A Novel Multi-Agent Planning System for Digital Interactive Storytelling

EFSTATHIOS GOUDOULAKIS, Liverpool John Moores University
PROF. ABDENNOUR EL RHALIBI, Liverpool John Moores University
PROF. MADJID MERABTI, Liverpool John Moores University

Digital Interactive Storytelling (DIS) is a relatively novel area of computer entertainment which aims at investigating interactive applications capable of generating consistent, emergent and rich stories. To provide new solutions for DIS, we designed and are implementing and evaluating a novel multi-agent DIS framework, DIEGESIS, which includes agents' coordination and new planning and re-planning solutions. In this paper, we discuss the design and implementation of DIEGESIS, explaining in detail the mechanisms of our planning algorithms, and the story execution and agent coordination algorithms, along with a planning methods evaluation, and agent planning and coordination examples. We are currently in the process of creating a large DIS scenario, involving the story of Homer's Troy, with several levels which will allow us to further evaluate and expand our system.

CR Categories: I.2.8 [Problem Solving, Control Methods, and Search]: Plan execution, formation, and generation—Heuristic methods I.2.11 [Distributed Artificial Intelligence]: Multiagent systems—Coherence and coordination;

Keywords: Interactive Storytelling; Planning Algorithms; Multi-Agents

1. INTRODUCTION

Video games are becoming more complex and so their use as a storytelling medium is growing in importance and popularity. The unique interactive nature of games means that stories and characters can become more personal and involving.

Computer game stories can be implemented in different ways: either linear, branching, parallel, or threaded. Most games typically follow a linear storyline, where the events of the story are presented in a predefined sequence. It can be argued that making a player follow a defined story can diminish the interactivity level of a game; the player is, after all, following a pre-set path already laid out for him by the author. In order to still convey a story and allow the player to feel a high degree of interactivity, the concept of interactive or non-linear storytelling has to be introduced. Simply put, Interactive Storytelling (IS) presents the opportunity for players to have a direct choice on what is happening in the game world in which they are placed, to be the ones who dictate how certain events may come to pass within the constraints set by the story author. Similar to other entertainment media, stories in games play a big role in increasing immersion, building tension and adding interest to the player. However, one main difference from the games to those other media is that games are interactive; they expect participation from the player and in turn, players expect to participate and get involved in the events the game is presenting and the outcomes of those events.

Many techniques, including Spierling [2005], Fairclough and Cunningham [2004], and Bangso et al. [2004], have been developed to generate the characters plan of actions. Indeed, in order to progress the story, a plan of actions must be created for the characters to follow; each action is composed of a precondition, used to decide if the action is coherent given the current state of the world, and an effect, which indicates the consequences of performing the action.

As we have discussed in Goudoulakis et al. [2011], although planning systems have been used intensively in DIS for many years, there has been no much novel solutions for DIS research with respect to planning algorithms. The most common approach is to use a general Artificial Intelligence (AI) planning algorithm but, as we have advocated in Goudoulakis et al. [2011], a new planning algorithm (combining features from existing algorithms with novel ideas) needs to be created specifically with DIS systems characteristics in mind.

To provide new solutions for DIS, we designed and are currently implementing and evaluating, a novel multi-agent DIS framework, DIEGESIS, which includes agents' coordination and new planning and re-planning solutions. We use constraints in the planning/re-planning solutions which enable us to extend the expressiveness of the representation language we use, which is one of the most important suggestions for the new planning algorithm we proposed in Goudoulakis et al. [2011].

Our proposal for re-planning aims to interleave plan generation and opportunistic plan execution, to re-plan when unexpected changes occur in dynamic environments whilst introducing minimal disruption to the original plan. In addition planning and re-planning for characters organised as a multi-agent system, requires managing and controlling the coordination of these characters' actions.

In this paper, we briefly discuss the design and implementation of DIEGESIS (documented in Goudoulakis et al. [2012a; 2012b], explaining in details the mechanisms of our planning algorithm, and our story execution and agent coordination algorithm. We also discuss two different approaches for the planning process, and evaluate them using a new scenario. Finally, we show an example of characters' coordination.

The remainder of the paper is organized as follows: in section 2, we briefly discuss the state-of-the-art in DIS systems, planning, and Multi-Agent Systems (MAS); in section 3, we discuss the design and implementation of DIEGESIS, paying particular attention to the planning and multi-agent coordination aspects of it; in section 4, we introduce a case study featuring a coordination example, we briefly summarise the scalability evaluation of our previous work [Goudoulakis et al. 2012b], and present an evaluation of two planning methods along with a discussion of its outcome; in section 5, we discuss possible routes for the framework research by extending it with other components, and finally, in section 6 we conclude the paper.

2. STATE-OF-THE-ART IN DIS, PLANNING AND MAS

In this section we discuss the related work in digital interactive storytelling, in planning, in multi-agent system, and we discuss the key features of DIEGESIS.

2.1 Digital Interactive Storytelling

Recently there has been an increase in the development of high level behaviour planning for storytelling. Recent examples of DIS systems include Fabulator [Barros and Musse 2005], Gadin [Barber and Kudenko 2009], I-Storytelling [Charles et al. 2003], and Othello [Chang and Soo 2009], each including its own feature set.

For example, in Fabulator, a planning algorithm is used to generate a sequence of actions (an actual story) performed by characters that is capable to transform the system's world.

The player controls one character (the protagonist) and every other character is a Non-Playing Character (NPC). All NPC's actions are determined by the system. If an action of the player renders the current plan invalid the system uses the planning algorithm to create a new plan (re-planning). This way, the story is adapted to the player's actions.

In DIEGESIS, there is not a main character that the player controls/observes; instead, the user can make choices (selected by the storyteller) for actions that can affect every character in the active story. Also, we plan to allow the user to select and view the story from the perspective of any of the characters, and to be able to switch between them.

Fabulator implementation treats the planning problem as a state space search problem and it uses the A* algorithm to solve it. The authors of Fabulator state that there are several planning algorithms specific for STRIPS-like domains that can achieve better performance than A* but for small storyworlds (the result of authoring process in DIS), performance is not an issue. They also state that the most important shortcoming of their work was its reliance on predicate logic to represent the world state.

The generator of adaptive dilemma-based interactive narratives (GADIN) dynamically generates interactive narratives which are focused on dilemmas to create dramatic tension. Its authors claim that the system addresses two open challenges: maintaining the dramatic interest of the narrative over a longer period and (story) domain independence. Its planner (which is based on the Graphplan [Blum and Furst 1997] algorithm) creates sequences of actions that all lead to a dilemma for a character (who can be the user). "The user interacts with the storyworld by making decisions on relevant dilemmas and by freely choosing their own actions. Using this input, the system chooses and adapts future storylines according to the user's past behaviour." The main problem of the planner is that as more characters and actions are included, the time spent planning becomes unreasonably long. The time increases exponentially with the number of characters and the number of actions. On the other hand, neither the number of locations nor the number of dilemmas has an impact on the speed. The authors of Gadin claim that a potential solution would be the use of a form of hierarchical planning.

2.2 Planning Algorithms

A review of planning algorithms that are commonly used in DIS systems such as Graphplan, can be found in Goudoulakis et al. [2011]. The following are some of the planning algorithms that have been extensively used in DIS systems.

2.2.1 Graphplan

Graphplan was the first planning algorithm that converted the planning problem into an intermediary data structure called a planning graph. Graphplan have moved the field of planning forward by obtaining impressive gains in performance compared to previous planning approaches, based on the experimental results documented in [Blum and Furst 1997]. Graphplan's main drawback is that although it is an optimal partial-order planner, its input language is quite limited. [Russel and Norvig 2010]

In Graphplan a plan is extracted from a graph. The graph consists of levels of literals which could be true and levels of actions of which the preconditions could be true. The graph is constructed starting at level 0 where all literals that are currently true are represented, these are true or false depending on the initial state and there are no other possibilities. Then a level of actions for which the preconditions hold in first level is added. This is followed by another level of literals that could hold if an action makes it true. Each level of literals gives the literals that could possibly be made true at that level depending on choices made earlier. Each level of actions gives all actions that could be used at that level depending on earlier choices. The Graphplan algorithm creates the graph in steps; if at the current level of literals all literals from the goal are present without mutex relations between them a solution plan may exist in the current graph. Otherwise the graph is expanded by adding a new level of actions and a resulting literals level. If the graph possibly contains a solution the algorithm tries to find it.

