Applying norms and preferences for designing flexible game rules

Edward Booth, John Thangarajah* and Fabio Zambetta

RMIT University,
GPO Box 2476, Melbourne,
Victoria, 3000, Australia
Email: edward.booth@rmit.edu.au
Email: edjbooth@gmail.com
Email: john.thangarajah@rmit.edu.au
Email: fabio.zambetta@rmit.edu.au
*Corresponding author

Abstract: Interactive storytelling is a strength of table-top role playing games as they are facilitated by a game master (GM) who directs the narrative and devises game scenarios. One difficulty with the implementation of a GM in computer games is the large amount of time, effort and specialist skills that can be required for the creation of such an agent. Another issue is that game rules become embedded in the agent implementation and thus may become difficult and time consuming to change. This article aims to address these issues by presenting a method for developers to shape the narrative by defining game behaviour in terms of norms and preferences. The system was evaluated with both a case study and a user experiment. The results showed that the users found out the system to be both user friendly and suitable for development of games with flexible narrative.

Keywords: AOSE in games; intelligent game design; norms; preferences.


Biographical notes: Edward Booth is a recent graduate from RMIT University with honours in Computer Science. His dissertation was in the area of artificial intelligence in games and is currently a research engineer in the games industry.

John Thangarajah is an Associate Professor in Artificial Intelligence at RMIT University, Australia. His main research interests are in agent-oriented software development, agent reasoning, intelligent conversation systems, agent testing, and intelligent games.

Fabio Zambetta is a Senior Lecturer at RMIT University, Australia where he coordinates the Games and Graphics Programming degree. His main research interests include artificial intelligence in games, reinforcement learning and virtual, augmented, mixed reality.
This paper is a revised and expanded version of a paper entitled ‘Flexible story
generation with norms and preferences in computer role playing games’
presented at Computational Intelligence and Games, CIG 2015, Tainan,
Taiwan, 31 August to 2 September 2015.

1 Introduction

In this article, we will investigate defining game rules in terms of norms and preferences
and then applying these concepts to externally govern an automated intelligent game
master’s (GMs) behaviour. This decouples the specification of these rules from the
implementation. In this way, we hope to present a method in which game rules can be
specified in more human readable form and can also be modified without making
changes to the GM code and vice versa.

Table-top role playing games (RPGs) are a style of game in which play is facilitated
by a GM who directs the narrative and creates the scenarios which the players will
negotiate (Wizards RPG Team, 2008). For example, a GM may devise a dungeon area
including specifications for the layout, number and type of monsters and any reward
items and their location within. The GM then provides a narrative motivation for the
players to enter the dungeon and the gameplay proceeds with the players relaying their
choices and actions to the GM and the GM subsequently updating the state of the game
world and relaying results back to the players.

In computer role playing games (CRPGs), the GM feature is not usually present and
thus the narrative is directed in a largely predetermined manner by designers specifying
the events and their sequence. It is also usual for gameplay environments to be built at
development time and remain static through each play through. This results in games for
which players have little motivation to replay in terms of narrative whereas, in contrast,
an infinite number of possible storylines are possible with a human GM.

In an attempt to address this shortcoming, Luong et al. (2013) implemented a system
using a table-top style GM agent within the CRPG Neverwinter Nights (NWN)
(BioWare, 2002). This GM guides the player through a given scenario by selecting a path
of actions via which the player can achieve the goal. The GM then manipulates certain
factors in the game world to guide the player through this sequence of actions. For
example, the GM agent may select a path involving the player bribing a specific character
with a favour to obtain a password. The GM would consequently have that character
announce that they are in need of assistance. Due to the number of possible actions and
their sequences a large amount of possible storylines for the quest are possible and each
play-through is likely to be quite different to the last.

Luong’s implementation used a BDI agent-oriented framework to implement the
NWN GM. Belief-desire-intention (BDI) (Rao and Georgeff, 1991; Rao et al., 1995)
agents are a popular and mature agent development paradigm based on cognitive
concepts such as beliefs, goals, intentions and plans. Agent-oriented programming allows
for the creation of autonomous agents to perform tasks which are capable of reasoning
and reacting to environmental changes.

One shortcoming of Luong’s approach is that the GM’s selection of narrative options
is essentially random. Using this approach, game designers would have no control over
Applying norms and preferences for designing flexible game rules

which of the implemented plot points the GM would present to the player. We address this issue with our preference specification framework which allows designers to describe the type of gameplay they desire for the game.

Another issue with the existing GM is that a game designer would need knowledge of agent-oriented and BDI concepts in order to design how game scenarios will function as the game rules are embedded in the BDI implementation. While agent-oriented approaches give significant advantages in the development of complex software systems (Jennings, 2000), building such systems can be time consuming and demanding of a high level of skill from developers. As game scenarios become larger and more involved, it may also be difficult for programmers to make changes to the game rules within a BDI context as such systems can easily become complicated.

We address these issues by implementing an innovative application of the well explored concepts of norms and preferences into the GM agent. We use norms and preferences because we are able to specify them separately from the agent code and they can dictate agent behaviour without the need for re-implementation and testing of agent definitions themselves. We also approach preferential reasoning in a novel way by using probability information to give the agent some information about the lower level implications of selecting higher level plans.

A norm is an application of a rule to one or more agents for which a sanction is incurred by an entity who breaks it (Alechina et al., 2012; Dybalova et al., 2014). These rules can take the form of obligations or prohibitions. For example, Amy is obligated to arrive at her job by the time her shift starts and will receive a warning from her employer if she fails to do so. She is also prohibited from driving faster than the specified speed limit and if she exceeds that limit she may incur a fine. When a norm restricts behaviour, it is necessary for some reasoning to occur against overall goals. For instance, in the above example, if Amy’s friend has been badly injured and it is necessary to get him to the hospital as quickly as possible, Amy may choose to break the speed limit in order to do so as that goal takes precedence over a potential fine.

In this context, preferences are defined as an agent’s predisposition to achieve a goal in a certain manner (Visser et al., 2011). These preferences come into play during reasoning when the agent has multiple plans it can select to fulfil an objective. For example, if Joe’s goal is to get to work on time, he may prefer to catch the train to get there. However, if Joe misses the train he can still catch a taxi to work to fulfil his goal. In this case, the preference was not satisfied but Joe was still able to achieve the goal, as opposed to if his goal was defined as ‘catch the train to work’, which he would have failed.

The remainder of this article is set out as follows: in Section 2, we discuss the background of and related work in areas such as the BDI framework, preferences and norms. Section 3 details the GM architecture, explaining the preference and norm specification languages and how the normative and preferential reasoning functions. Details of the system implementation follow in Section 4 and Section 5 discusses the evaluation of the GM using both a case study and a user experiment. We further discuss some areas of the project in Section 6, and conclude in Section 7 with some future directions.
2 Background and related work

The following is an overview of related work on the BDI agent architecture, norms, preferences and approaches to the problem of interactive storytelling, which our normative GM agent is to address.

2.1 BDI agents

In this work, we extend the work of Luong et al. (2013) which is based on using a BDI agent system to represent the GM that drives the narrative of the game. The BDI model is a mature framework which is commonly used for the development of automated intelligent agents. The model is based on the philosophical work of Bratman et al. (1988) which was later formalised by Rao and Georgeff (1991) and Rao et al. (1995). The framework models aspects of human reasoning processes such as beliefs, desires, goals, intentions and plans.

A typical BDI agent system consists of a plan library where for a given goal, there is one or more plans in the plan library that could possibly achieve the goal (this is an OR decomposition, as any of the plans can satisfy the goal). Each plan performs some actions and/or posts a number of subgoals (this is an AND decomposition as all subgoals must be achieved for the plan to succeed). A subgoal is in turn handled by some other plan in the plan library in a similar manner. This decomposition leads to a natural hierarchy which is termed a goal-plan tree. The root of a goal-plan tree is a goal node, and the leaves of goal-plan tree are plan-nodes. Each goal-node has one or more plan nodes as children, and each plan-node has zero or more goal-nodes as children. A goal is considered achieved if one of its child plans succeeds. A plan is considered achieved if every one of its child goals succeeds. Figure 1, taken from Padgham and Winikoff (2005), shows the structure of a goal-plan tree of depth 2. The goal-plan tree for our normative GM is described later in Section 3.3.1.

Figure 1 An example of a goal-plan tree of depth 2

![Goal-Plan Tree Diagram](source: Padgham and Winikoff (2005))

The BDI execution is such that the plan selection is performed on-the-fly. In other words, a path through the goal-plan tree is not predetermined as in planning. The plan choice is often determined by a context condition that is evaluated at the time of plan selection. However, there are situations where multiple plans are applicable to achieve the goal.
When this occurs in the GM agent of Luong et al. (2013), the plan is selected at random. The work presented in this paper provides a mechanism for selecting a plan that satisfies the preferences of the player, when there is more than one plan applicable.

