Generating Believable Stories in Large Domains

Bilal Kartal, John Koenig, and Stephen J. Guy
Department of Computer Science
University of Minnesota
Minneapolis, MN, 55455
{bilal,koenig,sjguy}@cs.umn.edu

Abstract

Planning-based techniques are a very powerful tool for automated story generation. However, as the number of possible actions increases, traditional planning techniques suffer from a combinatorial explosion due to large branching factors. In this work, we apply Monte Carlo Tree Search (MCTS) techniques to generate stories in domains with large numbers of possible actions (100+). Our approach employs a Bayesian story evaluation method to guide the planning towards believable stories that reach a user defined goal. We generate stories in a novel domain with different type of story goals. Our approach shows an order of magnitude improvement in performance over traditional search techniques.

Introduction

Within the last decade, there has been a growing interest in computer generated stories. Such stories are important to enrich the immersiveness of virtual environments used for entertainment, training and education. Also, as these automatically generated stories are brought into increasing use in modern computer games, they must be scaled to handle large domains with dozens of characters, each with many different actions across multiple environments.

Generating stories efficiently in large domains is a difficult problem due to the exponential growth in the search space. Furthermore, the number of possible actions each character can take changes as the story progresses and agents' states change. Monte Carlo Tree Search (MCTS) has shown promising results in several domains with large search spaces. Since it has been proposed, MCTS has been a game changer for several AI problems [Chaslot et al., 2006]. One of the most notable examples is computer Go, recently the program *FUEGO* beat a top human professional at 9x9 Go game [Enzenberger et al., 2010] despite the large search space inherent in Go. Inspired by this, we seek to apply MCTS to the story generation domain to generate believable stories for large scale story domains.

Main Result We propose new algorithms and heuristics to handle automated story generation by employing Monte Carlo Tree Search. This approach performs well both in terms of search time and memory usage. We also introduce a story evaluation metric capable of guiding MCTS to generate believable stories that meet user specified goals. Our method is flexible as it does not need predefined main characters, instead actions arise emergently as needed to satisfy the goals in a believable fashion providing more diversity and flexibility in stories.

Previous Work

In this section we briefly discuss previous work in the area of automated narrative generation and highlight some related work which makes use of MCTS.

Automated Narrative Generation

There are many approaches proposed for narrative generation. We discuss first some character-centric approaches as they are most closely related to our approach.

Character-centric The work of Theune et al. [2003] on Virtual Storyteller models autonomous agents and assigns them roles within the story by an external plot-agent. Actors plan collaboratively both in-character (traditional AI planning) and out-of-character. Once a story plan has been generated, it is passed to the plot-agent which then analyzes the output and determines the most interesting, believable narrative.

Brenner [2010] demonstrated narrative generation within the domain of multi-agent planning. Multi-agent planning is a rich domain which models multi-state variables, agent knowledge, information exchange between agents, and time. Brenner shows that by using multi-agent planning stories can successfully capture character failures. More recently, the work of Teutenberg et al. [2013] combined intentional planning with the multi-agent planning of Brenner [2010]. Instead of considering all possible actions from a given story-world state, agents first filter these actions against a set of intentions and consider only those actions which align with said intentions.

The above approaches focus on creating high quality stories in controlled domains. In contrast, we focus on narrative planning over large story-worlds with many actors, actions, items, and places, while augmenting narrative planning to include believability.

Other Approaches Author-centric methods such as analogy-based story generation of SAM [Ontanon and Zhu, 2011] and MEXICA [Pérez Ý Pérez and Sharples, 2001] attempt to generate stories from the point-of-view of the author. In the case of SAM, narratives are produced by finding analogies in a complete input story and an incomplete target story by use of a predefined knowledge domain.

Lastly, story-centric methods, such as Fabulist [Riedl and Young, 2010], reason over intentions and corresponding actions from the point of view of the audience. In doing so, story-centric methods generate narratives which more clearly communicate character motivations to the audience.

