

Planning for Reactive Behaviors in Hide and Seek¹

Michael B. Moore, Christopher Geib, Barry D. Reich

University of Pennsylvania
Department of Computer and Information Science
200 S. 33rd Street, Philadelphia, PA 19104-6389

Abstract

We describe the ZAROFF system, a plan-based controller for the players in a game of hide and seek. The system features visually realistic human figure animation including realistic human locomotion. We discuss the planner's interaction with a changing environment to which it has only limited perceptual access. A hierarchical planner translates the game's goals of finding hiding players into locomotion goals, assisted by a special-purpose search planner. We describe a system of parallel finite state machines for controlling the player's locomotion. Neither path-planning nor explicit instructions are used to drive locomotion; agent control and apparent complexity are the result of the interaction of a few relatively simple behaviors with a complex (and changing) environment.

¹This research is partially supported by ARO DAAL03-89-C-0031 including U.S. Army Research Laboratory; ARPA AASERT DAAH04-94-G-0362; DMSO DAAH04-94-G-0402; ARPA DAMD17-94-J-4486; U.S. Air Force DEPTH through Hughes Missile Systems F33615-91-C-0001; DMSO through the University of Iowa; and NSF CISE CDA88-22719.

1 Introduction

The game of *hide and seek* challenges the ability of players to plan for acquiring information and to react quickly to what they see. The player who is “it” (the *seeker*) must explore his environment attempting to locate other players (*hidiers*). Those players must select hiding places which are difficult to discover while providing access for them to run safely to home base when the way is clear. The goal of this work is to develop simulated agents that can play hide and seek (or more dangerous games) (Moore, Geib, & Reich 1995).

We describe a planning system for a player (which can change roles between hider and seeker during the game) and its vertical integration into a system called ZAROFF that selects reactive behaviors to execute in an animated simulation. Operation of the planner is interleaved with execution of the reactive behaviors so that the agent may adapt to a dynamic environment.

The software chosen for this work is *Jack*[®] (Badler, Phillips, & Webber 1993) running on Silicon Graphics workstations. *Jack* is a human modeling and simulation program developed at the Center for Human Modeling and Simulation at the University of Pennsylvania, that features visually realistic human locomotion based on both kinematic and dynamic techniques (Ko 1994; Ko *et al.* 1994). *Jack*'s LISP application programming interface (Becket 1994) was used to implement ZAROFF. This interface supports access to the environment (a database) and its behavioral simulation system.

2 Generating Behaviors

Human locomotion is performed by the Behavioral Simulation System (BSS) (Badler, Phillips, & Webber 1993; Becket & Badler 1993). ZAROFF controls this locomotion indirectly by binding

behaviors to human figures in the environment database. BSS, which constantly monitors the environment, immediately initiates locomotion based on the agent's bound behaviors.

A player utilizes a set of behaviors in interacting with its environment. ZAROFF includes the following behaviors: attraction, avoidance, field-of-view (to avoid areas visible to the seeker), path following (to draw an agent to a path) and chasing.

3 Action Execution

The Action Execution module is responsible for the control of all actions occurring in ZAROFF. Most actions such as opening and closing doors are performed directly by this module. (As noted earlier, human locomotion is performed by the Behavioral Simulation System (BSS) (Badler, Phillips, & Webber 1993; Becket & Badler 1993), so is controlled only indirectly by Action Execution.)

Non-locomotion actions are performed directly by Action Execution manipulating the environment. For example, a door is opened by rotating it about its hinges. This rotation is done incrementally, a small amount each frame of animation.

Locomotion is performed indirectly by Action Execution binding behaviors to human figures in the environment database, which means neither path-planning nor explicit instructions are used to drive locomotion: rather, agent control and apparent complexity are the result of the interaction of a few relatively simple behaviors with a complex (and changing) environment. An agent is made aware of its environment through the use of a network of sensors. Based on the information gathered by these sensors the path through the terrain is incrementally computed. This allows the agent to react to unexpected events such as moving obstacles, changing terrain, or a moving goal (Reich *et al.* 1994).

3.1 Sensors and Behaviors

A *sensor* senses an object or location in the environment. It is a function which returns the distance and angle to that object or location relative to the agent's position and orientation (his state). A *control behavior*, such as *attract* or *avoid*, is a function that maps distance and angle to a stress value, where lower values represent more desirable states. The combination of a sensor and a control behavior results is referred to as a *behavior*. The agent utilizes a set of behaviors in interacting with its environment. The following three classes of behaviors are currently in use.

