CHESS: Cohesion of Heuristic Elements in Structured Story

Nick Feeney, Mike Buerli, Eric Buckthal, and Connor Lange
Computer Science Department
California Polytechnic State University
San Luis Obispo, CA
{ndfeeney,rclange,mbuerli,ebucktha}@calpoly.edu

***Abstract*—In this paper we explore the possibility of using chess games to generate stories. Our approach, Cohesion of Heuristic Events in Structured Story (*CHESS*), contains a variety of components. The first is a chess game parsing system that could read and track chess games saved in the standard PGN format. Next, we defined eleven features which we believe describe the state of the chess game as it progresses. Using these features we attempted to generate stories that were based on the chess game through the use of custom story skins. To accomplish this, we built a plot tree iterator that is able to perform text replacement, plot tree iteration, and management of plot resources. As the game progresses and the plot graph is traversed, there are many plot events that will occur (or not occur), depending on the chess game. The stories that resulted from the project had plots that generally reflected the events in the chess game. However, we discovered that if the plot of the story is to closely match the events of the chess game, the input story skin must be very detailed and have many plot nodes. As a result, the current system requires a great deal of human input to function.**

***Index Terms*—story generation, drama analysis**

I. Introduction

Story generation and drama management have explored many fields attempting to create interactive and interesting drama. The goal of drama management systems is to organize a particular narrative to reflect particular actions of a user; the drama manager often attempts to steer drama in a particular way or change the environment, characters, and plot as if the story was being told by a human watching the same events. If a drama manager prefers to have its subjects follow a more specific story arc, the range of actions a user can perform to influence the system are limited. On the other hand, if the systems goal is an open-ended environment with many possible plot events and story outcomes, it should be possible to allow users to choose their own fate. Human emotion and character development is another aspect of story mediation that should be considered. To accomplish this, some systems interpret how the player converses with characters in the game and the game reacts.

Because our interface for story generation is chess, the control the user has over the course of the story has an interesting property – there are a limited set of possible moves the player can provide and there is no set translation from chess moves to story features. Chess games have a complex strategy in which player advantages and intentions are not easily inferred. The goal of this project is to explore the possibility of utilizing chess games to automatically generate stories. We theorize that events that transpire during a chess game could be translated into plot points which could then be combined to create a narrative. Our solution is comprised of five main subsystems: chess game parsing, chess game state reconstruction, chess move feature extraction, the plot iterator, and the story skins.

First, we built a PGN chess game parsing system that we used to parse meta data and chess moves from chess games we obtained from the internet. Next, we built a system that could keep track of the current state of the chess board after every chess move. We needed this system because the PGN file format does not include information about the current state of the game; it simply lists the moves made during the chess game. We then created a list of eleven features that could be pulled out of a series of chess moves. These features ranged from how dramatic a move was to how far a piece traveled in a set of moves. Next, we built a story plot iterator that assigned chess moves to story plot nodes and then generated text that was bound together to form our stories. To generate the text for a plot node we created a story skin system. These skins were responsible for the text generation. Each skin was capable of text replacement as well as resource management to generate text blocks. In the follow sections, we discuss each subsystem in more detail. For an quick overview of the overall system design see Figure 1.

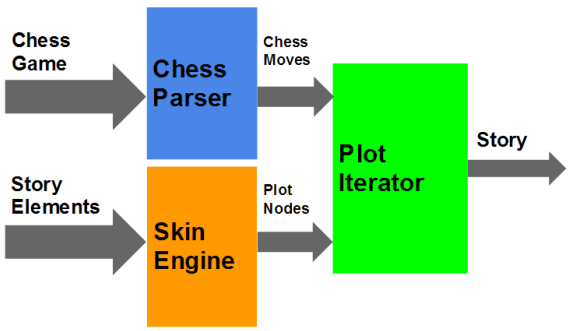

Fig. 1. A visual summary of the *CHESS* system

A. Paper Layout

The structure of the rest of the paper is as follows: Section 2 describes previous work, Section 3 explains the design and implementation details of the *CHESS* system, Section 4 analyzes the results of the *CHESS* system, Section 5 outlines future work and Section 6 concludes.

