Proceedings of the 3rd International
Planning in Games Workshop

PG 2013

Rome, Italy - June 10, 2013

Edited By:

Michael Buro, Éric Jacopin, Stavros Vassos

Organizing Committee

Michael Buro
University of Alberta, Canada

Éric Jacopin
CREC Saint-Cyr, Écoles de Coëtquidan, France

Stavros Vassos
“La Sapienza” University of Rome, Italy

Program committee

Marc Cavazza, University of Teesside, United Kingdom
Carle Coté, Eidos Montréal, Canada
Luke Dicken, University of Strathclyde, United Kingdom
Alan Fern, Oregon State University, United States
Peter Gregory, University of Teesside, United Kingdom
Carlos Linares López, Universidad Carlos III de Madrid, Spain
Christian Muise, University of Toronto, Canada
Jeff Orkin, MIT Media Lab, United States
Julie Porteous, University of Teesside, United Kingdom
Mark Riedl, Georgia Institute of Technology, USA
William van der Sterren, CGF-AI, Netherlands

Foreword

Artificial Intelligence Planning is successfully used in video-games: heuristic-based STRIPS-like planning and HTN Planning generate character behavior in several fast paced games since 2005, reaching millions of players. This certainly does not make planning in games a solved problem: from new game genres to next-generation consoles and new markets such as cloud gaming, the AI Planning research frontier is wide and open to any kind of planning technique in a gaming context.

This 3rd edition of the ICAPS Workshop on Planning in Games shall acknowledge the tighter link with the video-game industry, while aiming at inspiring traditional Game AI Planning research such as optimal planning in huge search spaces and temporal reasoning.

The papers gathered in these proceedings are divided in three themes: (*i*) planning about paths (papers 1-3), (*ii*) planning about stories (papers 4-5) and (*iii*) planning about actions (papers 6-8).

Table of Contents

Game Applications of HTN Planning with State Variables

Dana Nau

nau@cs.umd.edu

One reason why AI planning technology has not seen wider use in games is that the classical AI planning representation -- in which states of the world are sets of propositions, and actions modify states by adding and deleting propositions -- is different from the data structures used in ordinar computer programming, making it difficult to integrate AI planners with application programs.

Recently there has been renewed interest in state-variable representation, in which states are represented by assigning values to variables, and actions modify states by changing the values of those variables.

In this talk, I'll

(1) give a brief, informal introduction to state-variable representation, with examples.

(2) describe Pyhop, a Hierarchical Task Network (HTN) planning system written in Python that uses state-variable representation. The use of state variables has allowed close integration between planning and ordinary computational operations, which has helped to make the implementation of Pyhop quite simple.

(3) discuss ways that HTN planning has already been used in game environments. I'll discuss possible ways to modify Pyhop for use in game environments, and will solicit suggestions from the audience.

If there's sufficient interest, I can make Pyhop available to AI game developers as open-source software.

Planning for Game Characters

Alex Champandard

alexjc@aigamedev.com

This talk starts with an overview of AI applied to game characters, in particular the action games that use planners most extensively. Both the technology and authoring angles will be covered, emphasizing the tradeoffs between classical planners (e.g. STRIPS) and hierarchical task decomposition planners. Avenues for that are fruitful for research will be identified, as well as ideas for planning competitions suited to applications in the games industry.

Hierarchical Plan-Space Planning for Multi-Unit Maneuvers

William van Sterren

william@cgf-ai.com

In combat simulators and war games, coming up with a good plan is half the battle. Good plans make the AI a more convincing opponent and a more reliable assistant commander. This presentation describes the design of an AI planner capable of producing plans which coordinate multiple units into a joint "combined arms" maneuver on the battlefield. First it looks at how planning for multiple units is different from planning for a single unit, including some of the speaker's struggles with well known game AI and planning approaches. Then it introduces the basic ideas of hierarchical plan-space planning. Next, the presentation discusses the concrete implementation of these ideas in a combat maneuvers "mission generator" and the changes made to address more units on larger terrains. The presentation concludes with a brief evaluation of the chosen approach.

Local and Global Planning for Collision-Free Navigation in Video Games

Jamie Snape and **Stephen J. Guy** and **Ming C. Lin** and **Dinesh Manocha**
Department of Computer Science
University of North Carolina at Chapel Hill
Chapel Hill, North Carolina 27599
{snape, sjguy, lin, dm}@cs.unc.edu

1 Introduction

Collision avoidance and navigation among virtual agents is an important component of modern video games. Recent developments in commodity hardware, in particular the utilization of multi-core and many-core architectures in personal computers and consoles are allowing large numbers of virtual agents to be incorporated into game levels in increasing numbers and with increasing fidelity. As a result, there is a need for efficient techniques to automatically generate realistic behaviors for such groups of virtual agents.

Simple local collision avoidance behaviors, such as flocking (Reynolds 1987), have been implemented using force-based models in many recent video games and commercial game engines. These methods model groups of virtual agents as particle systems, with each particle applying a force on nearby particles. The laws of physics are used to compute the motion of the particles, along with a set of behaviors specified by game developers that influence properties of the system such as separation, alignment, and cohesion of particles. Examples of video games using this method include Capcom's Dead Rising (2006) and Ubisoft's Assassin's Creed (2007).

More recently, velocity-based methods (Fiorini and Shiller 1998) have exhibited improvements in terms of local collision avoidance and behavior of virtual agents, and improved computational performance, over force-based collision avoidance methods. Rather than using virtual forces to prevent nearby virtual agents from collisions, velocity-based methods use the current velocity of each virtual agent in the group and then extrapolate the position of each virtual agent for some short time interval under the assumption that the virtual agent will maintain almost a constant velocity over some short time interval. Based on predicting the future positions of other virtual agents, each virtual agent tends to choose an avoiding new velocity based on some optimization. THQ's video game Warhammer 40,000: Space Marine (2011) uses a velocity-based approach.

Reciprocal collision avoidance (van den Berg et al. 2008) is an extension of the velocity-based approaches. The main difference with prior velocity-based methods lies in the fact that reciprocal collision avoidance considers the reciprocity between pairs of virtual agents. Each virtual agent is assumed to be attempting to avoid a collision with the other, rather than seeing the other virtual agent as a moving obstacle. Incorporating reciprocity into velocity-based approaches typically ensures smoother motion for the virtual agents and may also cause emergent phenomena in groups of virtual agents, such as arching, jamming, bottlenecks, and wake formation (Guy et al. 2010).

2 Local Collision Avoidance

In this section, we present two different local collision avoidance algorithms based on reciprocal velocity obstacles. They use slightly different schemes to compute new velocities for each agent, as shown in Fig. 1.

2.1 Hybrid Reciprocal Velocity Obstacles

The hybrid reciprocal velocity obstacle (Snape et al. 2011) resolves the problem of reciprocal dances by combining the velocity obstacle and the reciprocal velocity obstacle, taking one side from each to form a hybrid reciprocal velocity obstacle that is enlarged on one side to discourage virtual agents from passing each other on different sides. If the velocity of a virtual agent is to the right of the centerline of its reciprocal velocity obstacle induced by some other virtual agent, then the virtual agent should choose a velocity to the right of the reciprocal velocity obstacle. To encourage such behavior, the reciprocal velocity obstacle is enlarged by replacing the edge on the side that the virtual agents should not pass, for example, the left side in this case, by the edge of the corresponding velocity obstacle. If the velocity of the virtual agent is to the left of the centerline, the procedure is mirrored, exchanging left and right sides. The geometric interpretation of a hybrid reciprocal velocity obstacle $HRVO_{A|B}$ for a virtual agent A with respect to a virtual agent B, including the location of the centerline and an indication of the enlarged area, is shown in Fig. 1.

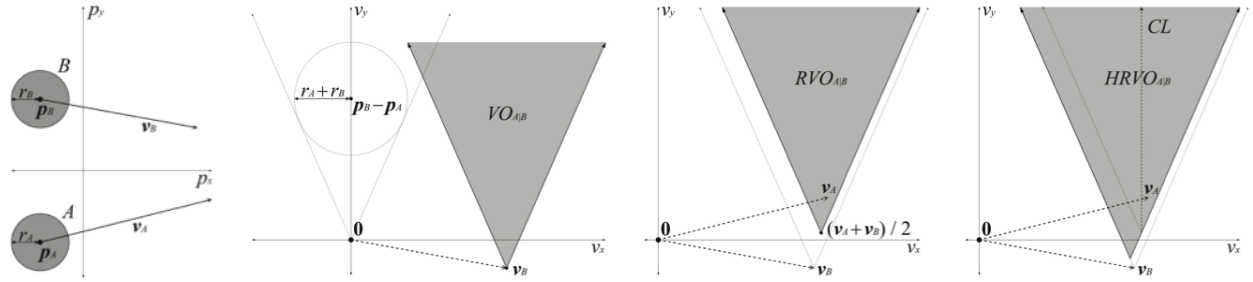


Figure 1: (a) Two virtual agents A and B. (b) The velocity obstacle $VO_{A|B}$ for virtual agent A induced by virtual agent B. (c) The reciprocal velocity obstacle $RVO_A|B$ for virtual agent A induced by virtual agent B. (d) The hybrid velocity obstacle $HRVO_{A|B}$ for virtual agent A induced by virtual agent B. The current velocity $\boldsymbol{v}_A$ is right of the centerline CL, so the left side of $HRVO_{A|B}$ is the left side of $VO_{A|B}$ and the right side of $HRVO_{A|B}$ is the right side of $RVO_{A|B}$.

2.2 Optimal Reciprocal Collision Avoidance

Optimal reciprocal collision avoidance (van den Berg et al. 2011) solves the problem of reciprocal dances addressed by the hybrid reciprocal velocity obstacle in a different way. This approach augments the velocity obstacle with a half-plane that defines a set of velocities that are both collision free and will additionally ensure that the motion of the virtual agents will be smooth in all but dense scenarios.

The optimal reciprocal collision avoidance half-plane $ORCA_{A|B}$ for a virtual agent A with respect to a virtual agent B is defined as follows. As shown in Fig. 2, let $\boldsymbol{u}$ be the vector from the relative velocity $\boldsymbol{v}_A - \boldsymbol{v}_B$ of the virtual agents A and B to the closest point on the boundary of the truncated velocity obstacle for virtual agent A induced by virtual agent B. Let $\boldsymbol{n}$ be the outward normal of the boundary of the velocity obstacle at $\boldsymbol{v}_A - \boldsymbol{v}_B + \boldsymbol{u}$. It follows that $\boldsymbol{u}$ is the smallest change required to the relative velocity of virtual agents A and B to avoid a collision. Incorporating reciprocity, each virtual agent should adjust its velocity by at least $\frac{1}{2}\boldsymbol{u}$ to avoid the collision. Therefore the velocities permitted by optimal reciprocal collision avoidance are in a half-plane in the direction of n starting at the point $\boldsymbol{v}_A + \frac{1}{2}\boldsymbol{u}$.

3 Global Navigation

We use standard well-known techniques for global navigation of agents. The simplest approach to global navigation in games is based on roadmaps (Latombe 1991). In roadmap-based methods, game agents are constrained to the edges of a graph between intermediate goal nodes (way points). Increasingly, navigation meshes (Snook 2000; Kallmann 2010; Van Toll et al. 2011) and similar methods (Pettré et al. 2005; Geraerts et al. 2008) have begun to supplant roadmaps in games. Navigation meshes are a decomposition of the freespace of game world into a mesh consisting of convex polygons and the connectivity or neighborhood information is represented using a graph as well. In practice, navigation meshes have advantages as all edges of a polygon are implicitly connected to each other. Moreover, a single navigation mesh can encode clearance for arbitrarily sized agents. Finding a global path with a navigation mesh consists of searching the connectivity graph for the shortest path between two polygons. Many techniques are known in the literature to combine local techniques based on reciprocal velocity obstacles with roadmaps (van den Berg, Patil et al. 2008) and navigation meshes (Curtis et al. 2012).

4 Implementation

4.1 Libraries

The hybrid reciprocal velocity obstacle approach and optimal reciprocal collision avoidance have been implemented as C++ libraries, *HRVO Library*[1] and *RVO2 Library*,[2] respectively.

Essentially, the algorithm in *RVO2 Library* computes the optimal reciprocal collision avoidance half-planes for a virtual agent induced by the other virtual agents, and then intersects these half-planes to form a region of permitted velocities for the virtual agent. The algorithm then computes the preferred velocity (see Section 4.2) of the virtual agent and computes a new velocity using two-dimensional linear programming (see Section 4.3) that is within the region of permitted velocities and as close as possible to the preferred velocity. If there are many virtual agents nearby and there is no velocity within the region of permitted velocities, then some constraints are relaxed and a new velocity is found using three-dimensional linear programming (see Section 4.4).

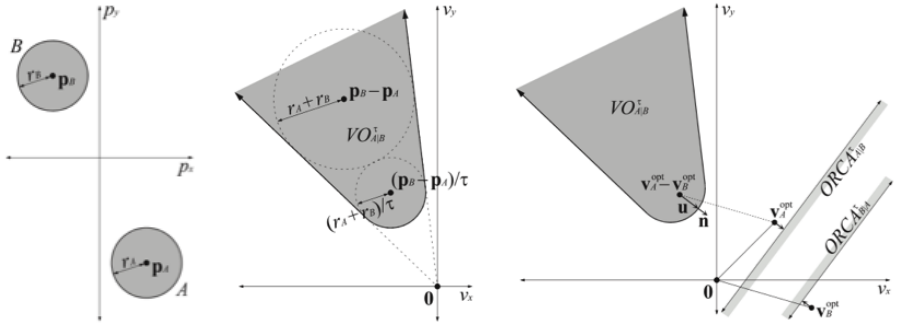
The algorithm used in *HRVO Library* is broadly similar except that it uses the ClearPath geometric algorithm (Guy et al. 2009) to compute new velocities.

4.2 Preferred Velocity

Both *HRVO Library* and *RVO2 Library* choose a new velocity by computing the velocity that is closest to the preferred velocity and is collision free. If the goal position of the virtual agent is visible, then the preferred velocity is in the direction of the goal. If the goal position is not visible, the preferred velocity should be directed to the nearest node on waypoint graph to the goal or to some point on the nearest edge on a navigation mesh path or a roadmap that leads to the goal.

[1] See http://gamma.cs.unc.edu/HRVO/.

[2] See http://gamma.cs.unc.edu/RVO2/.



(a) (b) (c)

Figure 2: (a) Two virtual agents A and B. (b) The truncated velocity obstacle $VO_{A|B}$ for virtual agent A induced by virtual agent B. (c) The optimal reciprocal collision avoidance half-planes of permitted velocities $ORCA_{A|B}$ for virtual agents A and B.

4.3 Linear Programming

RVO2 Library uses an efficient randomized linear programming algorithm (de Berg et al. 2008) that adds the constraints one by one in random order while keeping track of the current optimal new velocity for a virtual agent in the group. Linear programming is an optimization technique, commonly used in operations research, for finding one specific solution to a set of linear equality and inequality constraints that optimizes a given linear function of the variables. Geometrically, a linear programming algorithm computes a point in a polygon where the function has its maximum or minimum value if such a point exists. A randomized linear programming algorithm adds the linear constraints in a random order in order to compute the optimum solution.

For each virtual agent in the group, the randomized linear programming algorithm has a linear expected running time with respect to the number of virtual agents that are input into the algorithm. The algorithm computes the velocity in that is closest to the preferred velocity of the virtual agent, and reports failure if the linear program is infeasible.

4.4 Dense Scenarios

In dense scenarios, when a group of virtual agents is packed tightly together in part of the game level, there may not be a velocity that satisfies all the constraints of the linear program and the algorithm would return that the linear program is infeasible. When this occurs, *RVO2 Library* computes the safest possible velocity for the virtual agent, the velocity that minimally penetrates the constraints induced by the other virtual agents. This can be interpreted geometrically as moving the edges of the half-planes perpendicularly outward with equal speed, until exactly one velocity becomes valid. This velocity may be computed using a three-dimensional linear program. The same randomized linear programming algorithm as before may be used by projecting the problem down onto plane, such that all geometric operations can be performed in two-dimensions. The three-dimensional linear program is always feasible, so it always returns a solution. The running time of the algorithm is still linear with respect to the number of virtual agents.

Figure 3: A screenshot of the benchmark scenario for an integration of *RVO2 Library* with Unreal Development Kit, two hundred virtual agents navigating in real time between randomly chosen locations at the four corners of the game level.

4.5 Game Engine Integration

RVO2 Library has been integrated into several game engines to either perform local collision avoidance and navigation for groups of virtual agents or improve upon the default implementations provided by the game engine developers. Examples include a multi-platform package for Unity Technologies' Unity 3[3] written in C# and a DLL for Epic Games' Unreal Development Kit[4] written in C++ for Microsoft Windows with UnrealScript bindings. A screenshot from the Unreal Development Kit integration is shown in Fig. 3.[5] An approach broadly similar to the hybrid reciprocal velocity obstacle, as used in *HRVO Library*, has been incorporated into the Detour component of the game navigation toolset Recast and Detour[6] to provide local collision avoidance within a navigation mesh, as shown in Fig. 4.

[3] See http://rvo-unity.chezslan.fr/.

[4] As of mid 2011, Unreal Development Kit contains an implementation of the reciprocal velocity obstacle approach. Details of the integration of RVO2 Library into the game engine are available at http://gamma.cs.unc.edu/RVO2-UDK/.

[5] See http://youtu.be/x8dczNzxM0w for a video.

[6] See http://code.google.com/p/recastnavigation/.

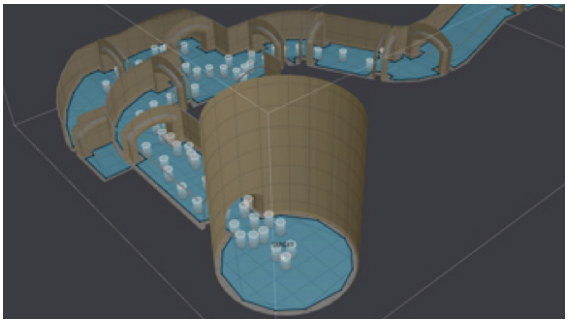


Figure 4: A screenshot of the Dungeon scenario included with *Recast and Detour*, fifty virtual agents navigating from one end of the game level to the other on a navigation mesh using *HRVO Library*.

5 Conclusion

We have presented the hybrid reciprocal velocity obstacle and optimal reciprocal collision avoidance methods for reciprocal collision avoidance and navigation in video games and described their implementations in C++ as *HRVO Library* and *RVO2 Library*. The libraries can efficiently simulate groups of twenty-five to one thousand virtual agents in dense conditions and around moving and static obstacles. *RVO2 Library* is on average at least twice as fast as *HRVO Library*, but *HRVO Library* results in fewer collisions between virtual agents of *RVO2 Library* and therefore results in better local interactions between the virtual agents.

6 Acknowledgments

This work was supported in part by Army Research Laboratory Contract W911NF-04-1-0088, by National Science Foundation Awards 0917040, 0904990, 100057, and 1117127, and by Intel Corporation.