2.2.2 Hierarchical Task Network (HTN) Planning

HTN-based planning [Cavazza et al. 2002], which is also known as task-decomposition planning, is among the oldest approaches for providing domain-specific knowledge to a planning system.

An HTN planner solves problems by decomposition. The initial problem statement, the initial state and goal are viewed as a single action that must be decomposed into lower level actions. On the lower levels actions are decomposed further until only primitive actions remain. There will often be choices available to the planner when choosing decomposition for an action. Action decomposition specifies a way to turn an action into a plan.

HTN is based on forward search, and thus can be searched to extract a task decomposition corresponding to a solution plan. It is also goal-directed at the same time, since the top-level task is the main goal. This brings the unique property that during planning itself the state of the world is known at all times. [Charles et al. 2003].

2.2.3 Heuristic Search Planner (HSP)

HSP uses a STRIPS-based representation for problem description and searches the space of states from the initial state, using a traditional heuristic search algorithm and a heuristic automatically extracted from the STRIPS formulation [Charles et al. 2003].

HSP is a state space planning approach that can run either forward or backward and is much like path-finding. According to Russel and Norvig [2010], HSP was the first state-space searcher that made state-space search practical for large planning problems.

2.2.4 FF (Fast Forward)

FF is a forward state-space searcher that uses the ignore-delete-lists heuristic, estimating the heuristic with the help of a planning graph. It then uses enforced hill-climbing search (modified to keep track of the plan) with the heuristic to find a solution. When it hits a plateau or local maximum – i.e. when no action leads to a state with better heuristic score- then FF uses iterative deepening search until it finds a state which is better, or it gives up and restarts hill-climbing [Russel and Norvig 2010]. FF was created by mixing some novel ideas with features of HSP and Graphplan among others [Barros and Musse 2007].

In this section we have reviewed some of the most commonly used planning approaches in DIS. We introduce now some Multi-agent system concepts applicable to DIS.

2.3 Multi-Agent System

A multi-agent system (MAS) is a system designed and implemented as a group of agents interacting with each other (i.e. cooperating, coordinating, negotiating and so forth). In such systems, the agents either work individually exchanging information and/or services with other agents trying to succeed to their individual goals or work together solving sub-problems so the combination of their solutions become the final solution. [Vlachavas et al. 2005].

As explained in Russel and Norvig [2010], when there are multiple agents in the environment, each agent faces a multi-agent planning problem in which it tries to achieve its own goals with the help (or not) of the others. As discussed in Vlachavas et al. [2005], in multi-agent planning, agents are generating a plan of actions and they will solve the problem based on that plan. During the execution, the plan is revised based on the new details and results.

Based on Vlachavas et al. [2005], there are two types of multi-agent planning:

- Centralised multi-agent planning, in which a central agent is responsible to collect the partial or local plans of the other agents, to combine them in one plan and solve any conflicts that may occur.
- Distributed multi-agent planning, in which all the agents communicate with each other to generate their plans and to negotiate any possible conflicts.

In this section we have introduced some MAS oriented concepts applicable to DIS. In the next section, we discuss the contrast of the related work with DIEGESIS.

2.4 Key Features of DIEGESIS

For our framework, we have created a planning algorithm based on the principles of Graphplan, extending it to be able to interpret and use language constructs which add to the extension of the expressiveness of the representation language used, PDDL [International Planning Competition 2001], and also including constraints such as time and importance of actions (i.e. salience).

Some of the relevant DIS systems include Mateas and Stern [2003], Spierling [2005], Fairclough and Cunningham [2004], and Bangso et al. [2004].

More details about the related systems, I-Storytelling, Façade, and Othello, and a brief discussion of their features can be found in Goudoulakis et al. [2011].

The main difference between the above mentioned systems and DIEGESIS is that we focus on creating and implementing new planning and re-planning solutions specifically designed for DIS, where the existing systems use generic AI planning algorithms to achieve their goals.

One other difference is that in some of the systems [Barros and Musse 2005; Charles et al. 2003; Mateas and Stern 2003] there is a main character/agent and the whole story is generated around him/her. In contrast, our system (as discussed earlier in this section) is independent of a main agent, with all of the agents being able to contribute equally to the story based on the storyteller's modelling and the user's choices.

In DIEGESIS, each agent is planning individually considering its knowledge of the current state of the world, and then, during execution, the framework is responsible of coordinating the agents, resolving any conflicts between their actions, and instructing them to wait or re-plan when needed.

3. DIEGESIS – DESIGN & IMPLEMENTATION

DIEGESIS is composed of several components. Figure 1 depicts the high level architecture, highlighting its main components. In the following we discuss some of these components in a sequence that shows how they work together.

A more detailed description of most of the components can be found in Goudoulakis et al. [2012a; 2012b].

3.1 Platform, Representation Languages, Parsing Component, and Game World

As an implementation platform, we have chosen to use Java, a language which is cross-platform (i.e. runs in any operating system) and includes all the modern programming techniques that we need. As a representation language, we use PDDL [International Planning Competition 2001]. More details about this choice and the aspects of PDDL that we find important to be used in DIS systems representation, can be found in Goudoulakis et al. [2011].

The author of the story (i.e. the Storyteller) is responsible of modelling the story world including locations, items, characters, goals, milestones, available actions, etc. and stores it in several PDDL & XML files. For the generation of the PDDL files, we use the PDDL editor from Cooper et al. [2011]. The Parsing Component analyses these files, creates a representation of the Game World, and communicates it to the World Manager so the Knowledge Base (KB) can be populated.

The Game World is organised in a hierarchical manner; each level is capable of including an “infinite” number of sub-levels, each involving their own sets of characters, locations, and resources.

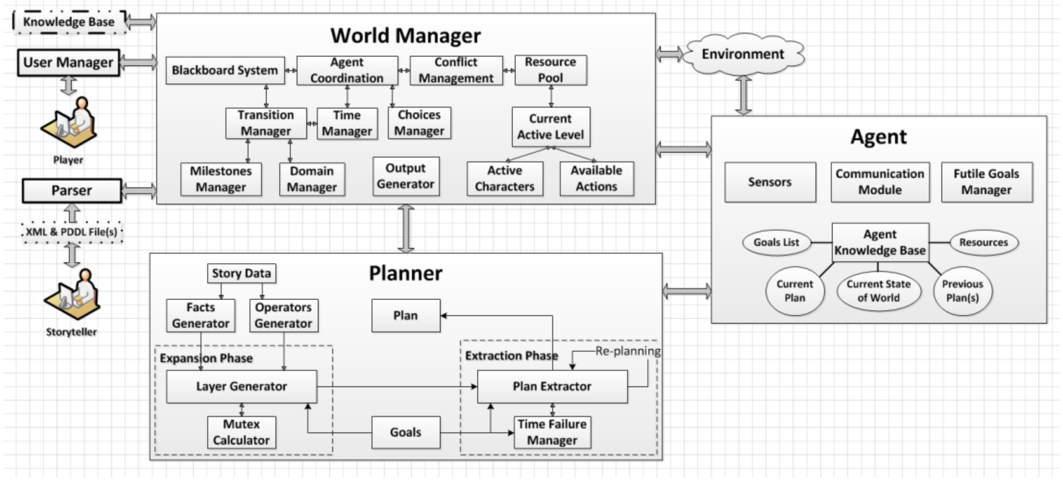


Fig. 1. DIEGESIS architecture.

3.2 World Manager & User Manager Components

The World Manager (WM) is the main component which coordinates the whole system. It has direct access to the KB and is responsible for keeping track of and updating the current world state, the state of the resources, and the environment that the agents are aware of.

The WM is also responsible for allocating resources to the planner (therefore to the agents) and retrieving each agent’s plans, solving any conflicts between the agents, so they can be coordinated and the story can be generated and progress.

