). This was our second iteration of step three ("design the new system") of Pane's four step user centered design process for programming systems [25]. Our implementation of the IFP is detailed in this chapter and our evaluation of the implementation (step four) is detailed in Chapter 5. We were able to incorporate many design considerations brought to light by our Cognitive Dimensions evaluation, but due to the scale of our implementation, we were not able to implement all features that we determined would aid the usability of the IFP.

A major design goal of the IFP is that it should enable coaches to create animated content by programming simulations using familiar notation. The notation within the IFP is drawn from notation coaches already use in their playbooks. This notation is augmented by additional information to allow coaches to control the animated player behavior over time. We will refer to synthetic animated agents as players. Since play notation is declarative in nature ("the wide receiver runs here" or the "defensive tackle blocks him"), we chose to implement a constraint-based, visual programming system. The coach associates rules (constraints) with players and spatial locations and the player behavior is determined by unifying those rules over time.

The player motion that results from the rules and interaction with other players makes up the player’s "performance". The set of rules associated with a player defines its "behavior". Our goal is to enable football coaches to program player behaviors in such a way that they can achieve a desired performance.

Defining player behavior in the IFP is not simply scripting out players performances, but declaring what we would like the player to do. This will cause the performances of players to respond as the players interact with each other, whereas with a script of animation commands, the players will not be affected by other players. Using a simple animation script approach creates a more rigid system where the coach is responsible for changing all the other players behavior in response to the changes to one player.
With constraints, the intent is expressed while allowing the unconstrained details of the actual performance to change in response to how the players interact with one another.

4.1 Visual Language Environment

During our interviews with the coaches and our classroom observations, we discovered that coaches often use the same arrangement of players on the field in a number of different instances, but what a player is expected to do depends on a number of variables such as the opposing team’s formation or the strategy for the current play. We decided to allow the coaches to define these “formations” once and then reuse them in a number of different “scenarios” where each scenario expresses the different desired variations. This design decision led to the creation of a “formation editor” and “scenario editor”.

The formation editor allows the user to physically layout and name an arrangement of players (Figure 4.1.) The players are dragged onto, around, and off of the field using direct manipulation. Grid lines and field markings may be displayed to assist the user in properly placing the players. Players snap to a grid to assist the user in aligning the players. Multiple formations can be edited simultaneously and switched between using tabs at the left of the window. Currently, an opposing formation may not be displayed opposite the formation being edited and two formations may not be juxtaposed for comparison.

After using the formation editor to define an offensive and defensive formation, the user may combine them into a scenario. Then, using the scenario editor (Figure 4.2), the user may associate rules with players and spatial locations and execute the scenario to see the resulting animation. To associate a rule with a player, a rule is selected from the list of rules on the left. A mouse dragging gesture is used to designate the starting and ending targets for the rule.

After laying out the players and specifying at least one rule for one player, users may run the resulting simulation producing animated motion. The user presses a play button to start the simulation. The simulation executes and the players animate around the 2D playing field according to the defined rules. The user may pause or un-pause the simulation, step it forward one time step, or stop and reset it. When stopped, the user may assign more rules to the players. See Figure 4.3 for an example of the animated motion.
4.2 Language Definition

The language is modeled after the notation coaches already use in their playbooks and classroom demonstrations. Presently, the language supports a minimal set of rules sufficient to generate a large number of plays: “run a route”, “block”, “guard”, “avoid”, and “pursue”. These rules have pure pictographic representations (Table 4.1), so the parameters of the rule are modified through direct manipulation. Future research may explore other input modalities.

4.2.1 Sequencing and Parallel Execution

Rules may be sequenced through chaining as seen in Figure 4.4. In the first case, the running back ("RB") is instructed to run a short route and then block the linebacker ("LB"). In the second case, the linebacker is instructed to run around the tight end ("TE") and then pursue the quarterback ("QB").
Figure 4.2: The scenario editor. Rules are associated with the players as well as spatial locations. The possible rule types are displayed on the left.

Though any rule may terminate a sequence, only the “route” and “block” rules may be used in the middle of a sequence. This is indicated by the open square at the end of the rule. In the current model, we have not found a need for chaining rules onto the end of either the “guard”, “avoid” or “pursue” rules since these behaviors are intended to apply for the duration of the play once they are invoked.

