Journal of Visual Languages and Computing 20 (2009) 16–29

Contents lists available at ScienceDirect

Journal of Visual Languages and Computing

journal homepage: www.elsevier.com/locate/jvlc

End-user strategy programming

Christoph Neumann *, Ronald A. Metoyer, Margaret Burnett

School of Electrical Engineering and Compurer Science, Oregon State University, Corvallis, OR, USA

ARTICLE INFO

Article history:
Received 20 September 2006
Received in revised form
20 March 2008
Accepted 28 April 2008

Keywords:
End-user programming
Natural programming
Visual programming
Computer animation
American football

ABSTRACT

Rule-based programming systems can be fragile because they force the user to account for all logical alternatives. If an unconsidered case does arise during execution, program behavior falls through the cracks into unspecified behavior. We investigate rule-based, end-user strategy programming by introducing our Interactive Football Playbook—a domain specific, end-user programming environment to allow American football coaches to create animated football scenarios by associating strategy information with virtual football players. We address the problem of rule explosion through "rule bending" to support a minimalist, scaffolding-driven programming environment. Additionally, we introduce visual language representations for logical and sequential "and" to mitigate end-user confusion with the semantic meaning of these "and" constructs.

1. Introduction

Computer generated content is richer than ever before, taking advantage of the greater capabilities of modern hardware. Scientific visualization experts can program complex software which generates visual, interactive content for users from data. Professional animators can use general purpose animation tools to create a vast array of content from instructional visualizations to life-like scenes for motion pictures. As the complexity of content increases, so do the challenges for content authors; the divide between the content creator and the content consumer grows.

Content consumers who want to bridge the gap and create content of their own are faced with a significant learning curve to get up to speed with readily available, general purpose content authoring tools. Lacking programming skills, an individual is limited to the domain-agnostic environment of the chosen content authoring tool (spreadsheet, animation tool, etc.), so the individual must learn a tool more abstract than the needed domain without supportive features relevant to the domain. This problem is compounded by the user's need to create content quickly and update it often. For example, an animation created using an animation tool is expressed at a low level—a set of concrete property changes over time (e.g. x–y location, orientation, scale, etc.) for a particular object being animated. Each of these points in time is termed a "keyframe" and properties are interpolated between keyframes in the "inbetween" frames. So a change to one object in part of the animation may result in large number of cascading changes to keyframes for other objects to keep the animation looking physically correct.

A simulation approach to animation allows a user to create animated content at a more abstract level than specifying concrete property changes over time. Rather than specifying desired properties for specific objects at points in time, the values of particular properties are determined by the simulation engine. This approach is used when there are many objects which are animated simultaneously and may have complex interactions between them. An example of such are particle systems which are used to model water, fur, and smoke. An end

* Corresponding author. Tel.: +1 541 758 4624.
E-mail addresses: christoph@neumannhaus.com (C. Neumann), metoyer@eecs.oregonstate.edu (R.A. Metoyer), burnett@eecs.oregonstate.edu (M. Burnett).

1045-926X/$ - see front matter © 2008 Published by Elsevier Ltd.
doi:10.1016/j.jvlc.2008.04.005

user could use extension mechanisms, such as macros or plugins, to customize a keyframe-oriented, general purpose animation environment to add domain-specific abstractions to support simulation-oriented authoring. However, developing macros and plugins often requires the user to learn another programming language (likely a scripting language) which represents information quite differently than the content authoring environment.

The research community has turned to visual programming languages to lower the barriers for end users to create domain-specific, interactive, animated content. Novices can use a visualization tool such as OpenDX [1] (a visual dataflow language) to create their own interactive visualizations that would otherwise require a strong background in a general purpose language such as C and a graphics API such as OpenGL. Agentsheets [2,3] allows end-user programmers to create domain-specific visual abstractions and use them to create simulation oriented, animated content. We believe rule-oriented visual programming environments hold promise for allowing end users to create computation-driven content such as what is required for visualization or animation.

We explore end-user creation of rule-based, computation-driven, animated content through our Interactive Football Playbook (IFP). The IFP is a simulation-driven, strategy-oriented approach to allow coaches to program animated football simulations. Fortunately, football coaches rely on a fairly standard symbolic language for specifying plays on static media such as paper or a whiteboard. The IFP builds on this standardized symbolic language and augments it with primitives and language enhancements that allow the coaches to specify strategy information for virtual players. Animations are then rendered by executing the programmed scenario and visualizing the parameters for the virtual players in the scene (Fig. 1).

Our approach is novel in a number of ways. Firstly, we contribute an approach for end-user strategy programming. Our approach differs from prior end-user programming work because we associate high-level, *strategy-oriented* rules with a virtual agent rather than using global, imperative rules to modify the *state* of the agents. The coaches' current use of static diagrams involves formulating a strategy in the coach's mind, translating that strategy into a drawn diagram, and then using the diagram as an aid to communicate the strategy to the football players. Only simple parts of the overall strategy are captured in the static diagram notation. Our approach elevates the semantic level of the captured information by allowing coaches to express more strategy information in the IFP than in the static playbook notation. For example, the coaches can use our notation to specify distance and sequencing information whereas in a playbook, that behavior is described using English. This richer language allows the IFP to generate animated simulations rather than simply representing static diagrams. Effectively, coaches author by using strategic concepts to create rather than simply scripting out animation actions. The outcome is more than a set of fixed animations; it is a repertoire of executable scenarios which exhibit specific strategies responding to the states encountered in the scenarios.

Our second contribution is our novel approach to rule bending. Like Repenning's work [4,5], we address the issue of rule explosion, but Repenning focuses on managing discrete permutations of rules. We apply rule bending in a continuous fashion.

We also contribute a notational device to express parallel and sequential "and" within a visual programming environment. Boolean expressions have been notoriously problematic for end users [6], so our device allows the user to express sequential and parallel "and" without the confusion that stems from using the overloaded English word "and". Prior work focused on logical "and" used in conditions for selection, but not on "and" used with regard to execution flow.