The communication between the WM and the agents is being done via a Blackboard system, which is a centralised component within the WM component. In our implementation of a Blackboard system, every agent communicates with the WM to access and update the shared knowledge base and coordination information, and not directly with each other. The WM continuously and in turns, updates each agent’s knowledge of the current state of the world for any updates, and then asks them if there is an action to execute. Other communication includes to check if an agent is busy, to instruct an agent to plan/re-plan, or wait, and to inject a new goal based on executed actions.

Our current implementation of the WM component uses an instance of the Parser (controlled by a WM module called Domain Manager) to parse the files of a level and is able to populate the KB using this data.

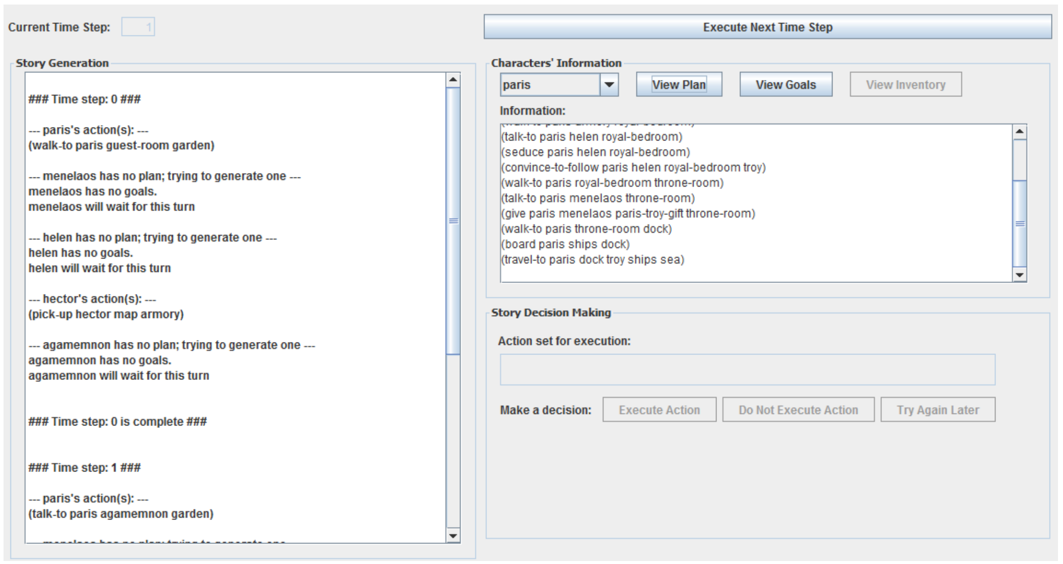
It uses an instance of the Planner to generate a number of high-level plans, which is still under development, and populates the Resource Pool with the state of the resources of the currently active level, while keeping an instance of it. Using this data, it also updates the environment that the agents are aware of.

The WM can also track the current time of the world, and is able to activate the agents which are active in the currently active level, coordinate and resolve conflicts between them so the story can be generated and progress properly.

Another responsibility of the WM is to send messages to the User Manager (UM) so they can be displayed to the user, as well as to process the user's input received by the UM. Figure 2 shows a screenshot of the current implementation of the UM's User Interface (UI). The generated story is displayed at the left half of the UI. The user can progress the story, view at any point during execution the current plan and goals for any of the activated characters in the level, and, when is required, make choices for the outcome of the story.

The story is executed in a turn-based way, based on the agents' plans. For each action an agent tries to execute, the WM queries the Resource Pool and checks the current state of the world to determine if it's applicable. If it is, then the Resource Pool and the current state of the world (and therefore the environment that the agents are aware of) are updated, to illustrate the changes made by the execution of the action. If it's not, depending on the conditions, the agent will be instructed to either wait for the next turn to try again, or re-plan.

The storyteller has the ability to mark actions as choices. When such an action is set to be executed, a decision needs to be made; either the action is going to be executed or not. Based on the storyteller's selection, either the system will make a yes/no/later decision, or the UM will be instructed to stop the execution of the story and ask the user to make this decision. Based on the outcome the decision, the action is either going to be executed, or not.



```
### Time step: 0 ###

--- paris's action(s): ---
(walk-to paris guest-room garden)

--- menelaos has no plan; trying to generate one ---
menelaos has no goals.
menelaos will wait for this turn

--- helen has no plan; trying to generate one ---
helen has no goals.
helen will wait for this turn

--- hector's action(s): ---
(pick-up hector map armory)

--- agamemnon has no plan; trying to generate one ---
agamemnon has no goals.
agamemnon will wait for this turn


### Time step: 0 is complete ###


### Time step: 1 ###

--- paris's action(s): ---
(talk-to paris agamemnon garden)
```

```
(talk-to paris helen royal-bedroom)
(seduce paris helen royal-bedroom)
(convince-to-follow paris helen royal-bedroom troy)
(walk-to paris royal-bedroom throne-room)
(talk-to paris menelaos throne-room)
(give paris menelaos paris-troy-gift throne-room)
(walk-to paris throne-room dock)
(board paris ships dock)
(travel-to paris dock troy ships sea)
```

Fig. 2. Screenshot of User Manager's User Interface.

While the story is executed, the Milestone Manager module constantly monitors the current state of the active level to identify if some (or all, depending of the level) of the milestones of the level, which are specified by the storyteller, are met. If they are, the Transition Manager module is used to perform a dynamic transitioning between two levels. Based on the milestones that are met and the current state of the current level, it activates a new level with the help of the Domain Manager, and dynamically transfers all the knowledge that needs to be transferred from the currently active level, to the newly activated one. The information that is not relevant to the new level is being kept in the KB since it may be used in domains that will be activated in the future while the story progresses.

3.3 Planner Component

The planner consists of a new planning and a new re-planning algorithm, able to generate plans of actions based on each agent's state and context, considering both the current world state and the available resources. The planning algorithm is based on Graphplan for solutions expansion, and backtracking heuristic search enriched with constraints satisfaction and dynamic opportunistic restart when required. By the term "planner" we refer to the component which deals both with the planning and the re-planning process.

A planner requires a domain definition, which lists the facts about the initial state of the world, its actions or operators, requirements and constraints. Actions are usually made up of three parts: parameters, preconditions and effects.

For the current implementation of the Planner, a planning agent based on the basic principles of the Graphplan algorithm is being used. The agent has a list of goals and tries to find opportunistically a valid plan considering the current world state (provided by the WM). We also consider a new approach, where the agent generates plans only for individual goals, prioritising them by a salience value specified either by the storyteller if the goal was specified as an initial part of the domain, or by the system, if the goal was injected during the story generation.

Each possible action has a specific duration and each goal has an "importance" (salience) value specified by the storyteller. The WM informs the agent of the available time at any stage during the execution, and the agent passes the information to the planner. The final plan cannot exceed the available time.

Since Graphplan is unable to interpret and use the language constructs that we consider key for DIS (introduced in Goudoulakis et al. [2011]) such as conditional effects, we have improved its graph generation and plan extraction processes to be able to interpret and use them properly. This adds to the extension of the expressiveness of the language that is being used (PDDL).

The planning process is illustrated in figure 3, and presented in Algorithms 1-4. The planner receives as an input a list of goals G, and its first step is to expand the layers of the planning graph until all the goals are present in the planning graph, or the algorithm has levelled-off, considering the constraints (mutexes) between them. This is shown in Algorithm 1 – lines 3-6. Two actions in the same level are considered mutex when their preconditions and effects are inconsistent. If two consecutive layers are identical (the algorithm has levelled-off) and the goals are not present and mutex free in the layers, the expansion stops.

