An Intelligent Storytelling System for Narrative Conflict Generation and Resolution

Youngrok Song
Department of AI
Sungkyunkwan University
Suwon, South Korea
id2thomas@gmail.com

Hyunju Kim
College of Computing
Sungkyunkwan University
Suwon, South Korea
julia981028@gmail.com

Taewoo Yoo
College of Computing
Sungkyunkwan University
Suwon, South Korea
woo990307@naver.com

Byung-chull Bae
School of Games
Hongik University
Sejong, South Korea
byungchull@gmail.com

Yun-Gyung Cheong
Department of AI
Sungkyunkwan University
Suwon, South Korea
aimecca@gmail.com

***Abstract*—Conflict is essential for the development of complex plots or more in-depth character design. Therefore, many literary works have employed a conflict-centered narrative structure. However, it is challenging to incorporate conflicts in a planning-based story generation because a planning algorithm constructs a plan that avoids conflicts, which can occur when one action negates the precondition of the other character's action. To address this issue, this paper presents an intelligent storytelling system that can automatically generate conflicts and their resolutions using the interleaved causal relationships between different characters in the story. We describe an example to show that the proposed algorithm can successfully generate a story that contains inter-personal conflicts and is presented in the Virtual Environment. We conclude with a discussion and future study.**

***Index Terms*—Narrative Generation, Conflict, Planning, Causal Threats**

I. Introduction

A common definition of narrative is "representation of a sequence of events" [11], where events can involve the change of states by an agent's actions. As game agents (or characters) perform actions to achieve the given goals, conflicts - either interpersonal or intrapersonal - is essential to create an interesting story.

Conflict in narrative refers to "the struggle in which the actors are engaged", which can be either *external* (e.g., protagonists fighting against fate or destiny, conflicts between protagonist and antagonist due to various reasons such as different beliefs or values, different goals or the same goals but with limited resources) or *internal* (e.g., protagonist's inner conflict between good and evil) [9]. Conflict is critical for distinguishing "narrative" from "non-narrative" - in other words, *narrativity* [9], where conflicts are consisted of "discrete, specific, and positive situations and events, and meaningful in terms a human(ized) project and world" [9].

Conflict is essential for the design of a plot in the story. In his book [5], McKee juxtaposes the story with life, as a good story reflects our real life in which conflicts always occur. McKee categorizes narrative conflict more in detail with three levels - inner, personal, and extra-personal for different types of plot such as "stream of consciousness", "soap opera", and "action/adventure" [5, p.213], respectively. Through the combination of different types of conflict, plots can become deeper and richer. For instance, the computational model in [4] generates a suspenseful story by rearranging a story plan that contains a conflict already.

While conflict is crucial for story development, only a few studies [12] [13] [14] have paid attention to it. Szilas has developed IDtension [12], an interactive story generation system based on the computational model of actions and obstacles in order to increase dramatic tensions between story characters. Ware and Young [14] presented CPOCL (Conflict Partial Order Causal Link) algorithm to support narrative conflict based on the previous partial-order causal link (POCL) planning algorithm. CPOCL creates conflicts by preserving the causal links that are threatened by other actions, still being able to generate a sound plan by marking those causally threatened steps as non-executed by adopting the notion of "character intentions" and "intention frames". With the CPOCL algorithm, however, plans that are free of conflicts can also be generated.

In this paper, we explore how conflicts can be formally defined and generated using a plan-based approach, which has been employed for (interactive) narrative generation [6] [1] [3] [7] [10] [2]. Planning in general is a problem-solving technique, consisting of a planning problem (i.e., initial state and goal specification) and a planning domain (i.e., objects, predicates, and action operators). Given the input of a planning problem, a sound planner produces a solution or a plan, that is, a sequence of actions with which all the specified goal conditions can be achieved without any causal threats.

The main contribution of this paper is to present a system that generates a story with conflict and its resolution, which is realized in an interactive virtual environment.