2.1.1 JACK intelligent agents

JACK Intelligent Agents™ (Busetta et al., 1999; Winikoff, 2005) by agent-oriented software (http://aosgrp.com/products/jack/) is an agent-oriented, Java-based language for the development of autonomous software. It incorporates BDI concepts such as beliefs, goals (referred to in JACK as a type of event) and plans and allows for the development of complex intelligent agents. JACK is specifically designed to be familiar, integrable, scalable, robust, documented and supported so as to be suitable for commercial and industrial applications.

JACK agents are built by defining (amongst other available concepts) the agent, the events which it can handle, and the library of plans which it can use to handle those plans.

- The agent object is extended to develop new agents by adding custom methods and members relevant to the problem domain. The events which the agent can handle and the plans it can use to do so are also declared in the agent definition.

- Events have numerous types, but are used primarily in our work for instantiating goals. Posted events are handled by selecting plans from the applicable set (i.e., plans which are relevant and can be used to handle the event). Further customised reasoning can be implemented using a metaplan, which can determine which plan is selected from the applicable set.

- Plans are used to implement the handling of events and are comprised of a number of reasoning methods. They can include methods to determine under what contexts and conditions they are relevant. If the plan is selected the body method of the plan is executed and pass and fail methods can be defined to be run depending on the outcome of the plan.

While JACK features its own syntax for agent declarations, standard Java code can be used in the methods within, allowing ease of integration with Java-based systems. The JACK compiler compiles the agent, event and plan files into Java files which can then be built into the final executable system or library with the standard Java compiler.

2.2 Norms

In the area of multi-agent systems, norms are used as constraints on the agent behaviours and are either personal or social. Personal norms are those that are regulated endogenously (i.e., the agent will enforce its own norm compliance) whilst social norms are regulated exogenously (i.e., enforced by entities other than the agents they apply to) via organisations (Hubner et al., 2006) or institutions (Esteva et al., 2002a, 2004), for example. Social norms are most common in practice which gives rise to issues such as norm emergence, and norm discovery.

In this section, we will briefly explore some concepts related to norms along with seminal work in the field.
2.2.1 Early work on norms

Norms in artificial intelligence originated from the concept of social norms and had early applications in social simulations. They were used to facilitate greater cooperation and to lower aggression (Castelfranchi et al., 1998) between agents. These simulations modelled other social concepts such as reputation, however, these agents were non-general in that their internal logics were pre-determined.

Other precursors to normative research on intelligent agents dealt with using protocols to manage multiple agents who shared common goals. Dignum (1999) argued that these techniques infringed on agents’ autonomy as the protocols dictated their behaviour and limited their ability to respond to environmental changes. He went on to propose the use of deontic logic to specify norms instead; rules which can be violated if the agent deems that doing so is necessary to achieve its own goals. This was later built into a generic framework (Castelfranchi et al., 2000) which described the management of norms and the way in which they interact with other BDI concepts. Here, norms can be used to both trigger the generation of new goals and influence the selection among existing goals. They also affect the generation and selection of plans in a similar way.

Another way of modelling norms was proposed in the beliefs, obligations, intentions, desires (BOID) architecture (Broersen et al., 2001). A BOID agent incorporates obligations as part of its internal feedback loop to consider the implications they will have on possible actions. The paper also proposes that a greater number of agent ‘types’ are possible with obligative reasoning, depending on the manner in which conflicts are resolved. For example, a ‘single-minded’ agent would only consider both obligations and desires when deliberating, whereas a ‘selfish’ agent would prioritise only desires.

2.2.2 Normative organisations

The formal framework of ISLANDER (Esteva et al., 2002a, 2002b) defines an electronic institution in terms of roles, scenes and norms, in which different types of agents can interact with each other. Here, norms are used as a way of specifying interactions which may not be detrimental to the agent. Whilst these norms can specify an undesirable outcome for the agent, this usage contrasts to other interpretations where norms have a sanction attached to them to specifically encourage adherence to an obligation or prohibition.

The AMELI infrastructure (Esteva et al., 2004) facilitates the execution of ISLANDER institutions and allows external agents to participate, handling (amongst other features) constraints such as maximum agent numbers. This framework is more concerned with the enforcement of such constraints than of norms, and does not explore norm violations in any detail.

Another approach is implemented by a middleware called SM (Hubner et al., 2006) which facilitated the interaction of MOISE organisations with agents developed in any architecture. It specifies both unbreakable hard constraints and soft constraints which agents may choose to violate, however, there is no in-built mechanism for dealing with such violations.
2.2.3 Norm-aware agents

The $\mathcal{J} - \mathcal{M}OISE^+$ architecture (Hubner et al., 2007) augments the Jason agent framework with support for providing organisational information to agents and allowing them to interact with these organisations. These interactions are dictated by the $\mathcal{S} - \mathcal{M}OISE^+$ middleware, which allows soft constraints to be broken, yet the underlying agent deliberation on whether to adhere to these norms or not is not explored.

Meneguzzi and Luck (2009) implemented an approach to real-time agent reasoning in AgentSpeak(L) whose major contribution was a mechanism by which agents could modify their existing plans in order to comply with norms if they choose to do so. They focused more on the implications of norms for the individual agent and their capabilities rather than overall multi-agent systems; however, the system has no means for reasoning about conflicting norms.

Another framework called N-2APL was presented by Alechina et al. (2012) which incorporates BDI-based specifications such as beliefs, goals and plans, with normative concepts such as obligations, prohibitions and sanctions. It allows agents to deliberate on whether to adhere to or to violate norms, including support for temporal factors such as deadlines and durations. However, this reasoning is based on priorities assigned to the agent’s goals and to the norms which the agent is subject to as opposed to such factors as the undesirability of the sanction itself and outcomes which would arise from said sanction. A more complete architecture has since been proposed which integrates N-2APL with a norm specification language 2OPL (Dybalova et al., 2013) via a JavaSpaces tuple space. This implementation allows norms to be activated (detached) and to expire at run-time and monitors and enforces these norms, applying sanctions in cases of violation.

van Riemsdijk et al. (2013) developed a framework whereby agents could reason about norms which were not known at the time of design. In their system, norms are defined in terms of the operational semantics by which they can be adhered to. This essentially describes to the agent how to adhere to the norm, allowing agents to deal with norms which they have not been programmed to deal with. This approach however, impacts the agent’s autonomy as they are essentially directed as to how to behave by these norms and normative reasoning incorporating the agent’s own goals is not discussed.

A very recent addition to the norm-aware agent language family is Lee et al.’s (2014) N-Jason. They extended the concepts of run-time norm compliance to allow agents to reason about new norms which they may not have encountered before and may not have specific plans to address. This differs to previous work which largely triggers specific plans or actions to deal with norms as they are activated, however, the approach is still limited in that it can only deal with norms defined in terms of the agent’s recognised plan ontology and syntax.

2.3 Preferences

Our system uses preferences to describe the kind of game scenario which will be presented to the player. In the planning domain preferences refer to the desirability of certain outcomes over others. These preferences may relate to desired states, actions or overall plan properties and may be ordered and/or dependent on certain conditions.
Preferences are typically specified in terms of certain properties which different states and actions satisfy and can be given numerical values to express measures of desirability.

2.3.1 Quantitative preferences

Boutilier et al. (2004) identified the problem of eliciting preferences from users as difficult as these users may be unable or unwilling to provide adequate preferential information for the system. They developed a qualitative graphical representation model, CP-nets, for specifying user preferences for the outcomes of decisions. This model included support for orderings of preferences and conditions under which these preferences are applicable. The representation of the preferential model is a directed graph on which outcomes are positioned highest to lowest in order of ascending preference (i.e., outcomes with higher preference appear at the bottom of the graph). The model uses conditional \textit{ceteris paribus} (all else being equal) semantics and is qualitative, with each outcome ordered based on conditions.

The preference specification language \textit{PP} was developed by Son and Pontelli (2040) and implemented in an action set planner (Lifschitz, 2002) which represents the planning problem as sets of actions, fluents (time dependent relations), predicates describing interactions between the world and the initial state. The planner then translates the problem into a logical problem described by a set of domain-dependent and domain-independent rules. The \textit{PP} language allows the user to specify a number of different types of preferences, namely:

- \textit{State preferences:} States with certain properties are preferred over states without these properties.
- \textit{Action preferences:} Certain actions are preferred over others when they are relevant to achieving the goal.
- \textit{Trajectory preferences:} Trajectories (solutions) that satisfy certain properties are preferred over others which do not.
- \textit{Multi-dimensional preferences:} Preferences are provided in an ordered set and solutions which satisfy more favourable preferences are more desirable than those which satisfy less favourable preferences.