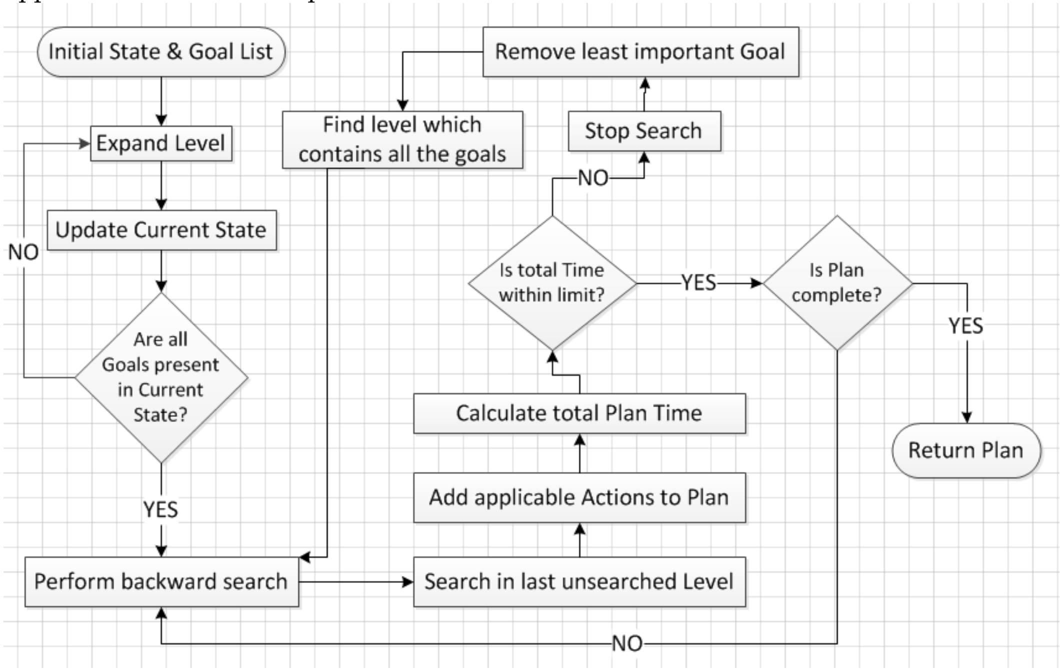
ALGORITHM 1. Planning Algorithm

Input: G: a list of goals
Output: The generated plan P

```
1:  L: layer
2:  initialise L to 0;
3:  while (G not present in L OR G not mutex-free in L) AND algorithm not levelled-off do:
```

```
4:          expand L;
5:          increment L;
6:    end while

7:    if G not present in L OR G not mutex-free in L then:
8:          return P = FAILURE;
9:    end if

10:   P = extract G from L;

11:   if P = TIME_FAILURE then:
12:         while P = TIME_FAILURE AND G not empty do:
13:               select goal from G with lowest salience value;
14:               remove goal from G;
15:               search for L where G present in L;
16:               P = extract G from L;
17:         end while
18:   end if

19:   return P;
```

The complete extraction phase is documented in Algorithm 2. The algorithm receives as an input a list of goals G, and the layer of the planning graph L which needs to be expanded. As it is shown in Algorithm 2 – lines 1-9, if the algorithm is not levelled-off, then the facts (possible states of the world) of layer L are retrieved, and a logical or is applied with the facts of the previous layer (L-1), so the previously applied actions can be copied.



Initial State & Goal List
Expand Level
Update Current State
NO
Are all Goals present in Current State?
YES
Perform backward search
Find level which contains all the goals
Remove least important Goal
Stop Search
NO
Is total Time within limit?
YES
Is Plan complete?
YES
Return Plan
Calculate total Plan Time
Add applicable Actions to Plan
Search in last unsearched Level
NO

Fig. 3. The planning process.

Afterwards, a check is being performed to identify which actions are applicable in L, the facts of the layer L are updated with the effects of these actions, the mutexes between the actions are calculated and stored, and the resolvers (used to speed up the search process) are updated. In the case that the algorithm is levelled-off, Algorithm 2 – lines 10-15, all the information for layer L is the same with the layer L-1.

ALGORITHM 2. Expand function

Input: G: a list of goals; L: planning graph layer
Output: The data in the expanded layer L

```
1:  if algorithm is not levelled-off then:
2:      get facts of L;
3:      if L > 0 then:
4:          apply logical or with facts in L-1 to copy results of previous actions;
5:      end if
6:      check which actions are applicable in L;
7:      update facts of L based on positive & negative effects of applicable actions;
8:      calculate mutexes of actions in L;
9:      update resolvers of actions in L (to speed up the search process);
10: else
11:     facts of L = facts of L-1;
12:     applicable actions in L = applicable actions in L-1;
13:     mutexes in L = mutexes in L-1;
14:     resolvers in L = resolvers in L-1;
15: end if
```

After the planning graph is created, the extraction phase begins, where a backward search is being performed starting from the last generated layer (containing all the goals) until a valid plan is discovered. This is shown in Algorithms 3 and 4. The search function searches iteratively for a set of actions that resolve a set of goals G in a layer L. These sets of actions form a complete plan. While searching, the total time that the plan needs to be performed is calculated (Algorithm 4 – line 21). If at some point the plan time exceeds the available time, the search stops.

ALGORITHM 3. Extract function

Input: G: a list of goals; L: planning graph layer
Output: A plan P

```
1:  if L = 0 then:
2:      return P = EMPTY;
3:  end if

4:  P = search for G in L;

5:  return P;
```

Then, as it is shown in Algorithm 1 – lines 11-18, the planner chooses the least important goal from the list (based on the “importance” value of the goals), removes it and restarts the search process from the level that all the remaining goals are present. This happens until a valid plan within the time constraints has been discovered.

ALGORITHM 4. Search function

Input: G: a list of goals; L: planning graph layer; A: a set of actions; M: a set of mutexes
Output: A plan P

```
1:  if G not empty then:
2:      select goal from G;
```

```
3:        get actions that resolve goal;
4:        while actions not empty AND P = NULL do:
5:            select action from actions;
6:            remove action from actions;
7:            select mutexes for action;
8:            newA = A + action;
9:            newM = M + mutexes;
10:           newG = G – positive effects of action;
11:           P = search for newA, newM, newG, in l;
12:       end while
13:   else
14:       for each A as action do:
15:           add action's positive precondition in G;
16:       end for each
17:       P = extract for G in L-1;
18:       if P is not FAILURE OR TIME_FAILURE then:
19:           for each A as action do:
20:               add action to P in layer L-1;
21:               calculate duration of P;
22:               if duration of P > available duration for P then:
23:                   P = TIME_FAILURE;
24:               end if
25:           end for each
26:       end if
27:   end if

28:   P = search for G in L;

29:   return P;
```

In the re-planning phase, the planning graph which provided the valid plan in the planning phase is kept. When a re-planning process needs to be performed, the previously generated planning graph is used to identify the action(s) needed to be changed to generate a new valid plan and to achieve the new goal(s). This approach helps to achieve the minimal possible disruption to the original plan.

Our proposal for re-planning in DIS aims at the following: to interleave plan generation and plan execution; to be able to re-plan when unexpected changes occur in dynamic environments; and to make minimal disruption to the original plan.

Our re-planning algorithm is illustrated in Algorithm 5. The first step is to get the set of actions related to the step of the plan which failed. The effects of each action which is either failed or not executed yet are added as the goals of the new plan. If there are more steps in the plan, then the effects of the actions of the next step are also added in the goals list.

ALGORITHM 5. Re-planning algorithm

Input: P: an existing plan; S: step of the plan which failed;
Output: A new plan P

```
1:   A: set of actions; p: partial plan; G: goals list;

2:   get A from P for S;
3:   for each A as action do:
4:       if action is failed OR action is not complete then:
5:           get effects of action;
6:           add effects in G;
7:       end if
```

```
8:  end for each

9:  if size of $P > S$ then:
10:     get A from P for $S + 1$;
11:     for each A as action do:
12:         get effects of action;
13:         add effects in G;
14:     end for each
15: end if

16: $p$ = generate plan for G;
17: if size of $P > S$ then:
18:     remove actions of $S + 1$ from P;
19: end if
20: remove completed and failed actions from P;
21: add actions of p at the beginning of P's actions list;

22: return P;
```

Afterwards, a new partial plan is generated for these goals via our planning algorithm. After the generation of the new partial plan, the completed and failed actions (as well as the actions of the step right after the failed step, if such a step exists) of the existing plan are removed, and the two plans (the existing and the new partial plan) are merged into one, which is returned to the agent for execution.