978-1-7281-4533-4/20/$31.00 ©2020 IEEE

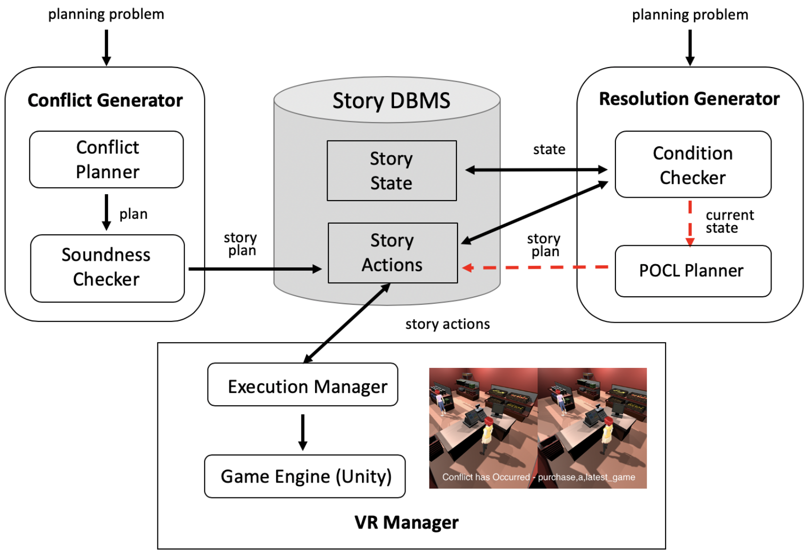


Fig. 1. A system overview (Dotted arrow in red color denotes that a conflict occurs.)

II. Our Approach

This section describes our approach and an overall architecture of the proposed system.

A. System Overview

Our system comprises three main modules - Conflict Generator, Resolution Generator, and VR manager. *Conflict Generator* is a story planner that generates story plans which contain conflicts; a conflict in this paper denotes a situation where a character's action threats the causal link established by another character. *Resolution Generator* is a POCL planner which generates a story plan with a conflict resolution in it. *VR Manager* is a module that retrieves story actions from story DB and automatically executes them on a virtual environment using a game engine such as Unity.

In our system, we mostly make use of the POCL planning formalism and algorithms for the generation of story actions that can include either conflict or resolution. A POCL plan is represented as a four-tuple $\langle S, B, C, O\rangle$, consisting of plan steps (S), binding constraints (B), causal links (C), and ordering constrains (O). A causal link is a link between two steps A and B in the plan when A establishes the precondition p of B and is written as $A \xrightarrow{p} B$. A conflict with the causal link can occur when a step C has the effect that unifies with the negation of the condition already established, $\neg p$, in our case. An ordering constraint takes the form of $A < B$ denoting 'A occurs before B'. Unlike a total-order plan where all the steps in the plan are ordered, ordering constrains in partial-order planning are added only when necessary. A solution (i.e., a *plan*) in partial-order planning is a sequence of steps (having partial ordering) that will achieve the goal state when executed in the initial state. In this paper we did not take binding constraint into account.

An overall process of generating conflict and resolution is illustrated in Figure 1. First, a planning problem consisting of the initial state and the goal state is given to both the *Conflict planner* and the *POCL planner*. Next, each planner generates a plan; *Conflict* planner generates an unsound plan that contains conflicts in it. The *POCL* planner builds a sound plan that contains no cycles in the ordering constraints and no threats with the causal links.

Among different types of narrative conflict, this paper proposes a planning algorithm to create inter-personal conflicts by making modifications to the POCL planner [8].

B. Conflict Generator

Conflict Generator modifies a partial-order planner in which the search algorithm looks for plans with conflict for a given planning problem with partially described ordering of actions. In this paper, a *conflict* is defined as a causal threat where one character's action can negate the precondition of the other character's action. Conflict Generator is composed of two submodules - Conflict Planner and Soundness Checker.