References

de Berg, M.; Cheong, O.; van Kreveld, M.; and Overmars, M. 2008. *Computational Geometry: Algorithms and Applications*. Berlin, Heidelberg, Germany: Springer, third edition.

Curtis, S., Snape, J. and Manocha, D. 2012. Way Portals: Efficient Multi-Agent Navigation with Line-Segment Goals. *Proc. of Symposium on Interactive 3D Graphics and Games*.

Fiorini, P., and Shiller, Z. 1998. Motion planning in dynamic environments using velocity obstacles. *Int. J. Robot. Res.* 17(7):760–772.

Geraerts, R., Kamphuis, A., Karamouzas, I., and Overmars, M. 2008. Using the corridor map method for path planning for a large number of characters. In *Motion in Games*. Springer, Heidelberg, 11–22.

Guy, S. J.; Chhugani, J.; Kim, C.; Satish, N.; Lin, M.; Manocha, D.; and Dubey, P. 2009. ClearPath: highly parallel collision avoidance for multi-agent simulation. In *Proc. ACM SIGGRAPH Eurographics Symp. Comput. Animat.*, 177–187.

Guy, S.; Chhugani, J.; Curtis, S.; Dubey, P.; Lin, M.; and Manocha, D. 2010. PLEdestrians: A least-effort approach to crowd simulation. In *Proc. ACM SIGGRAPH Eurographics Symp. Comput. Animat.*, 119–128.

Kallmann, M. 2010. Shortest paths with arbitrary clearance from navigation meshes. In *Proc. ACM SIGGRAPH Eurographics Symp. Comput. Animat.*, 159–168.

Latombe, J.-C. 1991. *Robot Motion Planning*. Springer, Heidelberg.

Pettré, J., Laumond, J.-P., and Thalmann, D. 2005. A navigation graph for real-time crowd animation on multilayered and uneven terrain. In *Proc. Int. Workshop Crowd Simul.*

Reynolds, C. 1987. Flocks, herds and schools: a distributed behavioral model. *ACM SIGGRAPH Comput. Graph.* 21(4):25–34.

Snook, G. 2000. Simplified 3D movement and pathfinding using navigation meshes. In *Game Programming Gems*. Charles River, Hingham, Mass., ch. 3, 288–304.

Snape, J.; van den Berg, J.; Guy, S. J.; and Manocha, D. 2011. The hybrid reciprocal velocity obstacle. *IEEE Trans. Robot.* 27(4):696–706.

J. van den Berg, S. Patil, J. Seawall, D. Manocha, and M. C. Lin. Interactive navigation of individual agents in crowded environments. *Proc. of ACM Symposium on Interactive 3D Graphics and Games*, pages 139–147, 2008.

van den Berg, J.; Guy, S, J.; Lin, M.; and Manocha, D. 2011. Reciprocal n-body collision avoidance. In Pradalier, C.; Siegwart, R.; and Hirzinger, G., eds., *Robotics Research: the 14th International Symposium ISRR*, number 70 in Springer Tracts in Advanced Robotics. Berlin, Heidelberg, Germany: Springer. 3–19.

van den Berg, J.; Lin, M.; and Manocha, D. 2008. Reciprocal velocity obstacles for real-time multi-agent navigation. In *Proc. IEEE Int. Conf. Robot. Autom.*, 1928–1935.

Van Toll, W., Cook, IV, A., and Geraerts, R. 2011. Navigation meshes for realistic multi-layered environments. In *Proc. IEEE RSJ Int. Conf. Intell. Robot. Syst.*, 3526–3532.

Way to go - A framework for multi-level planning in games

Norman Jaklin, Wouter van Toll and **Roland Geraerts**

Utrecht University, Department of Information and Computing Sciences

Abstract

Path planning is one of the classical computational tasks in video games. Virtual characters need to autonomously find a path from their current position to a designated goal position. This is usually solved by running the A* algorithm on a grid or a navigation mesh. However, in many modern games, strictly following the resulting path is not sufficient. More levels of planning are necessary to efficiently simulate realistic and advanced behavior, and the underlying data structure should support those levels. In this paper, we discuss a five-level hierarchy of planning in games. Furthermore, we present a framework that provides solutions for the three center levels: global route planning, route following, and local planning. It uses an efficient and flexible navigation mesh based on the exact geometry of the environment. Our framework can be extended to solve advanced path planning problems in future games. When used as an interface for higher-level semantic planning systems, it provides a comprehensive set of techniques for game developers and path planning researchers.

Introduction - the different levels of planning

Similarly to domains such as graphics, animation, or physics simulation, the field of path planning in video games has increased in complexity over the last decades. This aspect of game AI has been studied extensively (Rabin 2002). It might seem that path planning problems have been solved by algorithms such as A* (Hart, Nilsson, and Raphael 1968). However, in modern games, path planning is still limited with respect to an ever-increasing demand for new features that enhance player immersion. In addition, even when solving allegedly simple tasks such as letting a character reach a goal position, some games still suffer from flaws due to approximated graph-based representations of the navigable space.

A* is a valuable method to find global shortest paths in any graph structure. In simple cases, one could use a rectilinear grid, which does not lead to any information loss if the game world consists of rectilinear tiles. A* on a basic graph also works well if the game does not require any advanced features such as visually convincing and smooth trajectories, clearance from obstacles, collision avoidance between characters, path planning in environments with multiple height levels, reacting to dynamic changes in the environment, dealing with characters of various sizes, or taking other environmental factors into account, e.g. different terrain types or crowd density information.

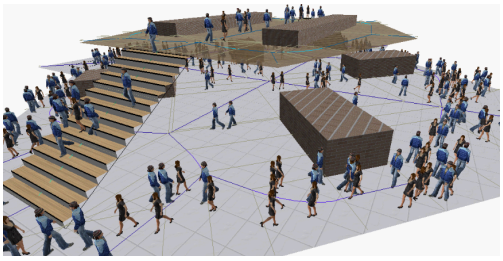


Figure 1: Autonomous virtual characters in a multi-layered 3D environment using the *Explicit Corridor Map* (ECM) (van Toll, Cook IV, and Geraerts 2011)

Modern games, however, do require such advanced features. Viewed from this perspective, computing a global shortest path from a start to a goal position is only one aspect in a multi-level hierarchy of planning systems. We propose five levels of planning that a modern video game might require. Figure 2 shows this five-level hierarchy.

At the top of the hierarchy, *high-level planning* (5) translates the desired *semantic* behavior of a character to *geometric* path planning problems. For example, the character could have an abstract task such as 'steal a stash of gold'. This can be converted to a list of more concrete tasks, e.g. 'enter the village, find character X, plunder its chest full of gold and leave the village without being seen'. Based on this plan, the character should compute an ordered list of goal positions. High-level planning is a research topic of its own, involving techniques such as *STRIPS* (Fikes and Nilsson 1971) and *Hierarchical Task Networks* (Kelly, Botea, and Koenig 2008).

Next, the *global route planning level* (4) uses the list of goal positions to compute geometric routes through the environment. This is where a classical method such as A* might be used, provided that the underlying graph structure does not yield any drawbacks with respect to the tasks at hand.

The three lower levels update the character in every step of the simulation. On the *route following level* (3), the global routes are being traversed. Depending on the application, this can be either a strict and simple following routine, or an advanced method that creates visually convincing tra-

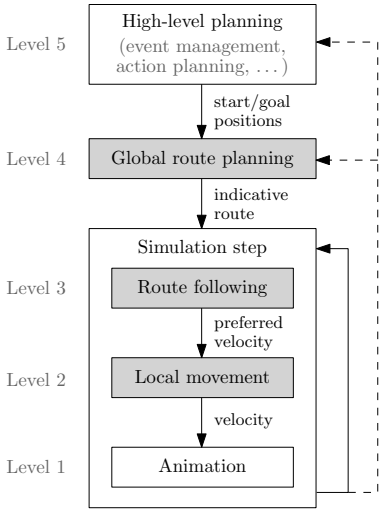


Figure 2: The five-level hierarchy of planning in games.

jectories while take other parameters such as terrain types into account. On the *local movement level* (2), the character might temporarily deviate from its global route to resolve local collision avoidance with other characters or to react to dynamic changes in the environment. Finally, the *animation level* (1) handles the actual animation down to the skeleton representation of the character model.

This planning process is not purely serial: events in the lower levels may cause a character to reconsider its global plans. For instance, if a part of the environment turns out to be too crowded, a character may choose to take a detour.

In this paper, we present an efficient and flexible framework for levels 4, 3, and 2, i.e. the levels that concern geometric path planning. We deliberately treat the high-level planning phase as a black box, and we argue that our framework can be plugged into any game AI system that follows the suggested hierarchy.

The Explicit Corridor Map framework

The core of our framework is a navigation mesh called the *Explicit Corridor Map* (ECM) (Geraerts 2010). We assume that the environment consists of polygonal obstacles. The ECM is based on the *medial axis*, which is the set of all points that have at least two distinct closest obstacle points in the environment. The medial axis is closely related to the Voronoi diagram, which is a fundamental data structure in the field of computational geometry (de Berg et al. 2000).

Figure 3 shows an example. The medial axis can be seen as a special type of waypoint graph in which all edges run through the middle of the free (or traversable) space. For each vertex of this graph (shown as big black discs), there are either at least three different nearest obstacle points, or the vertex is placed in a non-convex corner of an obstacle. An edge of the medial axis consists of a sequence of line segments and parabolic arcs, depending on the type of obstacles to the left and right. For a 2D environment with n obstacle vertices, the medial axis has $O(n)$ complexity and

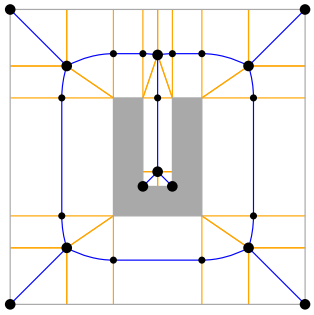


Figure 3: A 2D environment with obstacles (shown in gray). Its ECM is the medial axis (blue) annotated with closest-obstacle information (orange) at a selection of points. This subdivides the traversable space into polygonal regions.

can be constructed in $O(n \log n)$ time. Alternatively, one can use graphics hardware to robustly approximate the structure (Hoff III et al. 1999).

The ECM is an *annotated medial axis*: it stores the left and right closest obstacle points for each edge section. This partitions the environment into a set of polygonal walkable regions. Recently, we have extended the medial axis and the ECM to multi-layered 3D environments (van Toll, Cook IV, and Geraerts 2011). An example of a crowd in a multi-layered ECM is shown in Figure 1.

The ECM has many features that make it well-suited for our framework. All space is represented with respect to the exact geometry of the environment. This resolves the issues that are inherent to approximated representations such as grids or waypoint graphs. Furthermore, the ECM is space-efficient and supports time-efficient extraction of global paths with any desired amount of clearance from obstacles. It therefore supports characters with arbitrary sizes. The ECM is well-defined for both 2D and multi-layered 3D environments. In addition, we have shown that the ECM can be efficiently updated in response to insertions and deletions of obstacles (van Toll, Cook IV, and Geraerts 2012a). Finally, the concept of the ECM is general enough to allow for many extensions and advanced planning methods that build upon it, as will be illustrated in the next section.

Contributions to the planning hierarchy

Our framework comprises methods and techniques that provide efficient real-time solutions for the second, third and fourth levels of the hierarchy. Hence, it can be applied to *geometric* planning problems induced by a *semantic* high-level planner. We will now discuss the contributions in detail.

Global route planning

A global route planner should compute an *indicative route* from the character's start s to its goal position g. Formally, an indicative route can be any curve $\pi_{ind} : [0,1] \rightarrow \mathbb{R}^2$ through the free space of the environment. In practice, we implement such a route as a sequence of points connected by straight-line segments that do not intersect any static obstacles. The concept of using an indicative route for path

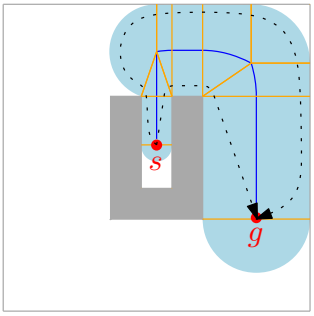


Figure 4: In the ECM, a path along the medial axis (blue) induces a corridor (light blue) due to its closest-obstacle annotations. Within the corridor, we can define any indicative route from s to g; two examples are shown in black.

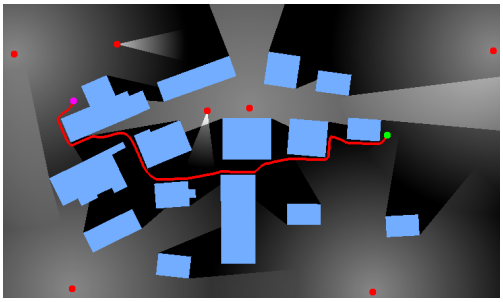


Figure 5: We have used an extended ECM to plan stealthy paths based on visibility information.

planning has first been introduced in the *Indicative Route Method* (Karamouzas, Geraerts, and Overmars 2009).

There are various approaches to compute a global indicative route. For instance, we have implemented A* on the ECM to find shortest paths along the medial axis. This is generally more efficient than performing A* on a grid due to the sparseness of the ECM structure. Furthermore, with the clearance information stored in the ECM, characters of all sizes can use the same graph without having to inflate the obstacles in a preprocessing step.

The optimal route through the ECM does not have to be the *shortest*; optimality can also be based on other criteria. For instance, we have shown how to map *crowd density* information onto the regions induced by the ECM (van Toll, Cook IV, and Geraerts 2012b). By using density information in the A* algorithm, characters can prefer paths with little expected delay. In practice, they will plan detours around congested areas, and the crowd will automatically spread over multiple routes of different homotopy classes.

Performing A* on the ECM always yields a *corridor*, which is a sequence of medial axis edges plus a description of the surrounding free space. A corridor represents a subset of the free space in which valid indicative routes that belong to the same homotopy class are contained. We can therefore create various indicative routes from s to g, e.g. a route that stays on the left or right side of the corridor, or the shortest route in the corridor with a preferred amount of clearance to obstacles (Geraerts 2010). Figure 4 illustrates this concept.

We have also created a method to find *stealthy global paths* with limited exposure to other characters, i.e. a path that lets the character stay unseen by other moving characters as much as possible. To this end, we computed *visibility information* on the GPU and mapped it onto an extended version of the ECM (Schager and Geraerts 2010).

Only recently, we added various *terrain types* to our virtual environments. Those can be used to ensure that a character plans its global route based on a set of individual terrain preferences. For example, a pedestrian might prefer to walk on the sidewalk while avoiding roads, puddles or muddy terrain. We refer the reader to Figure 6 for an example.

Lastly, we have developed a planning approach based on linear programming (Karamouzas, Geraerts, and van der Stappen 2012). It coordinates an entire crowd consisting of one or more independent groups of characters. The method efficiently computes the most promising paths in both time and space and yields an optimal distribution of the groups members over these paths. Thus, the characters' average traveling time is minimized. The computed space-time plan is then combined with an agent-based steering method to handle collisions and generate the final motions of the characters. The method runs at interactive rates and is able to solve complex planning problems involving one or multiple groups in gaming or crowd simulation applications.

The result of the global planning level serves as input to the *route following* level, which we will discuss next.

Route following

In this level, an indicative route π_{ind} is given. The character is supposed to follow the route, but it is allowed to deviate from it. Our framework is built to switch between different path following methods. Our methods use the concept of an *attraction point* p_{att} that lies on π_{ind} to generate smooth paths. In each step of the simulation, the character picks a new p_{att}, which directly leads to a *preferred velocity* v_{pref} for the character in the current step. The direction of v_{pref} is the vector from the character's current position to p_{att}; its magnitude is the character's preferred speed.

We implemented two different path following methods that use attraction points. Firstly, the *Indicative Route Method* (Karamouzas, Geraerts, and Overmars 2009) uses the clearance information provided by the ECM. It defines p_{att} as the last point along π_{ind} that intersects the character's clearance disk. When more free space is available, p_{att} lies farther along the route and larger parts of π_{ind} are skipped, i.e. the amount of smoothing increases.

Secondly, we have introduced a more general path following method named *MIRAN* (Jaklin, Cook IV, and Geraerts 2013 to appear), in which the user can control the character's look-ahead distance and its eagerness to take shortcuts. Furthermore, *MIRAN* lets characters plan their paths with respect to their individual terrain preferences. In other words, the amount of smoothing and route shortening depends on the local terrain costs for that particular character. Figure 6 shows an example of this method.

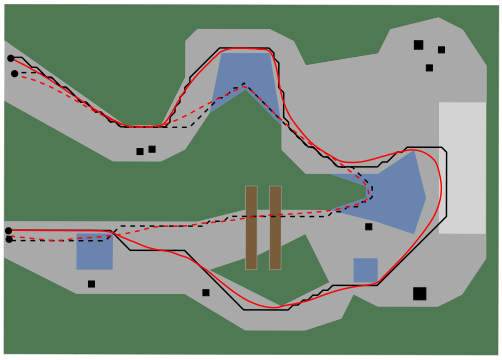


Figure 6: A path (gray) in a forest (green) with obstacle trees (black), puddles (blue), fallen trees (brown) and a spot with a panoramic view (light gray). Two characters (adult and child) follow automatically computed indicative routes (solid and dashed black). The smoothed paths (solid red for the adult, dashed red for the child) are computed with our MIRAN method. Both the indicative routes and the paths are based on the characters' terrain preferences.

The *local movement* level is the last remaining one before the actual *animation* on the skeleton level is handled. We will now discuss in what way our framework covers it.

Local movement

In a virtual crowd, the characters may need to adjust their velocities to avoid collisions with other characters. The task of the *local movement* level in our framework is to compute an actual velocity v for each character, based on its preferred velocity v_{pref} and other crowd members in its vicinity.

Many solutions for this *collision avoidance* problem are available. Early algorithms defined repulsive forces between characters (Helbing and Molnár 1995). Modern methods prevent future collisions based on the perceived velocities of other characters, while deviating from v_{pref} as little as possible. Our framework includes one such approach (Karamouzas and Overmars 2010), but it can support any other velocity-based algorithm, such as the popular RVO library (van den Berg, Lin, and Manocha 2008). Note that collision detection for *static* obstacles is trivial in our framework, because the ECM explicitly stores the nearest obstacle for any point in the free space.

Conclusion and future work

We have given an overview of our framework that covers the three center levels of a five-level planning hierarchy. Those levels comprise the *geometric* aspects of planning in games and can be combined with higher-level planning systems.

Our framework is general enough to support various future extensions and improvements. One possible extension is to take characters of different heights into account. For instance, big vehicles may not fit through small tunnels that regular characters can use. Another extension could be visibility-based planning, e.g. letting characters re-plan their paths based on the dynamic changes they can perceive visually. Finally, we could extend the *MIRAN* method so that it does not only affect the *global planning* and *path following* levels of the hierarchy, but also the *local movement* level, e.g. by including terrain-based collision avoidance.

As we have shown, our framework is flexible and enables future extensions. We therefore believe that it provides a comprehensive set of techniques for game developers and path planning researchers.

References

de Berg, M.; van Kreveld, M.; Overmars, M.; and Schwarzkopf, O. 2000. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, second edition.