Monte Carlo Tree Search

Monte Carlo Tree Search is a powerful method, that has shown particular success in searching over large domains by using random sampling approaches. It is an anytime algorithm that converges to optimal solutions given enough time and memory. Its success has been particularly recognized after its performance in the game Go [Enzenberger et al., 2010]. MCTS has been further employed in realtime games [Samothrakis, Robles, and Lucas, 2010] and Solitaire puzzles [Cazenave, 2007] where it can outperform humans. MCTS has also been used for other domains including optimization [Silver and Veness, 2010] and planning problems [Chaslot et al., 2008]. For more information, we refer reader to the excellent survey presented in [Browne et al., 2012]. We employed the MCTS with UCB (Upper Confidence Bounds) following the approach from [Kocsis and Szepesvári, 2006] which balances exploration vs. exploitation during planning.

MCTS for Story Generation

In this section, we first introduce a new story domain in which we evaluate our method. We then introduce our believability metric that guides the MCTS search, and provide a detailed explanation of our planning method.

Story Domain

Planning based story generation typically works over a user-specified story domain. We support a custom domain based on a simplified PDDL-type of environment specification. While our approach is generic, we demonstrate it using the following crime-story inspired domain.

Our domain has three types of entities: *Actors*, *Items*, and *Places*. *Actors*, which are intended to represent people or other characters, can pick up or use various *Items*, or move to other places. Each *Item* allows different actions for an *Actor*. *Items* and *Actors* can be located at different *Places*.

Each entity has several *attributes* which allows the planner to keep track of what effect various actions have on the *Actors*, *Items*, and *Places*. For example, actors have a "health" attribute which is decreased when they are attacked. Below is an abbreviated list of the various actions allowed in our story domain.

- **Move(A, P)** A moves to place P.
- **Arrest(A, B)** B's place is set to jail.
- **Steal(A, B, I)** A takes item I from B. This increase B's anger.
- **Play Basketball(A, B)** A and B play basketball. This decreases A's and B's anger.
- **Kill(A, B)** B's health to zero (dead).
- **FindClues(A)** A searches for clues at its current location
- **ShareClues(A, B)** A shares with B any clues he has found.
- **Earthquake(P)** An earthquake strikes at place P. This causes people at P to die (heath = 0), items to be stuck, and place P to collapse.

For *Actors* we have several citizens: Alice, Bob, Charlie, David, etc. There is also a detective named Sherlock, and a police officer named Officer McBraddy. For *Places* there are several homes, recreation areas (e.g., basketball courts), and a downtown. *Items* include flower vases, basketballs, baseball bats, guns and handcuffs. As discussed below, the believability of an actor taking a certain action will depend on where they are, what items they have, and their past experiences with other people.

We assume that the user specifies both an initial configuration and a goal for the story (e.g., who is in their own house, who is in downtown, where are the guns and vases). A common goal might be, "at least two people are dead and the murderer is arrested". For the purpose of running experiments, we can make the domain more complex by adding more citizens, items and places, and by changing the goal.

Believability

Our approach focuses on goal-oriented narrative generation. However, rather than searching to find any story which satisfies a user's goal we search for the best-possible story as evaluated by our metric. For this work, we chose a broad evaluation criteria based on how believable an action is given the current state of the story. The believability of each action is a user-defined measure on a scale from 0 to 1, which we treat as a (Bayesian) probability. That is, given the current state of the world, how likely is it that an event happens conditioned on the current state of the environment. For example, character A attacking character B may be more believable if A is angry. Likewise, a character arresting someone may be more believable if the character is a cop. Some key examples from our domain are presented below.

- **Arrest(A, B)** More believable if A is a cop. More believable if A has clues to a crime.
- **Steal(A, B, I)** More believable if item I is valuable.
- **Kill(A, B)** More believable if A is angry. More believable if A has previously killed someone.
- **FindClues(A, P)** More believable if A is a cop or a detective.
- **ShareClues(A, B)** More believable if B is a cop.
- **Earthquake(P)** Very low believability.