In addition to sequenced rules, multiple rules may be in effect at the same time for a single player. For example, a wide receiver (“WR”) may be instructed to avoid a corner back (“CB”) while running a route. Or, a WR may run a leg of the route and then be instructed to avoid a player while running the second leg of the route. As described in the implementation section (4.4), the performance of a player is determined by the forces applied to the player due to the rules associated with the player. Not all combinations make sense. For example, running a route and trying to block a player at the same time is unlikely, however the decision about what is desirable behavior is left to the user.
Figure 4.3: A fragment of a scenario. At the top left are the rules as specified by the user and the resulting animation is seen from left to right down the page.

Table 4.1: Visual representations of the rules.
4.2.2 Players

A player is the most fundamental primitive of the language. Every player has the following attributes: side (offense or defense), position (such as quarterback, defensive end, etc), location on the field, and maximum velocity. During animation, the player’s velocity vector is also stored as a property so that it can be used in rule calculations.

Offensive players are depicted with circles and defensive players are depicted with squares. The position of the player is displayed on the player’s icon using a common abbreviation for that position’s name. The player’s location on the field is represented spatially as in a standard play diagram (Figure 4.1.) The user is responsible for creating these formations, although we pre-loaded several standard formations into the IFP. Within the formation editor, the player icons snap to a grid allowing the user to accurately align the players. The maximum velocity of a player is not currently visualized.

4.2.3 Route

The route rule instructs a player to move along a user defined path and to try to stay on that path. Routes may be chained to form composite routes which simply act as one long route. Additionally, routes allow any other rule to be chained to the end of the route. This is useful for setting up a player’s location on the field before dispatching
the player to perform some other behavior.

A route rule has two parameters: 1) the player running the route and 2) the path the represents the route to be run. The path is a freehand line drawn on the canvas—this creates a very flexible notion of a route. The shape of the route and the interpretation of the shape is left up to the user. For example, the entire category of routes in COACH [26] may be expressed using our one route primitive, and new kinds of routes can be created by the coaches as needed without modification of the IFP. Internally, the path is represented by a piecewise linear function. The player’s position is tracked along the path using a marker on the path. Because external forces may move the player off the path (eg. another player blocking the player), the players actual movement must be mapped back onto the path to determine the progress along the path. This mapping is achieved by projecting the player’s movement vector onto the vector of ideal movement along the path. The resulting vector determines how far forward the player moved along the route (if at all), and the marker is updated. In the instance where the player makes no forward progress, the marker will not be moved backward along a route, but will remain as a goal for the player to move toward.

If forces push players away from the path, a correction vector is added to the ideal movement vector to pull the player back toward the path. The magnitude of this vector is small so as to not interfere with other rules that may be acting on the player, such as the avoid rule (Section 4.2.5).

4.2.4 Block

The block rule instructs a player to impede an opponent’s progress across a directional boundary. The angle of the boundary is shown by the perpendicular line at the end of the blocking rule.

The block rule has four parameters: 1) the player doing the blocking, 2) the player being blocked, 3) the blocking boundary angle and 4) the duration of the block. The angle is fixed based on the initial orientation of the block rule which is determined by the starting point of the rule and the initial location of the opponent. The duration of the block is not currently customizable. A player will continue to block indefinitely unless another rule is chained onto the block. If a rule is chained onto a block, the block will last for 0.5 seconds.
The player performing the blocking will shadow the blocked player with respect to the blocking boundary and will push the blocked player to keep the player from crossing the boundary. The block rule projects the velocity vector of the blocked player onto the blocking boundary. This determines the blocked player’s motion with respect to the boundary. The block rule mirrors this motion for the blocker and adds an adjustment vector that moves the blocker toward the other player.

4.2.5 Avoid

The avoid rule instructs a player to attempt to adjust his path in order to avoid contact with another player. This rule is generally applied to wide receivers and often to defensive linemen. The rule gives the coach the ability to specify that a player is to get around another player but to do so without having to specify the entire route that avoids the other player.

The avoid rule has three parameters: 1) the player doing the avoiding, 2) the player being avoided and 3) the minimum distance to maintain. The side on which to avoid is currently not customizable. As we will discuss in Chapter 5, a leverage rule could be used to designate a desired side on which to avoid.

When applying the avoid rule, the distance between the avoider and the avoided is calculated. If the avoider is outside this distance, no force is applied to the player. If the avoider is within the minimum distance, the avoid rule constructs a vector in the direction away from the avoided.

4.2.6 Guard

The guard rule instructs a defensive player to maintain a specific relative position with respect to an offensive player. This rule is generally applied to defensive linebackers whose job is to guard offensive receivers.

