Planning for an AI based virtual agents game*

Susana Fernández, Roberto Adarve, Miguel Pérez, Martín Rybarczyk and **Daniel Borrajo**
Departamento de Informática, Universidad Carlos III de Madrid
Avda. de la Universidad, 30. Leganés (Madrid). Spain†

Abstract

Computer games have become a big software industry. From the early days of the use of AI to solve classical games, such as chess or checkers, we are now seeing an intensive use of graphics to attract gamers. Currently, AI in games, normally refers to either the designing the behaviour of stock AI agents, like Bots (automated player characters), or refers to custom AI agents such as Non-Player Characters (NPCs). We present here our on-going work on building a game, AI-LIVE, that is oriented towards the intensive use of AI controlled Bots. The game borrows the idea from the popular THE SIMS, but with a strong focus on building characters based on different AI techniques. More particularly, we present the work on applying planning techniques for building one such agent.

Introduction

Games has always been a challenging domain for testing AI techniques. In the beginning of AI the focus was on classical games, such as chess (Newell, Simon, & Shaw 1972) or checkers (Samuel 1963). The type of AI-based techniques that were used for solving these games were mainly search and, sometimes, machine learning. Then, in the 80's and 90's the work on these classical games continued by intensive use of faster machines with more memory, such as the work on Deep Blue (Hsu *et al.* 1990) or CHINOOK (Schaeffer *et al.* 1996). Recently, video games have produced a renewed interest from the AI community on applying its techniques into games. They normally refer to automated players (Bots), either opponents or teammates, and NPCs. Bots are agents that act as if controlled by a human player. They are stock AI characters that will follow designed behaviours. Non-Player Characters are any artificial agent that is not a player, like Monsters that act only as an enemy to all players. A notable example of AI in games is the project called FEAR, which stands for Flexible Embodied Animat aRchitecture.[1] This is a framework for creating AI controlled systems for synthetic characters. The project includes reusable AI components, a portable framework, and interfaces to realtime 3D games. A similar system is Arianne, although it is not designed specifically for AI.[2] Arianne is a multiplayer online engine to develop turn based and real time games providing a simple way of creating the game server rules and game clients. There have also been several tasks in games that have been solved using a variety of AI techniques. Examples are production systems for QUAKE (van Lent *et al.* 1999), planning for bridge (Smith, Nau, & Throop 1998) or Real Time Strategy games (Chung, Buro, & Shaeffer 2005) and in Full Spectrum Command,[3] or genetic approaches in Blondie24 (Fogel 2001) (see (Rabin 2002) for some reported work). Perhaps, the most used technique has been different versions of the A* algorithm for path-planning purposes.

The idea inspiring AI-LIVE has been the commercial game THE SIMS[4] and how it could be generalised and modularised so that AI clients could be designed to play a game together with humans. Our goal is to build an architecture similar to the one proposed in (Buro & Furtak 2004) for Real Time Strategy games. Each AI client is developed as an architecture using one AI technique. So far, we have built a rule-based AI client and a planner one in AI-LIVE. However, it could be augmented to incorporate any other planner or AI technique. Furthermore, all agents can interact, leading to a more complex system that integrates social and psychological models in order to obtain believable emergent behaviours, as the work by Silverman (Pelechano *et al.* 2005).

*This work has been partially supported by the Spanish MCyT under project TIC2002-04146-C05-05, MEC project TIN2005-08945-C06-05, and CAM-UC3M project UC3M-INF-05-016.

†The use of planning for this domain appeared in the discussions carried out within the Artificial Intelligence Standards Committee of the International Game Developers Association, Working Group on Goal Oriented Planning. We would like to thank its members.

[1]http://sourceforge.net/projects/fear

[2]https://sourceforge.net/projects/arianne

[3]http://www.ict.usc.edu/content/view/56/108

[4]http://en.wikipedia.org/wiki/The_Sims

AI-LIVE architecture

AI-LIVE is a client/server application running over TCP/IP. It works similarly to modern online games such as Ragnarok Online,[5] World of Warcraft[6] or Guild Wars[7], where various users connect to a central server to play in a shared world , with a key difference: in our case, human clients share the game with AI controlled characters using a variety of AI-oriented techniques, playing in the same world. At this point in development, a basic universe has been implemented consisting of a simple room with objects to pick up, together with two different AI clients and a graphics renderer. Figure 1 shows a high level view of the architecture.