Fikes, R., and Nilsson, N. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2(3/4):189–208.

Geraerts, R. 2010. Planning short paths with clearance using Explicit Corridors. In *IEEE International Conference on Robotics and Automation*, 1997–2004.

Hart, P.; Nilsson, N.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* 4(2):100–107.

Helbing, D., and Molnár, P. 1995. Social force model for pedestrian dynamics. *Physical Review E* 51(5):4282–4286.

Hoff III, K.; Culver, T.; Keyser, J.; Lin, M.; and Manocha, D. 1999. Fast computation of generalized Voronoi diagrams using graphics hardware. In *International Conference on Computer Graphics and Interactive Techniques*, 277–286.

Jaklin, N.; Cook IV, A.; and Geraerts, R. 2013 (to appear). Real-time Path Planning in Heterogeneous Environments. *Computer Animation and Virtual Worlds*.

Karamouzas, I., and Overmars, M. 2010. Simulating human collision avoidance using a velocity-based approach. In *Workshop on Virtual Reality Interactions and Physical Simulations*, 125–134.

Karamouzas, I.; Geraerts, R.; and Overmars, M. 2009. Indicative routes for path planning and crowd simulation. In *International Conference on Foundations of Digital Games*, 113–120.

Karamouzas, I.; Geraerts, R.; and van der Stappen, A. 2012. Space-time group motion planning. In *Workshop on the Algorithmic Foundations of Robotics*, 227–243.

Kelly, J.; Botea, A.; and Koenig, S. 2008. Offline planning with Hierarchical Task Networks in video games. In *Artificial Intelligence and Interactive Digital Entertainment Conference*, 60–65.

Rabin, S. 2002. *AI Game Programming Wisdom*. Charles River Media.

Schager, E., and Geraerts, R. 2010. Stealth-based path planning in corridor maps. In *Computer Animation and Social Agents*.

van den Berg, J.; Lin, M.; and Manocha, D. 2008. Reciprocal Velocity Obstacles for real-time multi-agent navigation. In *IEEE International Conference on Robotics and Automation*, 1928–1935.

van Toll, W.; Cook IV, A.; and Geraerts, R. 2011. Navigation meshes for realistic multi-layered environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 3526–3532.

van Toll, W.; Cook IV, A.; and Geraerts, R. 2012a. A navigation mesh for dynamic environments. *Computer Animation and Virtual Worlds* 23(6):536–546.

van Toll, W.; Cook IV, A.; and Geraerts, R. 2012b. Real-time density-based crowd simulation. *Computer Animation and Virtual Worlds* 23(1):59–69.

The hybrid optimized path finding in MMOG

Sung June Chang

[1]Electronics and Telecommunications Research Institute (ETRI)
dyad@etri.re.kr

Abstract

This paper provides a hybrid optimized path finding method which consists of reactive and proactive path finding. Our system generates an optimal guide using proactive path planning models like A* and D*, and then executes a reactive path finding which is similar to Boids model and potential field model using the proactive path plan. The proactive path finding algorithm is optimal but time-consuming, reactive method is effective but not optimal. So we integrate them and get the near optimal and effective path finding algorithm.

Introduction

The path finding method in game field is traditionally A* using heuristic, which needs heavy recalculation time in dynamic environment. The D* and similar methods [Anthony95, Coenig04] use incremental search methods to decrease recalculation time. But they still need more time compared with reactive path planning methods.

In robot or Artificial Life field, Boids methods [Reynolds87, Reynolds99] or potential field methods [Kim91, Conner03] is very effective because it needs force generation in a narrow range. It is suitable for real time or massive application. But it often causes local minimum problem under complex circumstances.

Our approach uses the hybrid method to solve two kinds of problems. The algorithm generates an optimal path by proactive path planning then follows the path while coping with the dynamic environment using the reactive path planning.

To integrate two types of path planning, we use the sampling method which was used in the constraint animation [Jiayi04], where they generate forces by sampling points and integrate various kinds of forces.

System Overview

Our system consists of three parts.

The first part carries out a path planning by the proactive method. As a proactive method, we select A* algorithm where heuristic function is the straight-line distance. After path planning, we gather sampling points from the path and generate the attractive forces from them.

The second part uses a reactive path planning. Reactive forces are generated from dynamic environments where we use monsters. We use the repulsive potential field method to get the reactive forces from monsters.

The third is integration part. We generate attractive forces from a selected sample of the sampling list and integrate them with repulsive forces. Then the movement is generated by the summed forces.

Block-diagram style system overview is as follows.

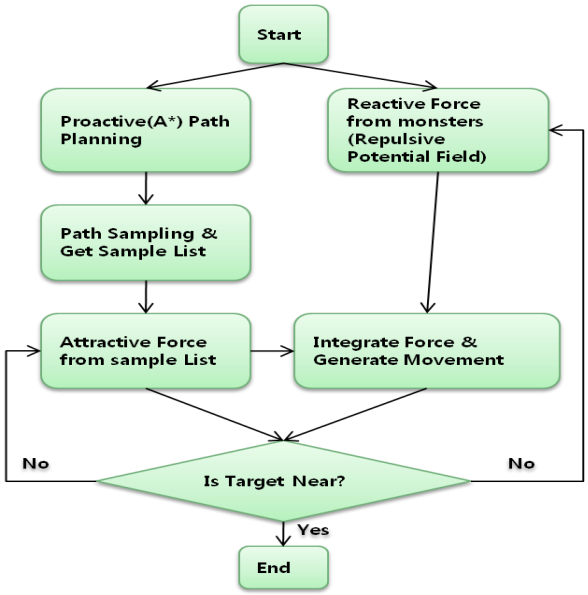


Fig 1: System Overview

Hybrid Method

We use classical A* algorithm as a proactive path planning. Firstly, it needs to span a map by grids to apply the algorithm. The cost function in A* is $f(x) = g(x) + h(x)$ as usual. We use the straight line distance as heuristic value $h(x)$ which is also common. We also choose the position which is near to the final target when breaking tie. The near grid is selected in breaking tie because it tends to shorten the time to arrive at the final target.

We sample points in every grid center of the above A* path, then make a sample list which is sequentially generated along the path. The force is generated from the distance

between the sample position in list and the current position. The selected sample is sequentially changed when being reached. The formulation to get the forces from the path is like the following.

$$F_{path} = NORM(P_{sample}-P_{current})*MAX(F_i)/2$$

In the formulation, F_{path} is the force vector by the selected path sample point in list, NORM is normalization, P is the sample point's position vector, $P_{current}$ is the current position vector, MAX is the max value of F_i, and F_i is the dynamic factor's force which will be explained in the next.

The direction of the path force comes from the difference between sample point's position and current position. The magnitude of the force is the half of maximum value of a dynamic force which means the reactive force. The magnitude graph is shown in the below

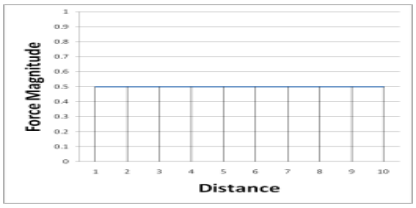


Fig. 2 Force Magnitude in F_{path}

As seen in Figure 2, the F_{path} direction depends on current position and target position but its magnitude is constant. So the agent (or robot) is attracted by the path's sample point with the same magnitude of force at any distance, even though the direction is different.

Reactive path planning method in this algorithm is similar to potential field and Boids method. It generates repulsive force from dynamic environment for which we use monsters. The formulation is in the below.

$$F_i = NORM(P_i-P_{current})*E_i$$

In the formulation, F_i is the force vector by the ith dynamic factor like monsters, NORM is normalization, P_i is position of the ith dynamic factor, $P_{current}$ is the current position vector, and E_i is the magnitude of force by the ith dynamic factor

The direction of the force comes from the difference between the ith dynamic factor's position vector and current position vector. The magnitude of the force which is E_i in the formulation is defined linearly. The details are in the below Figure 3.

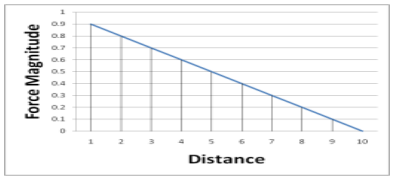


Fig. 3 Force Magnitude in F_i

The next formulation is the summation of reactive forces. $F_{dynamic}$ is generated by the summation of all F_i vectors.

$$F_{dynamic} = SUMMATION(F_i)$$

In the formulation, Fdynamic is the total reactive forces, and SUMMATION is the addition of all elements.

F_{path} and $F_{dynamic}$ are integrated into F_{hybrid} by simple addition because F_{path}'s magnitude is originally designed to reference $F_{dynamic}$, which is shown in the below.

$$F_{hybrid} = F_{path}+F_{dynamic}$$

In the formulation, F_{hybrid} is the final result of force calculation and will be used to generate the movement.

Simulation and Result

Simulation scenario is shown in the below Figure 4. The character 'A' in the Figure points monsters which are dynamic factors in the simulation. The character 'B' points obstacles which are used in proactive path planning. The character 'C' points the way-points including final target. The character 'D' points the path which is generated by the A* algorithm.

In the simulation, monsters are designed to move randomly. And two types of obstacles are designed. We select the first type of obstacles as a line style because it is generally used. The second type of obstacles is the concave style which often causes local minima in reactive path planning algorithm. For the way-points which are colored red in figure 4, we set two way-points. The left one is bypass way-point and the right one is the final target position.

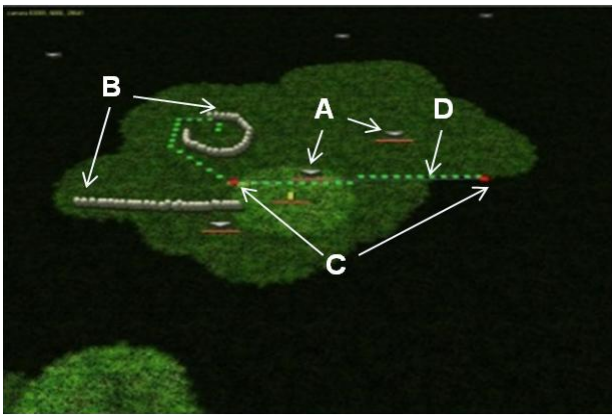


Fig. 4 Simulation Overview

In Figure 4, the generated A* path is shown in green dots. The A* path finding algorithm finds two kinds of path. The first is from the start position to the first way-point. It avoids concave type of obstacles. The second is from the first way-

point to the second way-point which is final target. The A* algorithm connects two kinds of A* path and is shown in green dots.

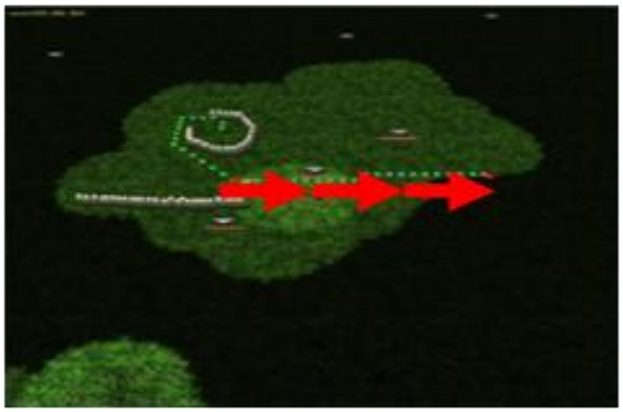


Fig. 5 F_{path} in simulation

The above figure's red arrow shows F_{path} which is the force attracting the agent into the proactive path. In our algorithm we use A*. Green dots are converted into sampling points in list, which will sequentially attract the agent. The selected sampling point is changed in the list when being reached until the final point. Because the magnitude of the force is constant, the force can make the agent consistently follow the A* path.

The repulsive force $F_{dynamic}$ is generated around dynamic factors which are monsters in the simulation. The force is shown in the Figure 6. The yellow arrow in Figure 6 shows $F_{dyanmic}$ which is generated as a repulsive force around monsters. In this simulation, only the nearest monster's force is displayed because other monsters are far from the agent. Although the threshold of effective distance to monsters is flexible, we choose very small threshold value. If the threshold is big value, many monsters affect the agent at the same time, which makes agent's chaotic behavior.

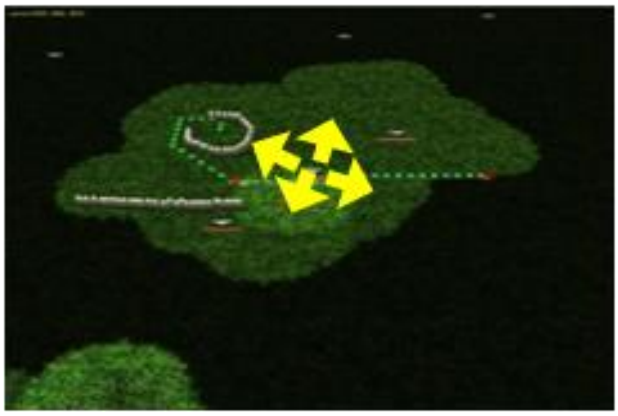


Fig. 6 $F_{dynamic}$ in simulation

In simulation, two types of force are integrated into the hybrid force like Figure 7's orange arrow which is same with the agent's movement. The agent keeps the green dots until a monster approach. As the monster goes near, it goes away while following the green dots. After the monster is away, it goes back to the green dots again, which means that the agent is following the proactively generated path while avoiding dynamic monsters in real time.

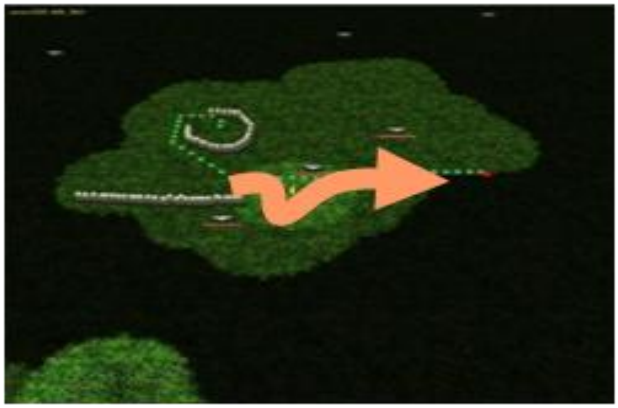


Fig. 7 F_{hybrid} in simulation

In the above simulation, the average calculation time of the path planning algorithm including proactive path planning algorithm and reactive path planning is about 20 milliseconds. The result is obtained from 1,000 times simulation using Xeon 3.20Hz as CPU.

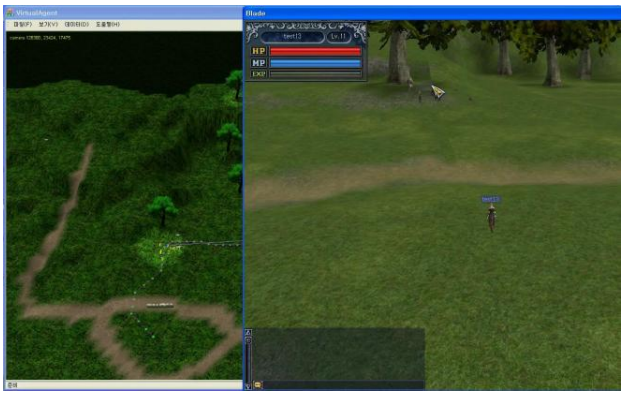


Fig. 8 Simulation in MMOG

In the algorithm, we use only one proactive path planning but its calculation time overwhelms the calculation time of reactive path planning. It needs almost 20 milliseconds to find a path by the A* algorithm. The agents controlled by our algorithm can follow the optimal path very effectively while avoiding monsters in real time.

The time complexity of the algorithm is same with that of the proactive path planning algorithm because time complexity of the reactive path planning algorithm is negligible. In this simulation, the complexity of our algorithm is O(logh*(x)) which is same with A* algorithm.

Conclusion

In this paper, we show the hybrid method which integrates proactive path planning and reactive path planning. Using the method, we can develop the algorithm which follows optimal path while reacting to the dynamic environment in real time. We also show that it is useful in the simulation. We make the proactive path planning by A* algorithm and develop the reactive path planning by the repulsive force from the monsters. We set the attractive force generated by A* algorithm with the same magnitude and the repulsive force by the reactive path planning. Finally we integrate two kinds of force and generate the hybrid force which makes the agent follow A* algorithm and react to the monster at the same time.

The benefit of our method is that it is effective. In simulation, our algorithm just takes about 20 milliseconds by 1,000 times testing. It is near-optimal at the same time because A* algorithm is used to make paths to avoid difficult obstacles like concave style.

Acknowledgments

This work was supported by Ministry of Culture, Sports and Tourism(MCST) and Korea Creative Content Agency(KOCCA) in the Culture Technology(CT) and Research Development Program 2013.

References

A new potential field method for robot path planning

Yunfeng W., Gregory S. Chirikjian (2000). Proceeding of the 2000 IEEE International Conference on Robotics & Automation.

Autonomous behavior for interactive vehicle animations

Jared G. Thuc V. and James J. K. (2004). Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation 9-18

Composition of local potential functions for global robot control and navigation

Conner D.C. Rizzi A. and Choset. H. (2003). In Proceedings IEEE AAAI-96. 530-535

Constrained animation of flocks

Matt A. Eric M. and Stephen C. (2003). Eurographics/SIGGRAPH Symposium on Computer Animation

Continuum crowds

Treuille A. Cooper S. and Popvic Z. (2006). ACM Transactions on Graphics 2006. 25: 1160-1168

Flocks, herds, and schools: A distributed behavioral mehod

Reynolds C. (1987). Computer Graphics 1987 4, 25-34.

Forward chaining for potential field based navigation

Graeme B. (2005). A thesis to be submitted to the University of ST Andrews for the degree of Doctor of Philosophy

Hierarchical A*: Searching abstraction hierarchies efficiently.

Holte R. Perez M. Zimmer R. and Macdonald A. Koenig S. Likhachev M. and Furcy D.(1996). In Proceedings AAAI-96. 530-535

Lifelong planning A*

Koenig S. Likhachev M. and Furcy D.(2004). Journal of Algorithms. 21:267-305

Near optimal hierarchical path-finding

Adi B. Martin M. and Jonathan S. (2004). Journal of Game Development. 1:7-28

Optimal and efficient path planning for partially-known environments

Anthony S. (1994). Proceedings of the International Conference on Robotics And Automation. 3310-3317

Path finding and collision avoidance in crowd simulation

Cherif F. Djedi N. Cedric S. and Yves D. (2009). Journal of Computing and Information Technology. 3:217-228

Pedestrian reactive navigation for crowd simulation: a predictive approach

Paris S. Pettre J. and Donikian S. (2007). Computer Grapchis Forum 26: 665-675

Real-time obstacle avoidance using harmonic potential functions

Kim J. and Khosla. P (1991). IEEE International Conference on Robotics and Automation. 1: 790-796

Shape-constrained flock animation

Jiayi X. Xiaogang J. Yuzhou Y. Tian T. and Mingdong Z. (2008). Computer Animation and Virtual Worlds. 19: 319-330

Steering behaviors for autonomous characters

Reynolds C. (1999). In Proceedings of Game Developers Conference, 763-782

The focussed D* algorithm for real-time replanning

Anthony S. (1995). Proceedings of the International Joint Conference on Artificial Intelligence. 1652-1659

On Compilations For Narrative Planning

Patrik Haslum
Australian National University
patrik.haslum@anu.edu.au

Introduction

The classical AI planning model, which assumes complete knowledge of and control over a deterministic world, is often considered too limited, as many planning problems appear to have requirements that do not fit in this model. Recently, however, it has been shown that some problems thought to go beyond the classical model can nevertheless be solved by classical planners through *compilation*, i.e., a systematic remodelling of the problem such that a classical plan for the reformulated problem meets also the non-classical requirements. A striking example is the work of Palacios and Geffner (2006; 2009), who showed that conformant planning (generating non-branching plans that are robust to uncertainty) can be compiled into classical planning.