II. Related Work

This work was originally founded on the dramatic theory of Vladimir Propp. [1] Propp analyzed Russian folk tales into a specific set of functions – independent plot structures deemed the minimal dramatic step of a story. These functions are understood as an act of a single character, defined with the significance of the course of action. Stemming from the idea of an independent point of plot drama, we transitioned to a more open-ended system because the Propp functions were restricting. Other implementations of Drama Management systems are based heavily on their architecture.

For example, the Interactive Drama Architecture (IDA) [2] proposes three key components to interactive drama: the Writer, the Director, and the User. IDA gives the Writer a reasonable amount of control in plot specification, while considering the desire of the User to control how the drama unfolds. IDA is a first-order logic representation of several components for each scene which consider the scene as the smallest dramatic moment; the Writers goal is to move the story forward as a whole. After a human supplies a loosely constructed plot, the following concept approaches are used in the drama solution:

- Annotate each possible object, character, etc. with attributes and then heuristically choose the binding for each variable.
- Have a default binding for each variable. This would put more work on the designer, but at the same time would allow the Designer to input an instantiation of the story as the default story.
- Randomly choose a binding. The Director still retains control of encouraging characters to perform certain actions such as ending the scene.

In Search-Based Drama Management (SBDM) [3], a players concrete experience in the world is captured by a sequence of Player Moves, abstract plot points that a players activity can cause. A single Player Move may encapsulate 5 or 10 minutes of concrete player activity in the world - moving around, picking up objects, interacting with characters and so forth. When the concrete activity adds up to a story significant event, then a Player Move is recognized. A SBDM has a set of System Moves available that can materially alter the world (e.g. move objects around, change goals in characters heads, etc.) in such a way as to encourage or obviate a Player Move. System Moves give the SBDM a way to warp the world around the player so as to make certain Player Moves more or less likely. Besides the System Moves, the author also provides the SBDM with a story specific evaluation function that, given a complete sequence of Player and System Moves, returns a number indicating the goodness of the story. Whenever the drama manager recognizes a Player Move (plot point) occurring in the world, it projects all possible future histories of Player and System moves, evaluates the resulting total histories with the evaluation function, and propagates these evaluations up the search tree (in a manner similar to game-tree search) to decide which system move to make next which will be most likely to cause a good total story to happen.

Declarative Optimization-based Drama Management (DODM) focuses on real-time drama integration in video games and it most closely represents the *CHESS* architecture. EMPath [4] specifically targets a dungeon-style game in which a player is able to explore and complete tasks. The goal is to create a dramatic element in a concrete and playable atmosphere. To configure DODM for a specific world, the author specifies plot points, drama manager actions, and an evaluation function. Plot points are important events that can occur in an experience. Different sequences of plot points define different player trajectories through games or story worlds. Examples of plot points include a player gaining story information or acquiring an important object.

Facade [5], another drama management system, represents the character focused approach rather than plot focused approach. Each character in Facade has a particular mood and certain contextual keywords trigger different moods rather than different plot structure.

III. *CHESS* Design and Implementation

A. Chess Game Extraction

To generate stories from chess games, we first needed to obtain a large number of chess games to use as our data set. We gathered 2,367 games from www.supreme-chess.com [6] in the form of Portable Game Notation (PGN) files. Since each game was in a known format, we simply extracted the information from each game using a regular expression and stored it in a Python dictionary. Each game included the following custom tags, in addition to the standard PGN set [7] and the set of moves in the game:

- WhiteELO - the ranking of the white player (not always present)
- BlackELO - the ranking of the black player (not always present)
- WhiteCountry - the home country of the white player
- BlackCountry - the home country of the black player
- PlyCount - the total number of half-moves played

B. Chess Game State Reconstruction

The chess moves retrieved from the chess parser represent a set state diagram of the chess game. The only information that is given is the type of piece and destination, as well as if there was a capture or some other special activity. Iterating through this list of moves, we are able to disambiguate the correct chess piece (in a case where multiple pieces could make a given move), as well as the starting location of the piece being moved. In this way we can construct a list of chess moves that not only keeps track of the current state of the game and the chess board, but provide features of each individual move. Moves can be further analyzed by calculating

features of the game, such as the total number of threatened pieces and the number of targeted pieces (pieces that can be captured in the next turn). Ultimately the set of chess moves are grouped by the plot iterator and features are extracted from groups of moves.