Unlike conventional POCL planners, the Conflict Planner manipulates threats differently for the purpose of conflicting-story generation. The algorithm initially builds an empty plan that contains $Start$ and $Finish$ steps. Then, it generates child nodes by refining the current partial plan. The refinement process includes causal planning and threat manipulation. The causal planning follows the original POCL algorithm [15], which creates causal links to establish open preconditions. However, the threat manipulation performs in a different manner. When the threatening or threatened actions (that prevent a character from achieving his or her goal) are performed by another character, the algorithm manipulates the ordering of the threats to ensure that conflicts occur. As a result, a character's actions to achieve a goal is threatened (i.e.,

Algorithm 1: The Planning Algorithm.

```
Result: A plan P = $\langle S, B, C, O\rangle$, where $S$ denotes plan steps,
  $B$ denotes binding constraints, $C$ denotes causal links,
  and $O$ denotes ordering constraints.
Initialization: The initial plan contains Start and
  Finish steps, the ordering constrain $Start < Finish$,
  and all the preconditions (i.e., goal conditions) in
  Finish as open preconditions. An open precondition
  refers to a precondition that is not achieved by some
  steps in the plan.
Termination: If the plan is consistent and contains no
  open preconditions, terminate and return the plan.
Plan Refinement: Non-deterministically do one of the
  following.
1. Causal Planning:
    (a) Select an arbitrary open precondition of a step
  $S1$.
    (b) Add a step $S2$ that achieves the condition. If no
  such step exists, backtrack. Otherwise, add the
  binding constraints required for the conditions to
  unify, an ordering constraint $S2 < S1$ that orders the
  new step $S2$ before $S1$, and the causal link
  $S2 \longrightarrow S1$. The step $S2$ can be either an existing
  step in the plan or a new step that is instantiated
  operator which has an effect that can be unified with
  the precondition to the step of the open precondition.
  If $S2$ is a new step, add it to $S$ and the ordering
  constraints $Start < S2$ and $S2 < Finish$ that orders
  it between $Start$ and $Finish$.
2. Threat Manipulation: Find any step $S3$ that might
  threaten to undo any causal link $S1 \xrightarrow{p} S2$ where $p$
  is a precondition of $S2$.
  for every such step do
    if the threatening action and the threatened
      actions are taken by same conflicting characters
      then
        Non-deterministically do one of the following
          to resolve conflicts with causal links:
        (a) Promotion: If possible, add the ordering
          constraint $S2 < S3$ to order the threatened
          steps to occur before the threat in the plan.
        (b) Demotion: If possible, add the ordering
          constraint $S3 < S1$ move the threatened steps
          to occur after the threat in the plan.
        (c) Separation: If possible, add binding
          constrains on the steps involved so that no
          conflict can arise.
    else
        Create conflicts with causal links by adding the
          ordering constraints $S1 < S3$ and $S3 < S2$
          that order the threatening step between the
          threatened causal link's source and destination
          steps.
    end
end
```

Algorithm 2: Determining Same Conflicting Character in Causal Threat.

```
Result: Boolean SameChar where True means same
  conflicting characters in threatening and threatened
  action of given causal threat.
Input: Causal Threat with threatened causal link
  $S1 \longrightarrow S3$ and threatening action $S2$. Conflicting
  Characters $CC$.
Determination: Initialize SameChar to True.
if $S1$,$S3$ taken by same character in $CC$ and $S2$
  taken by different character in $CC$ then
    SameChar = False
else if $S1$ not taken by character in $CC$ and $S2$,$S3$
  taken by different character in $CC$ then
    SameChar = False
else if $S3$ not taken by character in $CC$ and $S1$,$S2$
  taken by different character in $CC$ then
    SameChar = False
```

thwarted) by another character's actions. We excluded the threats that are taken by the same character, as it can produce some inconsistent or absurd actions (e.g., repeatedly making negating or negated actions by the same character). This can prevent the suggested algorithm from generating unnecessary flaws in the resulting plan.