I will argue for the potential of compilations into classical planning to play a role in narrative generation. Riedl and Young (2010) observe that "there are many parallels between plans and narrative at the level of fabula." Both are sequences of events that change the state of the (story) world. These parallels have inspired approaches to story generation based on planning or planning-like techniques (e.g. Meehan 1977; Riedl 2004; Riedl and Young 2010), but most have made little use of the capabilities of existing classical planners. Classical planning techniques have been used to drive the behaviour of individual characters in simulation-based approaches to story generation (e.g. Aylett, Dias, and Paiva 2006; Pizzi et al. 2007; Brenner 2010).

Intentional Planning and the Justification-Tracking Compilation

Despite their similarities, narratives are not just plans. A sequence of events must satisfy many criteria other than logical possibility and coherence before we can call it a story.

Riedl and Young (2010) isolate on one such additional criterion that separates stories from plans, viz. character intentionality. Some events in a story are actions performed by story characters. (Other events, such as accidents, coincidences, etc, have no associated actor; Riedl and Young call these *happenings*.) For characters to be perceived as believable, the actions they take must be seen to contribute to the characters' goals, which are not necessarily the same as the goal of the planning process (the latter is referred to as the *story outcome*). Character goals are included in the model by means of modal literals of the form (intends A f), where A is a character and f is a fact, i.e., a normal literal. Intentions can exist in the initial state, or arise as an effect of actions. Informally, an intentional plan is one in which each character action contributes, directly or indirectly, to achieving an intention that the character has.

Formally, intentionality is defined, in the context of partially ordered causal link (POCL) plans, through the concept of a *frame of committment*. This is a subset S' of plan steps, satisfying four requirements:
(1) Character A is an actor of every step in S'.
(2) There is a *final step* $s_{\text{fin}} \in S'$ that makes g true.
(3) There is a *motivating step* s_{m} which adds (intends A g) and which precedes all steps in S'. We say there is a *motivational link* from s_{m} to every step in S'. Note that s_{m} is not part of S'; it may be the intitial state.
(4) From each step in S' other than s_{fin} there is a path of causal or motivational links to s_{fin}.
The set of domain objects that are characters and the assignment of the role of actor(s) to parameters of actions are part of the domain theory.

Riedl and Young develop a specialised POCL planner, called IPOCL, that generates intentional plans. However, the intentionality requirement can also (almost) be achieved by a compilation into classical planning. The compilation has been presented in full detail elsewhere (Haslum 2012). Here, I will only make a brief sketch.

The compilation makes use of modal literals of the form (justified f I), where f is a fact and I an intends atom.[1] Suppose character A has (only) goal g, i.e., (intends A g) is true, and consider applying an action whose actor is A: If the action directly achieves g, it does not need further justification. If not, applying a creates an outstanding obligation that some later action taken by the character must make use of at least one effect of the action, to eventually achieve the intended goal. This is modelled by the justified modality. All justified atoms are true in the initial state, and required to

[1] Modal atoms cannot be expressed directly in a classical planning formalism like PDDL. In a PDDL model, they are replaced by a separate "modal predicate" for each predicate (resp. combination of two predicates) that can appear in a non-modal fact, whose arguments is the concatenation of all arguments in the modal literal. That is, (intends A (P $\vec{x}$)) is replaced by (intends-P A $\vec{x}$), and (justified (P $\vec{x}$) (intends A (Q $\vec{y}$))) by (justified-P-Q $\vec{x}$ A $\vec{y}$).

hold in the goal state; actions create an obligation by deleting the atom (justified e (intends A g)), where e is an effect of the action, and fulfil these obligations by adding (justified f (intends A g)) for all facts f in the actions' precondition. In the compiled problem, each (non-happening) action is associated with an intention for each of its actors, which marks the frame of committment that the action will be part of. Actions are further split into cases, based on whether the intention unifies with an effect of the action, and, if not, which of its effects becomes unjustified.

Characters can also "delegate" subgoals to other characters, through actions like (command A B g'). These actions create new intentions, (intends B g'), and are justified by the actor (character A) eventually making use of the fact g' achieved by character B. Some additional machinery is required for the compilation to handle delegations.

Beyond Intentionality

Character intentionality is just one of many aspects that must be incorporated into the planning process to generate plausible narratives. For example, the model formulated by Riedl and Young (2010) makes no distinction between the state of the story world and characters' state of knowledge about it, and does not allow stories in which a character tries but fails to achieve a goal (since the failed action does not contribute to the eventual fulfilment of a character intention, and thus cannot be part of a frame of committment).

The state of a compiled planning problem is not limited to representing the state of the (story) world. It can encode data structures, such as graphs, recording characters' plans or mental states. Likewise, actions are not limited those that actually take place in the (story) world, but can also represent (characters') "mental" actions, such as making inferences or plans. This can be seen in, for example, the work of Palacios and Geffner (2006; 2009) who use modal literals of the form (K f φ), meaning, roughly, "f is known to be true if φ was true in the initial state", to represent knowledge about uncertain facts, and actions that represent explicitly reasoning (by cases) over such statements.

A similar approach can be taken to representing story characters' knowledge and plans. Here, I will only sketch a possible compilation to illustrate the potential of the idea. There are many tricky issues in the details to be worked out.[2] For illustration, I will use the following story, adapted from one generated by Brenner's (2010) system:

> *Desiring the treasure, and believing it to lie unguarded in a cave, the king rode to the cave to steal it. Upon arrival, he saw it was guarded by a dragon. Knowing he could not defeat the dragon, he returned to the castle, and ordered his knight: Go get me the treasure! The knight, not knowing where the treasure was, asked the king: Where might I find this treasure? The king told him: It's in the cave. The knight rode to the cave, and saw the dragon. The dragon had also seen the knight and attacked him, believing it could defeat him. But, alas, the knight proved stronger, and the dragon perished. The knight brought the treasure back to the castle and presented it to the king, who was very pleased.*

[2] A more limited compilation based on the idea of meta-planning is described in the earlier paper (Haslum 2012).

Character Meta-Planning

Suppose character A intends a goal g: A *meta-planning* action, (plan-to-a A ...), allows the character to adopt the intention of achieving the preconditions of some action a that has g among its effects. That is, the meta-planning action does not affect any change to the story world, but models the character's mental process of planning. The committment is recorded by a modal fact (supports A f g), i.e., that the character's intention to achieve f is motivated only as a step towards achieving g. The character can recursively plan how to achieve f, until arriving at a set of intentions that can be achieved by taking "real" actions in the story world. To ensure that characters' actions are justified, their real actions are preconditioned on the character having an (unsupported) subgoal that is achieved by the action. Characters' beliefs can be represented with another modality, (believes A f). We have a lot of freedom in formulating meta-planning actions, for example to precondition them on the character believing facts that should hold for the planned action to make sense. In this, they somewhat resemble HTN planning methods.

The record of the character's plan, by means of supports facts, is useful for several reasons: meta-planning actions are preconditioned on the planned-for subgoal not being supported (to avoid characters making plans that they don't act on), and to avoid characters making cyclic plans. Using PDDL's derived predicates (Thiebaux, Hoffmann, and Nebel 2003), we can define complex conditions, such as

```
(:derived (subgoal ?who g)
  (or (intends ?who g) (exits (h) (supports ?who g h))))

(:derived (unsupported ?who g)
  (and (subgoal ?who g)
       (not (exists (f) (supports ?who f g)))))
```

expressing that g is a subgoal of character ?who, and that it is an open subgoal, respectively.

In the example story, initial facts include (intends King (has King Treasure)), (believes King (at Treasure Cave)) and (believes King (unguarded Cave)). We may apply the meta-planning action

```
(:action plan-to-steal-1
  :parameters (?who ?what ?where)
  :precondition (and (unsupported ?who (has ?who ?what))
                     (believes ?who (at ?what ?where))
                     (believes ?who (unguarded ?where)))
  :effect (and (supports ?who (at ?who ?where)
                              (has ?who ?what))
               (supports ?who (at ?what ?where)
                              (has ?who ?what))
               (supports ?who (unguarded ?where)
                              (has ?who ?what))))
```

to create the new character subgoal (unsupported King (at King Cave)). This justifies the king travelling to the cave, an action that is immediately applicable.

With only the condition that actions achieve some current character subgoal, characters can act hastily, before they have a complete plan (which may not even exist). To make

them more cautious, we can add to the precondition of real actions that any unsupported character subgoals are believed to be already true:

```
(:derived (complete-plan ?who)
  (forall (g) (imply (unsupported ?who g) (believes ?who g))))
```

Revising Character Plans

As the example story shows, characters' plans can fail, if based on invalid beliefs. To allow characters to replan, we need mechanisms to update characters' beliefs, and to decide when to retract plans. The first part is straightforward: an action like

```
(:action observe-guard
  :parameters (?who ?guard ?where)
  :precondition (and (at ?who ?where)
                     (guards ?guard ?where))
  :effect (and (not (belives ?who (unguarded ?where)))
               (believes ?who (guards ?guard ?where))))
```

allows changing characters beliefs about facts that they can immediately observe. To give some impression of persistence of characters' plans, replanning should only be allowed when a character's beliefs has changed in a way that affects their plan. The simplest approach to replanning is to retract all supports, letting the character replan from scratch. The general form of a replanning action is then

```
(:action replan
  :parameters (?who)
  :precondition (and (not (believes ?who f))
                     (unsupported ?who f))
  :effect (forall (p q) (not (supports p q))))
```

Character replanning is not suitable for all situations; sometimes we must allow characters to learn new beliefs through mistakes (and suffer the consequences). For actions that have uncertain outcomes (from the character's point of view), we need to model all possibilities. In the example story, the dragon attacks the knight, mistakenly believing it can win the fight. Here, we need at least two actions:

```
(:action attack-and-win
  :parameters (?who ?victim ?where)
  :precondition (and (subgoal ?who (dead ?victim))
                     (believes ?who (stronger ?who ?victim))
                     (at ?who ?where) (at ?victim ?where)
                     (stronger ?who ?victim))
  :effect (and (dead ?victim) (not (guards ?victim ?where))))

(:action attack-and-lose
  :parameters (?who ?victim ?where)
  :precondition (and (subgoal ?who (dead ?victim))
                     (believes ?who (stronger ?who ?victim))
                     (at ?who ?where) (at ?victim ?where)
                     (stronger ?victim ?who))
  :effect (and (dead ?who) (not (guards ?who ?where))))
```

(The inclusion of (not (guards ?victim ?where)) in the effects is a clumsy way to implement ramification. When a character dies, many facts about that character change, which, ideally, should be encoded in a more modular way.)

Knowledge Goals

A final challenge illustrated by the example story is planning to achieve knowledge goals. When the king commands the knight to get the treasure, the knight adopts the goal (intends Knight (has King Treasure)). The knight may plan to achieve this by giving the treasure to the king, leading to the unsupported goal (has Knight Treasure). But not having any beliefs about the whereabouts of this treasure, the knight cannot make any plan to acquire it. Here, we need a different meta-planning action:

```
(:action plan-to-steal-2
  :parameters (?who ?what ?where)
  :precondition (unsupported ?who (has ?who ?what))
  :effect (intends-to-know ?who (at ?what ?where)))
```

This allows the knight to motivate the action

```
(:action ask-answer-yes-truthfully
  :parameters (?who-Q ?who-A f)
  :precondition (and (intends-to-know ?who-Q f)
                     (trusts ?who-Q ?who-A)
                     (believes ?who-A f))
  :effect (believes ?who-Q f))
```

or the analogous action ask-answer-no-truthfully, if the character asked does not believe f to be true. (This is again somewhat clumsy, since it does not allow a character to ask "who/what/where" questions, for which other machinery may be needed (Petrick and Bacchus 2002). On the other hand, since the planner takes the part of the story's author, it knows the correct question to ask.) In this action, only the character asking the question is an actor. If we want to model an action in which the character who answers lies, it would perhaps be more appropriate to consider both of them to be actors, so that both need a motivation for their behaviour.

Discussion

Engaging and interesting characters have more dimensions than their knowledge and plans: They have emotions and personality, which manifests, among other ways, in their choice of actions (Bahamon and Young 2012). To what extent those aspects can also be captured through compilation (via meta-planning or other techniques) is an open question.

A key consideration in any form of automated narrative generation is originality. If we draw the boundaries of the problem too tightly, the system will only "generate" a story that we have scripted, and thus lack any element of surprise. Therefore, encoding narrative generation as a planning problem we should strive to give the planner maximum freedom to generate alternative stories, while ensuring that every plan meets the criteria we expect of stories (such as character believability). In the meta-planning compilation sketched above, this creates a tension between on the one hand allowing characters to achieve their goals in any way possible and on the other imposing preconditions on meta-planning actions so that they reflect "reasonable" problem-solving strategies (characters don't behave stupidly).

Classical planners often seek the simplest (i.e., shortest) solution to a problem, which is somewhat at odds with the aim of making stories "interesting". In the example story, why did the king first try to get the treasure himself, when he could just have ordered the knight to get it right away? One answer is, that would have made the story (even more) boring. PDDL offers a rich set of mechanisms to influence plan choice, including metrics, preferences and trajectory

constraints, which may be used to encode narrative control knowledge (Porteous, Cavazza, and Charles 2010). Alternatively, giving the planner more freedom and using techniques for generating diverse plans (e.g. Srivastava et al. 2007), we can generate many story variants to be evaluated for their "aesthetic" or "entertainment" value. The work of Porteous et al. (2011) on controlling "narrative tension" by rearranging the sequence of events in a story can be viewed as pursuing this approach (though they assume the arc of tension is provided as input). Work on "reader modelling" (e.g. Bailey 1999) could form the basis for such an evaluation.

Another question is whether handling full-scale narrative planning models is within the reach of current classical planners. If we want planners to generate many diverse stories for a scenario, domain models will almost certainly involve many possible facts and actions. Compiled problems often use advanced features, such as conditional effects and derived predicates, that are not as well supported as the basic STRIPS model, and sometimes have a structure that make them particularly difficult for current planning methods, such as delete relaxation-based heuristics (e.g. Palacios and Geffner 2009; Bonet, Palacios, and Geffner 2009; Brafman, Shani, and Taig 2012). Although the justification-tracking compilation of Reidl's and Young's (2010) example problem is solved by a state-of-the-art classical planner several orders of magnitude faster than the original problem is solved by IPOCL, adding just a few (irrelevant) objects and actions to the model increases runtime on its compilation by a factor of 7 (Haslum 2012). Thus, compilations of narrative planning problems may also provide a challenging benchmark for classical planning.

Conclusion

Classical planners are domain-independent, and sometimes highly effective, problem solvers. Through compilations, their performance can be directed at many more problems than those that on the surface appear to be classical planning problems, including narrative generation. Their efficiency, combined with the flexibility to impose and lift constraints on plans by merely changing the declartive problem model, suggests that classical planners can be a useful tool to explore ideas and methods for automated narrative generation.

Clearly there are many challenges, and of course there are limits on what problems can be compiled into classical planning. But I conjecture that the compilation of narrative planning problems can be pushed much further before those limits are reached, and encourage researchers interested in narrative generation to take part in that exploration.

References

Aylett, R.; Dias, J.; and Paiva, A. 2006. An affectively driven planner for synthetic characters. In *Proc. 16th International Conference on Automated Planning and Scheduling (ICAPS'06)*, 2–10.

Bahamon, J., and Young, R. 2012. A choice-based model of character personality in narrative. In *Proc. 3rd Workshop on Computational Models of Narrative*, 166–170.

Bailey, P. 1999. Searching for storiness: Story generation from a reader's perspective. In *Narrative Intelligence: Papers from the AAAI Fall Symposium*. AAAI Press.

Bonet, B.; Palacios, H.; and Geffner, H. 2009. Automatic derivation of memoryless policies and finite-state controllers using classical planners. In *Proc. 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*, 34–41.

Brafman, R.; Shani, G.; and Taig, R. 2012. Leveraging classical planners through translations. In *Proc. ICAPS'12 workshop on the International Planning Competition*, 6–9.

Brenner, M. 2010. Creating dynamic story plots with continual multiagent planning. In *Proc. 24th AAAI Conference on Artificial Intelligence*, 1517–1522.

Haslum, P. 2012. Narrative planning: Compilations to classical planning. *Journal of AI Research* 44:383–395.

Meehan, J. 1977. TALE-SPIN, an interactive program that writes stories. In *Proc. International Joint Conference on AI (IJCAI'77)*, 91–98.

Palacios, H., and Geffner, H. 2006. Compiling uncertainty away: Solving conformant planning problems using a classical planner (sometimes). In *Proc. 21st National Conference on Artificial Intelligence (AAAI'06)*.

Palacios, H., and Geffner, H. 2009. Compiling uncertainty away in conformant planning problems with bounded width. *Journal of AI Research* 35:623–675.

Petrick, R., and Bacchus, F. 2002. A knowledge-based approach to planning with incomplete information and sensing. In *Proc. 6th International Conference on Artificial Intelligence Planning and Scheduling (AIPS'02)*, 212–221.

Pizzi, D.; Charles, F.; Lugrin, J.; and Cavazza, M. 2007. Interactive storytelling with literary feelings. In *Proc. 2nd International Conference on Affective Computing and Intelligent Interaction*, 630–641.

Porteous, J.; Teutenberg, J.; Pizzi, D.; and Cavazza, M. 2011. Visual programming of plan dynamics using constraints and landmarks. In *Proc. 21st International Conference on Automated Planning and Scheduling (ICAPS'11)*, 186–193.

Porteous, J.; Cavazza, M.; and Charles, F. 2010. Applying planning to interactive storytelling: Narrative control using state constraints. *ACM Transactions on Intelligent Systems and Technology* 1(2).

Riedl, M., and Young, R. 2010. Narrative planning: Balancing plot and character. *Journal of AI Research* 39:217–268.

Riedl, M. 2004. *Narrative Planning: Balancing Plot and Character*. Ph.D. Dissertation, Dept. of Computer Science, North Carolina State University, Raleigh, NC.

Srivastava, B.; Kambhampati, S.; Nguyen, T.; Do, M.; Gerevini, A.; and Serina, I. 2007. Domain independent approaches for finding diverse plans. In *Proc. 20th International Conference on Artificial Intelligence (IJCAI'07)*, 2016–2022.