For a series of actions, we evaluate the overall believability as the product of the believability of each individual action:

$$B(a_1, a_2, ..., a_n) = \prod_{i=1}^{n} B_{a_i} \tag{1}$$

Approach Overview

Our approach uses the MCTS algorithm to find the chain of actions which accomplishes the user-defined goals with the maximum amount of believability. To apply MCTS we must first define a function which evaluates the extent that a given story believably reaches the user's goals. This function will shape the expansion of the Monte Carlo search tree. As an evaluation function we use the percentage of goals a story achieves times it's believability.

More formally, we represent a given story as a set of actions $\mathcal{A} = \{a_1 \cdots a_n\}$. We define a story evaluation function E as:

$$E(\mathcal{A}) = G(\mathcal{A})B(\mathcal{A}) \tag{2}$$

where $G(\mathcal{A})$ is the percentage of the user-defined goals the current story accomplishes, and $B(\mathcal{A})$ is the believability of the story as specified in Eqn 1.

The value of $E(\mathcal{A})$ will be maximized when the search algorithm finds a story which satisfies all the user's goals in the most believable way possible. Importantly, this formulation allows for a series of actions that are not very believable to occur in the story if it is the only way to achieve the user's specified goals.

While $E(\mathcal{A})$ provides a natural way to evaluate a completed story, it is of limited use for partial stories that will be encountered during a tree search. This is because, until a story satisfies some of the goals the evaluation will always be 0. We address this issue by adding a random *rollout* to the story, that is a series of random actions that is added to the partial story until all the goals are met (or until a story grows past a length threshold). We denote this randomized extension of $\mathcal{A}$ as $\mathcal{A}'$:

$$\mathcal{A}' = \{a_1, a_2, ...a_n, r_1, r_2, ...r_n\}. \tag{3}$$

where $r_1 \cdots r_n$ are randomly generated actions. This allows a probabilistic evaluation of $\mathcal{A}$ even when $\mathcal{A}$ does no yet reach the goal. We denote this probabilistic evaluation as E':

$$E'(\mathcal{A}) = E(\mathcal{A}'). \tag{4}$$

We can now formulate story generation as a Monte Carlo tree search problem. Each node in the tree will represent the complete state of the world. Each link in the tree represents one possible action from that state, and that child of the node represents the resulting world state after applying that action. The root of the tree is the initial state of the world. The MCTS algorithm proceeds by repeatedly adding one node at a time to the current tree. For each potential action, we keep track of how many times we have tried that action, and what the average evaluation was. Choosing which child node to expand (i.e., choosing which action to take), becomes an exploration/exploitation problem. We want to primarily choose actions that had good scores, but we also need to explore other possible actions in case the random rollout was "unlucky" and does not represent their true potential of that action.

The exploration/exploitation dilemma has been well studied in other areas. Here, we chose to use the Upper Confidence Bounds (UCB) approach proposed by [Auer, Cesa-Bianchi, and Fischer, 2002]. Applied to our framework, this means that each node choses its child n with the largest value of $f(n)$:

$$f(n) = E'(\mathcal{A}_n) + \sqrt{\frac{2 \ln v}{n_v}} \tag{5}$$

where $\mathcal{A}_n$ is the parent's story so far updated to include action n, v is the total number of times this node has been visited, and n_v is the total number of times that given child action has been previously tried.

Choosing which node to add is then a recursive process. For each node, a child action with the largest value of UCB equation (Eqn 5) is chosen and expanded. When a node with unexplored child is reached ($n_v = 0$) a new node is created for one of the unexplored children. The process then starts again from the root of the tree, each time adding one new node. This way, the tree can grow in an uneven manner, biased towards nodes with high value for $E'(\mathcal{A}_n)$, which are likely to be good stories. This process is summarized in Algorithm 1, the algorithm takes as input a *budget* of the maximum number of nodes to explore and returns a series of actions which comprise the story.

