Planning as a software component: A Report from the trenches *

Olivier Bartheye and Éric Jacopin
MACCLIA
Crec Saint-Cyr
Écoles de Coëtquidan
F-56381 GUER Cedex
{olivier.bartheye,eric.jacopin}@st-cyr.terre.defense.gouv.fr

An awarded claim While the Pengi paper (AGRE & CHAPMAN 1987) received a Classic Paper award at AAAI'2006 (News 2006), to our knowledge we have yet to see whether its main claim on classical planning is true (AGRE & CHAPMAN 1987, page 269): that "*a traditional problem solver for the Pengo domain [could not cope] with the hundreds or thousands of such representations as* (AT BLOCKS-213 427 991), (IS-A BLOCK-213 BLOCK), *and* (NEXT-TO BLOCK-213 BEE-23)". Or, stated differently (AGRE & CHAPMAN 1987, page 272): "*[The Pengo domain] is one in which events move so quickly that little or no planning is possible, and yet in which human experts can do very well.*"

The Pengo domain is that of a video-game of the eighties where a player navigates a penguin around a two dimensional maze of pushable ice blocks. The player must collect diamonds distributed across the maze while avoiding to get killed by bees; but the player can push an ice block which kills a bee if it slides into it.

The Pengi system described in the Pengi paper (AGRE & CHAPMAN 1987) is a video-game playing system which just happens to fight bees in the Pengo game. Pengi first searches for the penguin on the screen to register its initial position. Then searches for the most dangerous bee, an appropriate weapon to kill that bee (that is, an ice block) and then navigate the penguin towards that weapon to kick it. Both written in Lisp, the Pengo game and the Pengi system are in fact the same Lisp program: the search for the penguin and the most dangerous bee can be made directly by looking at the Lisp data structures. According to the on-going conditions of the game, various pieces of code are activated (for instance, you may wish to push an ice block several times before it becomes a weapon). We refer the reader to the Pengi paper for further information on the Pengi system. Finally, "*[Pengi] plays Pengo badly, in near real time. It can maneuver behind blocks to use as projectiles and kick them at bees and can run from bees which are chasing it*" (AGRE & CHAPMAN 1987, page 272).

Interpreted as a finite state machine, the Pengi system can easily be re-implemented and not only fight bees not badly but also collect diamonds even in non trivial mazes (DROGOUL, FERBER, & JACOPIN 1991).

The awarded claim eventually is about space and time complexity in the Pengo domain and of classical planning algorithms around 1987. But since 1987, processors are several hundred times faster and fastest classical planners are able to produce plans with hundreds of actions in a matter of seconds for certain problems. Consequently, we thought it would be interesting and, most surely, fun, to see how the current technology could cope with an 1980s video-game.

We here report on our very first steps towards the evaluation of the claim about classical planning.

Classical planning, really? As a testbed, we chose Iceblox (BARTLETT, SIMKIN, & STRANC 1996, pages 264–268), a slightly different version of the Pengo game for which there exists an open and widely available java implementation (HORNELL 1996). For instance (cosmetic differences): flames, and not bees, are chasing the penguin-player who must now collect coins, and not diamonds. Moreover (different actions), coins must be extracted from ice blocks. Extraction means kicking seven time at an ice block to destroy the ice and thus making the coin ready for collection. Such an ice block with a coin inside slides as well as any other ice block. So the player must kick in a direction where the ice block cannot slide (e.g. against the edge of the game) in order to extract an iced coin.

Instead of designing a new planning system, we decided to pick up an existing one, and eventually several, in order to compare their relative performance if they had any ability at playing Iceblox. We consequently decided to re-implement Iceblox in Flash (Adobe 2007). Not only would we provide a new implementation of the game, but also could we use the plug-in architecture of the Flash runtime: a call and return mechanism can run (and pass in and out parameters to) any external piece of executable code when put in the appropriate directory.

This deviates from the original Pengi system which was the same Lisp program as the Pengo game (and also deviates from (DROGOUL, FERBER, & JACOPIN 1991) where everything was implemented in the same SmallTalk program), but would eventually ease the comparison as classical planners are not necessarily written in Flash.

However, this dramatically changes the setting of the problem.

*Special thanks to Maria FOX, Jörg HOFFMANN, Jana KOEHLER and Derek LONG about the gripper domain.

On one side, a classical planner becomes an external component which happens to provide a planning functionality: fine, that's how we want it to work.

