ABSTRACT

In this paper we describe an approach for developing an intelligent game master (GM) for computer role-playing games. The role of the GM is to set up the game environment, manage the narrative flow and enforce the game rules whilst keeping the players engaged. Our approach is to use the popular Belief-Desire-Intention (BDI) model of agents to developing a GM. We describe the process for creating such a GM and how we implemented a prototype of it for a scenario in the Neverwinter Nights (NWN) game. We describe the evaluation of our prototype with human participants who played the chosen NWN scenario both with and without the BDI GM. The comparison survey completed by the participants shows that the system with the BDI GM was the clear winner with respect to game replayability, flexibility, objective setting and overall interest; thus, validating our hypothesis that a BDI GM will provide game players with a better gaming experience.

Keywords
BDI Agent Systems, Computer Role-Playing Games, Game Mastering.

1. INTRODUCTION AND MOTIVATION

In recent years, computer games and interactive storytelling research have been emphasising the need for a rational ‘coordination agent’ that orchestrates the behaviour of several autonomous characters according to some high-level, often plot-oriented, policy without taking direct control of all their actions. Examples of such a trend have been narrative controllers or director agents in interactive storytelling [9, 10], AI directors in computer games [1] and, directly inspired from real-life, game masters in a table-top or live-action Role-Playing Game (RPG) [13, 6].

A Game Master (GM) in table-top RPGs is a person in charge of organising engaging game sessions for cooperative multiplayer experiences, usually in the range of 4-8 participants [21]. The GM describes the events taking place in the game’s fictional world, she gives life to the characters populating it, and then finally communicates the outcomes of players’ decisions, enforcing any game rules as needed.

A table-top RPG game session is often regarded as an emergent theatrical production where players take the role of lead actors and the Game Master serves as the director that provides stage, scenery and a flexible plot that adapts as a result of the interaction between players and the fictional game world. Examples of interaction between Game Masters and players abound in specialised press, game books, academic publications [21, 5] as well as dedicated web sites1 and on-line gaming communities.

This paper contributes a solution to the design and implementation of an automated GM agent for computer role-playing games. The GM agent will set up the game environment, manage the narrative flow in a flexible fashion and it will enforce the game rules as needed while keeping the players engaged. Such a GM agent will help address issues in current computer role-playing games, which resort to scripting every possible event in the game’s narrative. Not only does this lead to a substantial increase in design and programming time for the game [24], but it also reduces the narrative flexibility, which will have a negative impact on players’ engagement.

Our design uses the Belief-Desire-Intention (BDI) model of agents to flexibly pursue the goals of the plot. BDI agents are designed and implemented using mental attitudes such as beliefs, goals and intentions [2, 17]. The use of such mental attitudes in software design allows developers to break complex decisions into smaller, more intuitive components. This technology is most useful when building systems that operate in highly dynamic environments as it allows for real-time online reasoning. The design of such systems allow incorporating alternatives strategies for achieving the one task, thus enabling flexible and robust systems. These traits, make BDI agent technology ideal for developing an automated software GM.

We describe a prototype of a BDI GM set in a particular scenario from the Neverwinter Nights (NWN) computer RPG [7]. Its implementation uses the Prometheus BDI design methodology [12] and the JACK agent programming language [22], incorporating it into the NWN game using its Aurora toolset [7]. We used this prototype implementation to validate the effectiveness of our approach by conducting experiments involving human participants playing the NWN game scene with and without a BDI GM. We show that the human participants chose the game with the BDI GM as the one that provided a better gaming experience and the preferred system for future game play.

1For instance, see http://www.darkshire.net/~jhkim/rpg/whatis/tabletop.html.
The remainder of this paper is organised as follows: we introduce the underlying guiding principles in the design of a BDI GM, together with an overview of BDI systems and the example RPG used in Section 2; in Section 3 we present our approach to designing a BDI GM; we present the experimental setup and results in Section 4; and finally conclude with related work in Section 5 and future directions.

2. BACKGROUND

In this section we present an overview of the features of a game master in the context of table-top RPGs. The BDI based GM that we develop will satisfy these features. We then provide an overview of BDI agent systems and the Neverwinter Nights RPG used for developing a prototype.