Algorithm 1: MCTS Story Generation

Input : budget
Output: best score and best story
while $budget > 0$ **do**
  Node ← ucbSelection(root) ;
  result ← rolloutStory(node) ;
  backpropagate(result) ;
  if $result > bestScoreSoFar$ **then**
    updateBestScore();
    saveBestStory();
  end
end
return Best Story;

Iterative Implementation

Because the MCTS algorithm keeps the entire tree in memory, in some cases, the approach can run out of memory (see Figure 5). This is especially true with domains that have large branching factors (e.g., many people, places, items or actions). This can be alleviated by pruning sections of the search tree that are unlikely to be productive. To this end, we propose an iterative approach which plans a story one action at a time. This approach first grows the tree for a fixed number of actions. Then, only the current best action kept, and its sibling actions' and their subtrees are pruned. This action forms the new initial condition and the tree search continues. Pseudocode for the iterative approach is presented in Algorithm 2.

Because a fixed number of nodes are added between each pruning step, the amount of memory used is bounded. We

Algorithm 2: Iterative Story Generator

```
Input   : budget and max_iterations
Output: best score and best story
for i ← 1 to max_iterations do
    while budget > 0 do
        Node ← uctSelection(root);
        result ← rolloutStory(node);
        backpropagate(result);
        if result > bestScoreSoFar then
            updateBestScore();
            saveBestStory();
        end
    end
    root ← root's most visited child;
    Prune all other subtrees;
end
return Best Story;
```

should note that this iterative approach is no longer probabilistically complete, as it is possible to prune a promising branch early on, leading to a local maxima rather than the global optimum. However, in practice we are still able to generate high scoring stories while using much less memory than the non-iterative approach.

Search Heuristics

Monte Carlo Tree Search can be improved by applying heuristics to help guide the search. We incorporate two domain independent heuristics. For both heuristics, we keep a history table that stores average evaluation results, E', for each action (independent of it's depth in the tree). We explore two ways of using this history table: *selection biasing* and *rollout biasing*.

Selection Biasing Here we modify Eqn. 5 to incorporate the average value for the action stored in the history table. We introduce a parameter α which weighs the history average value more strongly when very few (less than k) rollouts have been performed. Formally:

$$f(n) = \alpha E'(\mathcal{A}_n) + (1-\alpha)H(n) + \sqrt{\frac{2\ln v}{n_v}} \qquad (6)$$

where $H(n)$ is average value stored in history table and $\alpha = n_v/k$.

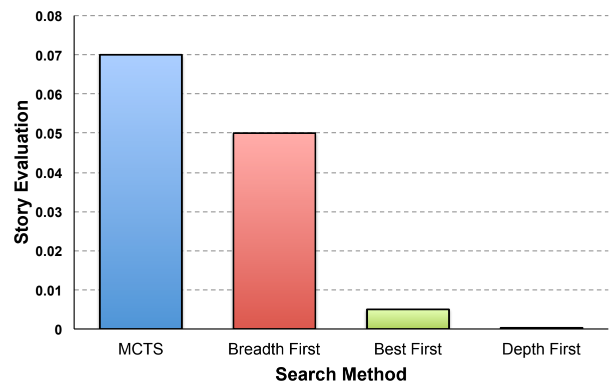
Rollout Biasing In this heuristic we use the history table to bias the random rollouts in Eqn. 4. Rather than choosing pure random actions, we preferentially choose actions which have had a higher evaluation score as stored in the history table.

Results

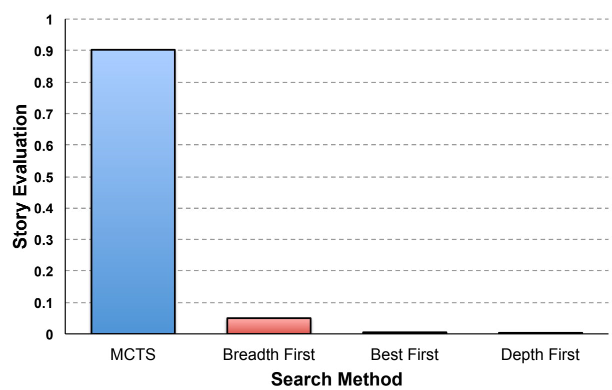
We tested our approach on an instance of the crime story domain described above. We utilized 5 actors (including 1 policeman and 1 detective), 5 places, and 5 items. The story goal is set as 2 people dead and the murderer arrested. Because each actor can use multiple items and travel to different places the resulting search space was fairly large with an average of 50 total actions available across all the actors at any given time (resulting in a search tree with an average branching factor of 50).