Attraction: An attraction behavior draws an agent toward a goal – either an object or a location. If a goal object moves, the point of attraction moves appropriately. The output (stress value) of an attraction behavior is high when the agent is far from the goal and decreases as the agent nears the goal.

Avoidance: An avoidance behavior is used to avoid collisions between the agent and objects or other agents. The sensor component of an avoidance behavior has a sector-shaped region of sensitivity. If there are no objects in this region, the output of the avoidance behavior is zero. Otherwise the output is proportional to the distance and size of the detected objects.

Field-of-View: A field-of-view behavior uses a sensor to determine whether or not the agent is visible to any other agents. The output is proportional to the number of agents' fields-of-view it is in and inversely proportional to the distances to these agents. This behavior is primarily used to support hiders in ZAROFF.

3.2 The Behavioral Simulation System

BSS provides general locomotion of objects in *Jack* (Becket & Badler 1993), and is used in ZAROFF to generate human locomotion. The central control mechanism of BSS is a perception control, and action loop. During the perception phase the behavior outputs are determined, during the control phase the next foot position is selected, and during the action phase the step is taken.

3.3 Behavior Scheduling

For control of the agents' behaviors we use a set of finite state **machines** which run in parallel with the simulation. These machines are responsible for behavior scheduling and act as a high level interface to the behaviors. They may instantiate other machines and either run in parallel with them or wait for them to exit. Communication among state machines is also possible. We make use of this communication among machines to implement interaction between players (Section 4).

When the planner commits to an action it invokes the Action Execution module. The Action Execution module instantiates and runs the machines necessary to carry out that action. Each machine is responsible for the scheduling and control of the appropriate behaviors.

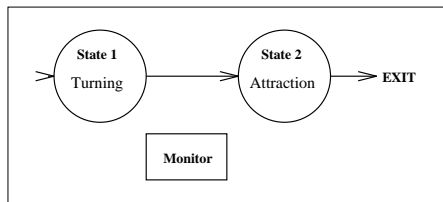


Figure 1: The Attract Machine

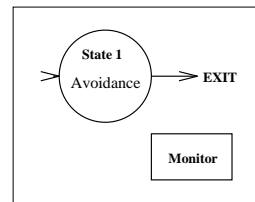


Figure 2: The Avoid Machine

3.3.1 Attraction

An *attract machine* consists of two states (Figure 1). In **state 1** the agent turns to face the object or point of attraction. In **state 2** an attraction behavior is bound to the agent. The agent walks to the goal object or location. When the agent arrives, this behavior is unbound and the machine exits.

An attract machine optionally includes a monitoring process which checks, in parallel, for an arbitrary condition (a LISP expression passed as an argument when the machine is created) to be true. If this condition evaluates to true at any time, the attraction behavior is unbound and the machine exits. The value returned by the machine when it exits indicates the cause of the exit.

We use the monitoring process primarily to control the behavior of the seeker. As the seeker explores the environment, moving from one location to another through the binding and unbinding of attraction behaviors, its progress is interrupted if at any time a hider becomes visible. This allows the seeker to stop and formulate a new plan, taking advantage of the opportunity.

3.3.2 Avoidance

An *avoid machine* consists of only one state (Figure 2). It binds an avoidance behavior to the agent for each avoidable object in the environment, and then exits. Unlike attraction, avoid-

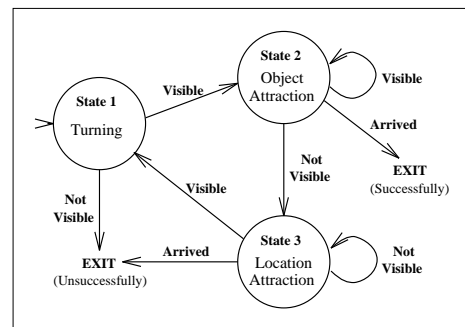


Figure 3: The Chase Machine

ance is maintained throughout the entire simulation. In ZAROFF the objects to avoid are doors, walls, and other agents. An avoid machine is used during the initialization of the simulation, once for each agent. This prevents the agents from colliding with each other, walls, and doors, and can easily be extended to any other objects that might be added to the system.

3.3.3 Chasing

The internal structure of a *chase machine* is a loop consisting of three states (Figure 3). If the target, the agent being chased, is not immediately visible, the machine exits unsuccessfully. Success is indicated by the exit value.

When the target is visible to the agent, the machine starts in **state 1**. The agent turns to face the target and the transition to **state 2** is made.