The framework is declarative and allows users to specify complex, high level preference combinations without having to encode them into the planning system. The system supports the following logical connectives when specifying preferences given two preference formulae $\varphi_1$ and $\varphi_2$:

- $\varphi_1 \land \varphi_2$: The user prefers both $\varphi_1$ and $\varphi_2$ to be satisfied.
- $\varphi_1 \lor \varphi_2$: The user prefers either $\varphi_1$ or $\varphi_2$ to be satisfied.
- $\neg \varphi_1$: The user prefers $\varphi_1$ not to be satisfied.
- $\text{next}(\varphi_1)$: The user prefers $\varphi_1$ occurs next.
- $\text{until}(\varphi_1, \varphi_2)$: The user prefers $\varphi_1$ to be satisfied until $\varphi_2$ is satisfied.
- $\text{eventually}(\varphi_1)$: The user prefers $\varphi_1$ to be satisfied by the end of the plan execution.
2.3.2 Quantitative preferential reasoning

\( \mathcal{PP} \) was extended by Bienvenu et al. (2006) to create their planner \( PPLAN \) by adding quantifiers, variables, non-fluent relations, a conditional construct, and aggregation operators (AgPF). Quantifiers and variables provide support for numeric preferential specification and reasoning. Whereas in \( \mathcal{PP} \), preference formulas were ordinal (i.e., one preference could be defined as more important to satisfy than another), here the relative differences between preference formulas can be quantified and thus incorporated into reasoning. Preference formulae are given values from a totally ordered set, \( \mathcal{V} \), with a minimum and maximum value. The authors use \( \mathcal{V} = [0, 1] \) for their examples, however non-numerical ordered sets can be used at the cost of some AgPF functionality.

\( PPLAN \) classifies preference formulae into four different categories, each being a subset of the next:

- **Basic desire formula (BDF):** A preferred situation specified by the syntax of \( \mathcal{PP} \) as summarised above.
- **Atomic preference formula (APF):** A BDF which is assigned a value \( \subseteq \) from \( \mathcal{V} \) to express its degree of preference over alternatives.
- **General preference formula (GPF):** An APF which may express conditionals and/or general conjunctions and disjunctions. For instance, a GPF with a general conjunction may state a preference for maximising satisfaction of both of two other preferences.
- **Aggregated preference formula (AgPF):** A GPF which may utilise numerous aggregation methods as summarised below.

AgPFs add support for more complex preferential relations over GPFs \( \Psi_1, \ldots, \Psi_n \) by evaluating the preference formulas which comprise them. These aggregation methods include:

- \( \text{lex}(\Psi_1, \ldots, \Psi_n) \): The GPFs are ranked in lexicographical order based on the value they produce within \( \mathcal{V} \).
- \( \text{lexand}(\Psi_1, \ldots, \Psi_n) \): \text{lexand} will try to satisfy all of the lexicographically ranked GPFs, but if it cannot do so it will prioritise those with higher values within \( \mathcal{V} \).
- \( \text{lexor}(\Psi_1, \ldots, \Psi_n) \): \text{lexor} will try to satisfy any of the lexicographically ranked GPFs, prioritising those with higher values within \( \mathcal{V} \).
- \( \text{sum}(\Psi_1, \ldots, \Psi_n) \): The satisfaction levels of the GPFs comprising the formula are summed, effectively attempting to maximise the average level of their satisfaction. This is only possible if \( \mathcal{V} \) is numeric.

We based some of our reasoning methods on these AgPF operations as described in Section 3.3.5. Preferences have been previously implemented in a BDI context with similar principles such as in Visser et al. (2011, 2016), where preferences could be specified in relation to plan properties. These properties provide more detailed information about what exactly will happen when a plan is used to achieve a goal. In the introductory example, a plan involving catching a train to work may set a variable to 100 e.g., \( \text{transportMethod.train(100)} \). Numerical plan property values are then compared to
preferences to sort the plans applicable to the goal in terms of how well they suit the preference set.

Visser et al. (2011, 2016) also deals with the upward propagation of plan properties through the tree so that any reasoning about plan properties takes into account the properties of the lower level plans which may need to be carried out in order to satisfy a plan’s sub-goals. This issue is also present in our work and we propagating plan properties in a similar fashion to the method described in the paper.

Another BDI implementation of preferential reasoning is BDI agent with objectives and preferences (BAOP), developed by Dasgupta and Ghose (2011). This framework allowed for the specification of both hard constraints and soft constraints. Hard constraints must be adhered to, yet have related objectives such as minimising cost, while soft constraints are preferences which are desirable to satisfy but not essential. BAOP includes support for adding preferences at run-time through use of a ‘preference store’. This store is updated when a new preference is introduced as an event and is monitored for inconsistencies. An inconsistency occurs when a new preference which is logically equivalent to an existing preference is inserted and in such cases the new preference replaces the existing one.

2.4 Flexible storytelling

The issue of flexible storytelling has been targeted by a many studies approaching different aspects of the problem such as automatic generation of content; use of virtual characters and player profiling (Riedl and Bulitko, 2013).

2.4.1 Automatic content generation

A common theme among flexible narrative studies is the balance between quality, coherent narrative and player agency. As the player is given a wider range of choices or possible actions, the number of possible narrative outcomes which an author must account for grows very quickly. Therefore, interactive narrative approaches often implement some variation on the concept of a ‘director’ (Bates, 1991) agent whose role it is to take authored content and arrange at run-time in reaction to player choices. In these implementations, the authored content is generally translated into a graph or tree representation, where nodes represent the possible narrative states (or ‘plans’ as they are termed when using planning methods) and edges or branches describe the transitions possible between these states. Narrative paths are then generated by the use of searching and planning algorithms on the narrative structure based on environmental states and player actions.

An extension of this type of framework was implemented by Porteous et al. (2010) who used state constraints to add further information to plans in the narrative structure which the planner used to select plans which would enhance the richness, pace or suspense of the narrative. The constraints take the form of requirements that other plot points have already occurred or will occur at some time during the execution, ensuring that the story is coherent.

One common problem in these approaches occurs when the player performs an action which will interfere with future plans the director agent has selected. For instance, if the player takes an item that another character requires to execute a plan which has been selected for later in the narrative, that narrative trajectory will fail. One solution to this
issue was implemented in the Mimesis (Young and Riedl, 2005) behaviour generation architecture. The process involves checking the player’s input commands for conflicts which may arise resulting from the action being attempted before updating the game state. If a conflict is detected it is dealt with via either intervention, where the player’s attempted action simply fails to execute with a realistic reason (e.g., the player’s gun jams), or accommodation, where plan modification or even re-planning occurs to ensure the narrative can continue.

The Façade (Mateas and Stern, 2003) environment attempts to find a middle ground between explicitly crafted and automatically or procedurally generated game content. It combines a narrative director (here termed a drama manager) with character agents which use reactive planning and are capable of sequential, parallel and simultaneous actions. The narrative is constructed of around 200 short interaction sequences between characters called ‘beats’ which contain preconditions and effects on the narrative state and together form a highly complex, intractable graph of possible story arcs. While the quality of the produced narrative is highly dependent on the author of the individual beats, Façade’s strength lies in its ability to generate a vast array of dramatic and coherent storylines with a well constructed arc of tension.

### 2.4.2 Virtual characters

The goal of virtual characters is to enrich the narrative with intelligent, believable actors for the player to interact with. These actors operate autonomously and independently from the director agent and build personal and emotional engagement in the player.

One such example is (Cavazza et al., 2002, 2003) employment of virtual actors who use planning to achieve their goals and to react to environment conditions. One scenario modelled a social situation via a hierarchical task network (HTN) in which the agent would seek other paths to the goal upon plan failure resulting from interference from other agents and users. Another, limited the agent’s reasoning about the effects arising from its actions other than goal satisfactions, resulting in situations where the agent failed subsequent plans due to earlier actions.

Another application for emergent narrative is education, where some learning objectives require emotional rather than purely knowledge-based material. The Fear Not! (Aylett et al., 2005) application presents an interactive drama aimed at children, where participants give advice to agents around the issue of bullying and then view the outcomes. The approach was shown to make the agent characters more believable in an attempt to increase emotional reactions in the users.

Barber and Kudenko (2009) modelled actor’s traits, relationships and principles along with sets of possible actions and dilemmas. These dilemmas are then selected based on the current state of the environment and a planner carries out the character actions necessary to bring the dilemma about. The user is then prompted to react, after which the state is updated and the cycle runs again. The user also is modelled as an actor and thus the user can be a participant in dilemmas which arise. The user model also allows the system to adapt to the users actions over time.

### 2.4.3 Player profiling

Other work on interactive storytelling has focused on gathering information about the player’s preferences to influence the narrative director’s future selection of events. Thue
et al. (2007) presented interactive storytelling as a decision making problem then proposed player modelling as a decision making technique. Their PaSSAGE interactive storytelling system profiles players by weighting their tendency towards five different play styles: fighters; power gamers (who prefer collecting powerful items and wealth); tacticians; storytellers and method actors (who like to take dramatic actions). These weights are updated based on player choices and then taken into account during director reasoning for the selection of events or encounters.

The case-based drama manager C-DraGer (Sharma et al., 2010) builds a library of play-through cases by mapping the sequence of plot points to feedback taken from the player of that sequence. The feedback is about the player’s feeling about how interesting different events of the play-through were to them and is taken upon completion of their play session. When a new player plays the game, C-DraGer performs case-based reasoning, comparing the current player’s sequence of events to the library of cases to produce a numeric ‘interestingness’ score for each possible future plot point. These scores can then be used to present event sequences to the player which they are likely to find interesting.

As below with norms, these preferences will be specified in a text file which the GM will read in at the start of its execution.