C. Chess Move Feature Extraction

The features that were developed for this project attempted to get a general overview of the effect that a group of chess moves had on the game. The exact effect of a feature on the chess game is entirely dependent on the skin that is provided for the story. The method in which a chess game feature is used is explained in the Plot Iterator section. In the end, we chose eleven different features, Dramatic, Danger, Hero, Travel, Unimportant Death, Important Death, Unimportant Kill, Important Kill, Check, Safety, and Defeat. These features are described below.

- Dramatic - used to measure how many pieces that a player had in threat
- Danger - measures how many pieces that a player owns are in threat
- Hero - signals when a single piece takes or kills two or more pieces in a set of moves
- Travel - signals when a large amount of the chessboard was covered by the pieces of a player
- Unimportant Death - signaled when a pawn was lost.
- Important Death - signaled when a non-pawn piece was lost.
- Unimportant Kill - signaled when a pawn was killed.
- Important Kill - signaled when non-pawn piece was killed.
- Check - signals that a player put the other players king into check.
- Safety - signals that a player won the game.
- Defeat - signals that a player lost a game.

D. Skins

1) Plot tree: A plot tree is defined as a set of plot nodes, or happenings, in a given story. In a given skin, the plot is constructed using a list of plot nodes that each contain a list of indices, which in turn describe the set of next possible plot nodes. Using this approach, we can map plot happenings, which restrict linear portions of a storys plot, while also modeling multiple plot branches. This architecture also allows for plot nodes to link back to previous nodes. An example of what a plot graph may look like for a given story is shown in Figure 2

2) Plot node: A plot node is the fundamental unit of the plot graph. It includes templates, all features associated with the template(s), resources associated with the node, wordsets used for populating the templates, and a list of nodes in the plot graph that are reachable from this node. Each plot node also contains a generateText() method that populates the chosen template with selected words from the template's wordset.

Text generation is done by selecting a random template from the list of template and doing tag replacement. Tag replacement is done recursively, replacing tags within tags

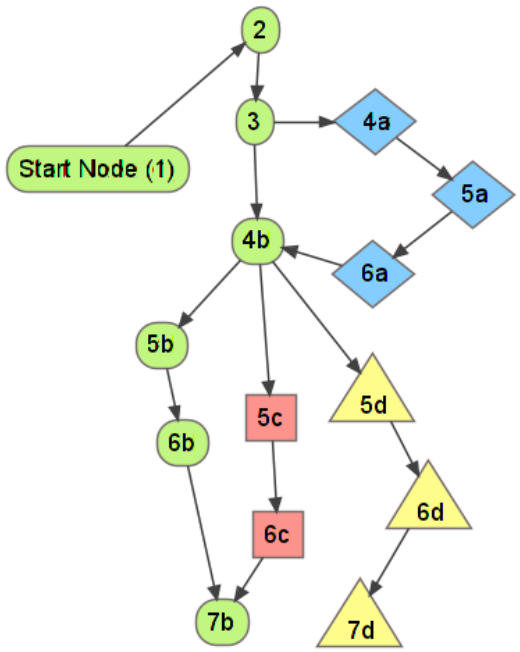

Fig. 2. A potential plot graph. Different shapes and letterings represent different paths in the story arc

with random choices from the skins wordset. While replacing tags, the plot node also keeps track of resources used, deleting a resource word or tag if it is used in a sentence. Lastly, after a template has been converted into a sentence, or sequence of sentences, the template used is removed from the list. Once a plot node runs out of templates, the plot iterator can no longer branch to that specific node. This allows for two levels of variability, both on the sentence level and on the word level, while never repeating the same template twice.