3.4 Agent Component and Multi-Agent Coordination

The system is a multi-agent solution. Each agent represents a character in the game world and uses an instance of the planner to be able to generate plans of actions and regenerate them if needed.

The agent's architecture, which is illustrated in a part of figure 1, follows a hybrid approach; it includes elements of reactive agents (the agent receives input, processes it and produces an output), elements of deliberative agents (holds an internal view of its environment) and elements of BDI agents (Beliefs – the agent's view of the world, Desires – the agent's goals, Intentions – the agent's plans) [Vlachavas et al. 2005].

In more detail, using its Communication Module, the agent receives the current state of the world relevant to it by the WM and stores it in its own KB, along with its own list of goals and generated plan(s). A set of sensors is able to monitor the agent's Area of Interest (i.e. the environment which the agent is aware of, which is updated by the WM), to be able to recognise other agents and resources, and to be able to report back to the agent (via communicating with the WM) any problems in the execution of the plan that needs to be resolved by re-planning. The planner is used both for planning and for re-planning purposes.

The agents communicate with each other using a centralized Blackboard system located in the WM. At some point during the implementation phase, this could change to become a decentralized direct messaging system. Regarding the communications between the agents, we will use a generic communication language such as ACL [Vlachavas et al. 2005].

We will not deal in much detail with the negotiation problem between the agents, since most of the potential conflicts are solved by resource reservation. That is, when an agent is asked to execute a set of actions in a time step, the system reserves the resources that are need to complete the actions (if they are available), so most of the conflicts are resolved before they occur since the agents need to make the resource reservation before trying to complete an action. Upon the completion of each action, the resources reserved are released, unless they have been consumed. The actions are therefore atomic and uninterruptable.

The agents receive a list of goals which correspond to the currently active level by the WM. Each goal has a salience value which is used to specify a hierarchy which can be extended by the storyteller, by also adding preconditions for them. Each agent will set as its current goals only the goals that have their preconditions met. After a plan is executed (or failed), the agent checks which of the goals are complete and re-populates its list of current goals, also considering any injected goals based on executed actions.

Algorithm 6 presents the execution of the agents' plans in a given time step. In brief, the WM queries all the agents one by one to identify if they have an active plan from which they want to execute a specific action, and if not instructs them to generate one.

If the agent is not busy at the given time step (i.e. if it is not involved in another action already), identifies if any other agents are involved in the action, and checks their availability as well.

ALGORITHM 6. Multi-agent execution of plans

Input: A: a list active agents; T: a time step; C: the current state of the world
Output:

```
1:  for each A as agent do:
2:      if agent has plan do:
3:          get next set of actions from plan;
4:      else
5:          generate new plan;
6:          get next set of actions from plan;
7:      end if
8:      if agent not busy at T do:
9:          for each actions as action do:
10:             identify agents involved in action;
11:             if agents not busy at T do:
12:                 if preconditions of action exist in C do:
13:                     if action marked as choice do:
14:                         get choice result (YES/NO/LATER) either from user or from
                            system;
15:                     else
16:                         choice = YES;
17:                     end if
18:                     if choice = YES do:
19:                         set involved agents busy at T;
20:                         mark action as executed;
21:                         update C with action effects;
22:                         inject new goals based on action;
23:                     else if choice = NO do:
24:                         mark action as failed;
25:                         remove agent's goal corresponding to action;
26:                         stop execution of all actions;
27:                         instruct agent to re-plan;
28:                     else if choice = LATER  do:
29:                         instruct agent to wait;
30:                     end if
31:                 else
32:                     mark action as failed;
33:                     stop execution of all actions;
34:                     instruct agent to re-plan;
```

```
                end if
            else
                instruct agent to wait;
            end if
        end for each
    else
        instruct agent to wait;
    end if
end for each
```

Afterwards, the preconditions of the action which is set to be executed are checked against the current state of the world, which is updated after every successful execution of an action. If the preconditions are met, then the system checks if the action is marked by the storyteller as a choice.

If it is, the system deals with the choice, and then, based on all the previously mentioned conditions, proceeds with one of the following:

- Executes the action, updates the current state of the world, and injects new goals if needed;
- Marks the action as failed, stops the execution of other actions for the specific agent, and remove the goal in which the action is related to if needed;
- Delays the execution of the action for the next time step, instructing the agent to wait.

As already mentioned, the storyteller is able to mark specific actions as choices, and select if the decision will be made by the system or the user. There are three different outcomes to a choice:

- Yes, where the action is set to be executed as usual;
- No, where the action is not going to be executed at all, and the goal related to this action is going to be removed from the agent's goal list;
- Later, where the action will fail for now, but the agent is free to try and execute it again at a later time step.

The storyteller is also able to assign a list of futile goals that are applicable in a level and they are controlled by a component called Futile Goals Manager. As a futile goal, we consider a goal that does not add something really useful to the story, but can keep an agent busy if there nothing important to do. Therefore, if at some point of the story an active agent does not have any goals, a futile goal is injected and a plan is generated for the goal to be complete.

4. CASE STUDY AND PLANNING METHODS EVALUATION

4.1 Troy Story Scenario

We are currently in the process of creating a large DIS scenario with several levels which will allow us to further evaluate and expand our system.

A brief summary of our scenario: It models the story of Troy (derived from Homer's epic poem, "Iliad" [Homer 2003]), where Paris, the Prince of Troy (brother of Hector; son of Priam) falls in love with Helen, the Queen of Sparta, seduces her and convinces her to join him and flee together to Troy. Menelaos, the King of Sparta and husband of Helen, along with his brother Agamemnon (King of Mycenae) decides to gather an army of great Greek warriors (such as Achilles, who despises Agamemnon) to attack and conquer Troy, each of them for their own reason. Menelaos because he's angered that Helen left him and Agamemnon because he always wanted to find a reason to conquer Troy.

The story is very rich in relations between characters, actions, choices, possible constraints and outcomes, making it a suitable setup for rich emerging narratives to be developed in a DIS system.

4.2 Troy Scenario Coordination Example

In this section we introduce as example, a scenario which includes characters' action sequence, agents' coordination and injection of goals. This scenario involves the following:

- Characters: Achilles, Odysseus, Patroclus, Hector, Paris, and Priam;
- Locations: Achilles' tent, Greek Camp, Greek Camp's Beach, Battlefield, and Troy Palace;
- Actions: go to, pick up, drop, ask for, allow to take, wear, talk to, tell, kill, prepare for funeral, perform funeral, attend funeral, find, wait.

The characters, locations, and actions vary in each level of our complete scenario, depending on the nature and context of the level, which can help managing the complexity of the scenario, for example by reducing the number of actions considered in the planning and re-planning phases.

Achilles and Priam are initially located at Achilles' tent and Troy palace respectively. Odysseus, Patroclus, Hector, and Paris are located at the battlefield, and they are in the middle of the battle.

At some point, Hector kills Patroclus. This triggers a goal injection for Odysseus, which now has the following goals: To bring Patroclus' body back to the Greek camp; and to inform Achilles that Patroclus was killed by Hector. So, Odysseus leaves the battle, picks up Patroclus' body, and brings it back to the Greek camp. Afterwards, he goes to the tent where Achilles is located, and shares the news about Patroclus' death.

Achilles now has the following goals: To prepare for battle, to find and kill Hector as well as to defile his body by taking it with him to take revenge for Patroclus' death, and, finally, to perform Patroclus' funeral. He gears up, goes to the battlefield where he joins the on-going battle (Odysseus does the same as well), and kills Hector. When he does, that triggers the end of the battle, and he takes Hector's body with him, and returns to the camp (Odysseus returns as well) to prepare for Patroclus' funeral. When the preparation starts, the goal to attend the funeral is injected to Odysseus.

The death of Hector triggers a goal injection for Paris. He has the task to return to Troy and give the bad news to Priam. When he does, Priam decides to find Achilles and ask him to allow for the body of Hector to return to Troy to have a proper funeral. So, he wears a disguise to be able to infiltrate the Greek camp, and goes there. He finds where Achilles' tent is, but since Achilles is at Patroclus' funeral, he waits there until it's over.