2.1 The Game Master

A comprehensive review of the activities of a GM in role-playing games is given in [21], but we highlight below the ones that have been typically carried out by human GMs in table-top RPGs. These are desirable in any software implementation of a game master in order to solve the aforementioned problem of trading off players control with coherent high-level narrative progression:

Managing narrative flow: The GM will often rely on pre-planned narrative, a series of quests (tasks for the players to fulfill), which can however be flexibly adapted in response to players’ actions and changing goals. For instance, a GM may want to lead players to a secret location, but may have to vary their strategy for achieving it, as players may wonder about in the opposite direction or they may be unable to find the way to the said location.

Enforcing game rules: The GM will arbitrate how the fictional game world operates and how the players interact with it, as well as with other players. For instance, if a player wants to bash a door, they will communicate their intention to the GM. The GM will consult the rule book to decide the outcome of the action, which will be then communicated back to the player.

Engaging players: If the game languishes and players are not making any progress in the game, it is in fact the duty of the GM to try and find a solution. For example, this can be achieved by having a game character assign an additional quest (a side-quest) to the players in an attempt to increase the pace and general interest of the game, enticing continued game play.

Setting up the game environment: The GM will provide players with a fictional setting, which includes the NPCs (non-playing characters) that inhabit it, e.g., her allies, the story villains, etc. She will impersonate the characters, voice them and lead them to cooperate or compete with the player, depending on their role in the game narrative.

Computer RPGs have implemented game rules fairly accurately: for instance, games such as Neverwinter Nights implement the vast majority of the d20 rule system employed in Dungeons & Dragons\(^2\). However, all other requirements have only been partially addressed in existing work [18, 8]. In this paper, we concentrate on managing narrative flexibly and setting up the environment with NPCs that posses different strategies so that player’s engagement is sustained.

Player’s engagement though is a complex topic in its own right [23] and we do not intend to claim that our strategy is by any means the best or the only possible in this context. It is however, a step towards achieving it.

2.2 BDI Agents

The BDI model of agent systems is based on the early philosophical work of Bratman [2], and the formal computational models of Rao and Georgeff [17]. The BDI approach is to model and implement agents using mental attitudes, such as beliefs, desires, goals, plans and intentions. Practically, BDI agent systems enable programmers to write abstract procedures, called plans, that are combined and used in real-time in a way that is both flexible and robust.

Figure 1 shows the general architecture of a BDI system. Briefly, a BDI system responds to events (i.e., inputs to the system), by selecting a plan from a pre-defined plan library based on the current beliefs. A plan is a set of steps to respond to a particular event. These steps could include actions that affect the environment, belief updates, or internal events (subgoals) that are in turn handled by different plans.

When an agent commits to a particular plan it is added on to the intention stack. The system continues this process of sensing, reasoning and acting.

Goals are often represented as (internal) events in BDI implementations. There is often more than one plan in the plan library that can be executed to achieve a goal, thus

\(^2\)Available online at http://www.wizards.com/default.asp?x=d20/article/srdarchive
providing 'flexibility'. If the chosen plan fails then the system will choose an alternative plan, if any, to achieve the goal, thus providing 'robustness' upon failure.

The fact that plans that handle goals may in turn post subgoals, naturally leads to a goal plan tree structure such as that shown in Figure 2. In essence, it is an 'AND/OR' tree where one of the plans is chosen to achieve a goal (‘OR’), and all of the subgoals must be achieved for the plan to succeed (‘AND’). Padgham and Winikoff [12, p.16] show the power of flexibility of these goal-plan structures by providing calculations that illustrate that a modest system with 72 sub-goals and 147 small plans (4 subgoals each) provides over two million ways of achieving the single top level goal.

There have been a number of BDI agent system design methodologies proposed with supporting tools [4]. For the purpose of illustrating our approach we chose the popular Prometheus [12] methodology and the supporting Prometheus design tool (PDT)[11]. There are 3 main design stages in Prometheus: System specification where the requirements of the system are captured including the goals of the system; Architectural design where the overview of the internals of the system are specified; and Detailed design where each agent’s internals are detailed in terms of the goal-plan trees for the goals they achieve. PDT provides a graphical interface for designing a system following the methodology, and provides code generation, amongst many other features, that generate code stubs in the JACK [22] agent language.

### 2.3 Neverwinter Nights

Neverwinter Nights is a third-person role-playing computer game developed by BioWare and published by Atari in 2002. The game was originally released on Windows, and subsequently on Linux and Mac OS. The game is set in a fantasy world, with the game rules based on the Dungeons & Dragons 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. It is the Aurora toolset that has made the game extremely popular. Aurora has been used for creating new game scenarios, dubbed modules in the original Dungeons & Dragons jargon, which have been counted in the thousands, including original expansions made by Bioware but most importantly by a very active community of gamers.