In addition to methods that populate the story, the plot node class also contains methods to determine properties of the plot graph such as shortest distance from the current node to a node that concludes the story and the maximum depth of the plot tree. These properties are ultimately used during the story generation phase of *CHESS* to ensure the constructed story matches the chess game.

3) Templating: The story skin contains a list of lists of templates that are associated with each plot node. Templates are patterns or structures that form a sentence or group of sentences. They are constructed similarly to a standard sentence, but have some words and phrases replaced by tags. Tags are defined as words(alpha-numeric) that begin with an @ symbol. These words or phrases are replaced at runtime with words or phrases from the corresponding tag in the wordset. Tags can be recursive, with tag mentions within a value for a given tag.

The wordset for a plot is implemented as a dictionary constructed from four parts in the skin; words, constants, choices, and resources. Words, the most general type in the wordset, are implemented as a dictionary of lists for each tag. Each list contains one or more strings, from which one will

be randomly chosen for every instance of the tag in the given template. Constants, a type very similar to Words, are also implemented as a dictionary of lists. Its main difference is that one string is chosen from the list at the beginning and remains constant throughout a given story. Next, the Choices type is a list of tuples that contain tags and options. At the beginning of story generation, the tags are randomly mapped to the option strings, so that no two tags have the same value. This allows for easy assignment of groups of names and objects to tags. Lastly, the Resources type is a dictionary of tags where strings from each list are deleted once they are used. If all values have been deleted the tag defaults to the Words set for replacement.

4) Story Skins: Story skins define the interaction of the English sentences, chess features, and plot development attributes. In each skin, a branching plot was developed by hand. Each node represented a plot event and nodes were tied together such that nodes dependent on certain events could only occur after those events. Every event was also given a list of applicable features from our chess feature set. We used feature analysis at each plot branch to determine the next story element. Text replacement substituted certain words in each sentence depending on the mood and level of excitement we wanted to convey in the story. We developed three different skins, each focusing on different elements. The zombie story is our most recent skin and focuses on feature analysis while still including plot branching techniques and text replacement.

E. Plot Iterator

The Plot Iterator is the component of *CHESS* that generates story content from chess move features and a plot graph. Each portion of the iterator is discussed in detail in the following subsections.

1) Chess Move Grouping: Since most of the games we extracted tended to have upwards of 50 moves in them, we couldnt map plot nodes (of which there are less than 50) to each individual move in a game. We originally attempted to use aggregate statistics about the game, in conjunction with examining the features of a particular move to determine importance, to influence a weighted drop rate when pruning moves. However, this lead to less than satisfactory results and we ultimately decided to utilize a uniform grouping of moves. The features for each individual move in the grouping are summed to produce the features of the move grouping.

2) Plot Tree Traversal: The traversal of the plot graph starts with the first plot node, which is fixed for a given skin. The plot iterator then traverses each node one-by-one, calling each nodes generateText() method to build parts of the story, until it gets to a potential branch (either an omission of an event or a different plot path) in the plot graph. To determine which path to take in the plot graph, the iterator calculates the total feature correlation of the move grouping by summing the weights of each feature for each move. Whichever plot node in the graph has the most similar value to the calculated feature weights of the move grouping is chosen. The exact method of calculating feature weights is discussed in the following section. The plot iterator then repeats this process until it gets to the end of the plot graph.

3) Finding the Best Marched Feature: We tried a few different matching algorithms when attempting to match the chess game features with the story skin features. In the end, we achieved the best results with a very simple direct matching method. Our algorithm compares the features of the chess game with the features of the story plot node and finds out how many overlap. We then take the overlapping score and subtract the difference in the total number of features. This method favors plot nodes that have the most features in common and have a similar amount of features to the chess move grouping.