When Achilles returns, the two men have a discussion, and Priam convinces Achilles to let him take Hector's body with him. Priam does that, returns to Troy, and performs a funeral for Hector, with Paris attending it.

Figure 4 displays the complete combined plan of Achilles.

```
PICK-UP ACHILLES SWORD ACHILLES-TENT
PICK-UP ACHILLES SHIELD ACHILLES-TENT
WEAR ACHILLES ARMOR ACHILLES-TENT
GO-TO ACHILLES ACHILLES-TENT BATTLEFIELD
ATTACK ACHILLES HECTOR BATTLEFIELD
KILL ACHILLES HECTOR BATTLEFIELD
PICK-UP ACHILLES HECTOR BATTLEFIELD
GO-TO ACHILLES BATTLEFIELD GREEK-CAMP
DROP ACHILLES HECTOR GREEK-CAMP
PICK-UP ACHILLES PATROKLUS GREEK-CAMP
GO-TO ACHILLES BEACH
PREPARE-FOR-FUNERAL GREEK ACHILLES BEACH
PERFORM-FUNERAL GREEK ACHILLES BEACH
GO-TO ACHILLES BEACH ACHILLES-TENT
ALLOW-TO-TAKE ACHILLES PRIAM HECTOR
```

Fig. 4. Achilles' combined plan.

Figure 5 illustrates all the interactions, coordination, and goal(s) injections between the characters involved in the scenario.

In future work, we will use Petri Nets [Hussein 2012] for the modelling of actions' execution, as well as agents' coordination, synchronisation, and communication.

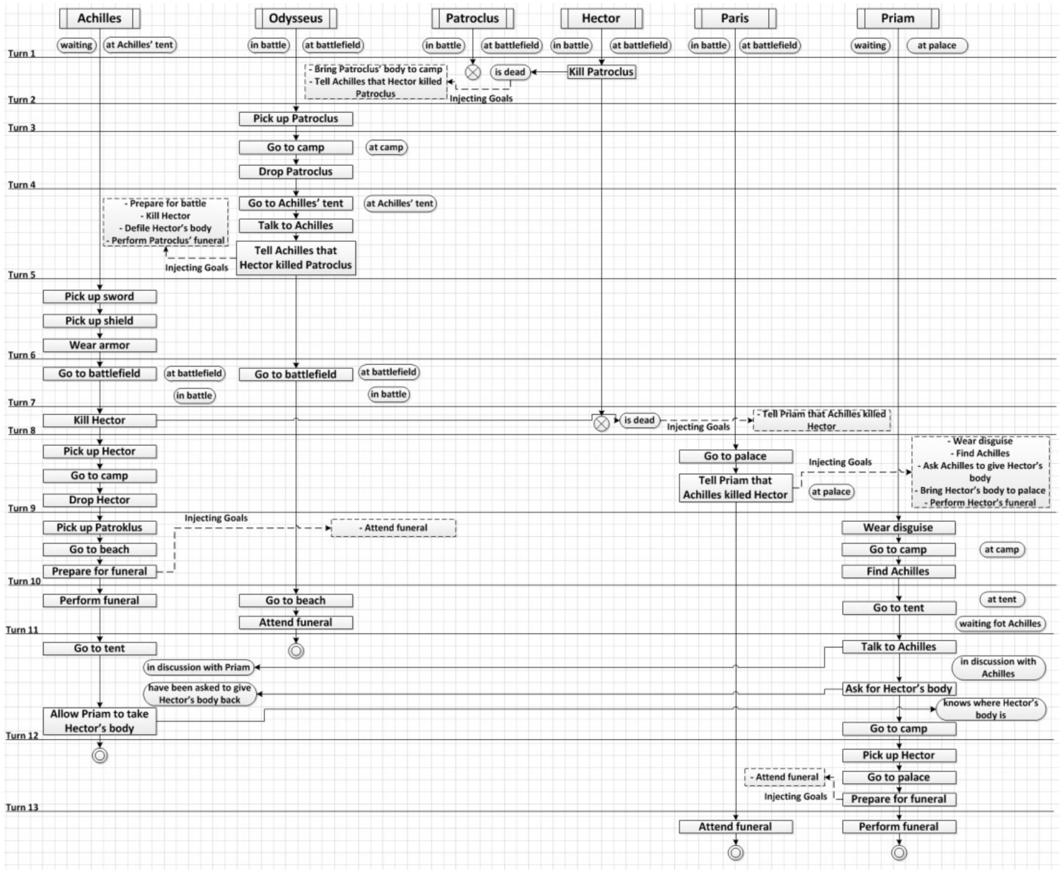


Fig. 5. Troy characters coordination.

4.3 Planning Methods Evaluation

In this section we discuss our planning methods evaluation for multi-agents and for individual agents with the different algorithms discussed in section 3.

4.3.1 Planning Performance for MAS

In our previous work [Goudoulakis et al. 2012b], we have introduced experiments to highlight the performance of our MAS when considering agents sharing the domain knowledge, versus agents each having their own domain.

The results revealed that when the domain knowledge is shared between the agents, there is a significant increase in performance. As an example, there was a 93.66% reduction in the total time needed to activate 100 agents and generate a plan for each one of them when the agents were sharing the domain knowledge, from when they did not.

In our current implementation, and in the following experiments, we are using this approach (i.e. all agents which are active in a given level share the domain knowledge), but in case we need to switch to the other approach sometime in the future, we solved its scalability issue by implementing a multi-threaded version of agent activation and initial planning. That reduced the activation and planning time by 62.18% comparing it with the non-sharing approach.

4.3.2 Planning Algorithm Performance

As we have discussed in section 3.3, we have considered two different planning methods. In the first one, each agent has a list of goals and generates a plan for all the goals at the same time, whilst in the second method a plan is generated only for the most important goal (based on the goal's "salience" value), and when the goal is complete (or not applicable any more), a plan for the next goal is generated, and so forth.

In this section we will discuss the evaluation we performed to identify which of the two approaches is better for our needs. To perform the evaluation, we used a scenario which includes the following:

- Characters: Paris, Hector, Menelaos, Helen, Agamemnon;
- Locations: Throne room, guest room, royal bedroom, kitchen, armory, dock, stable, and garden (all located in Sparta), as well as Troy's dock;
- Actions: talk to, walk to, travel to, pick up, drop, ask to give, give, kill, seduce, convince to follow, board, negotiate peace, drink, eat.

We are using Paris and Hector as our main characters, and the rest as "dummy" characters (they don't have any initial goals; the system injects goals to them during the execution based on other agents' actions, or just futile goals).

Hector has the following goals: first to give a present brought from Troy to Menelaos (since he is his guest), then to negotiate peace between Troy and Sparta, and finally to return back to Troy. Figure 6 displays a possible plan for Hector which achieves all his goals.

```
WALK-TO HECTOR GUEST-ROOM THRONE-ROOM
TALK-TO HECTOR MENELAOS THRONE-ROOM
GIVE HECTOR MENELAOS HECTOR-TROY-GIFT THRONE-ROOM
NEGOTIATE-PEACE HECTOR TROY MENELAOS SPARTA THRONE-ROOM
WALK-TO HECTOR THRONE-ROOM DOCK
BOARD HECTOR SHIPS DOCK
TRAVEL-TO HECTOR DOCK TROY SHIPS SEA
```

Fig. 6. Hector's plan.

Paris' goals are to give a present to Menelaos as well, but then to seduce Helen, convince her to follow him back to Troy, catch-up with Agamemnon who is around at the palace, and finally return to Troy. Figure 7 displays a possible plan for Paris which achieves all his goals.

```
WALK-TO PARIS GUEST-ROOM GARDEN
TALK-TO PARIS AGAMEMNON GARDEN
WALK-TO PARIS GARDEN ROYAL-BEDROOM
TALK-TO PARIS HELEN ROYAL-BEDROOM
CONVINCE-TO-FOLLOW PARIS HELEN ROYAL-BEDROOM TROY
WALK-TO PARIS ROYAL-BEDROOM THRONE-ROOM
TALK-TO PARIS MENELAOS THRONE-ROOM
GIVE PARIS MENELAOS PARIS-TROY-GIFT THRONE-ROOM
WALK-TO PARIS THRONE-ROOM DOCK
BOARD PARIS SHIPS DOCK
TRAVEL-TO PARIS DOCK TROY SHIPS SEA
```

Fig. 7. Paris' plan.