3 System architecture

Our architecture is comprised of a number of different aspects. First, there are the norm and preference specification languages which we have developed with the goal of making them human readable and easy to learn. Then, there is the GM agent itself, into which the specification languages were integrated, along with additional information and reasoning methods with which the GM can act upon them. Figure 2 illustrates the overall communications between architectural components.

Figure 2  High level system architecture

3.1 Gameplay preference specification

Preferences will be used to describe the type of experience the game will present to the player. The intention is that a designer is able to produce a range of different styles of game with the same GM agent by specifying those styles in terms of these preferences.
Applying norms and preferences for designing flexible game rules

For example, if the designer wishes the game to be an action RPG full of combat, they would describe a preference for plans involving violence.

These preferences relate to themes which apply to potential in-game events and gameplay outcomes, i.e., arbitrary interactions between the player and the environment, potentially involving other computer-controlled characters. These themes include: violence, persuasion, bribery, exploration and stealth. Another dimension is added in terms of the moral implications of the plan, classifying plans involving good-natured player actions as paragon and plans involving ‘evil’ behaviour as renegade. The preference specification syntax is defined in the preference ontology depicted in Figure 3.

Figure 3 Preference specification ontology (see online version for colours)

<table>
<thead>
<tr>
<th>Preferences</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Usage</strong></td>
</tr>
<tr>
<td>[property] [weighting (1-10)]</td>
</tr>
<tr>
<td><strong>Properties</strong></td>
</tr>
<tr>
<td>Outcomes</td>
</tr>
<tr>
<td>Violence</td>
</tr>
<tr>
<td>Instances of combat</td>
</tr>
<tr>
<td>Exploration</td>
</tr>
<tr>
<td>Searching the environment for objects or characters</td>
</tr>
<tr>
<td>Stealth</td>
</tr>
<tr>
<td>Theft, deceit or carrying out actions without the knowledge of key characters</td>
</tr>
<tr>
<td>Bribery</td>
</tr>
<tr>
<td>Using gold to buy goal outcomes</td>
</tr>
<tr>
<td>Persuasion</td>
</tr>
<tr>
<td>Using the persuasion skill to convince or deceive characters</td>
</tr>
<tr>
<td>Paragon</td>
</tr>
<tr>
<td>Using peaceful tactics to elicit outcomes such as doing favours</td>
</tr>
<tr>
<td>Renegade</td>
</tr>
<tr>
<td>Performing actions which will harm peaceful characters such as stealing or attacking</td>
</tr>
</tbody>
</table>

We define a preference as a specified theme with a numerical weighting out of 10. This definition is a simpler version of the similar concepts in Visser et al. (2011, 2016) and Bienvenu et al. (2006) which allow for more detailed specifications such as preferences which need only be satisfied once or until some condition has been met. In contrast, our preference system assumes that the specified preferences are applicable throughout the execution of the game. Additionally, the above implementations order preferences based on a numerical value V for which lower values are more desirable, whereas our system considers higher values of V preferential over lower values for the purposes of calculations involving probabilities which are described in Section 3.3.3.

3.2 Specifying game rules as norms

Norms will be used to dictate game behaviour by describing some rules for the game. In this work, the norms are specified outside of the GM agent by the user, but enforced internally by the agent itself at run-time, treating them as personal norms. Given that the norms are defined by the user, the norms are fixed for a particular execution of the GM. So, they could also be viewed as regimented norms (Alechina et al., 2013). The sanctions however, are actions that affect the player, via the narrative generated by the GM.

We specify a normative rule as: <type, action, sanction>, where type is either prohibit or oblige. Prohibit defines an action that must not occur, if it does, then the sanction is applied. Oblige defines an action that must occur, and if it fails, then the sanction is applied. As developing a generalised normative language is outside the scope of this
project, a list of possible normative options is provided for the user to create their own specifications. These options are defined in the ontology detailed in Figure 4, which also specifies some examples.

**Figure 4** Norm specification ontology (see online version for colours)

<table>
<thead>
<tr>
<th>Norm Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Prohibit</strong></td>
<td>Succeeding in the action triggers a sanction</td>
</tr>
<tr>
<td><strong>Oblige</strong></td>
<td>Failing the action when directed to do so by the GM triggers a sanction</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Norm Actions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defeat</td>
<td>The player defeats another character in combat</td>
</tr>
<tr>
<td>Talk</td>
<td>The player talks to a character</td>
</tr>
<tr>
<td>Persuade</td>
<td>The player uses the persuasion skill</td>
</tr>
<tr>
<td>Bribe</td>
<td>The player uses gold to buy an outcome from a character</td>
</tr>
<tr>
<td>Buy</td>
<td>The player uses gold to buy an item from a character</td>
</tr>
<tr>
<td>DoFavour</td>
<td>The player does a favour for a character</td>
</tr>
<tr>
<td>Deceive</td>
<td>The player lies to or tricks another character</td>
</tr>
<tr>
<td>Steal</td>
<td>The player steals an object</td>
</tr>
<tr>
<td>Find</td>
<td>The player searches for and finds an item or character</td>
</tr>
<tr>
<td>Bash</td>
<td>The player breaks an object open</td>
</tr>
<tr>
<td>Lockpick</td>
<td>The player picks the lock on an object</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Norm Sanctions</th>
<th>Description</th>
<th>Associated Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>GuardAttack</td>
<td>A group of Renegade guards spawns and attacks the player</td>
<td>Violence, Peril, Danger</td>
</tr>
<tr>
<td>Bloodstained</td>
<td>The player must defeat a bloodstained character who is threatening a patron</td>
<td>Peril, Danger</td>
</tr>
<tr>
<td>FightDrunkard</td>
<td>The player must defeat a drunkard who is provoking them</td>
<td>Renegade, Violence</td>
</tr>
<tr>
<td>DeFavourFind</td>
<td>The player must find an item for a patron to continue the story</td>
<td>Peril, Exploration</td>
</tr>
<tr>
<td>DeFavourReclaim</td>
<td>The player must steal an item which has been taken from a beggar to continue</td>
<td>Peril, Steal</td>
</tr>
<tr>
<td>StealItem</td>
<td>The player is blackmailed by a rogue and must steal an item from a beggar to continue</td>
<td>Renegade, Steal</td>
</tr>
<tr>
<td>PayBribe</td>
<td>The player must pay a bribe to avoid arrest by a guard</td>
<td>Renegade, Bribery</td>
</tr>
<tr>
<td>ResolveDispute</td>
<td>The player must resolve a dispute between 2 patrons in a dialogue sequence</td>
<td>Peril, Persuasion</td>
</tr>
<tr>
<td>PersuadeGuard</td>
<td>The player must talk their way out of incarceration in a dialogue sequence</td>
<td>Persuasion</td>
</tr>
<tr>
<td>ThreatenWitness</td>
<td>The player has been threatened by a boy and who they must threaten into silence</td>
<td>Renegade, Persuasion</td>
</tr>
<tr>
<td>FindGold</td>
<td>The player discovers they have dropped some gold and must find it to continue</td>
<td>Exploration</td>
</tr>
<tr>
<td>GiveGold</td>
<td>The player must give some of their gold to a beggar to continue</td>
<td>Peril, Bribery</td>
</tr>
<tr>
<td>WeaponConfiscated</td>
<td>The player's weapon is taken off them by a guard</td>
<td>Violence -1</td>
</tr>
<tr>
<td>Poisoned</td>
<td>The player is hit with a poison dart which drains their health for a time</td>
<td>Violence -1</td>
</tr>
<tr>
<td>Silence</td>
<td>A mage casts silence on the player and they are unable to talk to characters for a time</td>
<td>Bribe -1, Persuasion -1</td>
</tr>
<tr>
<td>Alert</td>
<td>An alarm sounds and characters in the scenario become alerted, making sneaking impossible</td>
<td>Steal -1</td>
</tr>
<tr>
<td>LoseGold</td>
<td>A thief runs through and steals the player's gold</td>
<td>Bribe -1</td>
</tr>
<tr>
<td>Injury</td>
<td>A bloodstained character brings the player's leg. Then leaves, the player's movement is slowed for a time</td>
<td>Exploration -1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Examples</th>
<th>Definition</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prohibit</td>
<td>Deceive, ThreatenWitness</td>
<td>If a player deceives or threatens a victim, they will be witnessed by a boy who they will have to threaten to ensure they keep quiet</td>
</tr>
<tr>
<td>Oblige</td>
<td>Steal, WeaponConfiscated</td>
<td>If the player fails to steal an item (e.g. gets caught pickpocketing) then a guard will appear and take their weapon</td>
</tr>
</tbody>
</table>

As illustrated in the ontology, each sanction is assigned properties in the same manner as other plans as described in Section 3.3.2, with each property in the figure assumed to have a value 1 unless omitted or stated otherwise. However, the properties with values of –1 represent a distinction from other non-sanction plans and this is elaborated on in Section 3.3.4.