To ensure that the generated plan contains conflict, only unsound plans (i.e., plans with threats to causal links by another character) are selected. Because the algorithm enforces inter-personal threats only, the unsoundness in the plans are derived from ignoring inter-personal threats. As a result, the actions that fail to be executed can be viewed as the points of conflict between the characters. Soundness Checker sub-module confirms this kind of conflict from the generated plans, and then writes those unsound plans (including unsound actions, the initial state, and the goal state) into Story Database.

C. Resolution Generator

The story plan containing conflicts cannot be fully executed in the virtual environment because some actions in the plan may have conditions that are not established. Therefore, Conflict Checker monitors whether the current state of the story can assure the execution of the next action by inspecting the preconditions of the next action retrieved from the Story Actions table. Only the actions approved as executable by the Condition Checker are executed in the virtual environment.

When the system detects an action that cannot be executed because its preconditions are not established as described in Algorithm 3, the system asserts that a conflict occurs and considers the action as the point of conflict. Then, it marks all the actions in the Story Actions table that are not yet executed at this point as invalid. Next, Resolution Generator creates a planning problem where the initial state is the state recorded in the database and the goal state is same as in the original planning problem. A solution to this planning problem is generated using the POCL planner. The actions of the solution are marked as safe and are appended to the Story Actions table in the database.

```
Algorithm 3: Resolution Generator
Result:
Input: Story Database with plan $P$, state $S$ and goal
  state $G$.
Initialization: All actions in $P$ is marked as Valid.
while $S$ not reached $G$ do
    Get next valid action $A$ in $P$
    if $S$ satisfies precondition of $A$ then
        Apply $A$ to $S$
        Update story database with updated $S$
        Mark $A$ as Executable(safe)
    else
        Mark $A$ as point of conflict
        Mark remaining actions from $P$ as Invalid
        Create planning problem $PP$ using $S$ as initial
          state and $G$ as goal state
        Generate sound plan with original planner by
          solving $PP$
        Append generated plan actions to $P$ and mark
          them as Executable(safe)
end
```

D. VR Manager

The VR Manager consists of Execution Manager and the Unity Game Engine (version 2019.2.12f1). The Execution Manager of the VR Manager module controls the character behavior in the VR envinronment. This retrieves the next action marked as executable in the story action table of the database and sends this to the VR game engine for realization. Then it marks the action as executed in the story actions table.

III. An Example

This section presents an example story to illustrate the conflict creation process. Table I shows the planning problem that contains the initial and the goal state. The initial state represents that the characters in the story A and B both need a recently released digital game title, which is in stock at a computer game shop. In the goal state both A and B purchased the game, B refunded the game, and A owns the game, and the game is no longer in stock.

TABLE I
A sample planning problem.

	Conditions
Initial State	(instock latest_game)(need A latest_game) (need B latest_game)
Goal State	(purchased B latest_game) (purchased A latest_game)
Characters	A B

TABLE II
A sample planning domain.

Operator Name	Condition Type	Conditions
purchase(?c, ?p)	precondition	(instock ?p) (need ?c ?p) (not (purchased ?c ?p))
	effect	(own ?c ?p) (not (instock ?p)) (purchased ?c ?p)
find_spare(?c, ?p)	precondition	(own ?c ?p) (need ?c ?p)
	effect	(not (need ?c ?p))
refund(?c, ?p)	precondition	(own ?c ?p) (not (need ?c ?p))
	effect	(instock ?p) (refunded ?c ?p) (not (own ?c ?p))

A. Plan Domain

Table II defines a set of plan operators used in this study. Its preconditions define the conditions that must be established before the action is executed. The effects are conditions that are made true after executing the action.

The $purchase(?c, ?p)$ operator denotes the character bound to the variable $?c$ purchasing the product $?p$. To execute the step, the product that is bound to $?p$ should be in stock, the character that is bound to $?c$ needs to purchase the product and has not purchased the product yet. After executing the $purchase$ action, the effects are changed to state that the character has the product, the product is not in stock, and the character purchased the product. Likewise, the $find\_spare(?c, ?p)$ operator represents that the character finds out that the product is obtained by another family member, and therefore the character $?c$ no longer needs the product. The $refund(?c, ?p)$ operator represents that the character $?c$ returns the product $?p$.