Search Method Comparison

We first compare our method to traditional search algorithms of Breadth-First Search, Depth-First Search, and Best-First Search. We chose these search algorithms because, like MCTS, none of these algorithms requires a search heuristic. Furthermore, Breadth-First Search and Best-First Search algorithms are guaranteed to find an optimal solution given sufficient time and memory. Additionally, Best-First Search and Depth-First Search will explore longer paths earlier which can potentially find optimal solutions earlier in the search process. All search algorithms are implemented such that they maximize score from Eqn 4. Figure 1 shows a comparison of the best story found by the different methods both for small search budgets and large ones (results averaged over 3 trials).



(a) Low Budget (100K Nodes)



(b) High Budget (3 Million Nodes)

Figure 1: **Comparison of Search Methods** Our proposed approach using Monte Carlo Tree Search (MCTS) outperforms other search techniques such as Breadth-First Search, Depth-First Search, and Best-First Search. (a) Even for a small search budget, MCTS outperforms other methods (b) The gains improve dramatically for larger budgets.

Depth-First search was observed to use very little memory, however, it failed to find stories which met any goals. Best-First search suffers from delay caused by trying to accomplish the goals through a set of believable actions due to its high exploratory behavior. As a result, it tends to require higher budget to eventually find the optimal solution.

While Breadth-First search outperforms the Best-First search and Depth-First search methods, it is unable to find a believable means to achieve the goal even with a budget of several million nodes. In contrast, our MCTS approach outperforms all the other search techniques for both small and large budgets, and it is able to find a high score story and performance difference increases exponentially in favor of MCTS given more budget.

The difference in stories generated by the various search approaches is highlighted in the illustrative sample stories below. These stories are direct outputs from our code. We note that we automatically combine two consecutive related actions into a single sentence to improve readability of the stories.

Figure 2 shows a sample of a high quality story, that has been generated by our MCTS algorithm. The story achieves the goals while containing several plausible actions (such as revenge killing).

Alice picked up a vase from her house. Bob picked up a rifle from his house. Bob went to Alice's house. While there, greed got the better of him and Bob stole Alice's vase! This made Alice furious. Alice pilfered Bob's vase! This made Bob furious. Bob slayed Alice with a rifle! Bob fled to downtown. Bob executed Officer McBrady with a rifle! Charlie took a baseball bat from Bob's house. Sherlock went to Alice's house. Sherlock searched Alice's house and found a clue about the recent crime. Bob fled to Alice's house. Sherlock wrestled the rifle from Bob! This made Bob furious. Sherlock performed a citizen's arrest of Bob with his rifle and took Bob to jail.

Figure 2: High Quality Story (Score: 0.68)

Figure 3 shows a story found by Breadth-First search. While the story is short and accomplishes the goal of two people being killed, it fails to achieve the more complex goal of somebody being arrested. Furthermore, the story makes use of an earthquake to reach its goals, which has a very low believability score.

Sherlock moved to Alice's House. An Earthquake occurred at Alice's House! Sherlock and Alice both died due to the earthquake.

Figure 3: Low Scoring Story (Score: 0.016)

Heuristic Comparison

We also experimented to determine the effect of our two proposed heuristics on search performance. Figure 4 summarizes our results (averaged over 3 trials). For low search budgets, the selection biasing heuristic improves performance over standard MCTS (Fig 4a). However, this heuristic gets stuck at a local minima and fails to improve the story even with large search budgets. In contrast, the rollout biasing heuristic leads to a substantial improvement over standard MCTS for large search budgets (Fig 4b).