Thiebaux, S.; Hoffmann, J.; and Nebel, B. 2003. In defense of PDDL axioms. In *Proc. 18th International Conference on Artificial Intelligence (IJCAI'03)*, 961–968.

Planning for Interactive Storytelling Processes

Stefano Cianciulli
Sapienza University of Rome
Rome, Italy
stefano.cianciulli@gmail.com

Stavros Vassos
Sapienza University of Rome,
Rome, Italy
vassos@dis.uniroma1.it

Abstract

In this paper we present some preliminary results experimenting with the AI method of behavior composition for the purpose of facilitating interactive storytelling in video games. The motivation is twofold: first, behavior composition is based on transition systems that are ubiquitous in video game development under the term finite state machines, and second, as the research community explores ways for a non-linear adaptive storyline in video games by means of automated planning and scheduling, the use of behavior composition may be able to offer added benefits by means of performing planning for a target desired process instead of a target desired state. We introduce JACO, a web service for behavior composition, and present a use case based on a simple conceptual example for interactive storytelling.

Introduction

In this paper we experiment with the AI method of behavior composition so as to facilitate interactive storytelling in video games. Behavior composition (De Giacomo, Patrizi, and Sardiña 2013) is concerned with *orchestrating* a set of available behaviors, each of which is expressed as a *transition system*, in order to accommodate a virtual target service also expressed as a transition system. The aim is to synthesize an orchestrator that is able to realize the target service by exploiting execution fragments of available services.

The motivation is twofold. First, transition systems are ubiquitous in video game development. Variants of transition systems, typically referred to as *finite state machines (FSMs)*, is one of the most widely used techniques for specifying the behavior of non-player characters (NPCs) in video games. This familiarity makes behavior composition well suited for orchestrating the behavior of NPCs. Second, as the research community explores ways for a non-linear adaptive and interactive storyline in video games by means of automated planning and scheduling, the use of behavior composition may be able to offer added benefits as transition systems essentially facilitate *planning for a target desired process* instead of a *target desired state*. For example, the target process may describe a recurring transportation activity connecting two areas, which may be realized by different available services, i.e., agents or devices.

In the setting we explore, each of the NPCs of the game may feature any preferred method for specifying and realizing their intended behavior, but we also assume that there is one additional interaction layer that specifies the *role of the NPC with respect to the storyline*. For each NPC a transition system or FSM is assumed that specifies which events in the storyline may be initiated and handled by the NPC and how they affect an internal state. For example, a particular NPC may be used to initiate a conversation with the player that reveals a clue or initiate a quest, but only if in the course of the game the player has not previously engaged in combat with the NPC in some previous encounter. Different states of the FSM may be used to represent the internal state of the NPC, and transitions may be used to encode available storyline interactions at each state. The set of these FSMs constitute the *available behaviors* for behavior composition.

As far as the intended storyline is concerned, a desired *target behavior* is constructed that describes how the events in the storyline may unfold. The desired target is not a fixed sequence of events, rather than another FSM that provides a high-level view of the process that the storyline should follow. Each state in the FSM corresponds to a *decision point* allowing a number of available storyline events to be invoked as transitions that lead to other states accordingly.

Finally, anticipating that service-oriented computing may become a useful paradigm for game development, we provide a RESTful web service for behavior composition, called JACO. The interaction with JACO is carried out by sending and receiving HTTP messages according to the REST principles. Our intention is to release JACO as a cloud-based tool that game developers can employ *offline* to compute an "AI orchestrator" for NPCs in a video game, which can then be used *online* to orchestrate NPCs according to the specified target and the choices of the "AI director".

A motivating example

We adopt a simplified game concept where the player embarks to a journey of becoming a mighty fighter or a powerful magician by pursuing various quests. A high-level view of this storyline is depicted as the desired target FSM in Figure 1. The storyline structure requires the player to begin his journey by completing a fixed quest, i.e., $quest0$. After

this initiation phase, he has the opportunity to pursue various quests, i.e., quest1–quest7, which influence the path he is taking toward becoming a fighter or a magician. For example, different sequences of quests may result to the storyline FSM being in state "Fighter", "Magician", or "Main". Note that only from the states "Fighter" and "Magician" can the storyline evolve to the ending through an appropriate quest.

The target FSM does not prescribe the desired sequence of events; it only specifies a *process* that the storyline should comply with. A so-called *AI director* component could use this as the basis for presenting available quests to the player to choose in order to progress the story. A more sophisticated AI director could look into the available options and choose to progress the story using the one that would be considered more fun for the player taking into account other information. In any case, each state in the target FSM is a *decision point* which specifies the possible ways to proceed. The actual realization of the decision though is to be performed by one of the available NPCs, also expressed as FSMs.

In our simple example there are six NPCs that the player may interact with for the purposes of achieving quests. Each NPC may be involved in more than one quest affecting the evolution of the storyline with respect to the target FSM of Figure 1, but also affecting the disposition of the NPC in a positive or negative way. Depending then its disposition, a different set of quests may be facilitated by the NPC in the course of the game. For example, Character1 may be used to facilitate quest3 and quest4 but not both. Similarly, Character2 may facilitate quests 3,5, and 6, but executing quest6 may get him into "Negative" state in which only quest6 could be re-invoked. Note also that the same quest may have a positive effect for one character but a negative effect on another, e.g., quest3 for characters 2 and 3. Also, for simplicity we have allowed quests to reoccur in this specification – in practice this may be reasonable only for some quests.

A problem that arises is that it is not easy to guarantee that the available FSMs corresponding to NPCs can *realize all possible runs* of the specified target FSM. For example consider the following scenario:

- Character3 facilitates quest0: this causes the storyline to evolve from "Beginning" to "Main" state, and Character3 to change its internal state from "Neutral" to "Positive";
- Character4 facilitates quest1: this causes the storyline to evolve from "Main" to "Fighter" state, and Character4 to change its internal state from "Neutral" to "Negative";
- Character4 facilitates quest7: this causes the storyline to go back to "Main" state, and Character4 to change its internal state from "Negative" to "Positive".

At this point, according to the target FSM the player should still be able to get involved with quest1 or quest2, but the only character that is capable of facilitating it, i.e., Character4, is in "Positive" state, from which he is not able to facilitate these quests.

The method of behavior composition that we described in the introduction is able to automatically construct a *global strategy* that specifies how each of the transitions of the target FSM should be delegated to available FSMs in order to avoid such situations *for any possible run of the target transition system*. In the next section we present JACO, a web service that provides this functionality.

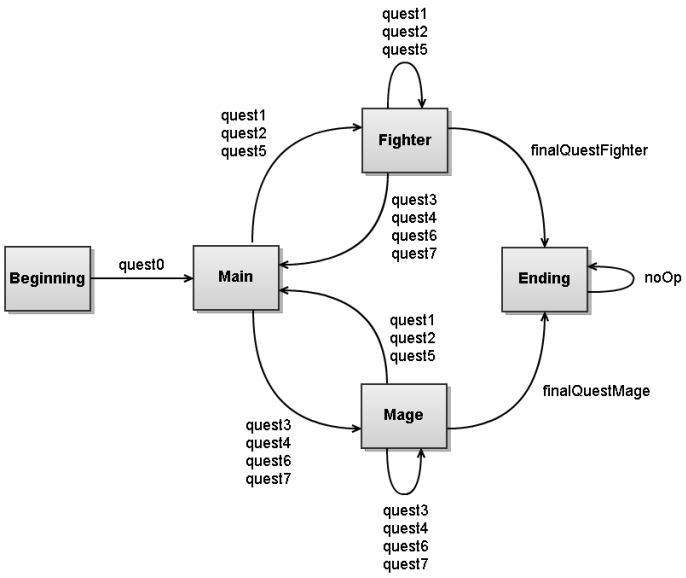


Figure 1: Storyline as a target FSM

Behavior composition with JACO

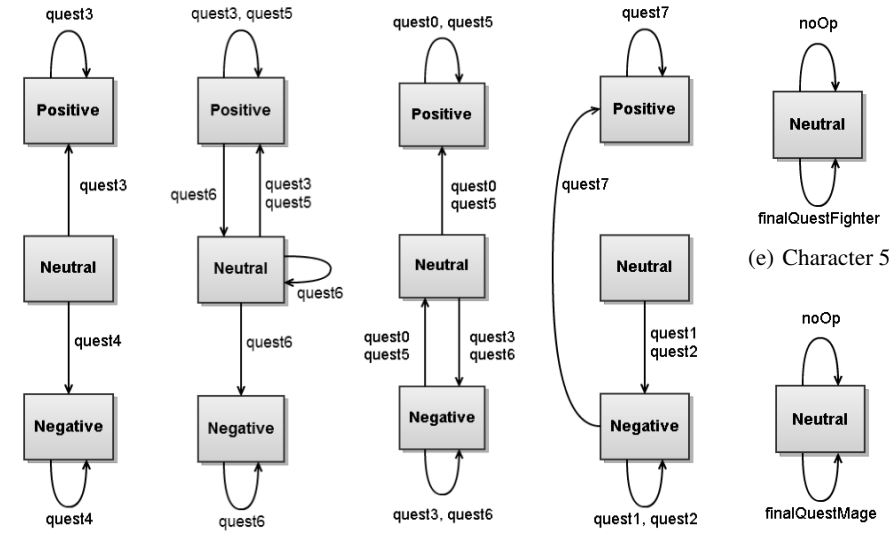
JACO is a web service for behavior composition following the REST software architecture. For a formal definition of behavior composition the reader is referred to (De Giacomo, Patrizi, and Sardiña 2013). The interaction with JACO is done by sending and receiving HTTP messages that handle information and requests. JACO's main computational component is built on top of *JTLV*[1],i.e., a Java implementation of the Temporal Logic Verifier (Pnueli and Shahar 1996).

JACO performs the following computation: given (*i*) a set of *available behaviors* as possibly nondeterministic FSMs and (*ii*) a *target behavior* as a deterministic FSM that is to be realized by combining execution fragments of the available behaviors, JACO provides a *composition* that specifies how the target can be realized. The composition can be used as a *look-up table* specifying for every collective state of the available behaviors and the target, which of the behavior can be used to realize the transitions supported by the target.

The JACO API identifies five endpoints with which the user can interact using HTTP verbs, such as GET and POST, in order to send or receive information expressed in XML:

- **/auth**: provides the user with a unique identifier **client_id** to be used with all requests to JACO as follows;
- **/{client_id}/behaviors**: allows the user to retrieve a list of available behaviors that have been submitted to the server by issuing an HTTP GET request, or add a new behavior with an HTTP POST request;
- **/{client_id}/behavior/{behavior_id}**: allows the user to perform the usual CRUD operations (Create, Read, Update, and Delete) on the behavior identified by **behavior_id** using HTTP GET, POST, PUT, DELETE;

[1] `http://jtlv.ysaar.net/`



(a) Character 1 (b) Character 2 (c) Character 3 (d) Character 4 (e) Character 5 (f) Character 6

Figure 2: The finite state machines of the six characters of the domain

- **/{client_id}/target**: allows the user to perform the CRUD operations to set, update or delete the desired target for behavior composition;
- **/{client_id}/composition**: allows the user to request the computation of the specified behavior composition problem by issuing a POST request, and obtain the resulting composition (or the status of the operation if it is still processing or in queue) using a GET HTTP request.

In a typical JACO usage scenario the user starts by getting a **client_id**, and then using appropriate POST requests specifies available behaviors and the desired target behavior, as well as initiates the computation of the behavior composition. Then the user goes on a loop of appropriate GET requests with which the status of the composition is retrieved, until the actual composition is returned as output. More information about the API and the XML data that is sent and received by the user can be found at `http://jaco.dis.uniroma1.it/`.

We now proceed to show how JACO can be employed to orchestrate the NPCs of Figure 2 in order to guarantee the realization of the target FSM of Figure 1.

JACO in action

Each of the NPCs of Figure 2 is specified as an available behavior using XML. The XML representation is very simple including a name and a finite state machine specified using nodes and transitions. For example, the following XML listing is used for Character1:

```
<behavior>
  <name>Character1</name>
  <finiteStateMachine>
    <state node="neutral">
      <transition action="quest3">
        <target>positive</target>
      </transition>
      <transition action="quest4">
        <target>negative</target>
      </transition>
    </state>
    <state node="positive">
      <transition action="quest3">
        <target>positive</target>
      </transition>
    </state>
    <state node="negative">
      <transition action="quest4">
        <target>negative</target>
      </transition>
    </state>
  </finiteStateMachine>
</behavior>
```

The target behavior is also represented using the same tags. The following is an excerpt used for the target of Figure 1:

```
<behavior>
  <name>Target</name>
  <finiteStateMachine>
    <state node="beginning">
      <transition action="quest0">
        <target>main</target>
      </transition>
    </state>
```

Following the usage scenario of JACO we get a client id and post the available behaviors and the target behavior one by one. We then request a composition that would provide an orchestrating strategy. If one such strategy exists, it will ensure that we can always continue progressing our storyline following the options formalized in the target FSM. Moreover for every available transition in the target FSM (and every corresponding possible state of all available FSMs), it will instruct exactly which of the available FSMs we should choose to realize the transition in order to ensure this.

When we get back an answer from JACO, the result is that such a composition is not possible. This means that some problematic runs for the target FSM (like the one we identified involving characters 3 and 4, and quests 0,1,2, and 7, in the section of the motivating example) cannot be avoided

by delegating the quests to characters in a different way. Essentially, *there exists some run for the target FSM* such that there is *no way to realize using the available FSMs*.

As a piece of information this is important to know but it is not very helpful for the purpose of specifying and executing an interactive storytelling experience as we intended. Nonetheless, in order to investigate which are all the problematic cases that lead to a deadlock, we can proceed as follows. We specify a simple "all-purpose" character that can facilitate all available quests always staying in the same "Neutral" state, and post it as an additional available behavior. Then we request a composition again. As expected, this time a composition is possible, but what is more interesting is that the details of the composition[2] point out the problematic cases. The composition shows at each step information about which of the available FSMs could facilitate a transition of the FSM based on the current state of the available FSMs. The cases then where only the "all-purpose" character comes out as an option are the ones that would originally lead to a deadlock. Essentially, this provides a simple way to "debug" storyline processes and behaviors at design time.

Related work

Our approach is similar in spirit to many other approaches in the literature that are based on automated planning, including STRIPS and HTN planning, for example the system I-Storytelling (Cavazza, Charles, and Mead 2002), GADIN (Barber and Kudenko 2009), and MIST (Paul et al. 2010) as well as the work on the framework Mimesis (Young 2001) and Zócalo (Young et al. 2011). Nonetheless, the methodology of behavior composition is different from planning both in conceptual and technical terms as we explain next.

Firstly, the target behavior is not a specification of a goal situation to reach but, rather, a description of a set of *routines* one would like to be able to carry on *at runtime*. Moreover, such routines cannot be seen as (classical or nondeterministic) plans, either, in that they do not prescribe the actions to execute, but leave the choice to the executor. Further, they may contain loops, which are typically ruled out in planning. From this perspective, target behaviors are more similar to IndiGolog programs (De Giacomo et al. 2009), i.e., high-level procedures definable on top of planning domains, for which one is typically interested to find an executable realization at runtime.

Secondly, in behavior composition, actions are not the subject of a planning task. Indeed, the controller does not select the actions to execute; instead it returns the index of the behavior that should execute the action selected by the AI director. In this sense, actions constitute the input, not the output, of the reasoning task, but in a way that takes into account *all possible narrative trajectories*. From a more formal perspective, we observe that both behavior composition and *conditional planning* are EXPTIME-complete problems (De Giacomo, Patrizi, and Sardiña 2013; Littman 1997), thus some way of reducing composition to (nondeterministic) planning must exist. Nonetheless, how this can actually be done is not as straightforward as one might expect, as shown by the above considerations.

Finally, our implementation of the behavior composition engine as the web-service Jaco is similar to the client-server based approach that is adopted in Mimesis and Zócalo. In fact as JACO is built as a pure behavior composition engine that can be accessed via a REST API, one interesting direction for future work is to explore how it can be used it as a service in such frameworks in order to provide high-level orchestration of characters, either as an alternative or in pair with the embedded narrative planner.

[2]XML files are available at JACO website.

Conclusions

In this paper we present some preliminary results experimenting with the AI method of behavior composition for the purpose of facilitating interactive storytelling in video games. We motivate the use of this method with a simple conceptual example that would require the orchestration of various non-player characters in order to facilitate different parts of the story. Anticipating that service-oriented computing may become a useful paradigm for this type of aspects of game development, we provide a RESTful web service for behavior composition, called JACO, and we use it to provide solutions for the motivating example. Our preliminary results show that such a service can be useful for use cases similar to our motivating example, and our current work focuses on identifying scenarios coming from commercial video games to validate our approach.

References

Barber, H., and Kudenko, D. 2009. Generation of adaptive Dilemma-Based interactive narratives. *Computational Intelligence and AI in Games, IEEE Transactions on* 1(4):309–326.

Cavazza, M.; Charles, F.; and Mead, S. J. 2002. Character-Based interactive storytelling. *IEEE Intelligent Systems* 17(4):17–24.

De Giacomo, G.; Lespérance, Y.; Levesque, H. J.; and Sardina, S. 2009. IndiGolog: A High-Level programming language for embedded reasoning agents. In *Multi-Agent Programming: Languages, Tools and Applications*. 31–72.

De Giacomo, G.; Patrizi, F.; and Sardiña, S. 2013. Automatic Behavior Composition Synthesis. *Artif. Intell.* 196:106–142.

Littman, M. L. 1997. Probabilistic Propositional Planning: Representations and Complexity. In *Proc. of AAAI 97 and IAAI 97*, 748–754.

Paul, R.; Charles, D.; McNeill, M.; and McSherry, D. 2010. MIST: An interactive storytelling system with variable character behavior. In *Interactive Storytelling*, volume 6432 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. 4–15.

Pnueli, A., and Shahar, E. 1996. A platform for combining deductive with algorithmic verification. In *Proceedings of the Eighth International Conference on Computer Aided Verification*.

Young, R. M.; Thomas, J.; Bevan, C.; and Cassel, B. A. 2011. Zócalo: A service-oriented architecture facilitating sharing of computational resources in interactive narrative research. In *Working Notes of the Workshop on Sharing Interactive Digital Storytelling Technologies at ICIDS*.

Young, R. M. 2001. An overview of the mimesis architecture: Integrating intelligent narrative control into an existing gaming environment. In *Working Notes of the AAAI Spring Symposium on Artificial Intelligence and Interactive Entertainment*.

Pushing the Envelope of Monte-Carlo Planning: Formal Guarantees Meet Practical Efficiency

Zohar Feldman
Technion & IBM HRL
Haifa, Israel
zoharf@tx.technion.ac.il

Carmel Domshlak
Technion
Haifa, Israel
dcarmel@ie.technion.ac.il

Abstract

Popular Monte-Carlo tree search (MCTS) algorithms for online planning, such as ε-greedy tree search and UCT, aim at rapidly identifying a reasonably good action, but provide rather poor worst-case guarantees on performance improvement over time. In contrast, a recently introduced MCTS algorithm BRUE guarantees exponential-rate improvement over time, yet it is not geared towards identifying reasonably good choices right at the go. We take a stand on the individual strengths of these two classes of algorithms, and show how they can be effectively connected. We then rationalize a principle of "selective tree expansion", and suggest a concrete implementation of this principle within MCTS. The resulting algorithm, BRUE$_{\mathcal{IC}}$, favorably competes with other MCTS algorithms under short planning times, while preserving the attractive convergence properties of BRUE.