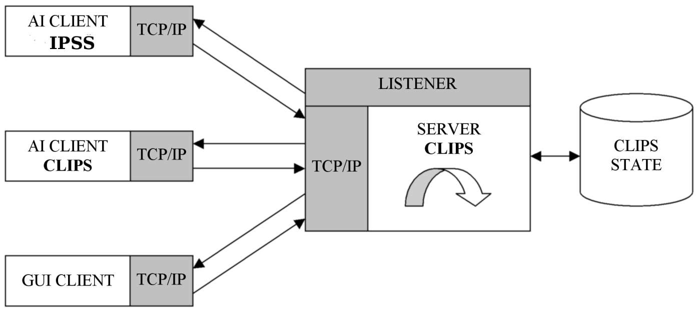

Figure 1: High level view of AI-LIVE architecture.

The server holds the state of the universe, which is divided in separate stages or realms where clients play. Each stage is made of objects in a cell-based 3D space. These objects are instances of classes from an ontology that is shared among all clients. In the future, clients will be able to travel from one stage to another, as players do in online games.

AI clients connect to the server to control an actor object in a specific stage each, while GUI clients connect to the server to open a window and display a graphical representation of a particular stage. In the future, GUI-based clients will allow human players to play and interact with AI clients.

The greyed application modules in Figure 1 are responsible for communication between both ends. All these modules are written in C. The server intelligence is done using the CLIPS tool for building knowledge-based systems.

After the server is initialised, it listens for incoming connections and runs a round-robin loop over the list of connected clients (we will define in the future asynchronous behaviour):

- Each AI client receives the CLIPS state corresponding to its stage from the AI server and gets a turn with unlimited time to decide on what to do. All other AI clients are paused as this happens. When the AI client is done, the server receives an action, which is passed to the AI engine and executed.
- Clients get all their turns during the game in strict order of connection.
- All GUI clients receive the CLIPS state corresponding to their stages from the AI servers in every turn the server executes, together with each AI client.

[5]http://en.wikipedia.org/wiki/Ragnarok_Online

[6]http://www.worldofwarcraft.com

[7]http://www.guildwars.com/

Clients connect to the server using a simple binary network protocol with little overhead, and at this stage of the project, they synchronise with the server simply by waiting for incoming data. Currently, there are two AI clients implemented, using CLIPS and IPSS (Rodríguez-Moreno *et al.* 2004) (an integrated planner and scheduler based on PRODIGY (Veloso *et al.* 1995)) respectively, as well as a 2D GUI client.

Given that we would like AI-LIVE to grow in the future and in order to be flexible, we have defined an ontology that is shared by all modules, that is described in the next subsection. Next subsections describe in more detail the ontology, the server and the CLIPS clients, while next section describes the IPSS clients.

The ontology

Figure 2 shows AI-LIVE class model. All objects in AI-LIVE universe are instances of one of these classes. A set of basic physical properties define all entities, and actors add personality properties and relationships with other entities (actors or not).

The main classes in the ontology are:

- `Stage`: it represents the different stages where the game can take place.
- `Entity`: abstract class to represent any possible entity in the stage. An stage is a collection of entities. There are four entity subclasses:
  - `Actor`: it represents game actors together with their personality and emotions. Now, we are not reasoning about this type of knowledge, but we would like to focus on it in the next future.
  - `Wall`: it represents walls that cannot be traversed.
  - `Object`: it represents any general object.
  - `ContentCapability`: it represents objects with capacity properties as a `Container` or an `Actor`.
- `Cell`: it represents the atomic space portions inside a stage.
- `Relationship`: it represents relationships among actors and objects.
- `ClientAction`: each action supported in the game has an associated class. So far, they are `MoveAction`, `PickUpAction`, `PutDownAction` and `AddClient`.