Finally, we contribute further evidence of the usefulness of the Natural Programming design process [7,8]. We successfully used the Natural Programming design process for creating the IFP and found the process to be

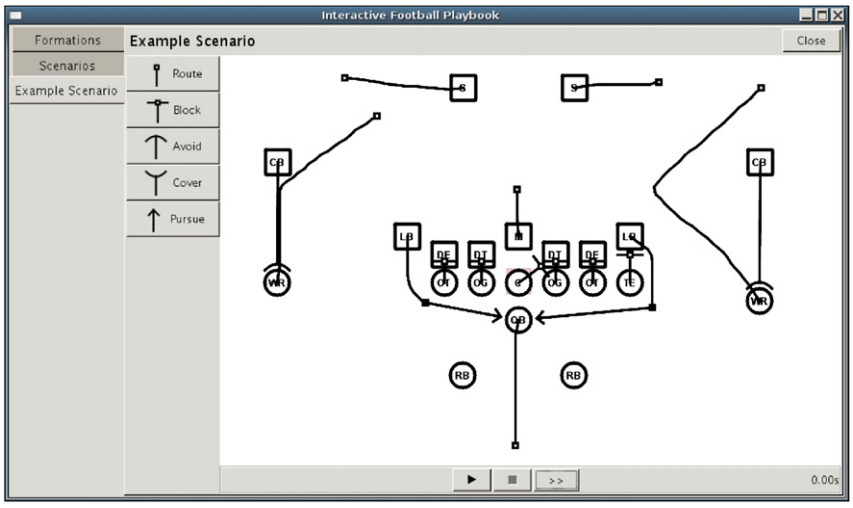

Fig. 1. A screenshot of the Interactive Football Playbook.

helpful. The process helped shaped our design decisions for the visual programming notation itself as well as the characteristics of the programming environment.

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 [9–11]. We also build upon our previous explorations into interfaces for programming by demonstration [12,13]. Although many commercial products exist for designing football plays, these products generally assist the coach in automating administrative tasks and building static plays [14] but do not allow coaches to create animated football simulations.

We extend previous agent-oriented programming approaches by allowing end users to associate strategy notation with agents via a domain specific visual programming notation. The RoboCode project [15] allows the user to associate strategy behavior with virtual agents in a 2D simulated environment, but the strategy is defined by Java code and the RoboCode tool is intended to be pedagogical. Likewise, the Alice programming environment [16] allows the user to associate behavior with a 3D agent, but Alice focuses on using textual code to teach programming and not on strategy programming for the virtual agents. We take this idea of agent centered programming, focus it on strategy, and make it accessible to end users.

End-user, visual programming environments that support simulation-driven animation are not new. Good examples of such environments include HANDS [17] and Agentsheets [2]. "Human-centered Advances for Novice Development of Software" (HANDS) [17], though specifically designed for children, supports a variety of event-oriented 2D simulations. For our design approach for the IFP, we used the Natural Programming design process [7] which Pane first outlined in his Ph.D. thesis [8] and applied when designing HANDS. Our use of the process is detailed in Section 3.

Agentsheets is an end-user, programming by example system [2]. 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.

Additionally, Repenning developed the idea of "bending the rules" [4,5] within Agentsheets' rule system. Agentsheets employs graphical rewrite rules to define agent behavior. Due to the demonstrational aspect of graphical rewrite rules, an example must be given for every circumstance the agent may encounter. This leads to a rule explosion because all possibilities must be defined explicitly. Repenning employed two higher-order constructs, rule bending and connectivity, to allow the user to generalize a rule to symmetric and similar cases. In this sense, the demonstrated rule is not being used solely as demonstrated, but is being "bent" to work in new situations. We extend the idea of rule bending into the continuous domain and use bending to avoid the fragility associated with rule-oriented approaches.

We use a physical model to generate animation from constraints defined by the rules associated with the virtual agents. Witkin and Kass were one of the first to apply constraints to generate animation [18]. Their optimization-driven approach uses a physical model of the object being animated, keyframe positions, and optimization preferences to create a physically realistic animation which passes through all of the keyframe positions. Myers et al. bring constraint-driven animation to an end-user environment by developing a toolkit (Amulet) which allows the user to declaratively specify animation effects [19]. Both of these approaches satisfy hard constraints whereas we use soft constraints which permit imperfect satisfaction to support our rule bending approach.

There is some published research pertaining to visual programming environments for the American football domain. Casner examined the semantics of football play diagrams to demonstrate how those diagrams could be represented within his general purpose diagramming tool, BOZ [20]. The semantic structure for our visual programming language in the IFP is similar to the structure identified by Casner. Also, Pickering's COACH system [21,22] is similar to the IFP. The primary exploration of Pickering's research was to develop an animated COACH playbook with an efficient and intuitive interface for entering 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 [23,24]. Pickering also used a similar model in COACH [21].

El-Nasr and Smith take a novel approach for creating football scenarios by using the game editor for the commercially available Warcraft III game [25]. 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.

3. Design process

We based our design approach on the Natural Programming design process for programming systems [7,8]. The Natural Programming design process applies the principles of user-centered design [26] to the design of programming languages for the purpose of treating usability as a first-class design objective to avoid subordinating usability in favor of historical convention, designer preference, or theoretical elegance. The steps of the Natural Programming design process are as follows: (1) identify the target audience, (2) understand the target audience's language, techniques, and thinking for problem solving, (3) design the new system, and (4) evaluate the system.

Fig. 2. An example of formations as drawn on a whiteboard during classroom instruction.

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 Fig. 2 for an example of the coaches' notation.) In contrast, Pane used empirical studies to further understand the domain when designing HANDS [8].

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 [27] on our paper prototype using the CDs Questionnaire [28]. A portion of the early prototype can be seen in Fig. 3. Our analysis is detailed in Dagit et al. [29].

Our rapid prototype evaluation process is similar to Cloyd's process [30]. Cloyd uses a paper prototype and a cognitive walkthrough [31] 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.