In our work, the norms specified by the user are applicable to the player only. This is because the NPCs in our game scenario are not autonomous and would either not be able to partake in the sanction events, or these events would mostly not be applicable to their scripting. For example, the NPCs are not programmed to navigate dialogue trees between each other therefore they cannot converse in any meaningful way, as some of the sanction
Applying norms and preferences for designing flexible game rules

3.3 The GM agent

The GM’s core structure is the goal-plan tree, which maps goals to the plans which handle them and plans to their sub-goals. We have added property summary information to the tree and developed algorithms for propagating this information throughout it in a similar fashion to Visser et al. (2011). Finally, we developed the reasoning mechanisms themselves, which we have developed so that the GM can reason about preferences, norms and their interactions to attempt to produce desirable gameplay for the player.

3.3.1 Goal-plan tree

The narrative possibilities from which the GM can select are contained within the agent’s goal-plan tree, the concepts of which are described in Section 2.1. Figure 5 shows the full goal plan tree for our normative GM agent, complete with the themes and actions which are relevant to each plan. Upon execution in Luong’s implementation, the main goal is posted and the agent then selects a course of action by selecting plans from the goal-plan tree to satisfy that goal.

Figure 5 The goal-plan tree for the normative GM with property information (see online version for colours)

3.3.2 Property summary information

In order for our agent to reason about plan selection based on the designer’s preferences and norms a property summary is added to each plan in the tree. We follow the work of Visser et al. (2011) by adding summary information which details what properties are applicable to the plan. In our case, these properties refer to either:
- **actions**: taken from the possible actions in the norm specification ontology detailed in Figure 4 and are required for normative reasoning.
- **themes**: taken from the possible themes in the preference specification ontology detailed in Figure 3 and are required for preferential reasoning.

The properties applying to plans which trigger an in-game event (as opposed to simply posting further goals) are assigned a value of either 1 or 0. These binary values can be interpreted as true or false. Properties are given a value of 1 if they are applicable to a plan as it is certain that these themes and actions will be present in the resulting gameplay. Values for themes and actions which are not applicable to the plan are omitted from the summary and assumed to be 0 as they will not eventuate as a result of that plan being selected.

However, the agent must be able to make a decision at all levels of the tree, even if the plans it has to select from do not directly bring about in-game events. In these cases, properties are not directly applicable to a plan but may be applicable to plans at a lower level of the sub-tree which extends from that plan. Thus, the agent must be given some information about the plans in the levels of that sub-tree in order to perform reasoning about which plan to select at a higher level.

**Figure 6** Example of plan summary information and its propagation throughout the goalplan tree

To facilitate this, we adapt the previous work of Visser et al. (2011) by assigning a probability to the property’s value. This value is the probability that, if the plan is selected, at least one lower level plan from the sub-tree will be subsequently selected which triggers an in-game event that satisfies the property. It is essentially the probability of a given plan leading to an outcome described by the property.
For example, in Figure 6, the plan \textit{GetNPCToDistractJalek} posts a sub-goal with the same name. There are two plans which handle this sub-goal, one of which involves having the player use the persuasion ability. At least one plan must be selected to handle the sub-goal, therefore if the \textit{GetNPCToDistractJalek} is selected, there is a 50\% chance that the ensuing gameplay will involve persuasion (before preferences are taken into account). Hence, the \textit{GetNPCToDistractJalek} plan is assigned the value of 0.5 for the \textit{persuasion} property.

Note that this probability value is a static calculation which is determined by hand and assigned to variables during implementation. It does not take into account preferences, as the actual probability of a given plan being selected would depend on the norms and preferences specified. However, we have used these probability values (as opposed to other possibilities explored in Section 6.1) because they are a clear way to provide information to the agent so that it can 'look ahead' at the implications of selecting any given plan.

### 3.3.3 Propagating probability information

Plan summary information is calculated for each goal and plan by recursively propagating values from the leaf nodes upwards through the tree. As stated above, the calculation results in a probability that properties will apply to in-game outcomes, however this calculation is done differently for each node depending on if the node is a goal or a plan. This is due to the way the two relate to each other as explored by Visser et al. (2011), namely the AND/OR relationships described in Figure 1.

Propagation to goal nodes is achieved by averaging a property’s child node probabilities for each possible property. We define the probability of a theme \( t \) resulting from selection of a goal \( g \) as \( Pr(g,t) \) in formula 1, where \( \text{children}(g) \) returns the child nodes of \( g \) and it denotes the \( i \)th child of goal \( g \) given theme \( t \).

\[
Pr(g,t) = \sum_{i \in \text{children}(g)} \frac{Pr(i,t)}{|\text{children}(g)|}
\]  

For example, in Figure 6, the theme probabilities for the goal \textit{GetNPCToDistractJalek} are calculated as follows:

- **Bribery**: \( 1:0 + 0:0 = 2 = 0.5 \)
- **Persuasion**: \( 0:0 + 1:0 = 2 = 0.5 \)
- **Paragon**: \( 1:0 + 1:0 = 2 = 1:0 \).

The goal \textit{GetNPCToDistractJalek} has two plans that could possible handle it, one which involves ‘bribery’. Hence, we deduce that the goal will have 50\% chance of triggering a gameplay involving ‘bribery’ (likewise for ‘persuasion’). However, since both of the plans contain the ‘paragon’ theme, we deduce goal has a 100\% chance of producing a gameplay with that theme.

Propagation to a plan node is more involved as it must account for AND connections between goals when calculating probabilities. Here, we must consider the probability formula for non-mutually exclusive events as detailed in Devore (2011) and Chow and Teicher (2003), which is the complement of the product of the complements of the probabilities. We define the probability of a theme \( t \) resulting from the selection of a plan
l as $\Pr(l)$ in formula 2 details how the probability is derived, with $l$ being the plan under evaluation and $t$ being one of the possible properties:

$$\Pr(l) = 1 - \prod_{i \in \text{children}(l)} (1 - \Pr(i))$$  \hspace{1cm} (2)

For example, in Figure 6, the action probabilities for the plan `Persuade Christov To Give Key` are calculated as follows:

- **Persuasion**: $1 - (1 - 0.75)(1 - 1.0) = 1.0$
- **Bribery**: $1 - (1 -0.25)(1 - 0.0) = 0.25$.

For the plan `Persuade Christov To Give Key` to succeed both of its subgoals must succeed. The subgoal `Persuade Christov` will definitely result in ‘persuasion’. So, irrespective of how `Get Jalek Away` is handled, the plan `Persuade Christov To Give Key` has a 100% chance of a gameplay involving persuasion. However, there is only a 25% chance that ‘bribery’ will occur as the goal `Persuade Christov` does not involve this action and thus the 0.25 probability of it occurring as a result of handling the goal `Get Jalek Away` remains.

### 3.3.4 Sanction properties

As stated above, plans in the goal-plan tree which result in in-game events are assigned a probability of either 0 or 1. However, plans which are executed as the result of a sanction may have another possible value, –1. Instead of being a probability, this value indicates that, while no themes are applicable to the gameplay outcomes of the plan, the outcomes are detrimental to the player’s ability to engage in certain behaviour.

For example, the `Weapon Confiscated` sanction leaves the player unarmed, thus severely impeding their ability to engage in violent actions. Therefore, this sanction is assigned the `violence` property with a value of –1, allowing the agent to reason about the implication of the player to triggering this sanction on any further plans involving violence.

### 3.3.5 Preferential and normative reasoning

When the GM is executed, the root goal of the goal plan-tree is posted and the agent begins traversing this tree, executing plans to address goals and posting their sub-goals in turn. For each goal that is posted, the GM evaluates the applicable plans which handle the goal and calculates a score for each of them. This score is calculated based on: the preferences specified; the inherent plan properties (actions and themes); and the properties of any norms which may be violated by the player in carrying out the plan.

First, the GM checks the actions (as defined in Figure 4) which the plan entails against the set of user specified norms. If any of these actions match an action which is *obliged* or *prohibited* through a specified norm, then that norm is considered to be *applicable*. For each applicable norm, the themes which are associated with the sanction are added to the set of the themes for the plan. These probabilities are then aggregated to calculate the probabilities for themes which appear more than once, again using the complement of the product of complements as we use for propagating plan summary information.

Once the set of plan themes has been aggregated, the GM compares this set to the preferences specified by the user. The GM finds the intersection of themes which appear
Applying norms and preferences for designing flexible game rules

both in the specified preferences and the set of plan themes, and for each theme, a value is calculated. This value is the product of the preference weight and the probability of that theme in the plan summary information. The GM then sums the values for all themes in the intersection to produce a score for the plan. We define the score for a plan $l$ as $S_l$ in formula 3, where: $r$ is the set of specified preferences; $\text{themes}(x)$ returns the set of themes relevant to a plan or a set of preferences $x$; $\text{weight}(r_i)$ returns the weight for a theme $i$ as specified within a set of preferences; and $\text{Pr}(l)$ is the probability of a theme $i$ resulting from selection of a plan.

$$S_l = \sum_{i \in \{ \text{themes}(r)=\text{themes}(l) \}} \text{weight}(n) \times \text{Pr}(l_i)$$

(3)

The GM ignores themes not present in the intersection of preference and plan themes. This is because themes not specified as preferences are assumed to have a preference weight of 0 and similarly themes not appearing in the plan properties are assumed to have a probability of 0. In both cases, the theme would contribute a value of 0 to the overall score.