We measured the following, for both agents for both of the approaches: the total planning time needed (in seconds), the average time needed to generate a single plan, the total number of nodes in the planning graph, the average number of nodes for each plan, and the number of successful, failed, and wait actions during the scenario execution.

The data is an average of five runs for each case in a PC featuring 4GB of RAM, and a Quad-core CPU @ 2.83 GHz, running Windows 7.

Figure 8 illustrates the total planning time for each approach, for each agent, as well as the average planning time for each plan. (a) is the case where a plan is generated for all the goals at the same time, where (b) the case where a plan is generated for only one goal at a time.

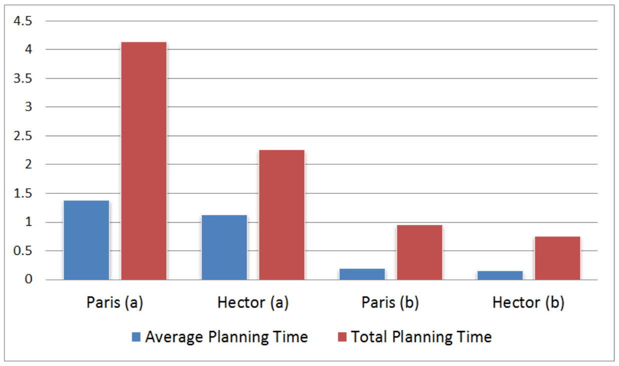


Fig. 8. Total and average planning times.

The total planning time for both of the agents in case (a) was 6.388 seconds, where in case (b) was only 1.703 seconds, a 73.34% decrease.

In case (a), the highest planning time that appeared was 2.216 seconds, and the lowest 0.735 seconds. In contrast, in case (b), the highest was 0.486 seconds, and the lowest 0.017 seconds.

The total average for both of the agents in case (a) was 1.2524 seconds, and in case (b) 0.1703 seconds, a decrease of 86.4%.

Based on the above numbers, it's clear that there is a significant performance increase using the case (b).

Figure 9 illustrates the total and average number of nodes generated in the planning graph while planning, for both agents, in both cases.

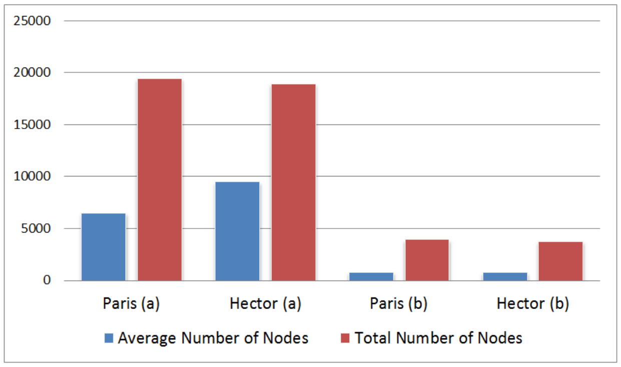


Fig. 9. Total and average number of nodes in planning graph.

The reduction in the number of nodes is consistent with the reduction in time. More specifically, the total number of nodes in case (b) is 80% less than in case (a), and the average number of nodes of case (a) are showing a 90.38% reduction comparing them with the average of case (b).

Figure 10 displays the successful actions execution, the failed actions, the number of times each agent was instructed to wait, and the total number of plans generated by each agent throughout the previously mentioned scenario. In case (a) the number consists of the initial plan plus the re-planning because of failures, and in case (b) consists of the plans for each individual goal, plus the re-planning because of actions' failure.

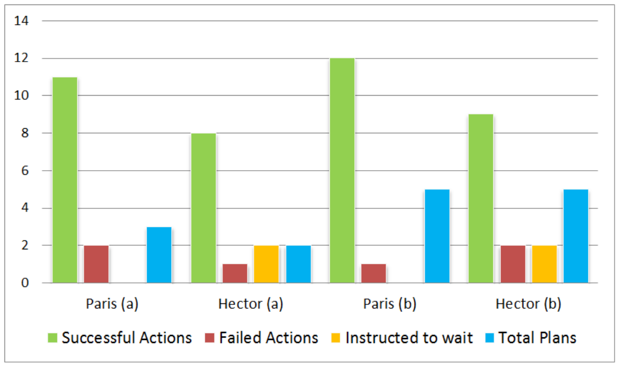


Fig. 10. Successful and failed actions, times instructed to wait, and numbers of total generated plans.

There were more plans generated in case (b), something which was expected since the agents needed to generate a plan for each individual goal, but since the complexity of each plan was quite lower than the plans of case (a), the planning times were quite lower as discussed before in figure 8, therefore the increased number of plans was not a negative impact on the performance.

The rest of the metrics in figure 10 are similar to each other between the two cases. The times an agent was instructed by the system to wait a turn before trying to execute its action again were identical, as well as the failed actions. There was a slight increase in the successful actions in case (b), which can potentially add to the emergence of the generated narrative.

5. DISCUSSION

Apart from implementing and evaluating the DIS framework and its algorithms, we are also considering alternative routes for it, such as extending it with other components.

One route would be to be connected with a 3D engine, able to visualise the outcome of the generated narrative. One potential candidate is the DISE framework developed in our previous work [Cooper et al. 2011].

For the game engine and implementation development aspect we will use our Homura Game Engine and development Framework [El Rhalibi et al. 2009]. We created the Homura project's game development framework to provide an Open Source (LGPL-Licensed) API for the creation of Java and Open GL based hardware-accelerated 3D games applications, which support cross platform, cross-browser deployment using Java Web Start (JWS) and Next-generation Applet technologies. Our framework bundles together several example applications and technical demos, which demonstrate and explain how to implement common games functionality in our applications; An application template, which acts as a great starting point for developing research applications and Homura related games; The APIs of both Homura and the key open-source projects it builds upon including the Java Monkey Engine scenegraph API, jME Physics Library, MD5 Model Importer, GBUI User Interface Libraries and many more; External Tools for the creation of Font Assets, Particle Effects and Levels for the games.

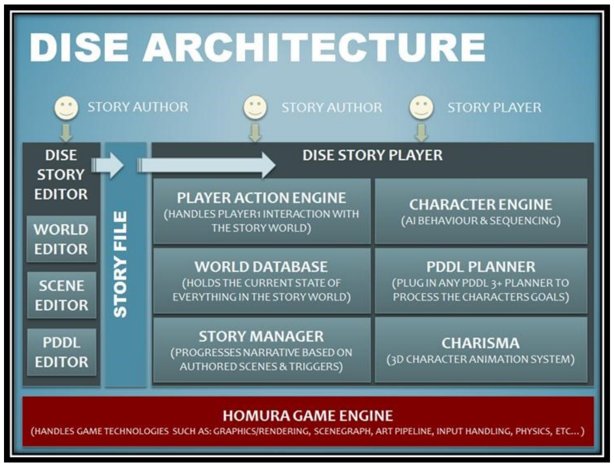
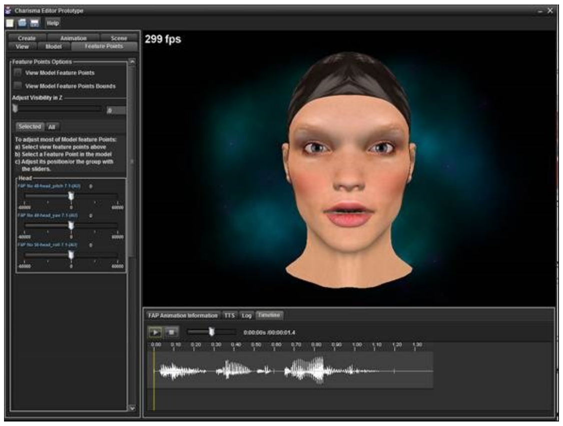