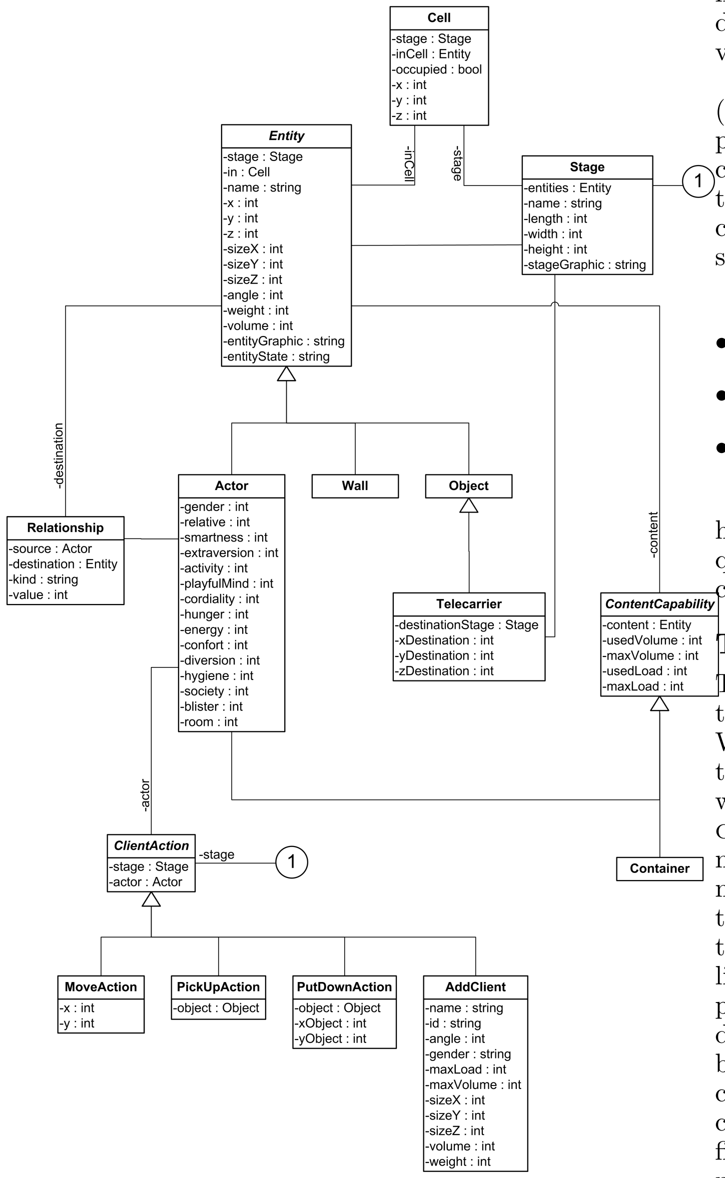

Figure 2: AI-LIVE ontology.

The server

The server is the central part of the game. It holds the state of the game with all of its defined stages and objects. It is in charge of maintaining a list of connected clients to serve states to and receive actions from in a round-robin basis. These actions are verified and executed against the state, producing a new state for the next client. The server is written in C language embedding the CLIPS production system to control the state, verify and execute rules.

When a client of any kind connects to the server (which is listening for incoming connections), both parts will identify. On success, the server will add the client to the loop. In this loop, all AI clients get one turn in strict order of connection, while GUI clients receive a copy of the state in every turn, but they do not send an action.

The actions currently supported by AI-LIVE are:

- Move an actor from one position to another
- Pick up an object as requested by an actor
- Put down an object as requested by an actor

To execute these actions, the CLIPS code in the server has a series of rules, that check for validity of the requested operations and alter the state. As the state changes, clients actions change.