Once the scores for all applicable plans have been calculated the GM simply selects the plan with the highest score and executes it, applying any in-game effects and posting the plan’s sub-goals if it has any.

4 Implementation

For our implementation, we chose NWN as the game for our evaluation as it was the basis for the work Luong in Visser et al. (2011) which we had access to. This made implementing our extensions easier and also allowed for a direct comparison in our evaluation which we describe in the next section. NWN is a third-person role-playing computer game developed by BioWare and published by Atari in 2002. The game is set in a fantasy world, with the game rules based on the Dungeons & Dragons (D&D) 3.0 rule system. NWN includes a game engine, a game campaign (the actual game itself) that can be played as single player or in multiplayer mode, and the Aurora toolset that has made the game extremely popular. It has also been employed in research work (Thue et al., 2008; Trenton et al., 2010) since it is still probably the best (almost) freely-available toolset for computer role-playing games to date.

Our implementation was undertaken in a number of stages. First, the GM was extended with a number of classes for reading and storing norms, preferences and plan summary information. Then, we built on these to add the information propagation algorithms and finally, implemented the normative and preferential reasoning mechanisms.

4.1 Server framework

To construct his implementation, Luong used a number of pieces of software to build a bridge between the Java-based GM environment and the NWN Linux Server. The Linux server (updated to its latest version 1.69) was selected over its equivalent Windows software due to the greater amount of available extensions to facilitate this construction. The NWN eXtender (http://www.nwnx.org/) was installed on the server which allows for
the execution of C++ extension modules via the Linux server, providing functionality such as SQL database support and performance monitoring.

The NWNX Java virtual machine (JVM) plugin (https://github.com/NWNX/nwnx2-linux/tree/master/plugins/jvm) was then installed to make the final connection of the bridge between NWNX and the JACK GM package. The NWNX JVM plugin allows for the calling of JVM event calls from within NWScript and vice versa thus facilitating communication between the Java-based JACK module and the Linux NWN server.

4.2 Luong’s GM

The original GM agent was prototyped with the Prometheus design tool (PDT) (Padgham et al., 2005, 2007; Sun et al., 2010), a toolset which facilitates the construction of agents with the Prometheus methodology (Padgham and Winikoff, 2003). The PDT allows users to graphically design agents, including their goal-plan trees, and then export skeleton JACK code in the form of plan and event files and this is how the GM began.

The JACK language calls goals events and, once an event is posted, the agent attempts to select a plan to deal with this event. However, in Luong’s implementation, the selection of plans was done randomly, as opposed to using intelligent reasoning. This still produced a variety of different gameplay threads, however for our purpose, we replaced the randomisation methods with normative and preferential reasoning.

4.3 Extending the GM

We created the NormHandler class which reads in the user’s preferences and norms, checking them for errors against the set of available options as defined in the ontologies. We also implemented a PropertyHandler class which stored the plan properties and handled propagation of plan summary information. Each of these classes allowed the reasoning methods to query them for preference, norm and property information.

We added an event for each of the sanctions and a plan to handle each of these events. These plans contained the code to present the sanctions in-game so that, to trigger a sanction, the GM needs simply post the sanction’s event like any other.

A SanctionHandler class was added to enforce the norms and trigger sanctions. When the agent selects a plan which may violate a norm, that norm is ‘activated’ and added to an active norm list in the sanction handler. Upon the success or failure of any plans, the GM sends the outcome to the SanctionHandler class, which posts sanction events if norm violations are found. These events are only posted one at a time and active norms are only removed once the sanction plan is completed, effectively queuing sanctions if more than one has been triggered. Once a sanction plan is completed, the SanctionHandler is checked for any further queued sanctions, and this queue must be empty before the GM continues with the story.

Next, we implemented the PlanSelector class, which contained the reasoning methods executed by the GM each time it had to select a plan to handle a posted event.

4.4 Reasoning implementation

For the PlanSelector class, the mechanism we used for the implementation of normative and preferential reasoning within JACK is called a ‘meta-plan’. A meta-plan can be used in cases when the JACK agent has multiple applicable plans it can choose to achieve a
Applying norms and preferences for designing flexible game rules

The meta-plan is defined in the agent like any other plan except that, instead of handling an event posted explicitly by the agent, it handles the ‘PlanChoice’ event. PlanChoice events are posted whenever an event is posted which multiple plans handle and when more than one of these events is applicable in context.

From within the meta-plan, the PlanChoice event can be queried to produce an applicable set of plans from which the meta-plan must make a choice. This applicable set is a list of plan signatures from which the properties of the plan can be accessed. The properties can then be combined with the preferences read from the file to calculate a numerical score for each plan describing how well they adhere to the preference list. Finally, the meta-plan need only select the plan with the highest score.

5 Evaluation

In this section, we detail the methodology used to evaluate our normative GM system. Firstly, we provide a case study which illustrates the inner workings of the new GM, then we move on to an explanation of the user experiment undertaken and analysis of the results. As mentioned above, our implementation was in NWN. In order to be able to compare it directly with the work of Luong et al. we chose the same ‘Hideout Quest’ as also utilised in their work.

5.1 Case study

We include a case study to illustrate the GM’s functionality, reasoning and the resulting gameplay outcomes by way of example. This execution thread\(^1\) was the result of extensive experimentation with combinations of specified preferences, norms and player actions and is representative of the GM’s overall functionality. Please refer to the ontologies in Figures 3 and 4 for details on the preferences and specified, including themes, actions and sanctions.

5.1.1 Setup

The preferences specified for this session were as follows:

- \(\alpha_1\): persuasion 10
- \(\alpha_2\): violence 8
- \(\alpha_3\): stealth 7
- \(\alpha_4\): paragon 10.

Please note that the labels \(\alpha_1\)–\(\alpha_4\) are provided for reference in the case study and were not included in the preference specification file. The norms specified for the case study were as follows:

- \(\beta_1\): oblige persuade DoFavourFind
- \(\beta_2\): prohibit lockpick BloodsailorDuel.
Again, the labels $\beta_1$ and $\beta_2$ are purely for reference. $\beta_1$ can be read as “If the GM directs the player to persuade someone they are obliged to. If the player fails to persuade the character they incur a sanction which involves doing someone a favour”. $\beta_2$ can be read as “The player is prohibited from picking locks. If the player violates this prohibition they must duel with a bloodsailor”.

5.1.2 Execution

Figure 7 displays the storyboard for the execution thread in this case study.

The server is executed and waits for the player to connect. Once the player connects from the Windows client, the GM is executed and reads the preferences and norms. The GM then begins reasoning about relevant plot points to present to the player.

The scenario begins with the player spawning at the entrance to the tavern [Figure 7(b)]. The GM traverses the goal-plan tree attempting first to handle the ‘Gain Access to Hideout Door’ event, calculating scores for each applicable plan along the way. The GM eventually decides on the ‘persuade the chef’ plan as it has the highest score at 20. This is because the plan fits the persuasion theme specified in $\alpha_1$ and will also possibly trigger $\beta_1$ because the persuade action is involved. Therefore, the themes for $\beta_1$ are added to the ‘persuade the chef’ themes, including the paragon theme, which satisfied $\alpha_4$. Due to $\beta_1$, the plan can potentially satisfy both $\alpha_1$ and $\alpha_4$ therefore the score is calculated with the formula below, in which $\text{Probability}()$ is a function which returns the probability of the preference being satisfied according to the plan and $\text{Weight}()$ is a function which returns the weighting specified within the preference:

$$\text{Probability}(\alpha_1) \times \text{Weight}(\alpha_1) + \text{Probability}(\alpha_4) \times \text{Weight}(\alpha_4) = 1 \times 10 + 1 \times 10 = 20$$

Note that, as this is a leaf node, the probabilities of the themes occurring are assumed to be 1. The possible triggering of $\beta_1$ raises the plan score above others under consideration and the player is directed to try and persuade the chef to give them access to the door [Figure 7(c)].

The player is unable to persuade the chef and thus fails the obligation specified in $\beta_1$. This triggers $\beta_1$’s sanction, and the GM directs the player to help a tavern patron find their lost necklace [Figure 7(d)]. The player must complete this task before the story can progress. The player finds the necklace [Figure 7(e)] and returns it to the patron.