It has also been employed in research work [19, 20] since it is still probably the best (almost) freely-available toolset for computer role-playing games to date. All of these features makes NWN a good choice for our prototype system.

### 3. DESIGNING A BDI GM AGENT

In this section, we detail the process of designing a BDI based GM agent. There are 3 main steps: (i) Constructing a ‘game-flow’ diagram out of a sample game scenario; (ii) Constructing a ‘goal tree’ based on the game-flow diagram; and (iii) Constructing the agent’s detailed ‘goal-plan tree’ by adding plans, to achieve the identified goals, and additional sub-goals as appropriate.

We use an example, the Hideout Quest occurring in Chapter I of the original Neverwinter Nights game campaign, to illustrate the process in practice. The main goal when playing this scenario is to gain access to the Hideout room inside a Tavern, whose door is well-guarded by a chef.

#### 3.1 Constructing the game-flow diagram

The first step, is to construct a game-flow diagram for the scenario. We begin with an abstract diagram as shown in Figure 3, where in order to Get inside the Hideout the player has to go through two main stages: **Gain Access To Hideout Door** and **Player Opens Hideout Door**. The game-
flow diagrams for each of them were subsequently developed and they are shown, respectively, in Figure 4 and Figure 5.

The construction of the game-flow diagrams was informed by observations made while playing the scenario multiple times, as well as by excerpts of the game progression contained in the Neverwinter Nights Official Adventure Guide [3].

The objectives in the scenario were interpreted as the game-flow diagram operation steps and they were represented by rectangles in the game-flow diagram. For instance, in Figure 5 Gain Access To Door is an example of such an objective; players have to find a way to gain access to the door guarded by the chef.

Each objective had a few different solutions, one of which would be triggered by specific game events. Consequently, the different solutions were translated as operation steps joined by a decision node in the game-flow diagram. A decision node is represented by a diamond. For example, the The Gain Access To Door has 4 different solutions, each represented as an operation step, joined at the decision node Which method? with the transition labels representing the choice that determines the particular operation step. The choices afforded to players in our example are persuading the chef, bribing him with gold coins, defeating him in a fight or telling him a password that would have been previously recovered. Each option is further decomposed into decision nodes and operation steps.

The success of the players actions, and consequently of the operation steps, depends on features such as the players skills; numerical properties that quantify the player's ability to perform a task, originally devised in pen-and-paper RPGs.

If a player fails a particular operation step in a particular stage, the player is forced to restart the stage. If the player succeeds to pass the stage then the game would proceed to the next stage. For example in Figure 3, if the player succeeds the Gain Access To Hideout Door then the player progresses to the Player Opens Hideout Door stage. If a particular operation step fails however, for instance if players are unable to persuade the chef or they are defeated by the guard, the Gain Access To Hideout Door stage is restarted.

3.2 Constructing the goal tree

In BDI agent design methodologies an initial task in the requirement analysis phase is to identify the goals of the system [4]. Each goal is decomposed into sub-goals which are further refined, leading to a goal tree structure. The second step to constructing our BDI GM agent is to develop this goal tree structure.

We do this by transferring all of the operation-steps in the game-flow diagrams, developed as described above, to goals as shown in Figure 6. The ‘OR’ and ‘AND’ labels indicate whether the sub-goals are alternatives or whether they all need to be successfully completed to achieve the goal. All operation steps connected by a decision node in the game-flow diagrams become ‘OR’ connected sub-goals in the goal tree, ‘AND’ connected subgoals otherwise. The dashed-arrows between subgoals (see Figure 6) indicate their order of execution, if any. This order can be obtained from the game-flow diagram.

In our example in Figure 6, the root top level goal of the GM agent is determined as Get Inside The Hideout which is the main objective of the scene as shown in the game-flow diagram in Figure 3. This main goal has two subgoals, both of which need to be achieved in sequence: Gains Access To Hideout Door and Player Opens Hideout Door. Hence, they are added with an ‘AND’ connective.

Each of these subgoals are then further expanded using the respective game-flow diagrams resulting in the goal tree shown in Figure 6.

3.3 Constructing the detailed goal-plan tree

In general when designing a BDI agent system, after having identified the goals of the system, the goals are assigned to roles which are in turn assigned to agents [12]. In this work, we are only concerned with a single agent; the game master agent. Hence, we avoid the need for detailing roles, and also inter-agent components such as communication protocols.