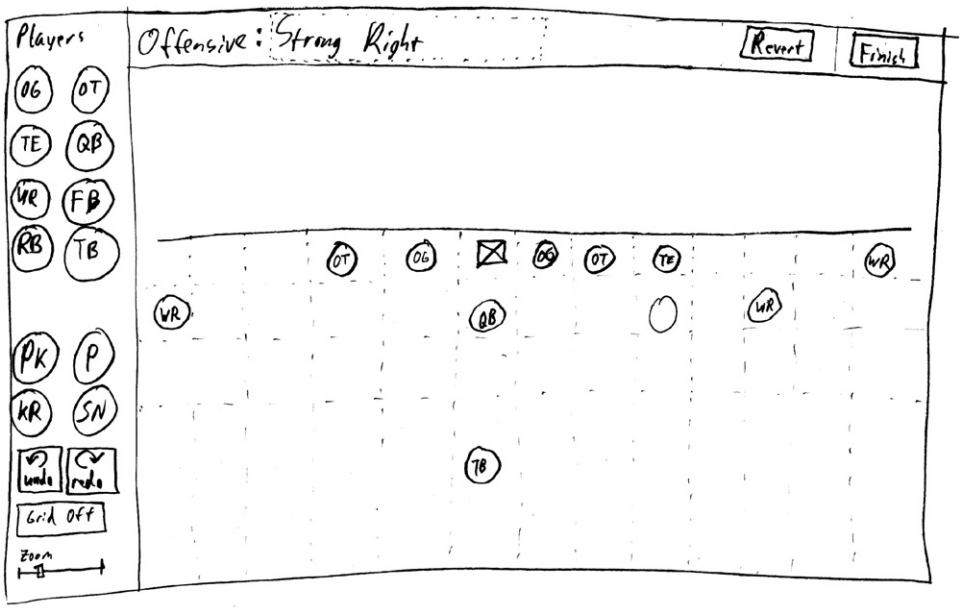

Fig. 3. A portion of the paper prototype: the offensive player formation editing screen for the Interactive Football Playbook. The user would manipulate a player by dragging it from the toolbox, around the field, or off the field.

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 process [8]. 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, but we were able to incorporate many design considerations brought to light by our cognitive dimensions evaluation. Our implementation of the IFP is detailed in the next sections and our evaluation of the implemented version (step four) is detailed in Section 7.

4. Visual language environment

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. Language additions include more player actions (“avoid”, “cover”, “pursue”), parameters for those actions (duration, distance, etc.) and notation for sequencing player actions. 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 those rules are unified over time to modify the states of the players.

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 the user 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 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.

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 (Fig. 4). A formation consists of players and their initial layout with respect to each other and the field. 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.

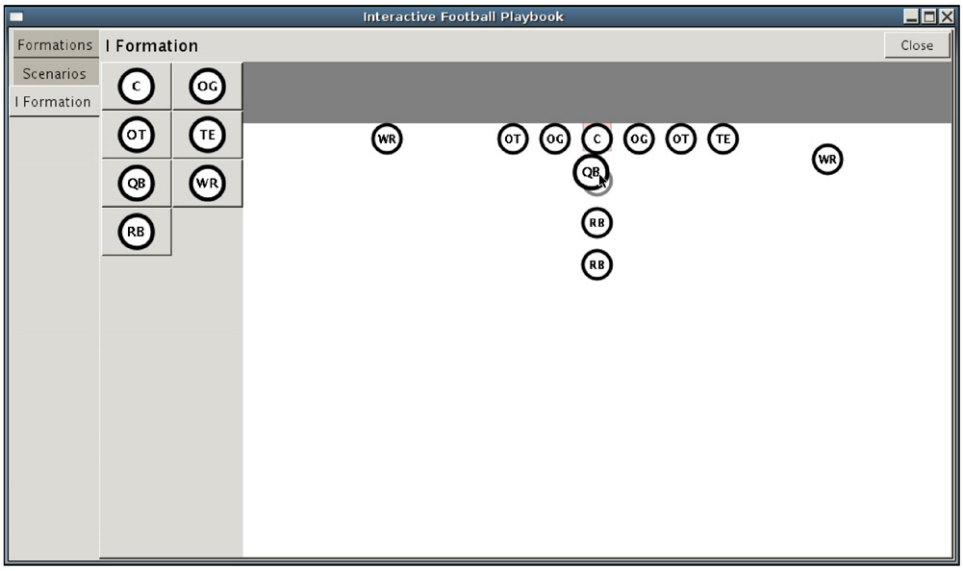

Fig. 4. The formation editor. A player is being dragged onto the field from the icons on the left.

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 (Fig. 5), 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

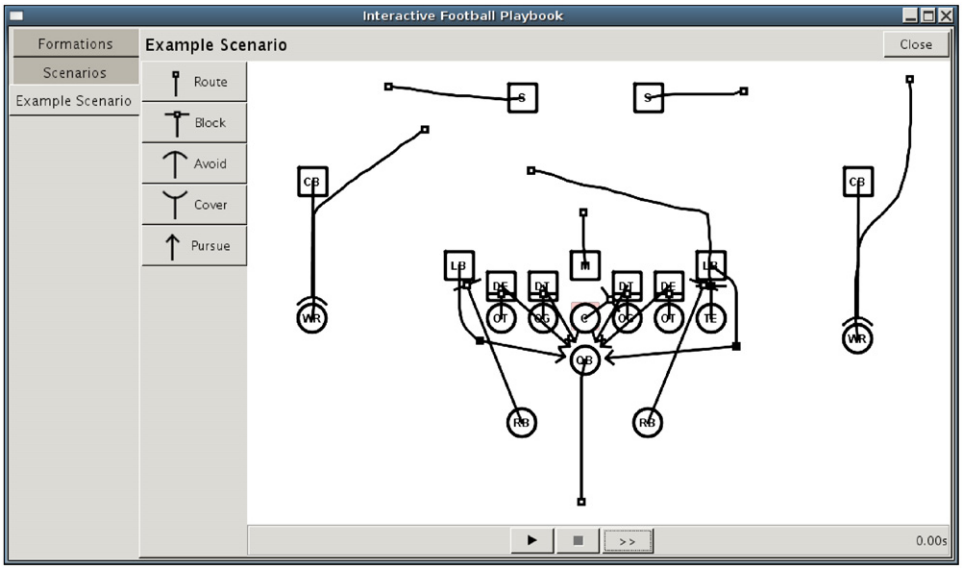

Fig. 5. The scenario editor. Rules are associated with the players as well as spatial locations. The possible rule types are displayed on the left.