The guard rule has three parameters: 1) the player doing the guarding 2) the player being guarded and 3) the distance to maintain from the player. The desired location with respect to the player being guarded is currently not customizable. Rather than parameterize this rule with an inside or outside bias, it could be combined with a leverage rule to more precisely specify the bias of the guarding player. (See discussion in
Chapter 5.) Although not currently implemented, the guard rule will also be parameterized by how “tight” or “loose” to guard. These terms are generally used to define how much vertical space (along the length of the field) is allowed between the receiver being guarded and the defender doing the guarding.

The guard rule vector is formed by locating a desired position with respect to the player being guarded. Currently, the desired position defaults to be horizontally offset (along the width of the field) two yards to the right or left of the guarded player. Once the desired position is identified, a vector from the guarding player to that desired position is computed and clamped to maximum velocity.

4.2.7 Pursue

The pursue rule instructs a player to “run after” another player. In football, this rule is generally given to defensive linemen whose main goal is to catch and tackle the quarterback.

The pursue rule has two parameters: 1) the player doing the pursuing and 2) the player being pursued. As we will discuss in Chapter 5, this rule could be augmented with a leverage rule to constrain the pursuit.

The pursue rule is implemented as a simple vector computation. A vector is constructed with the difference between the position of the pursuer and the pursued. This vector points from the pursuer to the purse and is clamped to a maximum velocity.

4.3 Limitations of the Language

The IFP does not prevent the user from constructing a scenario that violates the rules of football or a scenario that does not make sense from a strategy perspective. We leave it to the content creators to program behaviors that elicit appropriate performances for their visualization needs.

The domain specific visual language for the IFP is not intended to be Turing complete or appropriate for general purpose programming. Specifically, there are no mechanisms for conditionals, named abstraction, or iteration. We would like to explore these features, in particular, as we continue our research to allow coaches to reuse behavior between scenarios.
4.4 Implementation

Figure 4.5 depicts an overview of the Interactive Football Playbook's execution model. The model consists of an animation engine, a unification engine, a structured representation of the rules, a structured representation of the scene, and a view of the scene. The unification engine examines the rules and sends a vector for each player to the animation engine. The animation engine enforces physical constraints and then updates the scene. The view is then rendered from the contents of the scene.

4.4.1 Internal Representation

The IFP uses an object oriented data model approach for the internal representation. The players, rules and scenarios are modeled as their own classes with data members to store the properties. A scenario contains a list of rules. Each rule is associated with a source player and a target player except the route rule which only has one player associated with the rule. Within the rules, players are referenced using unique ids which are resolved at runtime—this prevents the need to bind references to player objects until the animation engine creates the objects for the animated players at runtime. Currently, rules may not be defined in a global manner or be reused from other scenarios. This problem of rule generalization is left to future research.

4.4.2 Unification

Execution is performed by unifying the rules and sending the results to the animation engine. (See Figure 4.5 for an overview of the execution model.) At run-time, the rules
are unified forward one step in time. Each rule has a class that defines its implementation. The program code for the rule determines the solution for the next step in time while satisfying the rule constraints. This code examines the parameters of the rules as well as the state of any relevant players and then generates a 2D floating point vector representing the direction and magnitude the player should move in response to the rule. A solution to satisfy each rule is independently calculated, and a player’s movement vector is the sum of all the solution vectors for the rules associated with that player. Scaling is applied when necessary to constrain the vector magnitudes to maximum velocities.

An example of the vector summation can be seen in Figure 4.6. The rules input by the user are shown in grey. The cornerback (“CB”) is directed to pursue the wide receiver (“WR”) and the wide receiver is directed to follow the given route and avoid the cornerback. The grey vector pointing up was generated by the route rule to keep the wide receiver on the user defined path. The grey vector pointing down to the right was generated by the avoid rule to keep the wide receiver from being too near the cornerback. The black vector is the result of adding the two vectors together. This produces motion that is not strictly along the path, nor strictly away from the cornerback, but something in-between.

Figure 4.7 shows a more complete sample of the animation shown in Figure 4.6. Notice how the wide receiver drifts to the right of the path to avoid the cornerback. Eventually, after going around the cornerback, the wide receiver is drawn toward the original path because the avoid vector no longer pushes the wide receiver away from the path.

Our unification algorithm only looks one step forward in time; it does not search forward through many time steps to find a solution that satisfies the rules optimally. We do not want the players to act on too much foresight within a scenario and thus exhibit unrealistic performances. Our approach results in run-states that are not optimum constraint satisfactions, but in the opinions of the coaches, do look reasonably realistic. Generally speaking, we employ soft-constraints with a very limited lookahead.
Figure 4.6: An example of the summation of vectors for the wide receiver (“WR”). The WR is attempting to avoid the CB while running a pass route and the CB is attempting to guard the receiver. The resultant unified vector is shown in black. (The vectors in the figure are approximations.)