On the other side, the world view of the Pengi paper (AGRE 1993) is that of the dynamics of everyday life (AGRE 1988) (plans do exist, but are better communicated[1] to people than built from scratch) and thus is opposed to the heavily intentional (BRATMAN 1987; MILLER, GALANTER, & PRIBRAM 1986) world view of planning.

In other words: the Pengi system is always in charge of the actions (moving the penguin, kicking ice blocks) whereas an external component is in charge only when activated and is harmless otherwise: a player must be able to play Iceblox when the planning component is not activated or no component is plugged-in. This generates supplementary questions: when is classical planning activated and for how long? One more constraint. To respect the dynamics of the domain of video-games, Iceblox must never stop and must run while the classical planning component is planning: flames keep on chasing the penguin and sliding ice blocks keep on sliding.

Consequently, the classical planning component is activated when the player presses the "*p*" key. This activation is ended as soon as the player presses the keyboard again: the arrow keys to move the penguin up, right, down and left; and the space key to kick an ice block.

Hopefully, an anonymous classical planner shall build a plan and return it to Iceblox. What shall Iceblox do with this plan? Please, note that this question does not immediately entail further questions of interleaving classical planning and execution (AMBROS-INGERSON & STEEL 1988). To begin with, there is a matter of level of detail: actions in Iceblox corresponds to keys pressed by the player. Is the classical planning component really expected to build plans with such actions?

Hints from a gripper video game On one hand, the classical planning component is expected to build plans with keys pressed. First because it seems part of the claim: if the classical planning component (that is, the "*traditional problem solver*" of the claim) has to cope "with hundreds or thousands" of detailed representations describing the initial and final situations, then we can expect action representations to be as detailed as the initial and final situations. However, the Pengi literature (AGRE & CHAPMAN 1987; AGRE 1988; 1993; 1997; CHAPMAN 1990) says nothing about this.

On the other hand, classical planners are used to cope with high-level action description. For instance, here is the classical planning Move operator from the well-known gripper (FOX & LONG 1999) domain:

$$\text{Move(X,Y)} = \begin{cases} \textbf{Preconditions} & : & \{\text{at-robby(X)}\} \\ \textbf{Additions} & : & \{\text{at-robby(Y)}\} \\ \textbf{Deletions} & : & \{\text{at-robby(X)}\} \end{cases}$$

[1]Official player's guides are good sources of plans communicated to video-game players that would otherwise take some time to build.

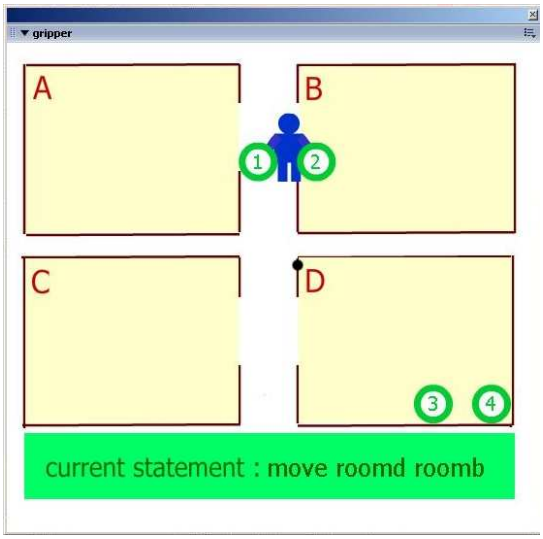

Figure 1: An anonymous classical planning system has built (actually, it's FF, plugged-in our Flash application as described earlier; but let's say we didn't tell you) a plan for the following the gripper video game problem: 4 balls must be moved from room B to room D. The on-going action (from the plan) is printed in the green area at the bottom of the window: robby is moving from room D to room B; details of the navigation (and of the picking up and down of balls) are left to the Flash application.

In the gripper domain robby-the-robot uses its arms to move balls from one room, along a corridor, to another. Neither bees nor flames prevent robby-the-robot from succeeding in transporting balls from one room to another. It is nevertheless easy to come up with a simplistic two dimensional gripper video-game: your task is to move as fast as possible a set of balls from their initial location to their final location (see Figure 1).

As stupid as this may sound, this gripper video-game isn't too far from, say, the popular Sokoban video-game (in a maze, blocks must be slided from one place to another, with no time limit) (CHARRIER 2007). In such a puzzle, the details of the block pushing activity are important: e.g. a wrong push at a corner can make the problem unsolvable. But more important is the block you push next, which sequences the player's next Move. Similar Iceblox situations where the player only needs to navigate towards iced coins and then extract them do exist (See Figure 2).