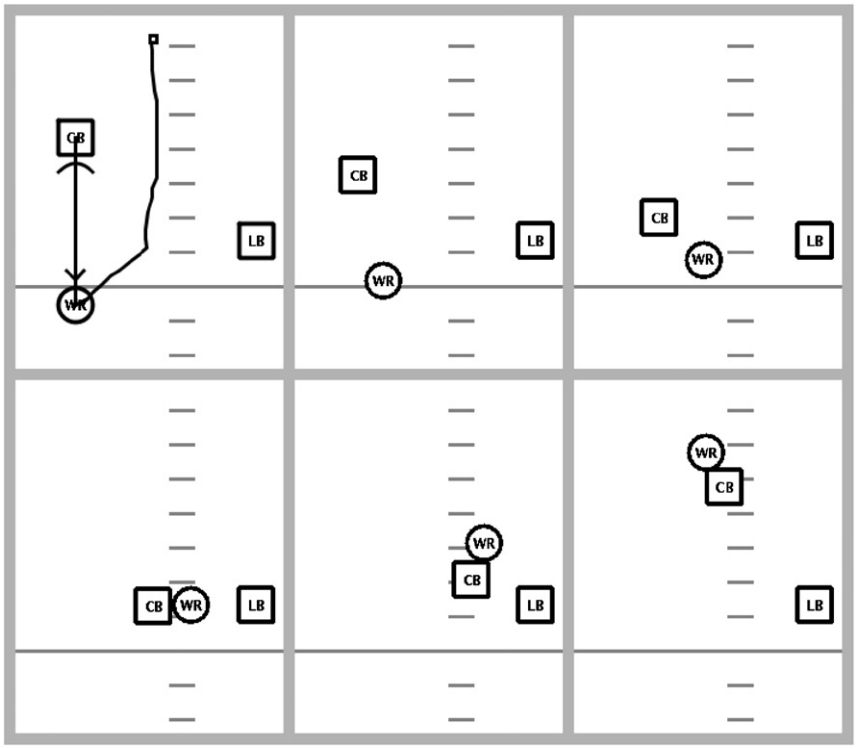

Fig. 6. A fragment of a scenario. At the top left are the rules as specified by the user and the resulting animation is seen from left to right, top to bottom.

and the players animate around the 2D playing field according to the defined rules. The user may pause or unpause 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 Fig. 6 for an example of the animated motion.

5. Language definition

The Interactive Football Playbook language consists of players (virtual agents) and rules (operations on those agents) and produces state changes in the players over time (player performance). An animation engine takes the state changes and visualizes them in real time to create the final animation. Each rule instance is associated with a player, so there are no "global" imperative rules that are applied via pattern matching (like the graphical rewrite rules in Agentsheets). Players are not limited to a single associated rule (like in COACH), but multiple rules may be combined and sequenced ("chained") to produce complex behavior for a player. Additionally, some rules react to the state of another player and therefore allow a player to respond to the behavior of other players.

During execution, each rule instance associated with a player produces a force vector for the player that "pushes" the player in the direction of the vector. Rules are unified by combining these force vectors into a single movement vector for the player. (The details of our implementation are in Section 6.) We use the vector-based approach to facilitate parallel execution of rules (Section 5.3) and rule bending (Section 5.4).

5.1. Players

A player is the fundamental primitive of the language. Every player has the following attributes: side (offense or defense), position (such as quarterback (QB), 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. For example, in Fig. 6, there is offensive player labeled "WR" for wide receiver and defensive player labeled "CB" for cornerback. The player's location on the field is represented spatially as part of a formation like in a standard play diagram (Fig. 4). The user is responsible for creating these formations. To aid the user, we have pre-loaded several standard formations (well known layouts of players on the field) into the IFP.

5.2. Rules

Presently, the language supports a minimal set of rules, but they are sufficient to generate a large number of plays: "run a route", "block", "cover", "avoid", and "pursue". A rule instance is associated with a player and may have a number of parameters to tailor the effect of the rule. Rules have pure pictographic representations (Table 1), so the user-modifiable parameters of the rule are set through direct manipulation.

Table 1
Visual representations of the rules

Avoid		Pursue	
Block		Route	
Cover			

5.2.1. Proactive rules

Proactive rules instruct a player to act, independent of the state of other players in the scenario. Our language currently only has one proactive rule, however, we will discuss more possibilities in Section 7.

Route: Follow a path on the field.

Parameter: Drawn path.

The path is drawn freehand on the field. The open-endedness of the representation creates a very flexible notion of a route since the shape and interpretation of the shape are left up to the user. For example, the entire category of routes in COACH 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. Multiple routes can be chained together to form a piecewise route. Other rules may be chained to the end of the route to set up a player's location on the field before dispatching the player to perform another behavior such as pursuit or blocking. (Rule sequencing is discussed in detail in Section 5.3.)

5.2.2. Reactive rules

Reactive rules instruct a player to act in response to another player. This target player is an implied parameter for any reactive rule, so it is not listed in the parameters below.

Block: Impede an opponent's progress across a directional boundary.

Parameters: Boundary angle, duration of the block.

The blocking player will mirror the movement of the blocked player with respect to the blocking boundary and will push the blocked player to prevent the player from crossing the boundary. The angle of the boundary is shown by the perpendicular line at the end of the blocking rule. (See the running back ("RB") in Fig. 5.) The angle is fixed based on the initial orientation of the starting point of the rule and the initial location of the opponent. If a player is instructed to block and then do something else, the duration of the block is 0.5 s, otherwise, the player will block for the remainder of the scenario.

Cover: Maintain a position relative to another player.

Parameter: Target distance to maintain.

Rather than parametrize this rule with an inside or outside bias, it could be combined with a leverage rule to

more precisely specify the bias of the covering player. (See discussion in Section 7.) Currently, the covering player stays on the closest side (right or left) of the covered player. This rule is generally applied to defensive players whose job is to cover offensive receivers.

Avoid: Change direction to avoid contact with another player.

Parameter: Minimum distance.

This rule can be used to specify that a player is to get around another player without having to specify a specific route that avoids the other player. This rule is generally applied to wide receivers and often to defensive linemen.

Pursue: Run after another player.

Parameters: none.

Our current implementation uses a "reckless" pursuit–no consideration is given to staying in a particular formation while pursuing. (See discussion in Section 7.) In football, this rule is generally given to defensive linemen whose main goal is to catch and tackle the offensive player with the ball.

5.3. Sequencing and parallel execution

Our notation allows the user to specify rules which execute consecutively or simultaneously—essentially representing parallel and sequential "and" without the confusion that stems from the use of the English word "and". Pane et al. documented English speakers' inconsistent use of "and", "or" and "not", noted the confusion that stems from using those words in an end-user query language, and developed a solution for expressing logical "and", "or" and "not" to mitigate confusion [32]. We extend this work by contributing a notation for sequential and parallel "and" in a visual environment.

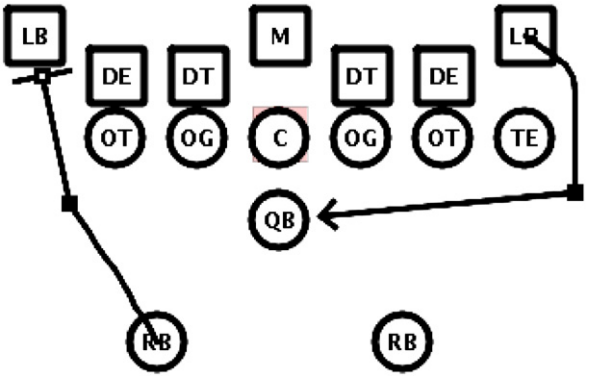

Fig. 7. Two examples of chaining.