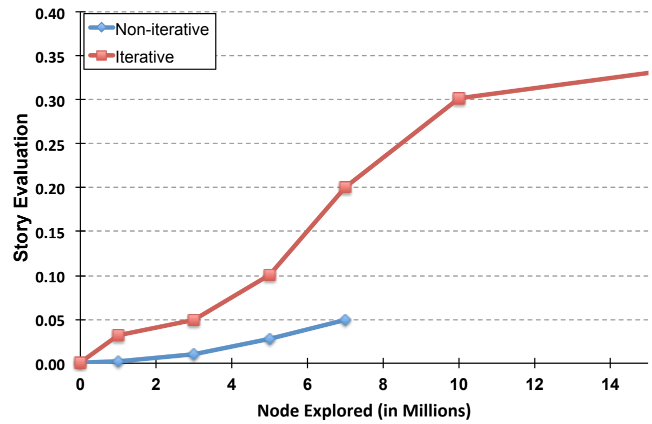


Figure 5: **Iterative vs Non-iterative** For very large stories domains, MCTS can run out of memory trying to store the entire search tree. In the 20-person domain, the non-iterative approach could only explore trees up to 5 Million nodes before failing. Our proposed iterative approach uses tree pruning to reduce memory and can explore much large trees (producing higher value stories).

Large Scale Scenarios

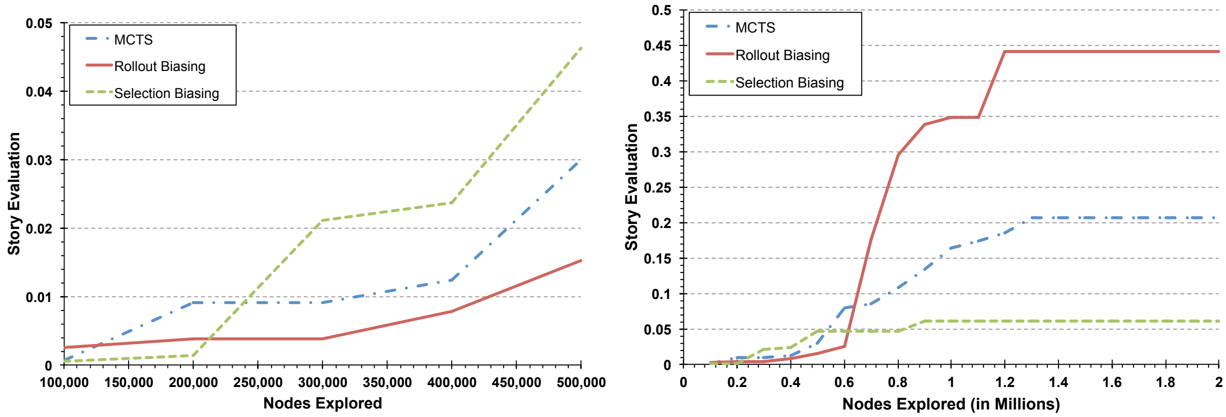
While the MCTS approach we described in Algorithm 1 works well, it consumes large amounts of memory. This large memory usage can restrict its applicability on very large scenes. To illustrate this limitation, we extend the crime story domain above to contain 20 actors, 7 places, and 7 items. This increases the branching factor to 150 potential actions on average.

Figure 5 compares standard MCTS with our iterative approach described in Algorithm 2. Importantly, the non-iterative approach fails to complete its execution when the search budget is larger than 5 million nodes. This failure happens because the non-iterative approach is using over 100GB of memory for such large search trees. In contrast, our proposed iterative approach produce better results for lower budgets, and can run much larger budgets without failure. In fact, we were able to run with a budget over 50 million nodes on the same machine with no memory issues.

Runtime For the 5 actor story domain, our method was able to find detailed stories in under 5 seconds, and find the optimal story in less than 1 minute (using a single core on an Intel 2.2 GHz laptop processor). For the 20 actors story domain, stories took much longer to generate, though a high quality story could generally be found in under 1 hour with the iterative approach.

Conclusion

We have presented a framework capable of generating believable stories satisfying goals provided by a user even in large domains. By using a Monte Carlo Tree Search approach, we were able to balance exploiting the most promis-



(a) Low Budget

(b) High Budget

Figure 4: **Effect of Heuristics** (a) For small search budgets (<500K nodes explored) the search heuristics tested had only a moderate effect on performance. (b) For large search budgets, the advantage of the rollout biasing heuristic can be clearly seen. Additionally, while the selection bias heuristic helps with small budgets it tends to get stuck in local minima.

ing branches along with exploring other potentially good choices at each level of the tree. The resulting framework generated complex, believable stories with only a few seconds of computation time for small domains, and a few minutes for larger ones.