Introduction

In online planning for MDPs, the agent focuses on its current state only, deliberates about the set of possible policies from that state onwards and, when interrupted, uses the outcome of that exploratory deliberation to choose what action to perform next. The quality of the action a, chosen for state s with H steps-to-go, is assessed in terms of the probability that a is sub-optimal, or in terms of the (closely related) measure of simple regret. The latter captures the performance loss that results from taking a and then following an optimal policy π^* for the remaining $H-1$ steps, instead of following π^* from the beginning (Bubeck and Munos 2010).

With a few recent exceptions developed for declarative MDPs (Bonet and Geffner 2012; Kolobov, Mausam, and Weld 2012; Busoniu and Munos 2012), most algorithms for online MDP planning constitute variants of what is called Monte-Carlo tree search (MCTS) (Péret and Garcia 2004; Kocsis and Szepesvári 2006; Coquelin and Munos 2007; Cazenave 2009; Rosin 2011; Tolpin and Shimony 2012). Most MCTS algorithms for online planning, such as ε-greedy tree search and UCT, aim at rapidly identifying a reasonably good action, but offer only polynomial-rate reduction of simple regret over the deliberation time. In contrast, a recently introduced MCTS algorithm BRUE guarantees exponential-rate reduction of simple regret over time, yet it does not make special efforts to home in on a reasonable alternative fast (Feldman and Domshlak 2012). Of course, "good" is often the best one can hope for in large MDPs of interest under practically reasonable deliberation-time allowances. This is precisely our contribution here: Reflecting on the differences between the two types of algorithms, we show that a redesign of BRUE, baptized BRUE$_{\mathcal{IC}}$, favorably and robustly competes with other MCTS algorithms under short planning times, while preserving both the attractive formal properties of BRUE, as well as the empirical strength of the latter under permissive deliberation-time allowances.

Background

An MDP $\langle S, A, Tr, R\rangle$ is defined over states S, actions A, a stochastic transition function $Tr : S \times A \times S \to [0, 1]$, and a reward function $R : S \times A \times S \to \mathbb{R}$. In the finite horizon setting considered here, the reward is accumulated over some predefined number of steps H. Henceforth, $\mathbf{\Pi}$ denotes the set of all valid policies for the MDP in question, $A(s) \subseteq A$ denotes the actions applicable in state s, the operation of drawing a sample from a distribution $\mathcal{D}$ over set $\aleph$ is denoted by $\sim \mathcal{D}[\aleph]$, $\mathcal{U}$ denotes uniform distribution, and $[\![n]\!]$ for $n \in \mathbb{N}$ denotes the set $\{1, \ldots, n\}$.

Canonical MCTS Scheme

MCTS, a canonical scheme underlying various MCTS algorithms for online MDP planning, is depicted in Figure 1a. Starting with the current state s_0, MCTS performs an iterative construction of a tree[1] $\mathcal{T}$ rooted at s_0. At each iteration, MCTS rollouts a state-space sample ρ from s_0, which is then used to update $\mathcal{T}$. First, each state/action pair (s, a) is associated with a counter $n(s, a)$ and a value accumulator $\widehat{Q}(s, a)$, both initialized to 0. When a sample ρ is rolled out, for all states $s_i \in \rho \cap \mathcal{T}$, $n(s_i, a_{i+1})$ and $\widehat{Q}(s_i, a_{i+1})$ are updated on the basis of ρ by the UPDATE-NODE procedure. Second, $\mathcal{T}$ can also be expanded with any part of ρ; The standard choice is to expand $\mathcal{T}$ with only the first state

[1] In MDPs, there is no reason to distinguish between nodes associated with the same state at the same depth. Hence, the graph $\mathcal{T}$ constructed by MCTS instances typically forms a DAG. Nevertheless, for consistency with prior literature, we stay with the term "tree".

MCTS: [input: $\langle S, A, Tr, R\rangle$; $s_0 \in S$]
search tree $\mathcal{T} \leftarrow$ root node s_0
for $n \leftarrow 1 \ldots$ time permits **do**
PROBE$(s_0, 0)$
return $\arg\max_a \widehat{Q}(s_0, a)$

PROBE (s : state, d : depth)
if END-OF-PROBE(s, d) **then return** EVALUATE(s, d)
$a \leftarrow$ ROLLOUT-POLICY(s)
$s' \sim P(S \mid s, a)$
$r \leftarrow R(s, a, s') +$ PROBE$(s', d+1)$
UPDATE-NODE(s, a, r)
return r

(a)

END-OF-PROBE (s : state, d : depth)
if $s \notin \mathcal{T}$ **then**
add s to $\mathcal{T}$ and **return** true
else if $d = H$ **then return** true **else return** false

UPDATE-NODE (s : state, a: action, r : reward)
$n(s, a) \leftarrow n(s, a) + 1$
$\widehat{Q}(s, a) \leftarrow \widehat{Q}(s, a) + \frac{r - \widehat{Q}(s,a)}{n(s,a)}$

ROLLOUT-POLICY (s : state)
if $n(s, a) = 0$ for some $a \in A(s)$ **then**
return $a \sim \mathcal{U}[\{a \in A(s) \mid n(s, a) = 0\}]$
else
$n(s) \leftarrow \sum_{a \in A(s)} n(s, a)$
return $\text{argmax}_a \left[\widehat{Q}(s, a) + c\sqrt{\frac{\log n(s)}{n(s,a)}}\right]$

EVALUATE (s : state, d : depth)
for $t \leftarrow d \ldots H$ **do**
$a \sim \mathcal{U}[A(s)]$
$s' \sim P(S \mid s, a)$
$r \leftarrow r + R(s, a, s')$
$s \leftarrow s'$
return r

(b)

Figure 1: (a) Monte-Carlo tree search template, and (b) the UCT specifics.

along ρ that is new to $\mathcal{T}$. In any case, once the sampling is interrupted, MCTS uses the information stored at the tree's root to recommend an action to perform in s_0.

Numerous concrete instances of MCTS have been proposed, with ε-greedy probably being the most widely known, and UCT (Kocsis and Szepesvári 2006) and its modifications (Coquelin and Munos 2007; Tolpin and Shimony 2011) being the most popular such instances these days (Gelly and Silver 2011; Sturtevant 2008; Bjarnason, Fern, and Tadepalli 2009; Balla and Fern 2009; Eyerich, Keller, and Helmert 2010; Keller and Eyerich 2012). Concrete instances of MCTS vary mostly along the implementation of the ROLLOUT-POLICY sub-routine, that is, in their policies for directing the rollout within $\mathcal{T}$. For instance, the specific ROLLOUT-POLICY of UCT is shown in Figure 1b. This policy is based on the deterministic decision rule UCB1 (Auer, Cesa-Bianchi, and Fischer 2002), originally proposed for optimal balance between exploration and exploitation for cumulative regret minimization in stochastic multi-armed bandit (MAB) problems (Robbins 1952). However, it has already been noticed that exploitation may considerably slow down the reduction of simple regret over time (Bubeck, Munos, and Stoltz 2011). Indeed, UCB1 (and thus UCT) achieves only polynomial-rate reduction of simple regret over time (Bubeck, Munos, and Stoltz 2011), and the number of samples after which the bounds of UCT on simple regret become meaningful might be as high as hyper-exponential in H (Coquelin and Munos 2007). In fact, none of the MCTS instances suggested so far breaks the barrier of the worst-case polynomial-rate reduction of simple regret over time.

Separation of Concerns

If fast convergence to optimal choice is of interest, then Monte-Carlo planning should be as exploratory as possible (Bubeck, Munos, and Stoltz 2011). However, what it means to be "as exploratory as possible" with MDPs is less straightforward than it is in MABs. In particular, recently it was observed that "forecasters" $s \in \mathcal{T}$ should be devoted to *two*, somewhat competing, *exploratory* objectives, namely identifying an optimal action $\pi^*(s)$, and estimating the value of that action, because this information is needed by the predecessor(s) of s in $\mathcal{T}$ (Feldman and Domshlak 2012).

Following this observation, previously we introduced MCTS2e, a refinement of MCTS scheme that implements the principle of "separation of concerns," whereby different parts of each sample are devoted to different exploration objectives (Feldman and Domshlak 2012). In MCTS2e (Figure 2a), rollouts are generated by a two-phase process in which the actions are selected according to an exploratory policy until an (iteration-specific) switching point, and from that point on, the actions are selected according to an estimation policy. A specific instance of MCTS2e, dubbed BRUE, was shown to achieve an *exponential-rate* reduction of simple regret over time, with the bounds on simple regret becoming meaningful after only exponential in H^2 number of samples (Feldman and Domshlak 2012).

The specific MCTS2e sub-routines that define the BRUE algorithm are shown in Figure 2b. Similarly to UCT, each node/action pair (s, a) is associated with variables $n(s, a)$ and $\widehat{Q}(s, a)$, but with the latter being initialized to $-\infty$. BRUE instantiates MCTS2e by choosing actions uniformly at the exploration phase of the sample, choosing the best empirical actions at the estimation phase, and changing the switching point in a round-robin fashion over the entire horizon. Importantly, if the switching point of a rollout $\rho = \langle s_0, a_1, s_1, \ldots, a_H, s_H\rangle$ is σ, then only the state/action pair $(s_{\sigma-1}, a_\sigma)$ is updated by the information collected by ρ. That is, the information obtained by the estimation phase of ρ is used only for improving the estimate at state $s_{\sigma(n)-1}$, and is not pushed further up the sample. While that may appear wasteful and counterintuitive, this locality of update is required to satisfy the formal guarantees of BRUE on exponential-rate reduction of simple regret over time (Feldman and Domshlak 2012).

```
MCTS2e: [input: ⟨S, A, Tr, R⟩; s_0 ∈ S]
  search tree T ← root node s_0; σ ← 0
  for n ← 1 . . . time permits do
      σ ← SWITCH-FUNCTION(n, σ)
      PROBE(s_0, 0, σ)
  return arg max_a Q̂(s_0, a)

PROBE (s : state, d : depth, σ ∈ [[H]])
  if END-OF-PROBE(s, d) then return EVALUATE(s, d)
  if d < σ then
    a ← EXPLORATION-POLICY(s)
  else
    a ← ESTIMATION-POLICY(s)
  s' ~ P(S | s, a)
  r ← R(s, a, s') + PROBE(s', d + 1, σ)
  if d = σ then UPDATE-NODE(s, a, r)
  return r
```

(a)

```
END-OF-PROBE (s : state, d : depth)
  if d = H then return true else return false

EVALUATE (s : state, d : depth)
  return 0

UPDATE-NODE (s : state, a: action, r : reward)
  if s ∉ T then add s to T
  n(s, a) ← n(s, a) + 1
  Q̂(s, a) ← Q̂(s, a) + (r − Q̂(s, a)) / n(s, a)

SWITCH-FUNCTION (n : iteration, σ ∈ [[H]])
  return H − ((n − 1) mod H) // round robin on [[H]]

EXPLORATION-POLICY (s : state)
  return a ~ U[A(s)]

ESTIMATION-POLICY (s : state)
  return a ~ U[{a | arg max_{a∈A(s)} Q̂(s, a)}]
```

(b)

Figure 2: Monte-Carlo tree search with "separation of concerns" (a), and the BRUE specifics (b).

Two Types of Forecasters

A comparative evaluation on *Sailing* (Péret and Garcia 2004) and *PGame* (Kocsis and Szepesvári 2006) domains showed that BRUE is continually improving towards an optimal solution, rather quickly obtaining results better than UCT (Feldman and Domshlak 2012). However, that evaluation also showed that UCT sometimes manages to identify reasonably good actions rather quickly, while BRUE is still "warming up". In fact, later we show that a very simple MC algorithm performs even better than both UCT and BRUE under tight planning deadlines. We now take a closer look at this gap between "fast optimal" and "fast good".

Consider the state/steps-to-go pairs (s, h) as a hierarchy of forecasters, all acting on behalf of the root forecaster (s_0, H) that aims at minimizing its own simple regret in a stochastic MAB induced by the applicable actions $A(s_0)$. In the setup of online planning, there is a conceptual difference between the exploration objective of the root forecaster and this of all other forecasters in the hierarchy. To see that, suppose there is an oracle that can provide each forecaster (s, h) *either* with the identity of the optimal action $\pi^*(s, h)$ but without revealing its value $Q_h(s, \pi^*(s, h))$, *or* with the value $Q_h(s, \pi^*(s, h))$ but without revealing the identity of $\pi^*(s, h)$. For the root forecaster (s_0, H), the first type of information is all he needs, while the second type of information buys him very little, if anything. In contrast, even if the oracle provides *all* the forecasters (s, h) *but* (s_0, H) with (only) the identities of the respective optimal actions $\pi^*(s, h)$, then the root forecaster (s_0, H) in some sense remains as clueless as it was before, and needs to explore the state space in order to obtain at least some ordinal information about the expected value of the alternative choices $A(s_0)$. However, if the oracle provides a non-root forecaster (s, h) (only) with the best Q-value among its alternative choices $A(s)$, then (s, h) can stop working since no further exploration of the sub-hierarchy rooted in (s, h) is needed.

In sum, what matters to the root forecaster is only what to execute, while all other forecaster care only about the value they can provide to their ancestors in the hierarchy, and *not about how this value can actually be acquired.* Of course, the reader may question this classification by arguing that these two objectives are just two sides of the same coin: estimating the value of optimal action assumes aiming at identifying an optimal action and vice versa. To some extent, that is true, but only to some extent. For instance, the very realization that this coin has two sides, and that these two sides are somewhat competing, is precisely what motivates the "separation of concerns" principle behind the MCTS2e scheme. Turns out that this classification of objectives suggests further insights into the dynamics of MCTS algorithms.

In all MCTS algorithms for online MDP planning, each iteration corresponds to examining a chain of forecasters within the overall hierarchy under (s_0, H), with the difference between the algorithm boiling down to two decisions:

(I) which chain of forecasters to examine, and

(II) how to estimate $Q_h(s, \pi^*(s, h))$ for each forecaster (s, h) in the hierarchy.

At first view, choosing the right strategy for (I) seems to be the key to rapid homing in on "good" decisions. The details of various MCTS algorithms suggest that their design was indeed primarily guided by choices for (I), with choices for (II) being implied by the former. Here, however, we suggest that decoupling these two decisions is important, and that the key to the quest of our interest actually lies in decision (II).

A closer look at different Monte-Carlo planning algorithms for MDPs reveals an interesting generalizing perspective on the way they all approach decision (II). Let $V_h^\pi(s)$ be the value of (s, h) under policy $\pi \in \mathbf{\Pi}$, $V_h^*(s) \equiv Q_h(s, \pi^*(s, h))$ be the value of (s, h) under the optimal policy, and let $\widehat{V}_h^\pi(s)$, $\widehat{V}_h^*(s)$ denote empirical estimates of these two quantities, respectively. In all MCTS algorithms,

at each point of time, the entire hierarchy of forecasters can be seen as consisting of *two types* of forecasters.

T_{OUT} forecasters (s, h) (possibly schematically) estimate $V^*_h(s)$ by an estimate of

$$\mathbb{E}_{\pi\sim\mathcal{U}[\mathbf{\Pi}]} V^\pi_h(s),$$

that is, of the expected total reward of a policy sampled from $\mathbf{\Pi}$ uniformly at random.

T_{IN} forecasters (s, h) distinguish between their alternative choices $A(s)$, and estimate $V^*_h(s)$ by an estimate of

$$\max_{a\in A(s)} \sum_{s'} P(s' \mid s, a) \left[R(s, a, s') + V^*_{h-1}(s')\right],$$

where the estimate of $V^*_{h-1}(s')$ is based on the information provided by s' to s.

Consider the way in which the specific MCTS algorithms approach decision (II) in terms of this T_{OUT}/T_{IN} partition of the forecasters. In both UCT and BRUE, *T_{IN}-forecasters correspond to the nodes of $\mathcal{T}$, while all other state/steps-to-go pairs correspond to T_{OUT}-forecasters*. Note that these T_{OUT}-forecasters are very much not virtual. For instance, in UCT they are queried by the EVALUATE sub-routine, and in BRUE they are queried, possibly in interleaving with T_{IN}-forecasters, by both EXPLORATION-POLICY and ESTIMATION-POLICY sub-routines.

At first view, T_{OUT}-forecasters appear to be strangely lazy and potentially very misleading, while T_{IN}-forecasters seem to be doing the right thing. However, it is not all that simple. First, while each T_{OUT}-forecaster samples a single random variable, each T_{IN}-forecaster (s, h) has to sample $|A(s)|$ random variables. Thus, T_{OUT}-forecasters converge to quality estimates of quantities of their interest much faster than their T_{IN} counterparts. Second, while T_{IN}-forecasters try to estimate the right thing, their success totally depends on the quality of estimates of $V^*_{h-1}(s')$ they receive from their successors. Hence, in general it is not clear that we should prefer all forecasters to be of type T_{IN}.

We return to this issue in more detail later on. For now, note only that both UCT and BRUE can be seen as *continuously reconsidering the typing of the forecasters*. Specifically, in both UCT and BRUE, (at most) a single forecaster is "converted" from T_{OUT} to T_{IN} at every iteration: in UCT it is the *shallowest* T_{OUT}-forecaster found along the rollout, and in BRUE, it is the T_{OUT}-forecaster that happens to lie at the rollout's switching point σ. This way, the set of T_{IN}-forecasters in UCT grows *incrementally* as a single community connected to the root forecaster (s_0, H). In contrast, T_{IN}-forecasters in BRUE evolve in H independent, equally sized sets, where each of these sets is distributed over the respective depth level of the forecast hierarchy according to the transition distribution induced by the *uniform* action selection at the preceding levels.

This specific difference between UCT and BRUE is directly related to their relative efficiency under different orders of deliberation time allowance. Populating T_{IN}-forecasters at all levels of the hierarchy is generally necessary to guarantee fast convergence to optimal choice at the root. However, the marginal value of T_{IN}-forecasters at different levels vary with the deliberation time allowance: Information gathered by T_{IN}-forecasters at deep levels takes time to be propagated to the root, making their near-term influence on the choices at (s_0, H) smaller than this of the T_{IN}-forecasters closer to the root.

In that respect, a modification of BRUE that suggests itself almost immediately is as simple as it gets: Instead of converting the T_{OUT}-forecaster at the switching point σ, we can resort to converting the shallowest T_{OUT}-forecaster on the exploratory part of the rollout, that is, up to the level σ. By offering both exponential-rate reduction of the simple regret at the root, as well as incremental conversion of T_{OUT}-forecasters as a connected set around (s_0, H), the resulting algorithm, BRUE$_\mathcal{I}$, substantially improves over BRUE in short-term effectiveness.[2] However, this simple modification of BRUE is not our final destination, and next we show that this simple bridge between MCTS and MCTS2e opens a much wider window of opportunity.