The CLIPS client

The CLIPS client is an AI-LIVE client implemented using the CLIPS tool for building knowledge-based systems. We have integrated CLIPS with the rest of client code that handles, as in the case of the IPSS client, all networking operations. At this stage of the project, each CLIPS client playing the game will have to pick up as many objects as possible, considering its actor maximum capacity. As in the case of the IPSS clients, setting up the goals of the AI characters is, perhaps, one of the most interesting parts of the project that we would like to work on. Given that IPSS is a backward chaining planner and CLIPS works in forward mode, there will be differences in how goals will be defined and pursued in both types of clients. In the case of CLIPS clients, they can afford having a more reactive behaviour, while IPSS clients will have to be goal oriented (in its simplest configuration). We are also considering adding other planners that might have a different behaviour, introducing new ways of looking at this problem.

To decide on which action to run, the CLIPS client will use the stage received from the server as state, matching every object and property (such as class, position, size, angle, weight, bulk...) with conditions of rules.

The GUI client

The first graphical client for AI-LIVE is a conventional 2D sprite-based renderer named CREND. It is modelled after successful 2D engines still in use by some of today video-games, and is written from scratch in pure C, using the portable SDL low-level multimedia framework for graphics.

CREND connects to an AI-LIVE server and obtains the state for a stage as an AI client would, but it does not send any action. Instead, it draws a graphical representation of the stage in a window where users can see what is happening in the server. Objects are drawn in a particular parallel perspective derived from traditional 2D scrolling, and used by most 2D role-playing games such as Tales of Phantasia.

The IPSS client

The IPSS client is the one that uses planning technology for deciding at each step which action is best to execute. It uses the IPSS system that integrates planning and scheduling (Rodríguez-Moreno *et al.* 2004). One can use any other planner for implementing this planning step, though there some features of IPSS that have been useful for building these clients (some of them will be described later). The client is divided in two modules:

- Main module: its tasks are to connect to the server, deal with the network, and invoke the AI module. This module is in charge of the low-level networking, receiving each state and interacting with the AI module to obtain an action, which is then forwarded to the server. It is written in C. The client can be configured with the following parameters: the server address and port; the stage the client is to play in; and the controlled actor profile.

- AI module: its main task is to decide the actor actions. The planning tool chosen for this task is the domain-independent planner IPSS. It is used to find a plan for the AI-LIVE domain and problem supplied by the main module. The main module receives the state with all the objects for the current stage from the server and parses it, translating it into IPSS description language (quite similar to PDDL2.1 (Fox & Long 2002)). The parsed state and goals form the input problem to the planner. The client domain is defined as a set of operators designed to be counterparts of the set of implemented actions the server can execute. At this stage of the project, the AI-LIVE actions are supported as operators: move, pick-up and put-down. These two last operators use the individual capacity constrains of characters. Then, IPSS provides a potential plan to achieve the goals from the current state, and outputs only the first operator in the plan. This operator is translated back to AI-LIVE actions scheme, and returned to the server.

Now, we will comment on two specific issues that have to be considered for applying planning to game playing in general and how we have solved them within AI-LIVE: how goals are generated, and how to solve efficiently the problem of selecting paths to go from one place to another in a given map, how to integrate path planning with task planning.

The first issue concerns selecting a goal to work on. Currently, AI-LIVE selects to maximize the number of objects taken by the actor, so goals consist on having the actor the objects that are in the room. Traditional approaches to planning assume that goals are given as input to the planners. However, we believe that, from a planning perspective, setting up the goal of an actor in this type of domains is precisely one of the key challenges of using AI planning here. Therefore, we want to study different types of goals generation schemes for this type of games. A related problem is the oversubscription in planning that occurs when agents do not have enough resources to achieve all of their goals. This requires finding plans that satisfy only a subset of them (van den Briel, Sánchez, & Kambhampati 2004).