Rules may be sequenced through what we term "chaining" as seen in Fig. 7. In the first case, the "RB" is instructed to run a short route and then block the linebacker ("LB"). In the second case, the LB is instructed to run around the tight end ("TE") and then pursue the "QB". Our representation for chaining is equivalent to how programmers use "and" in a textual programming language with short-circuit evaluation:

```
(task_x() AND task_y()) OR ...
```

In this example, `task_x()` is performed and upon returning a successful status indication of `True`, `task_y()` is performed. If `task_x()` is not completed, `task_y()` is not performed. Likewise, in Fig. 7, the RB will not start pursuing the QB until the RB has completed the route.

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 (see Table 1) to which the chained rule is attached (see Fig. 7). In the current model, we have not found a need for chaining rules onto the end of either the "cover", "avoid" or "pursue" rules since these behaviors are intended to apply for the duration of the play once they are invoked.

Rules to be executed in parallel are specified by placing both rules at the same point in the chain of rules for a player. In Fig. 8A, the wide receiver ("WR") is instructed to avoid a corner back ("CB") while running a route. In Fig. 8B, the WR runs part of the route and is then instructed to avoid the LB while running the second leg of the route. As described in Section 6, the performance of a player is determined by the forces applied to the player

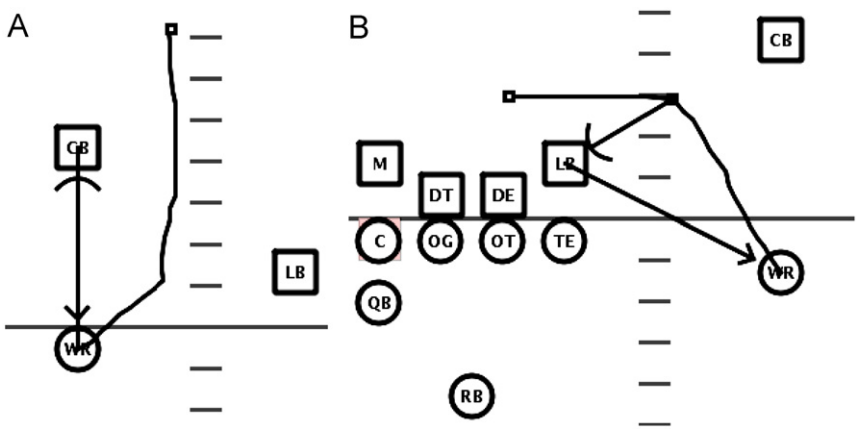

Fig. 8. Examples of parallel execution: (A) the CB is instructed to pursue the WR while the WR is instructed to avoid the CB while running the given route and (B) the WR is instructed to run a part of a route and then continue the route while avoiding the LB.

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.

Our approach leads to semantics in which the physical representation of the rules correlate to the intent of the rule—an example of "closeness of mapping" [27]. When the user draws a freehand path for a route, that resulting "line" not only means the player should run a route, but the player should run the route that passes through specific locations on the path. Consequently, the player's location on field determines the point in the execution chain—so in a sense, the player is a physical instruction pointer that activates sets of rules when the player physically arrives at them. This is similar to trigger-oriented programming that is present in video games where the player's physical interaction with the environment will activate code associated with special locations and actions.

An implication of this approach is that the user does not directly control timing. Unlike keyframe animation where the user may specify at which time the animated object is at a particular coordinate, the time a player arrives at a desired point in the field is determined by the simulation. An advantage to this approach is that the user can work at the level of rules and characters without being burdened with specifying timing details. A disadvantage is that indirect control over timing sometimes requires the user to iteratively change parameters to work out the desired timing appropriate for the scenario being constructed. We hope to include new parameters in the future that would allow users to more directly control timing when needed, such as speed and rule duration.

5.4. Rule bending

Early in our design, it became clear that if our implementation searched for an optimal unification of all the rules in effect on all players, the players would appear to be able to predict the future. Rules like "block" and "avoid" act on the state of other players, so if a LB is instructed to "block" a TE, the position of the LB is determined, in part, by the position of the tight end. If the unification process searched for an optimal solution, the behavior of the LB would not be fully determined by the current moment, but could be influenced by a future behavior that would better satisfy the unification. The behavior of the LB would be acting on knowledge of the future—not realistic human behavior. So, we started using an approach with a limited lookahead so players would not act as if they could predict the future. Our approach is a simulation-based approach which examines the current state and applies a set of rules to produce the next state. Unfortunately, our limited lookahead (a greedy algorithm) quickly lead to situations where some of the rules affecting a player could not be simultaneously satisfied, and play execution would come to a grinding halt. We ultimately took the approach of using soft constraints to allow the unsatisfiable rules to be bent to allow execution to move forward.

The intent of our soft-constraints approach to rule unification is to allow the user to specify what the player is to do without being concerned with guaranteeing the player does *exactly* what is specified. The alternate approach would be for the user to be required to add rules to mitigate dead-ends at each point where the simulation encounters them. These ad hoc corrections would have no semantic value in expressing the users intent, but would be necessary scaffolding to keep the scenario moving forward. Furthermore, the need for scaffolding rules leads to a rule explosion which, in the language of cognitive dimensions [27], increases viscosity. Having incidental rules diminishes semantic value and visibility for key rules while increasing accidental complexity.

Repenning noticed the effects of rule explosion with graphical rewrite rules (GRRs) and developed the notion of "rule bending" within the Agentsheets system to address the problem [4]. Agentsheets uses GRRs to allow end users to program by example. Given a particular situation, the user physically manipulates the agents in the situation to demonstrate the next state, and a rule is inferred from the user's changes. The GRR approach forced the user to demonstrate all possible situations an agent would encounter for the agent's behavior to be well defined.

Repenning created two higher-order constructs, connectivity and symmetry, to allow the user to declare new sets of rules by "bending" previously defined GRRs. For example, the user could define a "road" agent and bend the road in all directions the road should be able to be laid out. The road segment had an underlying connectivity representation which defined how this segment related to segments in adjacent cells, so when the road was bent, the connection to adjacent cells would also be bent. A "car" agent could then be demonstrated to move from one road segment to another and the inferred rules would act in accordance to the declared symmetric cases and the associated connectivity of those cases. Semantically, moving from one segment changed from "the car moves one cell to the right" to "the car moves along a connection from one road segment to another despite the specific direction the road is oriented". The demonstrated rule was "bent" to apply to many different circumstances than just the circumstance the user demonstrated.