The GM then removes the ‘persuade the chef’ plan from the instance of the goal-plan tree as it has failed and re-propagates all the plan summary information. Again, it traverses the tree and settles on the ‘buy the chef a drink’ plan. This plan inherently has the persuasion and paragon themes and thus also satisfies $\alpha_1$ and $\alpha_4$ to give a score of 20. Note that this is the same score as the previously selected plan and the previous plan was selected first purely due to the order of the tree traversal. The GM directs the player to find the chef a drink [Figure 7(f)] and present it to the chef. The chef then drinks the bottle’s contents and falls asleep [Figure 7(g)], giving the player access to the door and completing the plan successfully.
Applying norms and preferences for designing flexible game rules

Figure 7  Storyboard of the case study playthrough, d) patron in need (e) finding necklace (f) finding drink (g) chef drinking drink (h) selecting lockpick skill (i) picking lock (j) bloodsailor threatening beggar (k) duelling with bloodsailor (l) reaching the hideout (see online version for colours)
The GM then begins planning for the ‘open hideout door’ event, again traversing the sub-tree to find the best plan. This time the GM selects the ‘pick the lock’ plan with a score of 25. It arrives at this score due to the plan having the stealth theme and thus satisfying $\alpha_3$ and involving the lockpick action, which may trigger $\beta_2$. The themes of $\beta_2$’s sanction are
therefore added to the plan’s themes, including violence and paragon, which satisfy $\alpha_2$ and $\alpha_4$, respectively. The score is calculated as:

$$
\text{Probability}(\alpha_3) \times \text{Weight}(\alpha_3) + \text{Probability}(\alpha_2) \times \text{Weight}(\alpha_2) + \text{Probability}(\alpha_4) \\
\times \text{Weight}(\alpha_4) = 1 \times 7 + 1 \times 8 + 1 \times 10 = 25
$$

This means the plan is selected over other potentially desirable plans such as ‘Persuade Christov to give key’ which inherently satisfies $\alpha_1$ and $\alpha_4$ with varying probabilities, but does not have a chance of triggering any norms, and thus gives a score of 22.5. Here, we see the GM is attempting to exploit the norms by trying to trigger a sanction which will result in preferred gameplay outcomes in addition to those of the plan itself. In this case, the plan itself only satisfies a preference with a low weighting, however the GM selects it due to the potential sanction violations to produce desirable gameplay.

The player selects the lock pick skill (Figure 7(h)) and opens the door (Figure 7(i)). This violates the prohibition specified in $\beta_2$ triggering the sanction. The GM directs the player to rescue a beggar who is being threatened by a bloodsailor (Figure 7(j)), and this must be completed before the story can continue. The player fights (Figure 7(k)) and defeats the bloodsailor, allowing them to continue into the hideout (Figure 7(l)) thus completing the scenario.

In this play-through, we see how the GM selects plot points for the player to carry out, how these look inside the game environment, and how the GM attempts to exploit norms to achieve preferred gameplay outcomes.

5.2 User experiment

A user experiment was run to validate our hypothesis that the normative GM was to provide a user friendly and human readable method of controlling the behaviour of the GM agent. The experiment was setup in such a way that participants were unfamiliar with the system and given only a brief introduction and explanation of its use. This was done intentionally so that the evaluation allowed us to assess the usability of the system by game designers who were new to using it.

5.2.1 Experimental setup

Our experiment involved a mix of ten male and female participants who had a background in video game design. Given that access to professional video game designers is difficult to achieve, our recruitment focused mainly on game design program graduates. Our final participant pool comprised of eight recent graduates (two of which are currently working at game development studios) and two game designers who are currently in the industry. The criteria for selection was that subjects had to have studied video game design and participated in development of game projects over two years or more. Whilst recruiting designers from the industry would be ideal, we believe that these participants were able to give insights representative of the greater game design community. No subjects had played the game NWN prior to the experiment so some brief instructions were given around the controls and objectives of the game scenario.

Our experiment involved two systems: system A and system B. System A was the original GM on which our system was based while system B was the normative GM which we developed.
The experiment was conducted on a single laptop computer. The System A and B NWN servers were loaded onto an Oracle VM virtual box (https://www.virtualbox.org/) virtual machine (VM). These systems were executed as described below and had their console output redirected to a file to provide opportunity for later analysis gameplay outcomes. The server directory also contained the text files which contained the norms and preferences and which the GM read in upon execution. The participant was only given access to the VM for the purposes of editing these files; the rest of their interactions with the system were from within the NWN game. Participants connected to the NWN server from within the Windows game client and the GM executed as soon as the connection was accepted by the server.

A brief description of the experiment process and timeframe is as follows:

1. The participant was welcomed and given an overview of the experiment process.
2. The participant was then given an explanation of the system, including its purpose and expected behaviour.
3. The participant was given a printed copy of the system’s ontology and a detailed explanation of how the language is used to specify gameplay preferences and norms. This along with the previous steps took around 5 minutes in total depending on how much further explanation was required to help the participant understand the system.
4. The participant was then asked to think of a style of game they would like to create and then attempt to describe it to the system using the language described in the ontology. The participants were directed to specify 2 or more preferences and 2 or more norms. This step took around 3 minutes.
5. The participants then played through system A and system B. Users were not told which system was which and half of the experiments presented System A first, whilst the other half began with system B. Play-throughs of either of the systems varied in time but usually took 10 minutes or more.
6. Once the participant had tested both systems they were given the opportunity to do additional runs through either system and to specify new preferences and norms if they so wished.
7. When the participant had finished their testing session they were given a questionnaire to fill out. This took around 2 minutes to complete.

While it was initially intended for the participants to run through each system at least twice, this resulted in initial overall experiments running for nearly an hour. This made it difficult to attract a sufficient number of participants. In order to enlarge our pool of participants, we shortened the procedure to roughly 30 minutes by requiring only one run through each and gave the option to the participant of further experimentation with the system.

When specifying norms and preferences, participants were encouraged to attempt to create gameplay situations in which norms would be broken or exploited by the GM. This was so that the designers were more likely to observe the norm enforcement mechanics, as there was a good chance these features may not have been evident during the gameplay testing if the designers had not taken this into consideration.

Whilst this experimental setup is similar to that of a ‘player testing’ session, we draw an important distinction. The play-through of the game itself is not intended to be an
Applying norms and preferences for designing flexible game rules

Evaluation of the gameplay from a player’s point of view, it is intended to be a test run from the designer’s point of view. The questionnaire is reflective of this, its questions being around how well the GM conforms to their specifications as opposed to how fun the game is or how much ‘replay value’ it has. Our experiment is designed to be a shortened and simpliﬁed development iteration loop comprising design, implementation and testing phases and is intended to be reﬂective of the video game development process.

5.2.2 Questionnaire

The questionnaire which was given to the participants contained both quantitative and qualitative questions and was hosted as a form on Google Drive (http://drive.google.com/) which the subjects ﬁlled out from within the Google Chrome (http://www.google.com.au/intl/en-au/chrome/) web browser.

The questions comprising the questionnaire were as follows:

1. Which system produced the type of gameplay you described in your preferences? Answered on a ﬁve-point Likert-scale labelled ‘1 – deﬁnitely system A’ to ‘5 – deﬁnitely system B’.

2. Which system best enforced the norms you deﬁned? Answered on a ﬁve-point Likert-scale labelled ‘1 – deﬁnitely system A’ to ‘5 – deﬁnitely system B’.

3. Which system best produced the type of narrative you were trying to achieve with the preferences and norms you speciﬁed? Answered on a ﬁve-point Likert-scale labelled ‘1 – deﬁnitely system A’ to ‘5 – deﬁnitely system B’.

4. With regards to the system which performed better in the above 3 aspects, how well did the system satisfy the preferences and enforce the norms you deﬁned? Answered on a ﬁve-point Likert-scale labelled ‘1 – only slightly’ to ‘5 – very well’.

5. Given minimal training and experience with the system, how easy are the preference and norm deﬁnition languages to read and use? Answered on a ﬁve-point Likert-scale labelled ‘1 – very difﬁcult’ to ‘5 – very easy’.

6. How suitable do you think this approach of specifying gameplay via preferences and norms would be to designing video games with ﬂexible narrative? Answered on a ﬁve-point Likert-scale labelled ‘1 – not suitable at all’ to ‘5 – highly suitable’.

7. What are the positive aspects of the system, if any? Answered in a text box which accepted a paragraph.

8. What aspects of the system are in need of improvement, if any? Answered in a text box which accepted a paragraph.

5.2.3 Analysis of results

The results show that participants identiﬁed the difference between the two systems and that they believed the normative GM was doing a better job of satisfying their preferences, enforcing their speciﬁed norms and overall producing the type of gameplay they desired than the existing GM. They also show that the participants felt that the system was a suitable method of designing games with ﬂexible narrative. These results
are summarised by the box plot in Figure 8 and the Table of statistical properties in Table 1.

**Figure 8** Box plot detailing the spread of results

![Box plot](image)

**Table 1** Table of statistical properties of the data collected

<table>
<thead>
<tr>
<th>Question</th>
<th>Mode</th>
<th>Median</th>
<th>Wilcoxon signed-ranks test score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>5</td>
<td>5</td>
<td>0.005413461</td>
</tr>
<tr>
<td>Q2</td>
<td>5</td>
<td>5</td>
<td>0.002531016</td>
</tr>
<tr>
<td>Q3</td>
<td>5</td>
<td>5</td>
<td>0.003842897</td>
</tr>
<tr>
<td>Q4</td>
<td>4</td>
<td>4</td>
<td>0.005859343</td>
</tr>
<tr>
<td>Q5</td>
<td>5</td>
<td>4</td>
<td>0.005859343</td>
</tr>
<tr>
<td>Q6</td>
<td>5</td>
<td>4.5</td>
<td>0.005859343</td>
</tr>
</tbody>
</table>

We analysed the results with the Wilcoxon signed-rank test to test for significance as we had only a single ordinal dependent variable (user score) and only a single sample in each case. This test is also relevant because it does not assume data is normally distributed. We would reasonably expect that if there was no difference between systems A and B then the median score for questions 1 to 3 would be 3. For question 3, we would expect a median score of 3 if the system performed moderately well at satisfying the user’s preferences and enforcing their norms. We would also expect a median score of 3 for question 5 if the system was neither easy nor difficult to use, and for question 6 if the system was neither suitable nor unsuitable for game development. Therefore, we define the null hypothesis to be:

H1: The median of the population is its middle value, 3.