The second issue relates to the use of grids/maps in planning domains (that appear in most games), as it is the case of AI-LIVE. For many planners, reasoning about how to go from one place to another can easily make planning intractable, as it is the case of IPSS. This is specially true if we want to optimize the cost of the path to go from one place to another according to a quality measure. There are many articles in AI games about applying heuristics to path planning. However, planning domains pose the added problem of path planning integration with the operator definitions. For example, to define an action for moving a synthetic character is necessary to know how the character can move in the world. We have used two approaches. The first approach, which is based on a careful manual knowledge engineering of the domain, exploits one of IPSS main features: the support for user-defined heuristics, to efficiently guide the search. These domain-dependent heuristics are defined as control rules (if-then structures), that help the planner taking directions, checking adjacency, and deciding on a position for an actor to pick up an object from the stage. Figure 3 shows an example of one of these rules for selecting bindings for the `move` operator. Suppose that the current planner goal is to have an actor on a `goal-cell` (the one that has an object which the actor wants to pick up, with coordinates `(x,y)`). If the actor is currently at another cell, this rule selects the best adjacent cell to the `goal-cell`, `origin-cell` (coordinates `(x1,y1)`), to which the actor should move first. This decision is needed given that IPSS is a backward chaining planner. The meta-predicate `(adjacent-p x y x1 y1)` is true if position `(x,y)` is adjacent to position `(x1,y1)`. The meta-predicate `(best-cell-p x y x1 y1 x2 y2)` is true if `(x1,y1)` is the best adjacent position for reaching `(x,y)` starting from `(x2,y2)` (where the actor is). Repeated use of this control rule guides the actor within the map directly from its initial position to the goal cell.

This approach for solving the path-planning problem within the task planning requires to define by hand the appropriate control rules. Obviously, this depends on the user that defines the right knowledge. The second approach integrates the task planner (IPSS) and a

```
(control-rule select-cell-for-MOVE
  (if (and (current-goal (cell-inCell <goal-cell> <actor_id>))
           (current-operator move)
           (true-in-state (cell-x <goal-cell> <x>))
           (true-in-state (cell-y <goal-cell> <y>))
           (true-in-state (cell-x <origin-cell> <x1>))
           (true-in-state (cell-y <origin-cell> <y1>))
           (adjacent-p <x> <y> <x1> <y1>)
           (true-in-state (cell-inCell <actor-cell> <actor_id>))
           (true-in-state (cell-x <actor-cell> <x2>))
           (true-in-state (cell-y <actor-cell> <y2>))
           (best-cell-p <x> <y> <x1> <y1> <x2> <y2>)
           (or (true-in-state (cell-occupied <origin-cell> false))
               (true-in-state (cell-inCell <origin-cell> <actor_id>)))))
  (then select bindings ((<cell_id> . <origin-cell>))))
```

Figure 3: Example of a hand crafted control rule for selecting bindings for the `move` operator.

path planner (a standard implementation of the A* algorithm), by interleaving their execution following the ideas in (Fox & Long 2001). When IPSS needs to find a path during the search for the task planning solution, it calls the path planner. If there is a path between the current position of the actor and the goal position, the path planner returns a solution (together with all its associated quality metrics), that can use that information while solving the problem.

In relation to the integration of task and path planning, IPSS has two useful features:

- Functions can be called within the definition of variables on operators. If we want to know whether there is a solution and its quality between two nodes of a path-planning problem, we can define a variable `distance` in the preconditions of the `move` operator whose value is the result of calling the path planner function.
- Different cost metrics can be defined at each operator. In this paper, we are mainly interested in the distance quality metric, but both planners (task and path planners) can also use other quality metrics, and obtain *good* solutions according to them.

For describing the overall planning problem we need two separate files. A problem file for the task planner, in which we have abstracted the information on the map/path graph, and a problem file for the path planner with information about the map/path graph. Nodes of the path graph will also appear in the task planner problem file so that there is a connection between these two processes. In the next subsections we describe this process in more detail.

Path planner

The input to the path planner is a path-planning problem composed of an initial node, a final node, a quality metric, and a graph. In order to automatically specify the graph for each problem given to the IPSS client, we extract it from the problem definition. Given that we wanted the approach to be as domain independent as possible, for each domain we only need to specify the problem predicates from which the system will create the graph nodes and the predicates from which the system will create the graph arcs. Each arc can have a set of quality metrics defined. In our experiments, we have only used one: distance. But in other applications, such as planning tourist visits in the SAMAP architecture (Arias, Sebastiá, & Borrajo 2005), we have used others such as price (cost of the transportation method), utility (a subjective value that can represent user/agent preferences such as *I prefer to use the bus when possible*), distance, and time. Then, when calling the path planner we can specify the cost metric to be minimized.[8] The output of the path planner is a list of move actions in the form of: initial node, final node, and the values of the cost metrics for that arc (distance).