In the IFP, we bend the rules by allowing the scenario to move forward even if some of the rules are not satisfied—the IFP is bending the rules because the IFP is not doing exactly what the user told the IFP to do. If the user does not like details of the resulting animation, the user can add more rules to fine tune the performance. What is important is that the user is not forced to write unwanted rules up-front just to get the scenario to run. With rule bending, we avoid premature commitment and unnecessary rule explosion. We implement our rule bending by using vector-based unification as described in Section 6.

6. From rules to execution

Fig. 9 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.

6.1. Unification

Execution is performed by unifying the rules and sending the results to the animation engine, as depicted in Fig. 9. The animation engine operates on a single vector per player (the player's heading), so at each point in time, the rules must be resolved to one vector. The system 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. Clamping is applied when necessary to constrain the vector magnitudes to maximum velocities.

An example of the vector summation can be seen in Fig. 10. The rules input by the user are shown in gray. The LB is directed to run a short route up the field and then pursue the QB, and the QB is directed to follow the given route *while* avoiding the LB. The gray vector pointing to the left was generated by the route rule to keep the QB on the user defined path. The gray vector pointing down to the right was generated by the avoid rule to keep the QB from being too close to the LB. The black vector is the result of adding the two vectors together and it produces motion that is not strictly along the path, nor strictly away from the LB, but something in-between. Fig. 11 shows a more complete sample of the animation shown in Fig. 10.

Numerically speaking, for a set of n rules associated with a player, each rule produces a 2D vector, V_n, where

$$V_n = (x, y)$$

and (x, y) are the 2D components of the vector. These vectors are scaled and summed to produce the final resulting vector, V, where

$$V = \sum_{i=1}^{n} (s_n V_n)$$

and s_n is a scalar that will reduce the magnitude of the vector only when the vector magnitude is greater than the maximum velocity of the player. The value for s_n is computed as follows:

$$s_n = \min\left(1, \frac{M_n}{|V_n|}\right)$$

where M_n is the maximum velocity for player n. Likewise, the combined vector V is scaled in the same fashion if its magnitude exceeds the maximum velocity of the player.

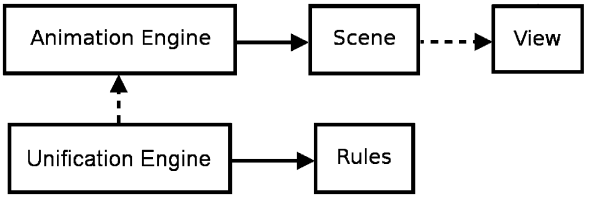

Fig. 9. Execution model of a scenario.

6.2. 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

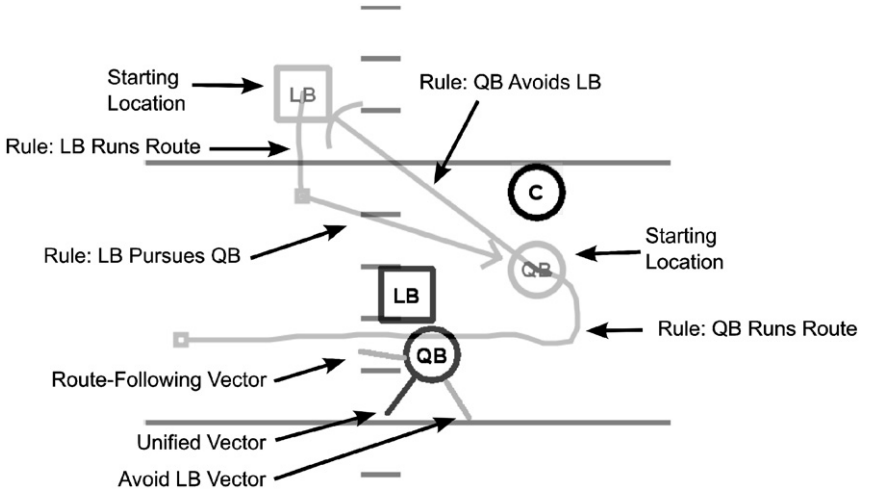

Fig. 10. An example of the summation of vectors for the QB. The scenario as constructed by the user is shown in gray (both the original positions and the rules). The current locations of the players are shown in black and represent the execution of the scenario at the current point in time. Three vectors are shown for the QB. Each gray vector is the result of one rule. The black vector is the combined result of the gray vectors.

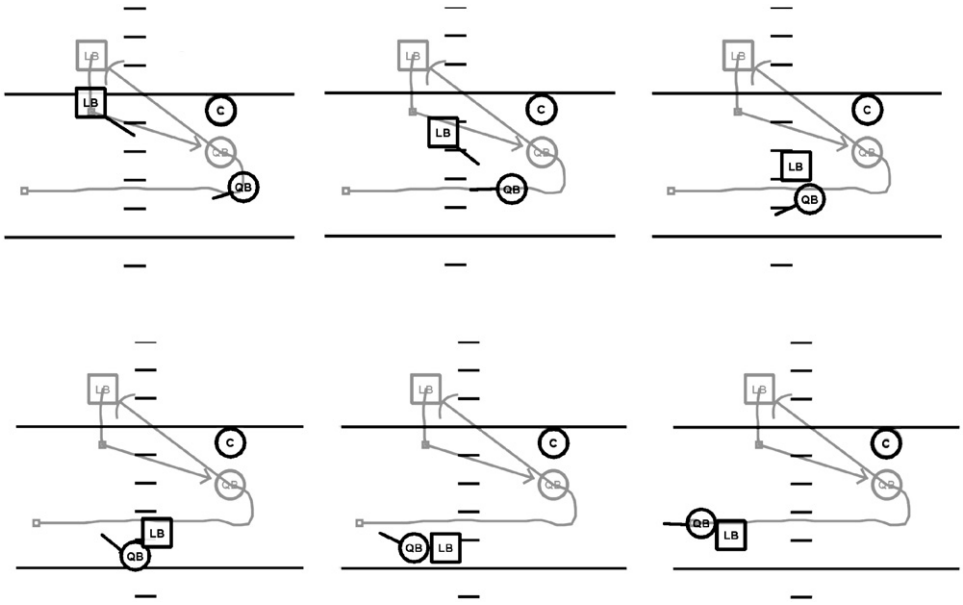

Fig. 11. Approximate resultant vectors for each player over time. The user specified rules are seen in gray. Play proceeds left to right, top to bottom.

using a 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 an approach similar to that used in Pickering's COACH [21,22]. 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 player is determined by the sum of the vectors.