The first three questions measured the performance of the normative GM against the original GM with regards to what degree they produced the desired behaviour as defined
Applying norms and preferences for designing flexible game rules

by the users’ specified preferences and norms. All three questions had a mode and median of 5 and this is a strong result also backed up by strong Wilcoxon signed-rank test scores of 0.005, 0.003 and 0.004 for questions 1, 2 and 3, respectively. This indicates that, overall, participants felt that system B behaved to their expectations and therefore shows that the normative GM works to some extent. We can also reject the null hypothesis of there being no difference between systems A and B.

From the feedback on the first three questions, it can be concluded that system B (our normative GM) did a better job of satisfying preferences and enforcing norms of the two. Question 4 gauged how well this system performed these functions itself, independently of the other system. Whilst the results were not as strong for this question as the first three, the feedback was still favourable, with a mode and mean of 4. The Wilcoxon signed-rank test score is also slightly higher at 0.006, however we can still reject the null hypothesis that the system performed only moderately well.

Question 5 dealt with the ease of use and human readability of the norm and preference specification languages and here the results were not quite as strong. The question had a lower mode and median score of 4. This indicates some room for improvement in the usability of the system, however, the scores for this question were still generally favourable. The Wilcoxon signed-rank test score is 0.006, allowing us to reject the null hypothesis that the system is neither easy nor difficult to use.

The final quantitative question related to the user’s opinion on how suitable the system would be in practice for the development of games with flexible narrative. The responses to this question had a mode of 5 and median of 4.5. Overall, this indicates that users felt the system could be adapted for use in the actual development of games. The Wilcoxon signed-rank test score was also 0.006, allowing us to reject the null hypothesis that the system was neither suitable nor unsuitable for game development.

Questions 7 and 8 requested qualitative feedback on the strengths and weaknesses of the system. Overall, the positive comments mostly dealt with the potential of the system and how easy it was actually specify the norms and preferences. Some comments include:

- “The system provides an effective process for procedural generation of narrative, which is one of the gameplay elements which has largely yet to benefit from procedural generation.”

- “It was easy to input and select outcomes.”

- “Even generic quests can be tweaked easily to incorporate the style of game the designer and/or players are interested in experiencing.”

Users identified some areas in need of improvement as the learning curve for the norm and preference specification languages; some of the terminology used in the ontology; and the fact that the GM restricts some of the player’s agency by specifically telling them what to do. Some comments include:

- “The system lacks subtlety, though that is indicative of its immaturity. Given time, this could be easily be resolved.”

- “With both systems the prompting from the GM may be too direct and restrict players from experiencing the problem solving elements of the quest.”
5.2.4 Discussion

Feedback from participants was encouraging in that they saw potential in the system’s functionality. However, users also expressed some views that the system was initially somewhat challenging to understand and this will be addressed in future work. Some learning curve is to be expected however, and we contend that a single 30 minute evaluation of the system is not reasonably sufficient for a user to become completely comfortable with it. Users also expressed some disappointment that the GM specifically directed them as to what players should do within the game and this should be addressed in future work. As the scenario which the GM can handle becomes larger, the potential increases for the GM to direct entire plot lines at a higher level, rather than directing the player in atomic actions.

Note that in the evaluation we included both norms and preferences in the normative GM system and the users found the system to implement them better than the system without them. We did not attempt to evaluate which component has a greater impact as they are complementary in the way we have implemented the reasoning. This may however be an interesting question for future work.

6 Discussion

Two areas of our architecture in particular warrant further discussion:

1. the different approaches to assigning plan property information
2. our method of summary information propagation which differs somewhat from the related work.

6.1 Plan property information

A number of other ways of applying weights to themes and actions were considered before settling on probability. Firstly, a degree of ‘satisfaction’ was considered, whereby execution traces through the goal-plan tree were defined to ‘satisfy’ a property more if they included more instances of that property than other traces. For example, if an agent selected a number of plans to satisfy its goals and a number of them triggered violent in-game events this trace would ‘satisfy’ the violence property more than an execution thread which produced only one violent outcome. The difficulties with this approach are numerous, including the fact that this would change the internal processes of the agent, effectively turning the problem into a more complex planning tree search. Apart from the additional work required to modify the agent, execution of such searches can quickly become intractable due to computational complexity. However, the main reason this approach was not used was due to the fact that our goal-plan tree only executes in-game events through leaf node plans. This mean that the advantages of this approach would be lost as the agent is not likely to trigger a large number of in-game actions unless the player repeatedly fails their assigned tasks and thus the satisfaction levels for different execution threads involving plans with a certain property would not vary to a significant degree.

Another way of assigning numerical weights to properties and actions attributed to plans would be to simply tally the number of leaf node plans in lower levels which can be
reached from that node. For example, the *PersuadeChef* plan would assign the value of 2 to the *persuade* property due to the fact that there are two leaf node plans which involve persuasion accessible from that node. However, this may skew values in an undesirable way due to the fact the only one of these plans is likely to be executed unless the player fails any selected plans. The problem lies in the fact that this would give larger calculated scores for plans with a large number of lower level plans containing highly preferred properties at the expense of potentially finding a plan which satisfies multiple preferences. For example, if plan A has the property summary \{*violence* = 0.5\}, plan B’s summary is \{*violence* = 0.2, *persuasion* = 0.2\} and both violence and persuasion are stated as preferences, it is likely that plan A will be selected, even though there may be a lower level plan following on from plan B which may satisfy the preference for both properties.

### 6.2 Propagation of summary information

Our propagation process differs from Visser et al. (2011) as there is no need to add null fields to properties of plan nodes whose children may not specify a value for that property. As discussed earlier, our properties come from a fixed ontology with the values assigned to them being a probability from 0 to 1. Therefore, instead of accounting for the possibility that a property will not be assigned in the sub-tree of the node, we assume each property to have a probability of 0 unless explicitly stated in the property summary information.

### 7 Conclusions

Flexible narrative in video games is a well investigated problem to which a number of solutions proposed. We have built upon one such solution (Luong et al., 2013) which uses an automated intelligent BDI agent to construct the narrative for the player. We did this by identifying two areas for improvement: the requirement for anyone desiring to use this system for game development to possess detailed knowledge of agent-oriented programming; and the fact that the existing narrative generation is essentially random.

In this article, we have proposed and implemented a preference and norm specification language for the purposes of allowing game designers to create games with flexible narrative using an automated intelligent GM. The system is designed to be human readable and accessible to developers who have little knowledge of the implementation details of the GM itself.

The normative GM we have implemented largely fulfils our objectives as confirmed by the results. A case study was conducted to show the system behaves in the desired fashion and generates game narratives which satisfy the user’s preferences specified. The system enforces and exploits the specified norms to do so. A user experiment was also undertaken to evaluate the usability of the system by testing it and then completing a questionnaire.
7.1 Future work

In our implementation, the GM assumes that if a plan has the potential to trigger a norm, then that norm will be triggered. That is, the themes from that norm’s sanction are added to the plan’s themes with a probability of 1; the GM reasons that these themes must come about. This is, of course, not quite true, as the player may well succeed in a plan whose action is obliged or fail in a plan for which the action is prohibited. Ideally, we would like to calculate the probability of the player succeeding or failing at any given plan, however, this is not always straightforward. For instance, the success probability for a plan involving a persuasion attempt could be calculated using the D&D rules (Wizards RPG Team, 2008), taking into account: the skill level of the player, the difficulty of the persuasion, and any dice rolls involved. However, how can we calculate the probability of the player finding a chest? Plans involving combat also present a problem. D&D has a ‘challenge rating’ system for numerically calculating the difficulty of defeating a certain opponent based on their and the player’s character statistics. However, this calculation only produces a number which categorises the challenge depending on which range it falls into. Calculating a true probability would be substantially more complicated. These success probability calculations are a potential focus for future work and would improve the GM’s capacity for normative and preferential reasoning.

Other implementations of preference (Bienvenu et al., 2006; Son and Pontelli, 2004) and norm (Alechina et al., 2012) specification languages are more expressive, allowing for more involved and complicated definitions. Our work focused on human readability and thus implemented an intentionally simplified system of preference and norm specification designed to be easily learnt and understood by users. Whilst more complex definition languages are outside the scope of this article, a larger and more expressive ontology may also make it more difficult for users to comprehend the system and learn how to use it. Future work may include attempting to expand the expressiveness of the languages whilst trying to maintain a focus on usability for game designers.

References


Applying norms and preferences for designing flexible game rules


Applying norms and preferences for designing flexible game rules


Notes

1 The full execution console output is available at: http://goo.gl/T4rxhi.