Limitations

While our method is capable of exploring large story spaces, our approach still has some limitations. The tree size being stored in memory still grows exponentially as the number of potential actions increases. Therefore, a story involving 100s of characters is likely to run out of memory on consumer hardware. We have also focused only on domain-independent heuristics, usage of domain specific heuristics can alleviate memory problems and reduce runtime.

Future Work

Beyond addressing the above limitations, we think there are exciting directions for future work. We would like to explore other forms of evaluation criteria beyond our believability metric. For example, a user might want to specify the pacing of a story to ensure rising actions and a climax. Additionally, we think our method is well suited for telling stories in an interactive domain where the goals change dynamically in response to the recent actions of a user. To this end, relevant heuristics such as the IFF (Intentional Fast-Forward) heuristic [Ware, 2012] may help further improve search times and achieve real-time story generation for large worlds. We are working on our first GUI prototype to enable interactivity with a user in domains with real time response times.

Acknowledgments

This work was supported in part by the University of Minnesota Supercomputing Institute. We also would like to thank Dr. Ioannis Karamouzas for the discussions and his feedbacks throughout this work.

References

Auer, P.; Cesa-Bianchi, N.; and Fischer, P. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine learning* 47(2-3):235–256.

Brenner, M. 2010. Creating dynamic story plots with continual multiagent planning. In *AAAI*.

Browne, C. B.; Powley, E.; Whitehouse, D.; Lucas, S. M.; Cowling, P. I.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; and Colton, S. 2012. A survey of monte carlo tree search methods. *IEEE Trans. on Computational Intelligence and AI in Games* 4(1):1–43.

Cazenave, T. 2007. Reflexive monte-carlo search.

Chaslot, G.; De Jong, S.; Saito, J.-T.; and Uiterwijk, J. 2006. Monte-carlo tree search in production management problems. In *Proceedings of the 18th BeNeLux Conference on Artificial Intelligence*, 91–98.

Chaslot, G.; Bakkes, S.; Szita, I.; and Spronck, P. 2008. Monte-carlo tree search: A new framework for game ai. In *AIIDE*.

Enzenberger, M.; Muller, M.; Arneson, B.; and Segal, R. 2010. Fuego—an open-source framework for board games and go engine based on monte carlo tree search. *IEEE Trans. on Computational Intelligence and AI in Games* 2(4):259–270.

Kocsis, L., and Szepesvári, C. 2006. Bandit based monte-carlo planning. In *Machine Learning: ECML 2006*. Springer. 282–293.

Ontanon, S., and Zhu, J. 2011. The sam algorithm for analogy-based story generation. In *AIIDE*.

Pérez Ý Pérez, R., and Sharples, M. 2001. Mexica: A computer model of a cognitive account of creative writing. *Journal of Experimental & Theoretical Artificial Intelligence* 13(2):119–139.

Riedl, M. O., and Young, R. M. 2010. Narrative planning:

balancing plot and character. *Journal of Artificial Intelligence Research* 39(1):217–268.

Samothrakis, S.; Robles, D.; and Lucas, S. M. 2010. A uct agent for tron: Initial investigations. In *Computational Intelligence and Games (CIG)*, 365–371. IEEE.

Silver, D., and Veness, J. 2010. Monte-carlo planning in large pomdps. In *Advances in Neural Information Processing Systems*, 2164–2172.

Teutenberg, J., and Porteous, J. 2013. Efficient intent-based narrative generation using multiple planning agents. In *AAMAS*, 603–610.

Theune, M.; Faas, S.; Heylen, D.; and Nijholt, A. 2003. The virtual storyteller: Story creation by intelligent agents.

Ware, S. G. 2012. The intentional fast-forward narrative planner. In *Eighth Artificial Intelligence and Interactive Digital Entertainment Conference*.