Given that the task planner can call the path-planner for solving the same path planning problem many times during the search (due to symmetries in the task planning search tree), the path-planner provides a caching mechanism. Every solution for the path planning problems is stored the first time. Then, in case of solving the same path planning problem is needed, it retrieves the previous solution. This assumes that the graph does not change between two calls to the path-planner. In case it can change, such as domains in which agents can act on the arcs, the nodes, or their quality metric features, then the caching mechanism will not be useful. For instance, if the actor leaves objects in the floor when building a plan, it creates obstacles in the graph. Therefore, a previous solution to go from one place to another in a previous call of the path planner can be

[8] In the case of maximizing cost metrics, such as utility, we can always convert them to minimization problems.

made invalid.

This approach can be applied to many other domains, such as DRIVERLOG, SOKOBAN, ... The DRIVERLOG domain, for instance, has a graph for drivers and a graph for trucks. So, our implementation of the path planner admits a set of graphs to be defined. Then, for each domain operator, the appropriate path-planning problem file will be selected.

Task planner

IPSS uses a backward search to solve planning problems. The types of decisions that it makes during search are: goal to work on, operator name that can achieve the selected goal, bindings for that operator, and decide whether to continue subgoaling or execute an applicable operator. In order to integrate it with the path planner, we redefined the `move` operator as specified in Figure 4.

Variables appear between brackets. The preconds section defines first the variables that are used in the operator, as well as the operator preconditions. The effects section defines other variables that are used only in the effects, as well as the postconditions. Finally, the cost section defines new variables only used for the costs computation, and the operator costs depending on the quality metric used by the task planner when solving a specific problem. In this case, we have defined one quality metric: steps (distance). The `gen-from-pred` function accesses the current state to provide values for variables. For instance, the room (stage) where the actor currently is, or the position it is on.

The connection with the path planner is done through the `path-planning-distance` function whose input parameters are the initial node and end node, that will also appear in the path-planner graph description. It returns `false` in case of no solution, or a numeric value with the total distance to go from the initial node to the final node. This value is assigned to the <steps> variable.

Conclusions

We have presented the first steps of building AI-LIVE, a game inspired in the popular game THE SIMS. We have defined and built an architecture based on a server and three types of clients: a rule-based one, a planning-based one, and a GUI. From a planning perspective, we have defined and provided initial solutions to two planning problems related to this type of domains: goal selection and integration with path-planning. Both problems are common to the application of planning technology to many games.

Currently, we are dealing with a simplified domain in terms of actions covered, though we have defined the architecture to be easily augmented with many more actions. So, we are using an ontology that can cope with the knowledge needed for many of the potential new actions. For instance, in the next future, AI-LIVE is expected to shift towards the social relationships among actors and the reasoning about psychological aspects, that have already been considered in the ontology. We also want to change our 2D cell-based world representation to a full 3D with dynamic physical objects.

References

Arias, J.D.; Sebastiá, L.; and Borrajo, D. 2005. Using ontologies for planning tourist visits. In *Working notes of the ICAPS'05 Workshop on Role of Ontologies in Planning and Scheduling*, 52–59. Monterey, CA (EEUU): AAAI.

Buro, M., and Furtak, T. 2004. RTS games and real-time AI research. In *Proceedings of the Behavior Representation in Modeling and Simulation Conference (BRIMS)*, 51–58.

Chung, M.; Buro, M.; and Shaeffer, J. 2005. Monte carlo planning in RTS games. In Kendall, G., and Lucas, S., eds., *Proceedings of the IEEE Symposium on Computational Intelligence and Games (CIG'05)*, 117–124. Essex (UK): IEEE.

Fogel, D.B. 2001. *Blondie24: Playing at the Edge of AI.* Morgan Kaufmann.

Fox, M., and Long, D. 2001. Hybrid stan: Identifying and managing combinatorial optimisation subproblems in planning. In *Proceedings of IJCAI'01.*

Fox, M., and Long, D. 2002. *PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains.* University of Durham, Durham (UK).