All the goals of the system are therefore assigned to the single GM agent. The next step is then to develop the internal details of the agent in terms of plan and subgoal structures to achieve the assigned goals.

Figure 6: The initial goal tree for the modified ‘Hideout Quest’.

Figure 7: Initial goal-plan tree for the ‘Hideout Quest’ (partial tree shown).
We develop the detailed goal-plan tree using a two-phase approach: (i) first, propagate all the goals from the goal tree developed in the previous stage to an initial goal-plan tree with appropriate plan structures, ensuring coverage of all operation-steps from the game-flow diagrams; (ii) refine the goal-plan tree to enhance run-time flexibility and robustness. We describe the steps of each phase below:

(i) Initial goal-plan tree (Figure 7):

- For each goal with subgoals that are ‘AND’ connected, such as the Get Inside The Hideout goal in Figure 6, a single plan is created to handle the top level goal and to post the subgoals - Lead Player Inside Hideout plan in Figure 7 that handles the top level goal and posts the Gain Access To Hideout Door and Open Hideout Door subgoals. Any subgoal ordering information annotated in the goal tree is also annotated to the plan.

- For each goal with subgoals that are ‘OR’ connected, such as the Gain Access To Hideout Door goal in Figure 6, alternative plans are created, each responsible for posting one of the subgoals - the alternative plans, Persuade The Chef, Bribe The Chef Defeat The Chef, and Get The Password, are added to handle the top level goal as shown in Figure 7. The ‘context condition’ of these plans, that is, the conditions that determine the plan choice, are also annotated in the design (though not shown because of space constraints).

(ii) Refining the goal-plan tree (Figure 8):

- Extend the goal-plan tree by adding alternative plans for achieving goals as appropriate, in particular, for subgoals with just a single plan. This is an important part of developing the GM agent and is the crucial step that determines the level of flexibility and robustness offered by the system. For example, we added the Distract The Chef plan as an alternative plan for the Gain Access To Hideout Door goal, and several others in Figure 8.

- Where appropriate extend the plans to post subgoals which can be achieved by different alternative plans. For example, the Player Meets Ulfnog plan is extended to post a subgoal Player Finds Ulfnog which is handled by two plans: Find Him Inside the Tavern and Find Him Outside the Tavern.
• Group together similar functionality as appropriate to enable better modularity but also importantly to enable greater cohesion in the game play. In our example, we determined bribing the chef as part of persuading the chef, and as such, moved the Bribe The Chef plan as an alternative plan for the goal Player Persuades The Chef in the final goal-plan tree.

• Collapse subgoals that do not have more than a single plan, with no prospect of expansion, into the plan that posts the subgoal. Subgoaling is only useful when there are alternative means of achieving the goal. For example, in Figure 7, the Player Defeats the Chef goal is collapsed into the Defeat The Chef By Fighting Him plan in Figure 8.

The final result is that the GM can run the modified NWN scenario generating many more execution traces than are available in the original implementation. Figure 3, 4 and Figure 5, show that there are four unique options to complete the Gain Access To Hideout Door phase and three distinct choices are presented for the Player Opens Hideout Door phase. Thus, the original scenario can only theoretically generate 12 unique execution traces (3 x 4 = 12) based on players’ choices to the end of the quest. In the modified scenario above, there are 108 traces that our GM agent makes available to players to complete the revised version of the game. We note that the goal-plan tree structure is by no means complete, and can be further extended providing a much larger set of traces.

4. EVALUATION

Our primary motivation in introducing a BDI based game master into computer role playing games is to provide game players with a better game playing experience. In this section we describe some preliminary experiments that we conducted to determine whether the prototype system that we developed, does indeed satisfy this objective.

We begin with a brief description of the implementation of our prototype game master with respect to the toolsets used. We then describe the experimental setup followed by the results with an analysis.

4.1 Prototype Implementation

The BDI game master was designed using PDT (Prometheus Design Tool [11]) which supports the Prometheus methodology [12] for developing agent systems. PDT allows for a goal-overview diagram to be created in the ‘System Specification’ stage and for the detailed goal-plan tree to be developed in the ‘Detailed Design’ phase. These correspond to the initial goal-overview tree and the final BDI goal-plan tree described in section 3.

PDT auto-generates skeleton code into the JACK agent programming language [22] which is a Java based agent programming language. The GM was therefore implemented in JACK version 5.6.