4.4.3 Animation

The animation engine is responsible for taking the unified direction for each player, enforcing physical constraints (such as collision), and then updating the scene. After receiving a movement vector for each player, the animation engine translates all the players and performs a pairwise search to find collisions between players. The locations of collided players are adjusted using a standard 2D collision-response. The response is weighted by the magnitude of the collision component of each player involved. For example, if two players collide, the player with the weaker force in the collision will be pushed in the direction opposite the force of the collision.

We use a similar approach to Pickering's approach in COACH [26, 27]. Like Pickering, we use vectors to determine the intended direction of the player and a collision-response to prevent players from moving on top of each other. Unlike Pickering, we allow more than one rule to be associated with a player and the total effect for the
Figure 4.7: Approximate resultant vectors for each player over time. The user specified rules are seen in grey.
player is determined by the sum of the vectors.

Rendering the animation is performed through a model-view-controller (MVC) approach [18]. The animation engine creates objects for each of the players (the scene), resolves the player ids to references, and then passes the references for the rules and the players to the unification engine. The unification engine uses the rules and the player state information to produce the unified vectors for the animation engine. When the animation engine updates the scene, the scene fires event notifications to the view which updates the screen. MVC simplified the implementation because the animation engine only needs to modify the player objects without concern for any graphical representation. It was simple for us to create alternate views of the scene to support our development of the IFP.

4.4.4 Platform

We implemented the IFP in Java using the Piccolo structured graphics framework [4]. We chose Java because its cross-platform nature allows us to deploy it in a heterogeneous environment and Java Web Start allows us to push new versions of the IFP to the football coaches. We chose Piccolo because the standard Java API does not have a 2D structured graphics framework, and we can still retain access to high-performance 3D bindings for future research needs.
Chapter 5 – Results

After creating a working prototype, we interviewed the football coaches at Oregon State University for an initial evaluation of the prototype. This was our second iteration of step four (“evaluate the system”) of Pane’s four step user centered design process for programmings systems [25]. Due to the prototype nature of the IFP, we have not yet performed a thorough user study. The feedback we received from the coaches during our interviews is detailed in this chapter.

5.1 Demonstration Feedback

Overall, the response was positive. The coaches’ initial reaction was excitement. The coaches quickly recruited other coaches to come see the demo and comment on the tool. When asked about the behavior of the players in response to the rules, they said that it looked good—for example the blocking delay “felt about right” (about two counts). Interestingly, they seemed more concerned with what could be expressed with the language than the animation of the players. They seemed to think the current animation was satisfactory.

From our discussions, we were able to determine how well we had captured the ability to specify real plays as well as identify some of the missing language elements. Like our initial information gathering interviews, the coaches reiterated the problem of finding video of an ideal demonstration of player behavior and were optimistic that a computer animated demonstration could be a suitable means for visualizing proper player behavior. They understood the goal of authoring and found the resulting performances to be compelling.

5.2 Identified Language Needs

After creating and viewing several scenarios, the coaches began to identify and articulate what types of things they would like to be able to do, but was not possible with our
current language. We identified several new rules and desired parameterizations of the existing rules.

**Leverage:** This rule allows the coach to specify a desired horizontal (width of the field) relative position of a player with respect to an opponent. This rule could be combined, for example, with the guard rule to further specify on which side to guard the offensive player.

**Reckless:** This rule is given to a defensive lineman that is to pursue the football without any regard to leverage. In other words, this is a default pursuit behavior. A more controlled pursuit would be obtained by combining “pursue” with the “leverage” rule. Although the default behavior of our pursue rule is essentially “reckless”, we must provide interface elements to distinguish between reckless pursuit and pursuit with leverage constraints.

**Wait:** This rule specifies that a player is to remain in its general start position. It would be parameterized by a time or a condition. For example, wait for a count of 2 or wait until a lineman is within two yards.

**Throw:** This rule allows the coach to designate when the ball is thrown and to which offensive player it is directed.

**Handoff:** Similar to “throw,” this rule designates when the ball should be handed off and to which offensive player it should be handed—typically a running back.

**Zone:** This rule involves the specification of areas of the field that must be guarded by a defensive player as opposed to guarding particular offensive players. Although this rule is designed around an area as opposed to a particular player, the coaches expressed several ways in which a defender must maintain the zone while also guarding players within and near that zone. This rule is clearly complex and will require more discussion with the coaching staff.