DISE ARCHITECTURE
STORY AUTHOR
STORY AUTHOR
STORY PLAYER
DISE STORY EDITOR
WORLD EDITOR
SCENE EDITOR
PDDL EDITOR
STORY FILE
DISE STORY PLAYER
PLAYER ACTION ENGINE (HANDLES PLAYER1 INTERACTION WITH THE STORY WORLD)
CHARACTER ENGINE (AI BEHAVIOUR & SEQUENCING)
WORLD DATABASE (HOLDS THE CURRENT STATE OF EVERYTHING IN THE STORY WORLD)
PDDL PLANNER (PLUG IN ANY PDDL 3+ PLANNER TO PROCESS THE CHARACTERS GOALS)
STORY MANAGER (PROGRESSES NARRATIVE BASED ON AUTHORED SCENES & TRIGGERS)
CHARISMA (3D CHARACTER ANIMATION SYSTEM)
HOMURA GAME ENGINE
(HANDLES GAME TECHNOLOGIES SUCH AS: GRAPHICS/RENDERING, SCENEGRAPH, ART PIPELINE, INPUT HANDLING, PHYSICS, ETC...)

Fig. 11. DISE architecture diagram shows the components which make up the system and the player/author interaction. [Cooper et al. 2011]

Another possible route is the integration with a system capable to generate a 3D facial model animation which will act as a narrator for the stories which our framework produces. A system which could be used for this purpose is our current work on developing a framework for MPEG-4 Based 3D Character with co-articulation and TTS - Charisma [El Rhalibi et al. 2010].



To adjust most of Model feature Points:
a) Select view feature points above
b) Select a Feature Point in the model
c) Adjust its position/or the group with the sliders.

Fig. 12. Charisma. [El Rhalibi et al. 2010]

When integrated with Homura [El Rhalibi et al. 2009] and Charisma [El Rhalibi 2010], our digital interactive storytelling system will provide a modern solution to develop interesting narratives.

6. CONCLUSION

In this paper, we briefly discussed the design and implementation of our framework, and then presented an agent coordination example, as well as a planning methods evaluation.

Some of the novel aspects of our framework are: the introduction of constraints in the planning/re-planning solutions that allows us to extend the power and expressiveness of the representation language that we use; and our proposal for re-planning, which aims to interleave plan generation and opportunistic plan execution, to be able to re-plan when unexpected changes occur in dynamic environments whilst introducing minimal disruption to the original plan.

In our future work we will continue the extension of our DIS framework and we will continue refining the planning and re-planning algorithms. We will proceed with a further evaluation of our framework and of the algorithms, using the complex DIS scenario that we will complete. We will also use Petri Nets for the modelling of DIS scenario to evaluate the properties of our coordination model, enabling actions' execution, agents' coordination, synchronisation, communication, and story resources sharing.

ACKNOWLEDGMENTS

The authors would like to thank the ACM Computers in Entertainment editors and reviewers for their constructive comments and suggestions which allowed us to improve this paper.

REFERENCES

BANGSO, O., JENSEN, O.G., JENSEN, F.V., ANDERSEN, P.B., AND KOCKA, T. 2004. Non-Linear Interactive Storytelling Using Object-Oriented Bayesian Networks. In *Proceedings of the International Conference on Computer Games: Artificial Intelligence, Design and Education*.

BARBER, H., AND KUDENKO, D. 2009. Generation of Adaptive Dilemma Based Interactive Narratives. In *IEEE Transactions on Computational Intelligence and AI in Games*, Vol. 1, Issue 4, 309-326.

BARROS, L.M., AND MUSSE, S.R. 2005. Introducing Narrative Principles Into Planning-Based Interactive Storytelling. In *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology*, 35-42.

BARROS, L.M., MUSSE, S.R. 2007. Planning Algorithms for Interactive Storytelling. *ACM Computers in Entertainment, Vol. 5, No. 1, Article 4.* DOI:http://dx.doi.org/10.1145/567752.567774

BLUM, A., AND FURST, M.1997. Fast planning through planning graph analysis. *Artificial Intelligence*, 90:281-300.

CAVAZZA, M., CHARLES, F., MEAD, S.J. 2002. Interacting with Virtual Characters in Interactive Storytelling. *In the Proceedings of the first international joint conference on Autonomous agents and* *multi-agent systems, AAMAS '02*.

CHANG, H.-M., AND SOO, V.W. 2009. Planning-Based Narrative Generation in Simulated Game Universes. In *IEEE Transactions on Computational Intelligence and AI in Games*, Vol 1 (3), 200-213.

CHARLES, F., LOZANO, M., MEAD, S.J., BISQUERRA, A.F., AND CAVAZZA, M. 2003. Planning Formalisms and Authoring in Interactive Storytelling. In *Proceedings of the 1st International Conference on Technologies for Interactive Digital Storytelling and Entertainment*.

COOPER, S. EL RHALIBI, A., AND MERABTI, M. 2011. DISE: a Game Technology-based Interactive Storytelling Framework. In *Proceedings of the 12th Annual Post Graduate Symposium on the Convergence of Telecommunications, Networking and Broadcasting (PGNet2011).*

EL RHALIBI, A., CARTER, C., COOPER, S., MERABTI, M., AND PRICE, M. 2010. Charisma: High Performance Web Based MPEG-4 Compliant Animation Framework, *Journal ACM Computers in Entertainment*, Vol. 8, Iss. 2.

EL RHALIBI, A., MERABTI, M., CARTER, C., DENNETT, C., COOPER, S., SABRI, M.A., AND FERGUS, P. 2009. 3D Java Web-Based Games Development and Deployment. *International Journal on Information and Communication Technologies*, Vol. 2 no. 3-4, 221-230.

HUSSEIN, K. 2012. *Petri Nets - State of the Art in Interactive Storytelling Technology: An Approach Based on Petri Nets*. Petri Nets - Manufacturing and Computer Science.

FAIRCLOUGH, C.R., AND CUNNINGHAM, P. 2004. AI Structuralist Storytelling in Computer Game. In *Proceedings of the International Conference on Computer Games: Artificial Intelligence, Design and Education*.

GOUDOULAKIS, E., EL RHALIBI, A., MERABTI, M., AND TALEB-BENDIAB, A. 2011. Evaluation of planning algorithms for digital interactive storytelling. *12th Annual Postgraduate Symposium on The Convergence of Telecommunications, Networking and Broadcasting*.

GOUDOULAKIS, E., EL RHALIBI, A., MERABTI, M., AND TALEB-BENDIAB, A. 2012. Opportunistic Multi-Agent Digital Interactive Storytelling System. *13th Annual Post Graduate Symposium on the Convergence of Telecommunications, Networking and Broadcasting*.

GOUDOULAKIS, E., EL RHALIBI, A., MERABTI, M., AND TALEB-BENDIAB, A. 2012. Framework for multi-agent planning and coordination in DIS. In *Proceedings of the Workshop at SIGGRAPH Asia WASA '12*, 65-71.

HOMER 2003. *Iliad*. Penguin books.

INTERNATIONAL PLANNING COMPETITION, 2011. PDDL 3.1 Specifications. http://www.plg.inf.uc3m.es/ipc2011-deterministic/Resources (accessed on 3/3/13).

MATEAS, M., AND STERN, A. 2003. Façade: An Experiment in Building a Fully-Realised Interactive Drama. In *Game Developers Conference Proceedings GDC '03*.

RUSSELL, AND S., NORVIG, P. 2010. *Artificial Intelligence: A Modern Approach*. Third Edition, Pearson Education Inc.

SPIERLING, U. 2005. Interactive Digital Storytelling: Towards a Hybrid Conceptual Approach. In *Proceedings of DiGRA 2005 Conference: Changing Views – Worlds in Play*.

VLACHAVAS, I., KEFALAS, P., VASILIADIS, N., KOKKORAS, F., AND SAKELLARIOU, E. 2005. *Artificial Intelligence*. Ed. Gartaganis.