7. Informal evaluation

We developed our working prototype until it had sufficient features that we were able to construct scenarios in which the players behaved in a way that looked reasonable. (See Fig. 12 for an example.) When we were satisfied with our prototype, we conducted demonstrations and a series of interviews with 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 the Natural Programming design process [7]. The feedback we received from the coaches during our interviews is detailed in the following sections.

7.1. User feedback

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 performance 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.

7.2. 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 that were not possible with our current language. We identified several new rules and desired parametrizations of the existing rules.

Leverage: This rule would allow 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 cover rule to further specify on which side to cover the offensive player.

Reckless: This rule would be given to a defensive lineman who 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 would specify that a player is to remain in its general start position. It would be parametrized 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 would allow the coach to designate when the ball is thrown and to which offensive player it is directed.

Handoff: Similar to "throw", this rule would designate when the ball should be handed off and to which offensive player it should be handed—typically a RB.

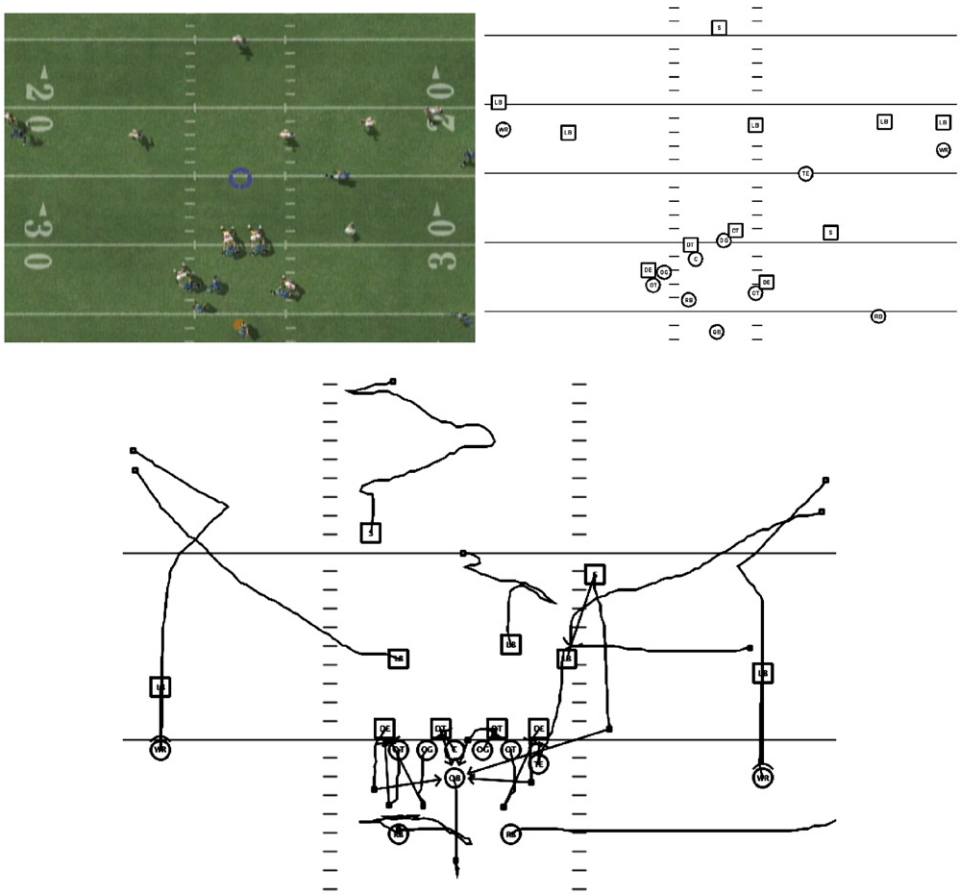

Fig. 12. A comparison with professionally created content. The top image is a snapshot of a football play sequence from Madden 2006 [33]. The middle and bottom are snapshots of the execution and notation within the IFP. The author of the play intended to create a play similar to the Madden play.

Zone: This rule would involve the specification of areas of the field that must be covered by a defensive player as opposed to covering particular offensive players (man-to-man coverage). Although this rule would be 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 covering 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 so much 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.

8. Discussion and conclusion

We used the Natural Programming design process to create a visual programming language for American football coaches to author strategy-oriented, simulation-driven scenarios for illustrating football player behavior. The clear focus on the end user, football coaches, helped shaped the language we designed and led us to design opportunities that, in our opinion, we would have otherwise missed. For example, to support coaches' adoption and comprehension of the language, we adopted constructs similar to the coaches' playbook notation. Starting with a pre-existing notation constrained our notational choices but allowed us to explore spatially oriented notation that is much different than the layout agnostic notation of box and line VPLs.

Defining our notation in a spatial way allowed us to work within a domain (location on the field) that was natural for applying rule bending in a novel way through vector unification. Since we consider deviations in player's locations on the field to not be as significant to the user as forward progress in the simulation, we were able to bend the rules with soft constraints. Ultimately, this benefits the end user since not having hard constraints, forcing the player to be at exact (x, y) locations, reduces the fragility of the system.

Furthermore, the Natural Programming design process led us to a notation that has a close mapping between the rule and the effect the rule will have on the location of the

player on the field. For example, from the user's perspective, the route rule is not manipulating an abstraction, an (x, y) coordinate, in order to move the player through the playing field, but rather the drawn path of the route rule is where the player will move.

The IFP's language could benefit from some improvement and further development. The interviews with the football coaches identified the rule modifications listed in Section 7.2. In addition to these modifications, the discussions identified a need for specifying conditional behavior and more user editable parameters of the current rule set. Conditionals and parametrization play important roles in expressiveness. 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. Or, one player can be authored to overpower or outrun another if players are parametrized by strength and speed. Currently, not all of the parameters identified in Section 5.2 can be modified directly by the user in the IFP's editing environment. Determining the visual editing mechanism for these is part of our ongoing development.

Our current implementation avoided conditional rules since we are focusing on basic authoring for a performance rather than reusable, abstract behavior. With a more expressive language, coaches could write program fragments that could be applied in more than one scenario so coaches would not be required to start from scratch for each scenario. Future research is needed to determine how to make the language more expressive while minimizing the additional cognitive load by introducing new abstractions.

The current IFP notation can lead to a fair amount of visual noise with 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 the view, 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 on debugging the constraints. An example of such tools for supporting authoring can be seen in the work of Ko and Myers [34].