**Annotation:** The coaches would like to annotate players and plays with supporting textual information (a secondary notation.) For example, the coach might want to label a particular defensive position with the actual name and tendencies of a player from an opposing team.
After our most recent interview, the coaches requested an installation of the IFP software. The coaches felt the tool had enough potential that they offered to help us build up a more complete playbook. We will continue to follow up with the coaches to collect feedback as they continue to use the IFP.
Chapter 6 – Discussion

We have presented a language designed to allow coaches to program a football simulation to produce a desired performance. The language was designed with the target users in mind from the beginning. As a result, it is based heavily on existing symbols and language used by American football coaches. This language and these symbols are used much like the coaches would use them to communicate directly with real athletes. In this case, the coaches are directing virtual 2D players to perform the plays.

The interviews with the football coaches identified the rule modifications listed in Chapter 5. In addition to these modifications, the discussions resulted in other key observations about the language. On several occasions the coaching staff mentioned the need to specify conditional behavior. For example, they wanted to program a particular player to exhibit a certain performance in one situation while exhibiting another performance in a different situation. This authoring effect can be achieved in two ways. One approach is to include a conditional rule to allow the coaches to specify behaviors that are selected based on the authored behavior of other players. For example, the receiver may run one route if the corresponding defender is authored to have inside leverage and run another route if the corresponding defender is authored to take outside leverage.

An alternative approach is to make it convenient for coaches to author the two variations of the scenario and save them out as different scenarios. In this approach, the coach would simply author the behavior for one scenario and save it. The coach would then modify that same scenario to reflect the behavior for the second option and save it as well.

In our implementation we avoid conditional rules since we are focusing primarily on authoring for a performance. Authoring for a specific performance is different from authoring for a reusable, abstract behavior. However, using a copy-modify approach instead of conditionals leads to having many different scenarios that are only slight variations on each other. Additionally, the coaches could benefit from rule combinations that generalized from one scenario to another. With generalization, the coaches would not
be required to start from scratch in each scenario. The players could behave according to a generalized behavior. Generalization would almost certainly require conditionals since the coaches would need to identify the important features of a situation that distinguish it from other situations. We may be able to support generalization without the user specifying conditionals by using machine learning techniques that identify specific situations and “learn” the conditionals.

Currently, our system allows the user to specify rules at \( time = 0 \) seconds: when the players have not yet moved. From our interaction with the interface and from our discussion with the coaches, it seems more intuitive to allow the specification of rules at any point in time. This is partly due to the iterative nature of specifying a performance. From experience, we have learned that to get a particular performance, the best workflow is to apply rules to subsets of players at a time and then review the motion. While reviewing the motion given the current set of rules, one may find that another rule should be applied to a particular player in parallel or applied in sequence at a particular point in time. Currently, the user may only insert rules at well-defined chaining points (the empty box at the end of a rule) and only when the players are in their initial positions. The ability to insert rules in parallel or sequence at any point during the animation adds additional complexity to rule specification and unification. Adding this capability is left as future work.

The current implementation does not provide a means for the user to specify parameter values such as strength, speed or delay. Parameterization, however, will play an important role in expressiveness. For example, a player should be parameterized by strength and speed so that the coaches can produce a scenario where one player overpowers another or outruns another. As another example, a coach may tell a player to “block” for 2 counts before running a route, therefore timing parameterization is necessary. The interfaces for these kind of parameters is left as future work.

The current IFP notation provides a fair amount of visual noise and no means for filtering. We plan to explore visualization techniques to help the coaches wade through the visual clutter to understand why a player’s performance is evolving as it is. To do so, coaches should be provided with tools for filtering, obtaining rapid feedback of the effect of changes, and understanding the sequence of events that caused a particular performance. Our goal is to allow coaches to focus on authoring content not debugging constraints. An example of such tools for supporting authoring can be seen in the work
of Ko and Myers [17].
Chapter 7 – Conclusion

The Interactive Football Playbook allows American football coaches to create digital playbooks with animated content. Coaches program the virtual players through a constraint-based, visual programming language which is modeled after the notation coaches already use in their playbooks and classroom demonstrations. The language was developed through iterative use of Pane’s four step user centered design process for programming systems. The system has been evaluated both through a formal Cognitive Dimensions evaluation and through an informal review by the Oregon State University football coaches themselves. Through continued interaction with the coaches, we hope to develop a richer language for specifying player behavior, a more supportive user interface, and visualization tools to aid the coaches in understanding and refining player behavior.
Bibliography