Hsu, F.; Anantharaman, T.; Campbell, M.; and Nowatzyk, A. 1990. *Computers, Chess, and Cognition.* Springer. chapter Deep Thought, 55–78.

Newell, A.; Simon, H.A.; and Shaw, J. 1972. *Human Problem Solving.* Englewood Cliffs, NJ: Prentice-Hall.

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

Rodríguez-Moreno, M.D.; Oddi, A.; Borrajo, D.; Cesta, A.; and Meziat, D. 2004. IPSS: A hybrid reasoner for planning and scheduling. In de Mántaras, R.L., and Saitta, L., eds., *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI 2004)*, 1065–1066. Valencia (Spain): IOS Press.

Samuel, A. 1963. Some studies in machine learning using the game of checkers. In Feigenbaum, E., and Feldman, J., eds., *Computers and Thought.* New York, NY: McGraw-Hill.

Schaeffer, J.; Lake, R.; Lu, P.; and Bryant, M. 1996. Chinook, the world man-machine checkers champion. *AI Magazine* 17(1):21–29.

Smith, S. J.; Nau, D.S.; and Throop, T.A. 1998. Computer bridge - a big win for AI planning. *AI Magazine* 19(2):93–106.

van Lent, M.; Laird, J.E.; Buckman, J.; Hartford, J.; Houchard, S.; Steinkraus, K.; and Tedrake, R. 1999. Intelligent agents in computer games. In *AAAI/IAAI*, 929–930.

```
(OPERATOR move
   (params <actor_id> <cell_id> <destination_cell_id>)
   (preconds
     ((<actor_id> ACTOR)
      (<stage_id> (and STAGE (gen-from-pred (stage-entities <stage_id> <actor_id>))))
      (<cell_id> (and CELL (gen-from-pred (cell-inCell <cell_id> <actor_id>))))
      (<destination_cell_id> (and CELL (diff <cell_id> <destination_cell_id>)))
      (<steps> (and STEPS (path-planning-distance <cell_id> <destination_cell_id> <steps>)))
      (<xx> (and COORDINATE (gen-from-pred (cell-x <destination_cell_id> <xx>))))
      (<yy> (and COORDINATE (gen-from-pred (cell-y <destination_cell_id> <yy>))))
      (<x> (and COORDINATE (gen-from-pred (cell-x <cell_id> <x>))))
      (<y> (and COORDINATE (gen-from-pred (cell-y <cell_id> <y>)))))
     (and (cell-inCell <cell_id> <actor_id>)
          (cell-occupied <destination_cell_id> false)))
    (effects ()
     ((del (cell-inCell <cell_id> <actor_id>))
      (del (cell-occupied <destination_cell_id> false))
      (del (cell-occupied <cell_id> true))
      (del (actor-x <actor_id> <x>))
      (del (actor-y <actor_id> <y>))
      (add (cell-inCell <destination_cell_id> <actor_id>))
      (add (cell-occupied <destination_cell_id> true))
      (add (cell-occupied <cell_id> false))
      (add (actor-x <actor_id> <xx>))
      (add (actor-y <actor_id> <yy>))))
    (costs ()
      ((steps <steps>))))
```

Figure 4: Example of the `move` operator for the AI-LIVE domain in the IPSS language.

Veloso, M.; Carbonell, J.; Pérez, A.; Borrajo, D.; Fink, E.; and Blythe, J. 1995. Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental and Theoretical AI* 7:81–120.

Champandard, A.J. 2003. *Synthetic Creatures with Learning and Reactive Behaviors.* New Riders Games.

Pelechano, N.; O'Brien, K.; Silverman, B.; and Badler, N. 2005. Crowd simulation incorporating agent psychological models, roles and communication. In *First International Workshop on Crowd Simulation (V-CROWDS '05).*

van den Briel, M.; Sánchez, R.; and Kambhampati, S. 2004. Over-Subcription in Planning: A Partial Satisfaction Problem. In *ICAPS Workshop on Integrating Planning into Scheduling.*