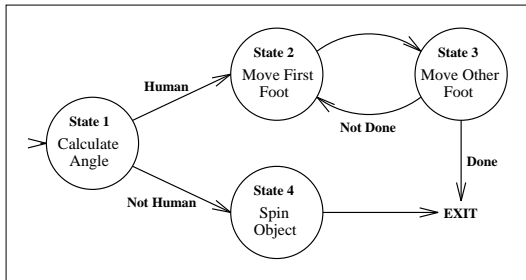


Figure 4: The Turn Machine

An attraction to the target is bound. The agent walks toward the target until one of two things happen. Either the agent arrives at the target in which case the machine exits successfully, or the target ceases to be visible to the agent. The target may have walked around a corner. In the latter case the transition to **state 3** is made. An attraction to the last known location of the target is bound to the agent. The agent walks toward this location until it arrives or the target becomes visible again. In the former case the machine exits unsuccessfully. In the latter case the transition to **state 1** is made.

A potential improvement to this machine would be to add an Extrapolation State. If the machine is in **state 3** and the agent arrives at the last known location of the target without seeing the target, instead of exiting unsuccessfully the agent would walk in the direction the target was walking before disappearing. This would require maintaining two pieces of information about the target: its last known location and its vector velocity.

3.3.4 Turning

A *turn machine* is used to rotate an object or human to face an object or location. The rotation may be either clockwise or counterclockwise, whichever is smaller, or may be specified. A turn machine consists of four states (Figure 4).

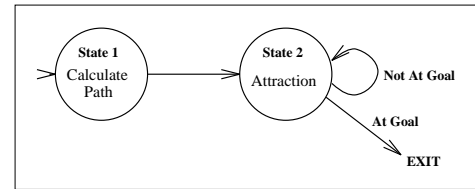


Figure 5: The Path Machine

In **state 1** the angle and direction of rotation are calculated. If the object is not a human, it is spun about its center of mass and the turn machine exits.

Humans are a special case. The human agent rotates the first foot to the right for clockwise, or to the left for counterclockwise. The leg and body follow, maintaining the agent's balance. The rotation angle for the first foot is the minimum of the goal angle and ninety degrees. The agent then follows the first foot with its other foot. If the goal angle is greater than ninety degrees, this cycle repeats until the goal has been reached. An optional argument can specify that the agent not follow with the other foot in the final cycle. When locomotion is to follow turning, it looks more natural when the final cycle is not completed.

3.3.5 Path Following

The idea behind a *path following machine* is to generate a path from the agent's current location to a goal location somewhere in the world and have the agent follow the path to the goal. A path following machine consists of two states (Figure 5). In **state 1** the path is calculated as a series of connected segments. The path has three important properties: it connects the agent to the goal, it avoids obstacles by a clearance specified by the user, and it is a path of minimal length given the first two constraints. In **state 2** the agent is attracted to successive vertices along the path. As the agent approaches a vertex, the attraction to

is accomplished by combining avoidance of the seeker and attraction to base.

The seeker begins by going to home base (**state 4**) and synchronizing the start of the game by forcing the other players to transition from **state 1** to **state 2**. The seeker delays the start of its search by counting **state 5** to permit the hiders time to reach their hiding places. Then the seeker begins to explore the environment to find a hider (**state 6**). When a hider is discovered, it becomes the target for a chase (**state 7**). The outcome of the chase determines whether the player will be a hider next game (when a hider is tagged before reaching base) or a seeker again (when all the hiders safely evade the seeker). If a single hider successfully evades the seeker, the seeker continues to search for other hiders until they are all safe. This may result in different hiders being chased at different times.

A consequence of limited perception is the occasional need to find objects. Our approach is to isolate this reasoning in a specialized module, a *search planner* that translates information acquisition goals to high-level physical goals to explore parts of the environment. Our approach to search planning requires that each player maintain information about the state of a heuristic search on an internal map. The heuristic search has finding a desired object as its goal.

5 Example

Having given an overview of the system’s components, we now illustrate ZAROFF with an example drawn from a two-player game of hide and seek. To illustrate the conduct of a search, we will use an example environment with two buildings, one of which has two internal rooms separated by a door (Figure 8). Our example begins in the middle of a game, after the hider has hidden and the seeker has finished counting. The hider is in **state 2**,

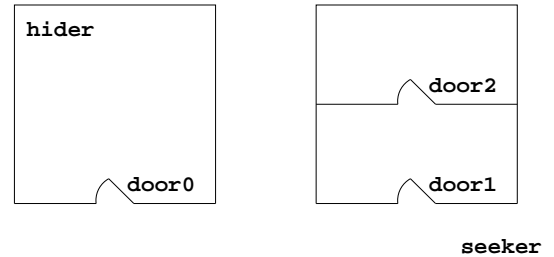


Figure 8: Plan view of example environment

waiting to be seen. The seeker changes to **state 6** of its goal machine and begins to seek. The goal machine state specifies a “seek” goal to the planner as `goto(X)` with the added constraint that `type(X) = HUMAN`.