Allowing end users to create significant programs at a high level remains a challenging goal for visual language researchers. By focusing on the end user and working within the constraints imposed by the needs of the end user, we were able to find notation, that through rule bending, is able to express a variety of American football scenarios at a strategy level and can be executed to produce an animation of the scenario. 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.

References

[1] Opendx ⟨http://www.opendx.org/⟩, Last accessed: September 20, 2006.
[2] A. Repenning, Agentsheets: a tool for building domain-oriented dynamic, visual environments, Ph.D. Thesis, University of Colorado at Boulder, January 3, 1993).
[3] Agentsheets Project ⟨http://agentsheets.com/⟩. Last accessed: September 20, 2006.
[4] A. Repenning, Bending the rules: steps toward semantically enriched graphical rewrite rules, in: VL '95: Proceedings of the 11th International IEEE Symposium on Visual Languages, Washington, DC, USA, 1995, p. 226.
[5] A. Repenning, C. Perrone, Programming by example: programming by analogous examples, Communications of the ACM 43 (3) (2000) 90–97.
[6] J.F. Pane, B.A. Myers, Improving user performance on boolean queries, in: CHI '00: CHI '00 Extended Abstracts on Human Factors in Computing Systems, ACM Press, The Hague, The Netherlands, 2000, pp. 269–270.
[7] B.A. Myers, J.F. Pane, A. Ko, Natural programming languages and environments, Communications of the ACM 47 (9) (2004) 47–52.
[8] J.F. Pane, A programming system for children that is designed for usability, Ph.D. Thesis, Carnegie Mellon University, Computer Science Department, Pittsburgh, PA, CMU-CS-02-127, May 3, 2002.
[9] C. Neumann, End-user programming of 3D virtual agents, in: IEEE Symposium on Visual Languages and Human Centric Computing, 2004, pp. 285–286.
[10] C. Neumann, Assisting end-users in understanding and programming simulations, in: IEEE Symposium on Visual Languages and Human Centric Computing, 2005, pp. 339–340.
[11] R. Metoyer, L. Xu, M. Srinivasan, A tangible interface for high-level direction of multiple animated characters, in: Proceedings of Graphics Interface, 2003.
[12] R.A. Metoyer, J.K. Hodgins, Reactive pedestrian path following from examples, in: Proceedings of the 16th International Conference on Computer Animation and Social Agents, IEEE Computer Society, Washington, DC, USA, 2003, p. 149.
[13] R.A. Metoyer, J.K. Hodgins, Animating athletic motion planning by example, in: Proceedings of Graphics Interface, 2000, pp. 61–68.
[14] Coach's Office ⟨http://coachsoffice.com/⟩, Last accessed: September 20, 2006.
[15] S. Li, Rock 'em, sock 'em robocode! ⟨http://www-128.ibm.com/developerworks/java/library/j-robocode/⟩, January 2002.
[16] S. Cooper, W. Dann, R. Pausch, Alice: a 3-D tool for introductory programming concepts, in: Proceedings of the Fifth Annual CCSC Northeastern Conference on The Journal of Computing in Small Colleges. Consortium for Computing Sciences in Colleges, 2000, pp. 107–116.
[17] J. Pane, B. Myers, L. Miller, Using HCI techniques to design a more usable programming system, in: Proceedings of IEEE Symposia on Human Centric Computing Languages and Environments, 2002, pp. 198–206.
[18] A. Witkin, M. Kass, Spacetime constraints, in: SIGGRAPH '88: Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques, ACM Press, New York, NY, USA, 1988, pp. 159–168.
[19] B.A. Myers, R.C. Miller, R. McDaniel, A. Ferrency, Easily adding animations to interfaces using constraints, in: UIST '96: Proceedings of the Ninth Annual ACM Symposium on User Interface Software and Technology, ACM Press, New York, NY, USA, 1996, pp. 119–128.
[20] S. Casner, Building Customized Diagramming Languages, Plenum, New York, 1990, pp. 71–95.
[21] J.M. Pickering, COACH, Master's Thesis, Brown University, May 1999.
[22] J.M. Pickering, D. Bhuphaibool, J.J. LaViola Jr., N.S. Pollard, The coach's playbook, Technical Report CS-99-08, Department of Computer Science, Brown University, May 1999.
[23] C.W. Reynolds, Flocks, herds and schools: A distributed behavioral model, in: SIGGRAPH '87: Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques, ACM Press, New York, NY, USA, 1987, pp. 25–34.
[24] D. Helbing, P. Molnar, Social force model for pedestrian dynamics, Physical Review 51 (5) (1995) 4282–4286.
[25] M.S. El-Nasr, B.K. Smith, Learning through game modding, Computers in Entertainment (CIE) 4 (1) (2006) 7.
[26] D.A. Norman, S.W. Draper (Eds.), User Centered System Design: New Perspectives on Human–Computer Interaction, Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1986.
[27] T.R.G. Green, M. Petre, Usability analysis of visual programming environments: a 'cognitive dimensions' framework, Journal of Visual Languages & Computing 7 (2) (1996) 131–174.
[28] A. Blackwell, T. Green, A cognitive dimensions questionnaire optimised for users, in: A. Blackwell, E. Bilotta (Eds.), Proceedings

of the PPIG 12, 2000, pp. 137–152 ⟨http://www.cl.cam.ac.uk/afb21/CognitiveDimensions/CDquestionnaire.pdf⟩.
[29] J. Dagit, J. Lawrance, C. Neumann, M. Burnett, R. Metoyer, S. Adams, Using cognitive dimensions: advice from the trenches, Journal of Visual Languages & Computing 17 (4) (2006) 302–327.
[30] M.H. Cloyd, Designing user-centered web applications in web time, IEEE Software 18 (1) (2001) 62–69.
[31] C. Wharton, J. Rieman, C. Lewis, P. Polson, The cognitive walkthrough method: a practitioner's guide, Usability Inspection Methods (1994) 105–140.
[32] J.F. Pane, B.A. Myers, Tabular and textual methods for selecting objects from a group, in: VL '00: Proceedings of the 2000 IEEE International Symposium on Visual Languages (VL'00), IEEE Computer Society, Washington, DC, USA, 2000, p. 157.
[33] Electronic Arts ⟨http://www.easports.com/⟩, Last accessed: September 20, 2006.
[34] A.J. Ko, B.A. Myers, Designing the whyline: a debugging interface for asking questions about program behavior, in: CHI '04: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM Press, New York, NY, USA, 2004, pp. 151–158.