B. Resulting Plans

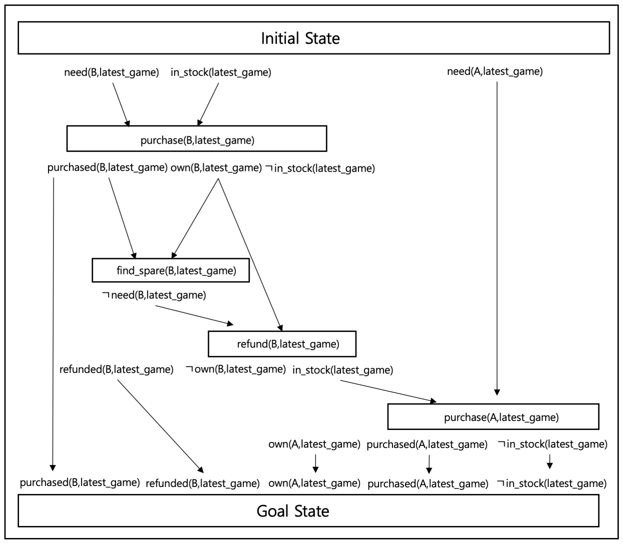


Fig. 2. A plan generated using the original PyDPOCL (Decompositional Partial Order Causal Link planner implemented with Python). The ordering constraints proceed from the top to the bottom.

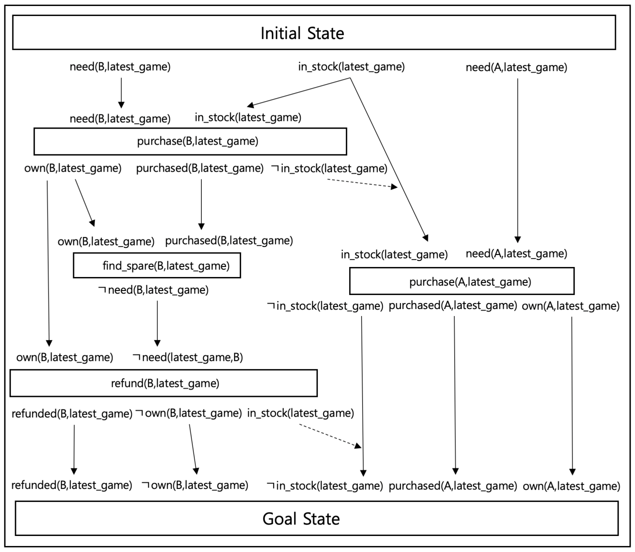


Fig. 3. A plan generated using the proposed algorithm which enforces causal threats (denoted by dotted lines). As a result, the character A attempts to purchase the game title when its precondition is not established, which we call conflict in this study.

Figure 2 illustrates a sound plan that is generated using the PyDPOCL planner, the Python version of DPOCL (Decompositional Partial Order Causal Link) planner[1]. The plan does not include the decomposition functionality of the planner, as our focus is on its characteristics as a partial-order planner that utilizes the causal relationship in the plan. In the plan, the character B purchases the game title, finds a spare, and then refunds it. Then, A purchases the game title when it is in stock after B refunds it. As shown in the figure, a sound solution that has no conflicts can be generated with the given planning problem.