The ITPLANS planner considers the action `goto(X)` to be primitive but underspecified since the variable `X` is not bound to a particular object of type `HUMAN`. In order to bind the variable, the search planner must be called to generate a plan for locating a `HUMAN`. To this end, ITPLANS adds to the plan a *find node* and calls the search planner to instantiate a search plan (Figure 7a).

The search planner reasons from this knowledge acquisition goal of locating a `HUMAN`, to the goal of exploring regions where a `HUMAN` might be. Satisfying this goal requires physically searching through possible regions.

ITPLANS asks the search planner to expand the find node. Each time a find node is expanded, the search planner first examines the *Jack* environment to determine if an object of the specified type is visible to the agent. If not, the search planner selects a region to explore next, generates a goal to explore that region, and adds it to the plan (Figure 7b). The initial map (Figure 9) of regions contains all the regions in the environment except the interior room of the building on the right. Regions that are completely visible are marked as having been explored; partially visible regions are marked

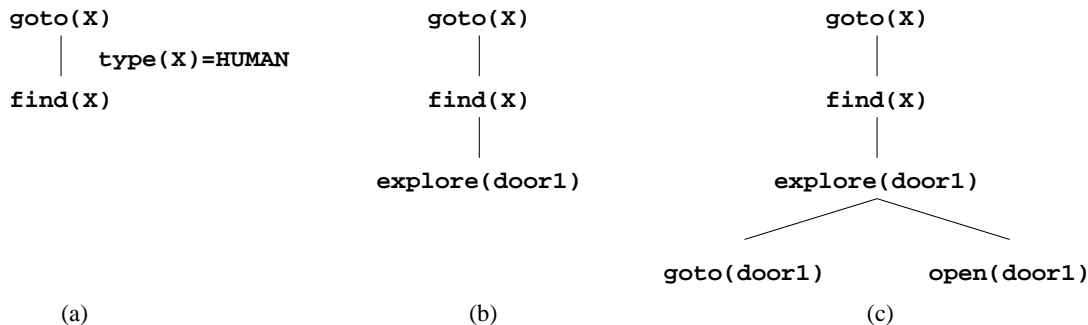


Figure 7: Evolution of the plan graph

for future exploration. The closest available unexplored region is the first room in the building on the right; it is recommended for exploration. This goal is then further expanded by ITPLANS to go to `door1` and open it (Figure 7c). Since all of the arguments in the first action are bound to specific objects, it can be carried out. Action Execution performs this action indirectly by binding an attraction sensor to the seeker. When the seeker arrives, `door1` is opened directly by Action Execution.

After `door1` is opened, ITPLANS uses the search planner to evaluate the progress of the search by examining the world for objects having the property `HUMAN`. If one is located, the search is considered successful. If not, the search planner selects a new region for exploration and the searching process repeats until there are no more regions to explore.

In this case, opening `door1` does not reveal a `HUMAN`, but does permit the agent to see another region that is automatically added to the search planner’s internal map. As this new region is the closest unexplored space, on the next iteration the planner will plan to explore it. Opening `door2` does not reveal the `HUMAN`, so the search proceeds to the next closest unexplored region, the right side of this building.

Here we see the advantage of maintaining a map. Immediately after opening `door2`, the agent is inside one building and decides to go to a non-neighboring region. Since this destination region is known (from having seen it previously), we could simply go there. This would result in the agent walking directly toward the destination until stopped by the wall. To avoid getting caught in this local minimum, the search planner uses its internal map (Figure 10) to plan a path to the next region.

The only known path there is to exit the current building through `door1`. The search planner returns this sequence of intentions to ITPLANS, which then invokes Action Execution to generate locomotion along this path. Eventually, after opening `door0`, the seeker finally sees a `HUMAN` and can go to it.

6 Conclusion

We have implemented a plan-based controller for a player in the game of hide and seek. The complete seeker is implemented, we are extending our architecture to implement the hider. Our agent dynamically reacts to changes in the environment, from avoiding collisions with obstacles and other players to exploiting changes in information about where

the other players may be hiding. The implementation combines general purpose planning, special purpose reasoning about conducting a search, and reactive control of human behaviors.