Selective Tree Expansion

Similarly to UCT and BRUE, each iteration of BRUE$_\mathcal{I}$ either finds no candidate for type conversion, or *unconditionally* converts a concrete single T_{OUT}-forecaster to type T_{IN}. However, suppose that we somehow know that, for that specific forecaster (s, h),

$$\mathbb{E}_{\pi\sim\mathcal{U}[\mathbf{\Pi}]} V^\pi_h(s) = \max_{a\in A(s)} \sum_{s'} P(s' \mid s, a) \left[R(s, a, s') + V^*_{h-1}(s')\right].$$

Since direct Monte-Carlo estimation of the quantity on the left-hand side is substantially easier than this of the right-hand side, converting (s, h) to type T_{IN} is clearly not a good idea. In fact, both (s, h) and all of its exclusive descendants in the hierarchy would better remain T_{OUT}-forecasters for the entire deliberation process, no matter how long it is. Of course, this equality rarely holds, and, more importantly, we have no prior knowledge about the size of the gap between the quality of the best policy under (s, h) and the expected quality of the randomly picked policy. However, this extreme example still hints on the promise of *selective* type conversion, and below we examine the prospects of this direction.

The variance of a Monte-Carlo estimator $\widehat{Q}_h(s, a)$ of the value of action a at state s stems from two sources. The first source of variance comes from following different policies (aka action selections) along different rollouts. The other source of variance comes from the stochastic nature of the action outcomes. That is, if r is the reward obtained by following policy π for h steps starting from state s, then

$$\text{Var}[r] = \mathbb{E}\left[\text{Var}[r \mid \pi]\right] + \text{Var}\left[\mathbb{E}[r \mid \pi]\right]. \qquad (1)$$

At one extreme, we have all policies yielding the same expected reward, and thus all the variance comes from the action outcomes. In that case, distinguishing between the policies under (s, h) is not only useless, but also computationally harmful. Thus both (s, h) and its descendants should be

[2] The specific empirical results for BRUE$_\mathcal{I}$ are shown later in the paper.

Planning and Execution Control Architecture for Infantry Serious Gaming

Alexandre Menif, Christophe Guettier[1] and Tristan Cazenave[2]
[1]SAGEM, 27, Rue Leblanc, 75012 Paris, France
[2]LAMSADE, Université Paris-Dauphine, Paris, France
{alexandre.menif, christophe.guettier}@sagem.com
cazenave@lamsade.dauphine.fr

Abstract

Serious gaming is developing among all modern armies for teaching and training as well as for developing new concepts of engagement. To reach a realistic level of simulation, online planning techniques provide an expressive and constructive approach to define basic tactical activities. To achieve a mission goal, a virtual soldier must follow a short-term plan that can be quickly reprocessed in order to follow changes in the environment, orders or situation awareness. This paper presents a planning and execution control architecture for simulating the behaviour of virtual infantry soldier. The planning approach relies on the frequent generation of short plans using a Hierarchical Task Networks approach. Execution control handles synchronisations of soldiers and trigger replanning whenever action cannot be executed in simulation. Applied to two generic types of action, preliminary results show that response times match the level of reactivity needed for serious gaming.

Introduction

Automatic planning has always given major challenges in defence domains. Many problem models and search techniques have been considered at strategic, operative or tactical command levels. However, simulation of low level tactics and basic soldier behaviours refer in general to action scripting. In order to match the expected level of realism required by modern armies, this practice tends to become a tremendous and time-consuming engineering activity.

In video game, however, planning techniques have become more and more popular for a decade. The experience of planning in video games has started with FEAR, a first person shooter issued in 2005, which implements an algorithm called GOAP (Goal Oriented Action Planning) to provide an efficient coordinated behaviour to the enemies. Although GOAP was inspired on STRIPS, one of the oldest algorithm in actions planning, this experience has successfully illustrated the possibility to compute short linear plans for a few coordinated agents in real time (Orkin 2006).

Planning techniques provide an expressive way to model tactical behaviour, by developing a compositional approach to basic activities. Several dedicated planning domains can be developed according to the type of mission, environment or military action. Using search techniques, automatic plan generation can be used at different hierarchical levels to simulate both the command chain and soldier activities.

The paper exhibits a planning and execution control architecture that integrates mission management along the command-chain, automatic tactical sequence generation and execution control. The architecture makes use of Hierarchical Task Networks (HTN) representations, that facilitate command-chain modelling, automatic goal breakdown as well as sequential task synthesis. To match mission execution tempo and contingencies, the planning functionality can generate very short plans over small horizons. Once a plan is generated, each action is tentatively executed by the execution controler which interacts with the simulated environment. Execution control also handles synchronisations of soldiers and trigger replanning whenever action cannot be executed.

Applied to two generic types of action, preliminary results show that response times match the level of reactivity needed for serious gaming.

First section focuses on the relation between serious gaming and military requirements for infantry warfare at tactical level. The second section describes a software planning architecture intended to meet those requirements. The next section presents the modelling of a planning domain for infantry tactical behaviours and provide details about a new implementation of SHOP2. Finally the last parts are dedicated to the state of the art and conclusion.

Soldier Level Serious Gaming

Rather than using old fashion tactical simulation, serious gaming is a very promising approach to teach, train and develop new concepts of engagement. Serious gaming reuses video game design and offers immersive and interactive environment to armies end-users. To be effective in professional tasks, these environments necessitate realistic and intelligent behaviour for simulated soldiers.

Different requirements stress both agents behaviour as well as the architectural design of such serious gaming:

- Teaching: Basic infantry scenarii must be easy to program, and soldier behaviour must be realistic enough to stimulate the end-user. Basic action sequences can be analysable to give explanations during after action review.

- Training: Such training stresses coordination, orders and reporting capabilities of the end-user. The serious game must be able to handle a large set of soldiers, and to construct complex situation. Again, in spite of their complexity, user performances can be analysed during after action review.
- Concept development: the development of new operational concept or the integration of new systems need doctrinal evolution. On one hand, serious gaming can help understanding the impact of new tactics, operational techniques and procedures. On the other hand, it provides a strong support to evaluate new system performances and man-machine interactions.

Planning and Execution Control Architecture

Figure 1 gives an overview of the Planning and Execution Control Architecture. It features two different levels of planning functionalities. Mission management solves medium and long term planning from brigade down to the platoon level. Below this level, squads and soldier sequence of actions are generated by a dedicated HTN planner. This core component drives the quality of virtual soldier behaviour and is detailed in a dedicated section.

The planning architecture is meant to be integrated in a real time simulator, which means that it is not possible to spend much processing power in planning procedures that may alter frame rate, gameplay and graphical rendering. The architecture must scale up to several dozens of entities to be managed in parallel, potentially leading to many planning queries to be conducted simultaneously (see figure 2).

Both planners takes inputs from a situation awareness and threat assessment components that gathers and fuses data from the simulation environment. The execution control component processes the realisation of a sequence of actions in the simulator kernel. Whenever a basic sequence of action cannot be correctly executed in the simulation environment, a replanning event is triggered.

Mission manager, situation awareness, threat assessment and execution control modules are fundamental software components for the architecture design, however their detailed design are out of the scope of this paper.

Mission manager

The mission manager combines mission planning and scheduling as well as plan decomposition for lower units. Given initial conditions (on both friendly and enemy units), mission planning and scheduling (P&S) defines the course of actions to reach mission objectives. P&S takes in account terrain structure, unit capabilities and their coordination :

- Terrain representation involve axis of advances, observation points, covers and concealments.
- Capabilities refer to mobility, engagement, communication and observations.
- Coordination of actions is needed in time and space for self protection, or to reach an expected effect.

The problem solved by the mission manager P&S is to find, for each unit, a course of actions and movements (e.g.

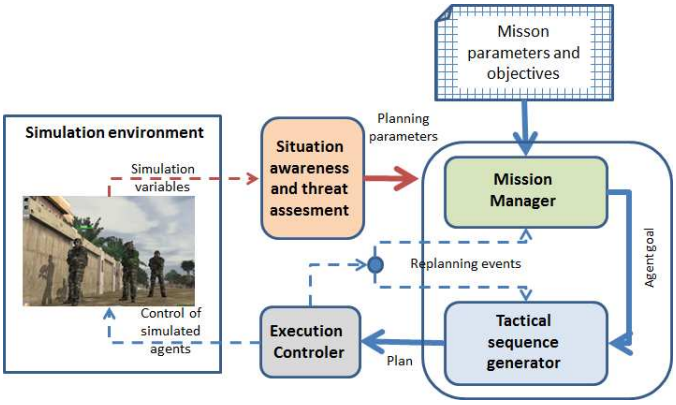


Figure 1: Global planning architecture for agent gaming

	squad	platoon	company	battalion
timeline	$<$ second	$<$ 5 mins	5 to 15 mins	$>$ hour
units	10	4	16	70

Figure 2: This table gives the timeliness with respect to the number of units to plan for

a plan) with an associated schedule from the initial position to the mission objective. The mission planning problem can be modelled and solved following a complete constraint programming approach (Guettier 2007).

The mission manager must also handle plan breakdown along the tactical command chain. Each hierarchical level defines an operational order (OPORD) using its own mission goal. The OPORD tasks / organises units of a lower echelon and allocates resources. This part is not automated yet, since many constructive parameters have to be taken in account simultaneously (organisational, logistics, assessment of enemy situation and associated course of action, complex coordination procedures...). However, it is possible to automatically structure the different OPORD by using the outcome of both mission planner and units organisation.

When major changes occur during mission execution, global replanning might be necessary. This has two impacts:

- A new problem instance is provided to the planner. Then, plan repair or local search can be used to find a plan that cope with the new situation.
- OPORD are updated, using so-called FRAGmentary Orders (FRAGO).

Execution Controller

The execution controller executes action provided by the sequence generator. In several ways it interacts with the simulation environment by controlling the virtual soldier. For a given action, it will set the virtual soldier mobility, orientation, and posture. Through the simulator, the controller can request a waypoint to reach, assess visibility of a line-of-sight, or find threatening objects within its field of view. To evaluate the success of an action, the execution controller uses the following logic:

- Preconditions are verified. Their scope are mobility (way points), previous action termination, observation results, or event occurring in the environment.
- Request to the simulator succeeds (for instance, the path finder has been able to find a waypoint).
- Time / space coordination can be met. These coordination can be either defined by the mission planner or by collaborative type of actions (for instance a mutual protection or a synchronised mobility action).

Whenever an action cannot be successfully achieved, a replanning event is triggered.

Situation awareness and threat assessment

Situation awareness is maintained by a Red Force Tracker (RFT) directly inspired from target acquisition and tracking systems (Sella & al. 2011). The RFT service tackles observation reports, target association, short term position estimate, long term enemy course of action prediction. Data fusion algorithms such as Kalman Filtering (KF), Interactive Multiple Models, or Multiple Hypothesis Tracking are integrated to associate observations, remove inconsistencies, and to manage an active list of tracks. Delayed reporting and tracking are also simulated such that two units does not necessarily have the same immediate awareness (note that in real life, this knowledge is actuated by issuing an OPORD through the chain of command).

Based on these outputs, whenever the enemy situation changes, each soldier or squad evaluates its own threat level in order to potentially recompute a new plan sequence.

Tactical sequence generator

Each hierarchical unit uses planning for its level and then assigns orders for its sub-level units. Units coordination is verified by planning at the upper level, so that each level can behave more autonomously within its own action area. It also results in time / space synchronisations, enforced by the execution controller.

In combat simulation, the tactical environment is constantly evolving, so that plans can be quickly invalidated. Replanning is necessary to comply with the up to date situation awareness. Relying on a planning domain where actions would be interleaved between hierarchical levels would be hard to manage. Indeed this would likely produce complex and expensive plans that would be compromised by the first unexpected event.

Instead, the proposed approach is closed to the Command Hierarchies concept (Pittman 2008). This results in a much more flexible way of planning: not only the global plan search complexity would be reduced, but an unexpected event would only affect a tiny sub-part of the overall plan and not trigger a re-planning for the whole operation. Another aspect is that low-levels units would probably be much more subjected to planning events than high-levels ones. For example, the detection of obstacles, traps or enemies would surely alter the way a fireteam had planned to conduct its mission, but not necessarily the global manoeuvre of a platoon.

```
; monitor problem definition
(defproblem problem monitor
  ; this line describe the environment as a set of predicates
  ((down soldier1) (detected soldier1 target sector))
  ; the task assign to the soldier
  ((monitor soldier1 sector)))

; a computed plan for the monitor problem
(:task !report soldier1 target)
(:task !stand-up soldier1)
(:task !use-weapon soldier1 target)

; patrol problem definition
(defproblem problem patrol
  ((sector front) (sector sector1) (unsafe sector1) (front fireteam soldier2)
   (back fireteam soldier1) (covering soldier1 front) (covering soldier2 sect
   (up soldier1) (up soldier2))
  ((patrol fireteam)))

; a computed plan for the patrol problem
(:task !cover soldier2 sector1)
(:task !cover soldier1 front)
(:task !find-cover-point soldier1 sector1)
(:task !go-to-cover-point soldier1 sector1)
(:task !pass soldier1 soldier2)
(:task !cover soldier1 sector1)
```

Figure 3: Planning request and plan answer for monitoring actions

Infantry domain modelling

It sounds crucial that the lower a unit is in the command chain, the simpler its behaviour should be defined. For example, a single soldier would probably be constantly planning to adapt his activity to a constantly changing environment, so his behaviour should be extremely cheap to plan. Therefore, our requirement for a planning domain relies on two aspects: actions at one hierarchical level should only focus on its level and the synchronization of the direct sub level, and at the bottom of the hierarchy, planning should be extremely simple. With those rules in mind, we have started to design a simple planning domain that could match our expectation, using SHOP formalism (Nau & al. 2003).

The planning domain (see appendix) describes a set of very simple actions for monitoring and patrolling tasks, carried out by one soldier or a two-soldiers fireteam. The ”monitor” action illustrates how planning would interact with events from the simulator. Here the simulator would request planning for the ”monitor” task each time an event modifies the situation (in this case, simply either there is an enemy which is detected or not). Then, the virtual soldier has a few possible behaviours. He may or may not engage the target, according to its ability to do it or if it is allowed to do it. The ”patrol” action is collaborative (it involves two soldiers from a fireteam) and handles the coordination between the two soldiers (so one soldier only moves if it is covered by the other), and will assign very simple tasks to each soldier agent. Figure 3 representes two typical queries for both examples (planning requests for actions ”monitor” and ”patrol”) as they would be issued by the simulated environment, and the replies from the planning system: two plans as sequences of tasks.

Search Algorithm

In order to constitute a benchmark of popular planning algorithms, a first implementation of SHOP2 is under development. The C++ programming language is used as it is

the one employed in the targeted simulator product. At its current state, the planner implements the main core functionalities of the original one from Nau (Nau & al. 2003), alongside with a parser component that is able to read planning domains and problems written with a subset of the formalism described for SHOP.

On different situations, all the previously detailed examples (figures 3) are computed in less than a millisecond with our current implementation of SHOP2. Even if actual planning domains would probably be more sophisticated, this is the target time we will have to achieve for such basic level hierarchical units.

Conclusion

This paper proposed a way to apply planning techniques from the video games experience on basic military units behaviour to professional simulation tools. The main objective is to demonstrate that tactical behaviours for low level entities of military hierarchy (individual soldier, fireteam, squad...) could be encoded and executed with low computational power through two main considerations.

On one hand, hierarchical planners seem adapted to model complex behaviour for coordinated units. They affords a segregation of the chain of command from the local virtual agent behaviour, alongside with a convenient expression of action sequences. On the other hand, a reactive architecture, aware of any triggered events from the simulated environment, can restrict the time horizon for planning request considerably, and thus reduces the need for complicated planning processes.

The dedicated planner fits easily with other components (long term mission planning, situation awareness and execution control). In terms of software engineering, the approach provides a strong gain compared to scripting methods.

References

Guettier C. Solving Planning and Scheduling Problems in Network based Operations. Proceedings of CP'07, USA. 2007.

Sella, G.; Cherrier, O.; Guettier, C. & Yelloz, J. Development and Experimentation of Collaborative Red Force Tracking in Service Oriented Architecture for Tactical Networking Systems Proceedings of MILCOM'11, USA. 2011

Orkin, J.. Three states and a plan: the AI of FEAR. In Game Developers Conference. 2006.

Nau, D. S.; Au, T. C.; Ilghami, O.; Kuter, U.; Murdock, J. W.; Wu, D. & Yaman, F. SHOP2: An HTN planning system. J. Artif. Intell. Res. (JAIR), 20, 379-404. 2003.

Pittman, D. Command Hierarchies Using Goal-Oriented Action Planning, AI Game Wisdom 4, Charles River Media (2008), pages 383 to 391. 2008.

Planning domain

```
(defdomain monitor (
  (:operator (!use-weapon ?soldier ?target) ((up ?soldier)) () ())

  (:operator (!watch ?soldier ?area) ((up ?soldier)) () ())

  (:operator (!report ?soldier ?target) () () ((can-engage ?soldier ?target)))

  (:operator (!bend-down ?soldier)
             ((up ?soldier))
             ((down ?soldier)))

  (:operator (!stand-up ?soldier)
             ((down ?soldier))
             ((down ?soldier))
             ((up ?soldier)))

  (:operator (!cover ?soldier1 ?sector1)
             (and (sector ?sector2) (covering ?soldier1 ?sector2))
             ((covering ?soldier1 ?sector2))
             ((covering ?soldier1 ?sector1)))

  (:operator (!follow ?soldier1 ?soldier2) () () ())

  (:operator (!go-to-waypoint ?soldier1) ((have-waypoint ?soldier1)) () ())

  (:operator (!find-cover-point ?soldier1 ?sector1)
             ()
             ()
             ((have-cover-point ?soldier1 ?sector1)))

  (:operator (!go-to-cover-point ?soldier1 ?sector1)
             ((have-cover-point ?soldier1 ?sector1))
             ()
             ((at-cover-point ?soldier1 ?sector1)))

  (:operator (!pass ?soldier1 ?soldier2)
             (and (front ?fireteam ?soldier2) (back ?fireteam ?soldier1))
             ((front ?fireteam ?soldier2) (back ?fireteam ?soldier1))
             ((front ?fireteam ?soldier1) (back ?fireteam ?soldier2)))

  (:method (use-weapon ?soldier ?target)
           ; soldier cannot use weapon if down
           ((down ?soldier))
           ((!stand-up ?soldier) (!use-weapon ?soldier ?target))
           ; soldier already up
           ()
           ((!use-weapon ?soldier ?target)))

  (:method (watch ?soldier ?area)
           ; soldier cannot watch if down
           ((down ?soldier))
           ((!stand-up ?soldier) (!watch ?soldier ?area))
           ; soldier already up
           ()
           ((!watch ?soldier ?area)))

  (:method (engage ?soldier ?target)
           ; soldier must protect itself
           ((under-fire ?soldier))
           ((!bend-down ?soldier))
           ; target is neither hostile nor can be engaged
           ; according to rule of engagement
           ((not (hostile ?target)) (not (can-engage ?soldier ?target)))
           ((!report ?soldier ?target) (use-weapon ?soldier ?target))
           ; else engage target
           ()
           ((use-weapon ?soldier ?target)))

  (:method (monitor ?soldier ?area)
           ; a target is detected while monitoring
           (and (detected ?soldier ?target ?area))
           ((engage ?soldier ?target))
           ; else, keep watching
           ()
           ((watch ?soldier ?area)))

  (:method (patrol ?fireteam)
           ; if a target is detected in a given sector, engage it
           ((detected ?target ?sector1) (covering ?soldier1 ?sector1))
           ((engage ?soldier1 ?target))
           ; if there is an unsafe sector, use parrot-like move
           ((sector ?sector1) (unsafe ?sector1)
            (front ?fireteam ?soldier1) (back ?fireteam ?soldier2))
           ((!cover ?soldier1 ?sector1) (!cover ?soldier2 front)
            (!find-cover-point ?soldier2 ?sector1)
            (!go-to-cover-point ?soldier2 ?sector1)
            (!pass ?soldier2 ?soldier1) (!cover ?soldier2 ?sector1))
           ; else progress normally
           ((have-waypoint ?soldier1) (front ?fireteam ?soldier1)
            (back ?fireteam ?soldier2))
           ((!cover ?soldier1 front) (!cover ?soldier2 far)
            (!follow soldier2 soldier1) (!go-to-waypoint ?soldier1)))))
```

BlocksWorld: An iPad Puzzle Game

Minh Do* **and Minh Tran†**

* SGT Inc., NASA Ames Research Center, Mail Stop 269-3, Moffett Field, CA 94035
† TranCreative, 788 Stern Ave, Palo Alto, CA 94303

Abstract

Blocksworld is arguably the most well-known and simplest planning domain. Classical planning is the most researched branch of AI planning. In this paper, we describe an iPad puzzle game, appropriately named *BlocksWorld*, in which users solve a variation of the traditional blocksworld problems in the form of anagram matching. User performance is graded by comparing against the baseline score produced by a state-of-the-art classical planner using the exact same scoring function employed by the recent International Planning Competitions (IPC). The app has been released in the Apple AppStore and accumulated more than 1000 downloads. Our game demonstrates that the most basic planning concepts can be utilized in a game/application that people are willing to play.