Here are two operators which can combine into a plan and solve the simple situation of Figure 2: first MoveToCoin(6,4), then Extract(6,4).

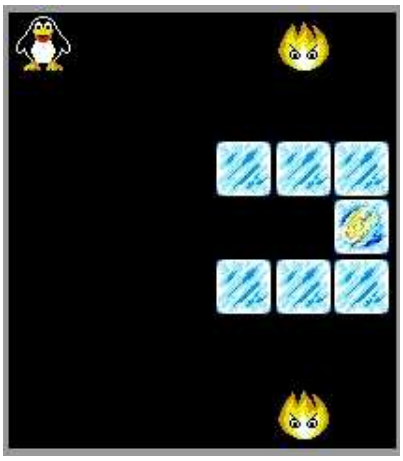

Figure 2: A simplistic level in the Iceblox domain: move to the ice block containing a coin and then extract it. Details of the navigation, as far as possible from the flames, and of the extraction of the coin (seven kicks to the ice block) are again left to the Flash application.

$$\text{MoveToCoin(X,Y)} = \begin{cases} \textbf{Preconditions :} & \{\text{at(X,Z)}\} \\ \textbf{Additions :} & \{\text{at-coin(X,Y)}\} \\ \textbf{Deletions :} & \{\text{at(X,Z)}\} \end{cases}$$

$$\text{Extract(X,Y)} = \begin{cases} \textbf{Preconditions :} & \{\text{at-coin(X,Y)}, \\ & \quad \text{iced-coin(X,Y)}\} \\ \textbf{Additions :} & \{\text{at(X,Y)}, \\ & \quad \text{extracted(X,Y)}\} \\ \textbf{Deletions :} & \{\text{at-coin(X,Y)}, \\ & \quad \text{iced-coin(X,Y)}\} \end{cases}$$

Since we have neither implemented flame-fighting nor fleeing operators, flames must be un-aggressive so that the coin of Figure 2 can be extracted. And because of the simple path from the Penguin to the coin, the initial and final situations are simply described: {at(1,1), iced-coin(6,4)} and {extracted(6,4)}, respectively.

We won't discuss this extremely low number of formulas needed to describe what could be called a *minimal* Iceblox problem: up to now, the biggest part of our work has been devoted to stay as close as possible to the spirit of classical planning and video-games, while designing a satisfying testbed. In the future, we hope to concentrate more on designing classical planning predicates and operators in order to cope with more complex Iceblox situations.

References

Adobe. 2007. Flash. `http://www.adobe.com/`.

News, A. 2006. Classic paper award. *AI Magazine* ***27(3)*** 4.

AGRE, P., and CHAPMAN, D. 1987. Pengi: An implementation of a theory of activity. In *Proceedings of* 6^{th} *AAAI*, 268–272.

AGRE, P. 1988. *The Dynamics of Everyday life*. Ph.D. Dissertation, MIT AI Lab Tech Report 1085.

AGRE, P. 1993. The symbolic worldview: Reply to vera and simon. *Cognitive Science* ***17(1)*** 61–69.

AGRE, P. 1997. *Computation and Human Experience*. Cambridge University Press.

AMBROS-INGERSON, J., and STEEL, S. 1988. Integrating planning, execution and monitoring. In *Proceedings of* 7^{th} *AAAI*, 83–88.

BARTLETT, N.; SIMKIN, S.; and STRANC, C. 1996. *Java Game Programming*. Coriolis Group Books.

BRATMAN, M. 1987. *Intentions, Plans and Practical Reason*. Harvard University Press.

CHAPMAN, D. 1990. *Vision, Instruction and Action*. Ph.D. Dissertation, MIT AI Lab Tech Report 1204.

CHARRIER, D. 2007. Super sokoban 2.0. `http://d.-ch.free.fr/`.

DROGOUL, A.; FERBER, J.; and JACOPIN, E. 1991. Viewing cognitive modelling as eco-problem solving: The PENGI experience. In *Proceedings of the 1991 European Conference on Modelling and Simulation Multiconference*, 337–342.

FOX, M., and LONG, D. 1999. The detection and exploitation of symmetry in planning problems. In *Proceedings of* 16^{th} *IJCAI*, 956–961.

HORNELL, K. 1996. Iceblox. `http://www.tdb.uu.-se/~karl`.

MILLER, G.; GALANTER, E.; and PRIBRAM, K. 1986. *Plans and the Structure of Behavior*. Adams-Bannister-Cox.