Our prototype, ‘Hideout Quest’, was then implemented in a Windows 7 Neverwinter Nights (NWN) client, while the NWN Server was running under Linux Ubuntu 12.04 with the JVM (Java Virtual Machine) plugin extension for NWN Extender.

In PDT, goals are translated to events in the detailed design. See http://www.nwnx.org/

4.2 Experimental setup

Our experiments involved human participants playing the ‘Hideout Quest’ of Neverwinter nights in its original form and with our modified prototype system that includes the BDI game master described in 4.1. Hereafter, we term the former ‘Orig.Sys.’ and the latter ‘GM.Sys.’ for spatial convenience.

There were 10 participants in total, 2 that were experienced in playing Neverwinter nights and 8 that were not, however, they had extensive experience playing other computer RPGs. Each participant was asked to play one of the Orig.Sys. and the GM.Sys. (selected randomly) 3 times and then the other 3 times. The systems were labelled ‘A’ and ‘B’ and the participants were not told which of them was the BDI game master implementation and which one was not. This resulted in 30 game plays of each system.

The Neverwinter nights game server stores a comprehensive log of game activity, which is useful when analysing the results of the experiments. In addition to this, the GM.Sys. was augmented with code to log the activity with respect to the plans selected and actions executed by the system. The participants played the games locally on a Windows 7 PC, hence there were no network issues that interfered with the game.

Questionnaire:

After both systems were played the participants were requested to fill out a short survey that compares the two systems with respect to the game play they had completed. The questions were designed to evaluate each user’s game playing experience in terms of replayability, flexibility, and game objective setting. These criteria are reflective of the

The NWN Extender is an open source library that allows C++ extension code to be plugged in to the NWN server, using DLL injection. Typical extensions include more complex data structures not available in NWN Script, ODBC bridges to store state persistently on the game server or bridges to several language runtimes (e.g., Ruby, Java, C#).

In particular, the JVM plugin implements a bridge to the JVM, making it possible to call NWN scripts from within a JVM process and to also fetch the results in the same process. In other words, this plugin allowed us to write Java code to send or receive messages to and from an NWN module. This was instrumental to interfacing our BDI game master agent, which was written in the JACK Java based agent programming language, with the NWN runtime engine (see Figure 9).

Figure 9: The communication layer between the GM and the NWN game.
features of a game master as outlined in Section 2.1 as follows: **Flexibility** provides the player with a number of options, whilst managing the narrative flow as well as enforcing game rules. **Replayability** is crucial in maintaining player engagement over a period of time. **Game objective setting** relates to the overall game setup such that the player is aware of what needs to be done which in turn creates a more engaging game play experience.

We also added two questions to gauge which system was overall more interesting to the players and which they would choose for future game play. The survey questions are shown below, along with the hints given to the respondents:

1. **Between A and B, which one did you find more flexible?** HINT: Does the game accommodate your preferences by changing itself, or did it require you to follow its choices? Further, if you fail a step, does the game recover and provide different options for you?
2. **Between A and B, which one do you think has a better replayability value?** HINT: In video games, replayability is used to describe the entertainment value of playing a game more than once. A game has high replayability if it can be played as many times as possible and still generate fun for players (wikipedia).
3. **Between A and B, which one demonstrates its objectives clearer?** HINT: That is, in which game did you better understand what you need to do to finish the quest.
4. **Between A and B, which one did you find more interesting?**
5. **Between A and B, which one will you be more likely play in the future?**

The responses were recorded in a 5 item categorical scale: 1 - definitely A, 2 - most likely A, 3 - they are both equal, 4 - most likely B, 5 - definitely B. In the survey questions, System A and System B correspond to Orig.Sys and GM.Sys. respectively. Each survey (and game play), recorded the participant ID for the purpose of being able to match the survey with a particular game play log. No personal information was collected or stored.

### 4.3 Results Analysis

Figure 10 summarises the participants’ responses to the survey. All the participants selected the GM.Sys. (B) as the preferred system (item 4 or 5) in terms of replayability (Question 2) and the one they would most likely play in the future (Question 5). In terms of the most interesting (Question 4), flexible (Question 1), and clear objectives (Question 3), 70% of the participants selected GM.Sys. (B) while the other 30% considered both systems as equal.

None of the participants selected the Orig.Sys. as the preferred system on any of the questions in comparison to the GM.Sys. This simple evaluation shows that the BDI game master does indeed provide a better gaming experience in terms of the chosen criteria.