Introduction

Blocksworld is one of the oldest benchmark domains in planning research. It's simple, easy to understand, challenging, and is representative of a large class of "construction" domains (Hoffmann 2005). Classical planning, the simplest branch of planning, dominates the planning research landscape based on both the number of papers accepted and the number of best paper awards won at recent ICAPSs. However, the most popular complaint at the Festivus events is that there are not enough work on planning application and simplistic settings such as classical planning and simple domains such as blocksworld are the leading examples of the disconnection.

The debate between the connection/disconnection between important topics in planning research and planning application developments motivated us to build a planning application based on solving the simplest planning domain with a classical planner. With a strong belief that getting planning technology into applications is the most effective way to grow the field, we want to build a planning application that can serve as a new type of motivation: instead of pointing out the gap between basic and applied planning research, we would like to encourage planning researchers to look for new types of not-so-complex applications where techniques such as classical planning can readily be applicable and can make impact with minimal efforts.

For the rest of this paper, we will describe *BlocksWorld*, an iPad puzzle game, and the roles within this game of the familiar planning concepts such as: PDDL modeling, classical planning, IPC's objective and scoring functions. The app has been released in the Apple's AppStore last year with limited success.

Background & Game Setup

BlocksWorld: is an artificial planning domain with actions moving unique blocks between different towers located on a table. The goal is to reconfigure the initial block-tower setting to match with a predefine goal tower configuration. Because it is clear and simple, it has been by far the most frequently used example in the AI planning literature since the 1960s and has been rather thoroughly investigated (Gupta and Nau 1992; Slaney and Thiebaux 2001). The two most cited versions of blocksworld are: (1) 4-operator version: that involves a robot hand that can *pick-up* or *drop-off* a single block from the top of a tower or from the table; and (2) 3-operator version: in which there is no robot hand and blocks can be moved in parallel between different towers and the table. In both versions, the table has infinite number of free spaces. Gupta & Nau (1992) discusses some other variations such as with limited number of table spaces or blocks having different sizes.

Classical planning: is the the simplest setting in planning research where: (1) state variables are discrete; (2) actions are instantaneous and deterministic; (3) the world state is fully-observable; and (4) the planner is the only agent that can change the world state. The planner's objective is to find a sequence of actions that leads from the fully-specified initial state to a state that satisfies all goals.

After exploring different potential ideas for making an application out of the traditional blocksworld domain and classical planning, we settled on developing an iPad app[1] in which users try to solve blocksworld problems with various levels of difficulty. Specifically, an user will use his fingers, through the iPad's touch interface, to move and rearrange the blocks from the initial block arrangement to the final/goal configuration. User can also rearrange the full towers. However, the game score only keeps track of the number of block-moving actions, not the number of tower-rearrangement actions.

Figure 1 shows several variations of the blocksworld domain that we have considered to be used in the app. The first one (showing the famous Sussman Anomaly problem (Sussman 1975)) is the traditional variation where blocks are unique and the initial and goal states are random configurations. We discarded this version due to our believe that random configurations are not interesting for normal human players. The second variation that we considered has blocks with different properties such as having different colors or are

[1] iPad is a new type of tablet computer made by Apple Inc. iPad uses iOS operating system and has access to the Apple's App Store, from which a published iOS application can be downloaded and installed on the iPad.

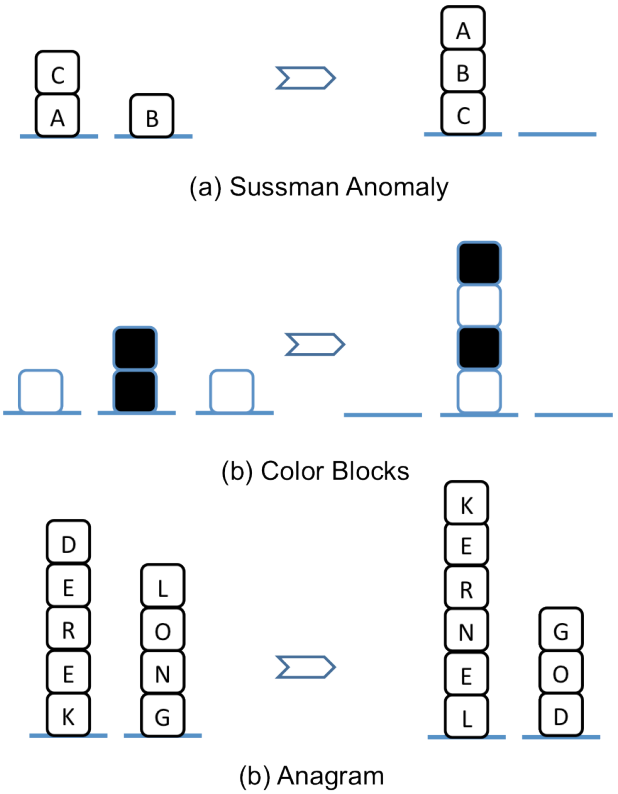


Figure 1: *BlocksWorld* Variations

built of different materials (e.g., glass, wood) and thus carrying different properties. Figure 1(b) shows an example of a colored-block variation. We believe that they are interesting enough. However, one critical problem is that we are not artist and it is hard for us to create by ourselves enough interesting problems of different difficulty levels. We finally settled on using anagrams[2]. Figure 1(c) shows an ICAPS-friendly anagram example in which users need to reconfigure the initial block configuration: "DEREK LONG" into the goal configuration "KERNEL GOD"[3]. In our app, we use two sets of anagrams ranging from easier ones like "THE EYES" → "THEY SEE" to the harder problems like "STATUE OF LIBERTY" → "BUILT TO STAY FREE". The anagrams were collected and handpicked from hundreds of anagrams available freely on various anagram-enthusiast websites.

Figure 2 shows the app's screenshot of the playing windows of one example: "VACATION TIME" → "I AM NOT ACTIVE". The main difference from the limited table space variation of the blocksworld domain (Gupta and Nau 1992) is that: blocks are not all unique and can be *duplicate*. they are represented by the same letter and should be considered "interchangeable" (e.g., there are two "I" , two "A" , and two "T" blocks in the example shown in Figure 2). This is necessary due to the nature of anagrams. Note that we did not choose the traditional unlimited table space setting because we believe it will degrade the game play experience. That version allows the possibility of having too many towers, thus enforcing us to either: (1) reduce the size of the playing windows, which can make the blocks tiny and not touch friendly; or (2) the user will need to scroll left or right to be able to see the whole set. Both options are not attractive.

[2]An anagram is a type of word play involves rearranging the letters of a word or phrase to produce a new word or phrase.

[3]This example came from Malte Helmert.

Problem Ranking & PDDL Modeling

As all multi-level games, the problems need to be ranked in the order of increasing hardness. In this case, it is easy to use the number of block moves from the initial configuration to the desired configuration as the measurement of problem hardness. Thus, we can safely use the plan-length in terms of the number of block-moving actions in classical planning setting; which is the standard object function in the IPC for the classical planning track. To measure that, we simply need to first model this blocksworld variation in PDDL and use a state-of-the-art classical planner, preferably optimal classical planner, to find the shortest plan. Thus, the problem of lower level is the one with shorter (optimal) plan-length.

PDDL Model: As mentioned above, we need to adjust the traditional PDDL no-hand blocksworld model with three block-moving actions: *MoveToTable*, *MoveFromTable*, and *MoveToBlock* to account for two changes: (1) limited number of table spaces; and (2) interchangeable blocks. The first issue can be handled quite easily while the second required a surprisingly large amount of work to get a clean PDDL model. We indeed had to seek advices from Patrik Haslum and Malte Helmert, two of the best researchers on classical planning to come up with the final model. Our full PDDL model is included in the Appendix.

To handle the limited number of table spaces, we use an approach similar to modeling rover's battery level when the Mars Rover domain was first introduced in the 3^{rd} IPC in 2002 (Long and Fox 2003). Specifically, we introduced:

1. an object type *count*, and the predicate *(next ?n - count ?n-plus-one - count)* to represent consecutive integer numbers
2. a predicate *(table-space ?n - count)* that tracks the number of "free" table spaces; this predicate changes value whenever a *MoveToTable* or *MoveFromTable* action executes.

To handle interchangeable/duplicate blocks, after several iterations, we settled on the version in which blocks have types (e.g., two blocks each in Figure 2 are of the same type "I" , "A" , or "T"). The goals describe the final tower configuration with a given block *B* at a particular position *P* on a tower *T* needs to be of type *X*. For example, in Figure 2, the goals specify that the block at the second location (from the bottom) on the first tower ("VACATION") is of type "O".

We then add two more actions to the existing set of three existing *MoveToTable*, *MoveFromTable*, and *MoveToBlock* block-moving actions:

- *Start-Tower*: that starts the process of building a particular goal tower, basically establishing that the first block that is on the table should match the block type specified in the goal to be the foundation of that tower. For example (refer to Figure 2), if the block on the table is of type "N", then it can *start* the first tower.
- *Extend-Tower*: that extends the tower T that has been built up to level L to the new level $L+1$. Basically, it checks:
  1. if a given tower T that has been built up to L correctly;
  2. if the block B at level $L+1$ matches the block-type at level $(L+1)$ of T specified in the goal condition.

  For example, in Figure 2, if the first tower has been "built" successfully until level $L = 3$ (i.e., the lowest three blocks

Figure 2: Screenshot of the playing pane of one problem: "VACATION TIME" → "I AM NOT ACTIVE".

are of correct types "I O N" in that order) and the planner checks that if the block at level $L + 1 = 4$ is of type "T", then this tower can be extended from level 3 to level 4.

Each of the goal tower will be *started* and then keep on being *extended* with blocks of the correct types until it reaches its full height. Obviously, adding the tower starting/building actions will lengthen the final plan. However, those extra actions can be filtered out easily through post-processing to reveal the actual number of block-moving actions. One additional (critical) nice property of this particular PDDL model is that for a given problem P, every valid plan has the same number of tower start/building actions (equals to the total number of blocks + total number of towers). Therefore, an optimal plan for P also represents the optimal solution when excluding the "dummy" tower building actions. In the Appendix, we includes the full PDDL model.

Problem Hardness/Ranking: We wrote several scripts to automatically generate the PDDL problem files for all the candidate anagrams. We then use off-the-shelf classical planners to find the best solution for each problem. Specifically, we run FastDownward with LAMA-2011 setting (Helmert 2011) which uses an anytime algorithm to incrementally finds better quality solutions. We ran LAMA-2011 with 30 minutes of running time on an early 2010 Macbook Pro machine with Core I7 and 8GB of RAM[4].

In summary, for each problem P, we find the best possible plan in terms of plan length $L(P)$ by running LAMA for 30 minutes. We then discount the number of extra tower building actions from $L(P)$ to get the number of only block-moving actions $L^{'}(P)$. We then rank all problems P in our final problem set in the increasing order of $L^{'}(P)$. The easiest problem in our set is P_1: "MUMMY" → "MY MUM" with $L^{'}(P_1) = 3$ and the hardest problem is "TELEVISION PROGRAMMING" → "PERMEATING LIVING ROOMS" with $L^{'}(P_{55}) = 69$.

Near the top-left corner of Figure 2, the line "Best Known: 14 moves" represents the best solution $L^{'}(P_{22}) = 14$ found

[4]Besides LAMA, we also ran another optimal-guarantee planner (Zhou and Hansen 2006). However, that planner can only solve very small problems and for all problems that it can solve, LAMA-2011 also returns best solution of the same quality. Another option is to use the domain-specific algorithm (Slaney and Thiebaux 2001). However, we believe that existing domain-specific algorithms do not work for our blocksworld variation with interchangeable blocks.

by LAMA for this problem. This serves two main purposes: (1) to challenge the users; (2) to act as the basis to grade the user's performance (details in the next section).

Measuring User Performance

Like other games, the user's performance needs to be measured and for that we turn to the way the last two IPCs have used to score competing planners. Specifically, for a given benchmark classical problem P, assume that the best-known solution for P is $L^*(P)$. If a given competing planner X can return within the allotted time the plan with length $L^X(P)$ then the score given to X for P is: $S = L^*(P)/L^X(P)$.

Our scoring function is exactly the same with the value $L^{'}(P)$ produced by LAMA serves as the baseline score $L^*(P)$ and the length of the user's actual "plan" (i.e., number of block moves that the user made until the final goal configuration is made) serves as $L^X(P)$ in the IPC scoring function above. In short, each user acts as a planner competing in our *BlocksWorld* IPC. We also integrated our *BlocksWorld* app with Apple's Game Center so that users can report their best scores and see how they rank against other players. For us, we are mostly interested in seeing if any user can indeed beat our baseline planner LAMA in any problem. Given that LAMA does not guarantee optimality, it is totally possible to see a user performs better than the best-known-score provided by LAMA. However, we haven't seen that so far.

Near the top-left corner of the Figure 2, the line "Your Best: 24 moves - 58 points" indicates that the best found solution by this player X is $L^X(P) = 24$ and thus the score for this player for this problem is: $14/24 = 0.58 = 58/100$ points with $L^*(P) = 14$ shown right above that line. Another game-related extension is that we added the star-based performance-grade visualization for easier categorization of user performance. Specifically:

- no star iff $S \leq 0.6$;
- one $\star$ iff: $0.6 < S \leq 0.75$ (i.e., within 40% of the best solution);
- two $\star\star$ iff: $0.75 < S \leq 0.9$ (i.e., within 25% of the best solution); and
- the full three $\star\star\star$ iff: $0.90 < S$ (i.e., within 10% of the best solution).

The star system is popular and easier for human players to understand than the numerical scoring function. Figure 2 also has example of how we display the star score to the user (again, near the top-left corner).

Current Status & Future Work

We have released *BlocksWorld* on the Apple's AppStore around March, 2012 and priced it initially at $2.99 (then later changed to $0.99), the AppStore URL for this game is: *https://itunes.apple.com/us/app/blocksworld/id593706027?mt=8*. It has accumulated 110 downloads. We then changed it in February, 2013 to the free-with-ad model and at the moment accumulated around 1020+ downloads (thus total of 1120+ downloads). Given the fact that there are more than 200 millions iTunes accounts, that is a tiny number. Thus, our app so far is not a successful iPad app. However, from the planning application point of view, we believe that having at least 1120 unique users indeed make it one of the more used applications/games utilizing AI planning technologies directly.

From the user-acquisition point of view, there are several lessons learnt. So far, we have not used any marketing mechanism and totally depended on "word-of-mouth" marketing. Thus, if a given user likes it, he may tell his friends about that app. Obviously, it haven't really worked in this case. With more than 1 million iOS apps (with 300,000+ exclusively built for the iPad), app-discovery is a big problem and we know that most popular apps spent heavily on many different marketing channels.

There are a couple of things that we intend to do. We would like to spend some time thinking about the best way to market the app, which is none so far. There are many paid services for app-marketing but we are not sure at the moment which one is the most effective for this type of games.

Conclusion

In this paper, we describe *BlocksWorld*, our effort to create a game/application from some of the most basic components of our planning research community: the simplest planning benchmark domain + the simplest classical planning setting + off-the-shelf classical planner + IPC objective and scoring function. Our app is available on the Apple AppStore. While it is not a successful iPad app, it has paid users and is a legitimate planning application.

Given the contrast between our *BlocksWorld* game and most traditional planning applications that are complex, expensive, and single-user, we would like to see *BlocksWorld* acting as a motivation for planning researchers to look for applying their work to a new type of real-world applications. The raise of ecosystems such as the Apple's AppStore has enabled $0.99-AngryBird-making Rovio to become a multi-billion-dollar company and flashlight apps that do nothing more than turning the whole phone screen into a pure-white windows to be used by millions of users. Maybe AI planning researchers can look at those opportunities to see if we can get planning technologies and concepts, no matter how "simple" it is, into the new style of applications that can impact the world in a different way: small impact to a single user multiplied by millions of unique users.

Acknowledgement: We would like to sincerely thank Patrik Haslum and Malte Helmert for helping us building a clean PDDL model for our variation of blocksworld and also with information on how to setup and run FastDownward/LAMA.

References

Gupta, N., and Nau, D. 1992. On the complexity of blocks world planning. *Artificial Intelligence* 56:223–254.

Helmert, M. 2011. The fastdownward planner. In *http://www.fast-downward.org/*.

Hoffmann, J. 2005. Where ignoring delete lists works: Local search topology in planning benchmarks. *Journal of Artificial Intelligence Research* 24:685–758.

Long, D., and Fox, M. 2003. The 3rd international planning competition: Results and analysis. *Journal of Artificial Intelligence Research* 20:1–59.

Slaney, J., and Thiebaux, S. 2001. Blocks world revisited. *Artificial Intelligence* 125(1-2):119–153.

Sussman, G. 1975. A computer model of skill acquisition.

Zhou, R., and Hansen, E. 2006. Breadth-first heuristic search. *Articial Intelligence Journal* 170:385–408.