ZAROFF is an effective system for animating humans carrying out tasks that require locomotion. Limiting the human agent's awareness of its environment by simulated perception increases the realism of the behavior generated.

7 Acknowledgements

We wish to thank our advisors Norm Badler and Bonnie Webber for their support of this work. Thanks also to them, Welton Becket, Jonathan Crabtree, Brett Douville, and Jeff Nimeroff for commenting on drafts of this paper.

References

- Badler, N.; Phillips, C.; and Webber, B. 1993. *Simulating Humans: Computer Graphics, Animation and Control*. Oxford University Press.
- Becket, W., and Badler, N. I. 1993. Integrated behavioral agent architecture. In *Proceedings of the Third Conference on Computer Generated Forces and Behavior Representation*, 57–68.
- Becket, W. M. 1994. The Jack LISP API. Technical Report MS-CIS-94-01, University of Pennsylvania, Philadelphia, PA.
- Geib, C. 1992. Intentions in means-end planning. Technical Report MS-CIS-92-73, Department of Computer and Information Science, University of Pennsylvania.
- Ko, H.; Reich, B. D.; Becket, W.; and Badler, N. I. 1994. Terrain navigation skills and reasoning. In *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representations*.

Ko, H. 1994. *Kinematic and Dynamic Techniques for Analyzing, Predicting, and Animating Human Locomotion*. Ph.D. Dissertation, University of Pennsylvania.

Moore, M. B.; Geib, C.; and Reich, B. D. 1995. Planning and terrain reasoning. In *AAAI Spring Symposium on Integrated Planning Applications*. (also University of Pennsylvania CIS department Technical Report MS-CIS-94-63/LINC LAB 280).

Reich, B. D.; Ko, H.; Becket, W.; and Badler, N. I. 1994. Terrain reasoning for human locomotion. In *Proceedings of Computer Animation '94*, 996–1005. Geneva, Switzerland: IEEE Computer Society Press.

8 Biographies

Michael B. Moore is a Ph.D. candidate in Computer and Information Science at the University of Pennsylvania. His current research in Artificial Intelligence is planning for information acquisition. He received his A.B. degree in Philosophy in 1986 from the University of Maryland and his M.S.E. degree in Computer and Information Science in 1990 from the University of Pennsylvania.

Christopher Geib is a post-doctoral research fellow at the University of British Columbia. His current research in Artificial Intelligence is planning. He received his Ph.D. degree in Computer and Information Science from the University of Pennsylvania in 1995.

Barry D. Reich is a Ph.D. candidate in Computer and Information Science at the University of Pennsylvania. His current research includes providing high-level behavioral control over the animation of human locomotion. He received his B.S. degree in Mathematics and Computer Science in 1989 from the University

of Maryland and his M.S.E. degree in Computer and Information Science in 1991 from the University of Pennsylvania.

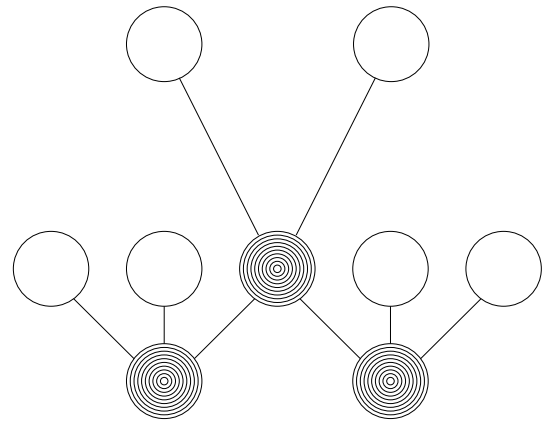


Figure 9: Initial map

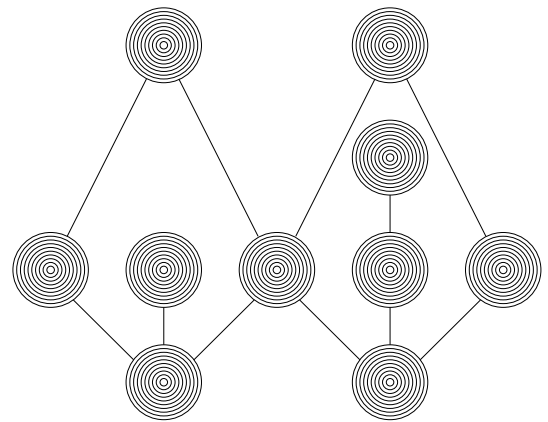


Figure 10: Final map