IV. Results

Quantifying validity of the experiment is not a trivial task. The goal was to make an interesting story, but the ultimate task was to reflect the excitement of the game. To measure how well our generated story actually reflected the input game, we surveyed players of various chess skill levels. To present the data, we built a list of 100 PGN translations and paired them with a randomly generated story. It was the users task to determine which story was generated from the chess game given. The chess game was viewed with an online PGN viewer which realized the chess board and piece positions after each move. We gathered results from 50 different experiments from 15 different people. Not surprisingly, when we asked participants to rate their own chess skills, many people put themselves between 1-3 out of 5 and no one rated themselves a full 5 (chess aficionados are hard to come by).

Of the 15 people surveyed, 4 classified themselves as rank 1, 5 as rank 2, 4 as rank 3, and 2 as rank 5. There were 50 total experiments of which 31 were correct. The breakdown is shown in Table I below.

TABLE I
Experiment Results

Chess Proficiency	# Correct	# Total	# of People
1	8	15	4
2	7	14	5
3	10	12	4
4	6	9	2
5	0	0	0
Total	31	50	15

V. Future Work

This project has many pieces that could be expanded for future work. The most obvious and probably most worthwhile would be to automate the story skin generation. The main limiting factor for the entire project is that the story skins that are needed to generate new stories are handwritten. Developing a method to automatically generate these skins could result in better and more varying stories being generated with less human input.

Another main area for expansion on this project would be feature extraction. We developed a small, yet potent feature set, but there is still room for improvement. Smaller features or less obvious features could be developed which could add more detail into the stories. Character tracking could also be implemented and provide a plot that matched the chess game

more accurately. Character tracking would mean that each chess piece is assigned to a character in the story and the fate of that chess piece would dictate that characters actions.

VI. Conclusion

CHESS was our first approach to test the validity of generating stories based on the events that transpire in a chess game. Our implementation shows that it is in fact possible, but there is still much room for improvement. The gap in distinguishing features in English sentences and distinguishing them in chess games could certainly be reduced as well. Currently, we have sentences classified as boring that arent always quite as boring as they seem. The difficulty in increasing variability and interesting plot developments comes at the cost of potentially vague or ambiguous theatrical elements. The chess game feature extraction does a decent job of getting the central themes of the game but could be expanded to get more detailed features. More expertise in chess theory could also help to isolate certain unseen elements in the games. In the future, we would attempt to have more dynamic stories and try to find certain chess themes that really set games apart, instead of focusing on features present in every chess game.

In terms of system improvements, the current version of the story skins require large amounts of user input to create, and this could be reduced. At the time of this writing, each story skin takes about about two hours to generate. We believe that with a few modifications a similar system will perform very well and require much less human input. This topic area shows a great amount of potential and hopefully will be expanded in the future.

References

[1] V. Propp, *The Morphology of the Folktale*. Austin: University of Texas Press, 1968.

[2] M. Mateas, "A neo-aristotelian theory of interactive drama," http://www.etc.cmu.edu/projects/DialogEngine/Neo-Aristotelian%20Theory.pdf, 2000.

[3] M. J. Nelson and M. Mateas, "Search-based drama management in the interactive fiction anchorhead," http://www.aaai.org/Papers/AIIDE/2005/AIIDE05-017.pdf, 2005.

[4] M. Nelson, M. Mateas, D. Roberts, and C. Isbell, "Declarative optimization-based drama management in interactive fiction," *Computer Graphics and Applications, IEEE*, vol. 26, no. 3, pp. 32 –41, may-june 2006.

[5] M. Mateas and A. Stern, "Integrating plot, character and natural language processing in the interactive drama façade," in *Proceedings of the 1st International Conference on Technologies for Interactive Digital Storytelling and Entertainment (TIDSE-03)*, 2003.

[6] "Chess game collection," http://www.supreme-chess.com/chess-games/chess-games-a34.zip, 2012.

[7] Wikipedia, "Portable game notation," http://en.wikipedia.org/wiki/Portable_Game_Notation, 2012.