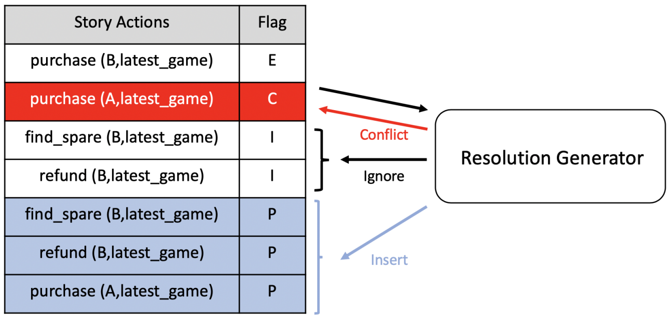
On the other hand, Figure 3 shows an unsound plan that contains conflicts, which is generated by Conflict Generator. In the figure, the conflicts are denoted by dotted lines; $purchase(B, latest\_game)$ threats the causal link $Init \longrightarrow purchase(A, latest\_game)$ by undoing the effect $in\_stock(latest\_game)$ that establishes the precondition of the $purchase$ action taken by A. As a result, A'will fail to take the purchase action. Another conflict occurs when B refunds the game after A purchases it by negating the precondition of the goal step, $\neg in\_stock(latest\_game)$.

C. Realization

To realize the generated story via the VR environment, the plan actions shown in Figure 3 and its initial state are recorded in the story database. Then Resolution Generator and VR Manager are executed. The Resolution Generator program's story state is initialized with the state recorded in the database.

Each action in the database has an execution flag value, which indicates whether it can be executed or not. When the story actions are added to the Story Actions database, all the actions' flags are set to 'P' denoting that it is pending to be checked by the Resolution Generator module. If the module determines an action as safe, the flag is set to 'S' denoting safe for execution; otherwise, the flag is set to 'E.' Then it updates the story state applying the action's effects to the current story state. The VR Manager can only present the actions that are flagged as 'S.' After presenting the action, its flag is updated to 'E.'

[1]https://github.com/drwiner/PyDPOCL

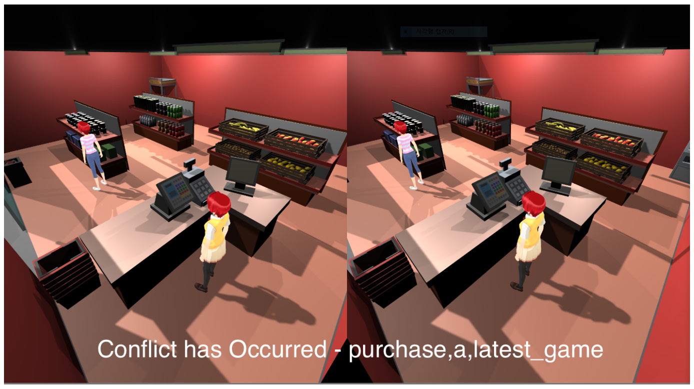


Story Actions	Flag
purchase (B,latest_game)	E
purchase (A,latest_game)	C
find_spare (B,latest_game)	I
refund (B,latest_game)	I
find_spare (B,latest_game)	P
refund (B,latest_game)	P
purchase (A,latest_game)	P

Fig. 4. Plan Resolved by Resolution Generator. Red column indicates conflict generating action. Blue columns denote sound plan actions generated by POCL planner of Resolution Generator.

Figure 4 illustrates when a conflict has occurred. When Resolution Generator detects an action that generates conflict, its flag is set to 'C' denoting conflict. All the following actions' flags are set to 'I' to be ignored by the VR Manager. In order to resolve the conflict, a new modified planning problem is created utilizing the state in the database as its initial state. A new sound plan is generated by the POCL planner in Resolution Generator with the modified planning problem. Generated plan actions are then appended to the database with its actions' flags set to 'P.'

When the VR Manager detects a conflict flag, it notifies the viewer that a conflict has occurred and specifies which action has been attempted. Figure 5 shows a VR screenshot when a conflict occurs. A video footage describing this example can be accessed via the link at https://youtu.be/gd1wHpmlpfY.