One of the key features of BDI agent systems is the ability to design a system that provides greater flexibility and robustness by virtue of the goal-plan structures. In section 3 we showed that for our example prototype the GM agent’s goal-plan tree for the ‘Go To Hideout Quest’ has 108 different paths in which the game could possibly be played. Analysing the game play logs for each of the 30 game plays from the experiments conducted revealed 26 unique traces of game play. This highlights the flexibility of the GM whilst retaining the overall objectives of the game.

When a plan fails in a BDI system, if there is an alternative plan to achieve the associated goal of the failed plan, it will be attempted by the agent. This mechanism makes these systems robust to failure. In the 30 game plays analysed, there were 38 instances when a plan failed and an alternative was attempted with success (1.27 on average). This shows that plan failure does indeed occur in practice and it is important for systems to provide the players with alternatives.

## 5. RELATED WORK

Academics and professional game developers alike have been wrestling with the problem of trading off players’ control with the necessity of maintaining high-level coherence for the game progression, most importantly for narrative purposes. The fundamental nature of the issue lies within the players’ ability to interact with a game in unexpected ways. This may jeopardise the ability to establish, build and resolve dramatic tension in the narrative which is a key tenet of the widely studied and accepted theory of Aristotelian drama. We highlight existing approaches to address the problem below.

A large amount of work around narrative management has come from the interactive storytelling research community, in the form of narrative controllers or director agents [9]. Several forms of automated planning have been instrumental, over the last few years, to the construction of fully realised interactive storytelling systems. The critically acclaimed Facade [10] puts the user in the role of an old time friend of a married couple on the verge of divorce. The user will, through her choice, decide the future of the marriage. The system is driven by reactive planning, in the form of the ABL (A Behavioral Language). There are also several other planning algorithms used to control narrative, from the point of view of different characters [15], with temporal constraints [16] or using state constraints [14], reinterpreting classic literary work in an interactive fashion. None of these systems though has to manage game play resources (e.g., amount of treasure, number of monsters, etc.) at the lower level, which makes high-level control harder to exert.

The games development community instead has contributed AI directors to vary the pace and the shape of game play in a few computer games. ‘Left 4 Dead’ [1] and ‘Cloud-
bbery Kingdom” are such systems. Both games approach the problem by optimising, in an ad-hoc manner, specific high-level game play parameters. Such parameters have direct bearing on the player experience: for instance, in Left 4 Dead the AI Director can decide to vary how many zombies are spawned in a specific section of the game is the player is performing poorly. However, despite the term, it is pretty clear that AI directors have a very simplistic view of what constitutes narrative, especially compared to the complex narrative systems listed before.

Finally, a few attempts to produce game masters (GM) inspired after table-top Role-Playing Games (RPG) have started [8, 18, 13]. To the best of our knowledge, none of those systems are full implementations of a GM, yet. As such, they do not provide the ability to manage both game play resources and narrative at the same time. While this is a very complex task, it is one that we believe is worth solving because it will ensure players’ satisfaction due to improved narrative and game play management at the same time. Moreover, it will provide an extremely interesting testbed for several algorithms devised in the agents community and artificial intelligence at large.

6. CONCLUSION

In this work, we presented a systematic approach to developing an automated GM agent for computer role-playing games. We utilise BDI agent technology for developing the GM, leveraging off the inherent flexibility and robustness in the way these systems are developed which play an important role in realising the features of a GM.

We implemented a prototype GM for a particular scenario of the Neverwinter Nights computer RPG. We used this prototype to evaluate the effectiveness of our BDI-based GM agent. The human participants that played both, the game with the BDI GM and without, found that the BDI GM provided greater replayability, flexibility, and clearer game objectives. They also found the version with the BDI GM more interesting overall and was the unanimous choice for future play.

As future work, we intend to investigate how to engineer knowledge about different scenarios in such a way that the GM can be more easily pluggable across many, without the need for heavy modifications. Clearly, knowledge elicitation is a crucial issue for the GM which we have not discussed in this paper. We envisage that while some of the knowledge could be generalised and will thus be domain-independent, there will certainly be some dependency on the domain (scenario). We would also consider incorporating knowledge about the player into the GM, possibly learning player preferences. These preferences could then be used by the GM to guide the the paths along the goal-plan tree (essentially the game narrative) based on these preferences.

7. REFERENCES