Conflict has Occurred - purchase,a,latest_game

Fig. 5. VR screenshot and Story DB when a Conflict occurs

IV. Conclusion

This paper address a computational model of conflict generation for storytelling. To create inter-personal conflicts, we present an intelligent story generation system based on AI planning algorithms. The system consists of three main modules: Conflict Generator, Resolution Generator, and VR Manager. When a planning problem given, Conflict Generator generates conflicts by manipulating the causal links in the plan using a proposed algorithm that modifies the POCL (partial-order Causal Link planner) algorithm [15]. When a conflict occurs, Resolution Generator leverages the original POCL planner [15] to generates a complete story that eliminates the conflicts to achieve the goal as stated in the planning problem. The VR Manager retrieves the story from the database system and realizes the story in the VR environment using the Game Engine. We implement the system and generate a simple example to show that the algorithm can successfully generate a story plan that contains conflicts arising from the interleaved causal relationships among characters in the story. We implemented the system to realize the example story in a VR environment.

In the future, we will carry out formal evaluations to show the efficacy of the proposed approach. Currently, the system can generate a linear story. We will extend the system to allow the user to interact with the VR environment.

Acknowledgement

This research was partly supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education(2019R1A2C1006316), the Ministry of Science and ICT (2017R1A2B4010499), and the Institute of Information communications Technology Planning Evaluation(IITP) grant funded by the Korea government(MSIT) (No.2019-0-00421, AI Graduate School Support Program).

References

[1] Cavazza, M., Charles, F., Mead, S.J.: Character-based interactive storytelling. IEEE Intelligent Systems **17**(4), 17–24 (July 2002). https://doi.org/10.1109/MIS.2002.1024747

[2] Cavazza, M., Young, R.M.: Introduction to Interactive Storytelling, pp. 1–16. Springer Singapore, Singapore (2016)

[3] Charles, F., Lozano, M., J. Mead, S., Fornés, A., Cavazza, M.: Planning formalisms and authoring in interactive storytelling. Proceedings of TIDSE **3** (01 2003)

[4] Cheong, Y., Young, R.M.: Suspenser: A story generation system for suspense. IEEE Transactions on Computational Intelligence and AI in Games **7**(1), 39–52 (2015)

[5] McKee, R.: Story. Mathuen (1998)

[6] Michael Young, R.: Notes on the use of plan structures in the creation of interactive plot (12 1999)

[7] Michael Young, R.: Story and discourse: A bipartite model of narrative generation in virtual worlds. Interaction Studies - INTERACT STUD **8** (01 2007). https://doi.org/10.1075/is.8.2.02you

[8] Penberthy, J.S., Weld, D.S.: Ucpop: A sound, complete, partial order planner for adl. pp. 103–114. Morgan Kaufmann (1992)

[9] Prince, G.: A dictionary of narratology. University of Nebraska Press, Lincoln, Neb., rev. edn. (2003)

[10] Riedl, M.O., Young, R.M.: Narrative planning: Balancing plot and character. CoRR **abs/1401.3841** (2014)

[11] Ryan, M.L.: Toward a definition of narrative. The Cambridge Companion to Narrative pp. 22–35 (01 2007). https://doi.org/10.1017/CCOL0521856965.002

[12] Szilas, N.: Idtension: a narrative engine for interactive drama (09 2003)

[13] Szilas, N., Richle, U.: Towards a Computational Model of Dramatic Tension. In: Finlayson, M.A., Fisseni, B., Löwe, B., Meister, J.C. (eds.) 2013 Workshop on Computational Models of Narrative. OpenAccess Series in Informatics (OASIcs), vol. 32, pp. 257–276. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2013). https://doi.org/10.4230/OASIcs.CMN.2013.257, `http://drops.dagstuhl.de/opus/volltexte/2013/4164`

[14] Ware, S.G., Young, R.M., Harrison, B., Roberts, D.L.: A computational model of plan-based narrative conflict at the fabula level. IEEE Transactions on Computational Intelligence and AI in Games **6**(3), 271–288 (Sep 2014). https://doi.org/10.1109/TCIAIG.2013.2277051

[15] Young, R.M., Moore, J.D.: DPOCL: A principled approach to discourse planning. In: Proceedings of the Seventh International Workshop on Natural Language Generation (1994), `https://www.aclweb.org/anthology/W94-0302`