7

Possibility Maps for Opportunistic AI and Believable Worlds

John Manslow

7.1 Introduction

The state of a game world is the combination of the states of all the objects within it: the state of the player (his or her location, orientation, health, inventory, etc.), the states of all NPCs (nonplayer characters—their locations, emotional states, etc.), the states of inanimate objects (which doors are locked, which buildings have collapsed, etc.), and so on. Games usually simulate a game world in order to track the evolution of its state, perhaps using level-of-detail approximations for unobserved elements of game state to reduce the complexity of doing so.

This chapter will introduce the idea of maintaining probability and possibility distributions over some unobserved elements of game state rather than simulating their evolution explicitly. If such distributions are used correctly, they allow the artificial intelligence (AI) to defer decisions relating to unobserved elements of game state without the player realizing that anything unusual is happening. This makes it possible to identify opportunities for action that might otherwise have required the creation and execution of complex plans and hence to create opportunistic AI that appears much smarter than it actually is.

7.2 What Are Probability and Possibility Maps?

A probability map provides a mapping from game states to probabilities that represent the relative likelihoods of the game being in different states. For example, the location of a character in a village is part of the game state and can be approximated by discrete locations such as the character's home, the stables, the tavern, the well, the shop, and the stretches of road between them. A probability map can be used to associate these locations with probabilities so as to model the likely location of the unobserved character. The map might change with time of day so that the character is more likely to be found at home at night or in the tavern in the evening.

Possibility maps are simplified versions of probability maps that dispense with probability values and only map states to indications as to whether they are possible. A possibility map for a character in a village, for example, would be true for every location where the character could possibly be located and false everywhere else. Because possibility maps aren't concerned with the relative likelihoods of states, they are typically easier to use and better suited to applications that are concerned with creating the illusion of intelligence rather than level-of-detail approximations.

7.3 Using Possibility Maps

For an application of probability and possibility maps to be convincing, it is important that they are updated in a way that is consistent with the player's observations of the game's state. We will first consider how to do this for possibility maps because the mechanisms for updating them are easier to understand than are those for updating probability maps.

A possibility map must regularly be updated to propagate possibility information according to its propagation rules. The precise form of those rules depends on what the possibility map represents: if it represents the location of an NPC, for example, the propagation rules would reflect how the NPC moves around the environment. If an update would propagate possibility information into a state that would affect the player's observations, or the player makes an observation that determines a state that is marked as possible, the AI must immediately decide whether to put the game into that state. If it does decide to put the game into that state, all mutually exclusive states must be marked as impossible; if it decides not to put the game into that state, then that state must be marked as impossible.

The decision as to whether to put the game into a particular state is taken based on the AI's assessment of the state's desirability from a gameplay perspective except when it is the only possible state remaining, in which case the AI has no choice. A simple example of an element of game state that can usually be represented by a possibility map is the location of an NPC. Figure 7.1 shows a possibility map for the location of an NPC on a simple level where P represents the player's location, N the NPC's location, the Xs represent walls and the ?s represent possible locations of the NPC.

Initially, in Figure 7.1a, the player can see the NPC and hence there are no possible alternative locations for it. If the NPC moves east and the player remains stationary, the player loses sight of the NPC and the possibility map starts to track the NPC's possible movement, as represented by the question marks in Figure 7.1b. The progress of the question marks is determined by the map's propagation rules and, since the map represents the possible location of the NPC, the rules are simply set up to reflect its range of possible movement. Figure 7.1c shows

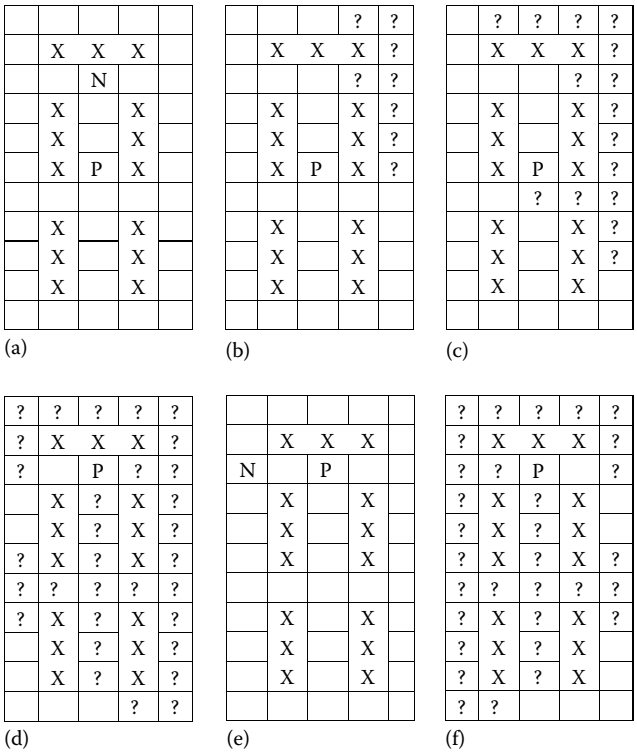

Figure 7.1

Possibility maps of an NPC's location. (a) A possibility map with a player looking north at an NPC. (b) The state of the possibility map a few seconds after the NPC moves out of sight to the east. (c) The state after a few more seconds. A melee attack from behind the player is now possible. (d) After the player moves north three squares, the AI can choose from a variety of melee and ranged weapon attacks. (e) The state of the map if the AI instantiates the NPC behind the player as the player turns to face east. (f) Possible locations for a second NPC that started in the same location as the first but moved west.

that, after several seconds, it becomes apparent that the NPC could've crept up behind the player and the AI has the opportunity to instantiate it and perform a melee attack.

If the AI chooses not to instantiate it and the player moves north three squares and starts to turn to the right, as shown in Figure 7.1d, the player will be about to observe two squares that are possible locations of the NPC. The AI must therefore decide whether to instantiate the NPC or to keep it hidden and continue propagating possibility information. It has a rich set of options if it chooses to instantiate it—it can instantiate it south of the player for a melee or ranged weapon attack, a couple of squares west of the player for a ranged weapon attack from behind, or to the east of the player for either a melee or ranged weapon attack. Figure 7.1e shows the state of the possibility map if the AI chose to instantiate it west of the player. If it chose not to instantiate it, the possibility map would look the same as in Figure 7.1d except that the two squares east of the player would be blank to indicate that the NPC cannot possibly be present at those locations.

In this example, the AI was forced to decide whether to instantiate the NPC when the player observed possible locations for it. It might also be the case that the player could infer the location of the NPC by other means, such as by hearing sounds that might be emitted when the NPC walks over noisy surfaces. If the player is out of earshot of the surfaces, then possibility information can propagate over them as if they didn't exist. If the player is within earshot, however, the surfaces block the propagation of possibility information and the AI would need to instantiate the NPC to allow it to cross. Similarly, if the NPC had no stealth capability, then the rules for propagating possibility information would've prevented locations adjacent to the player being marked as possible in Figure 7.1c and d and the AI would've needed to instantiate the NPC to get it close to the player. In general, the AI can choose to instantiate an NPC at any time though doing so prevents it from deferring decisions about the NPC's behavior and hence early instantiation is usually undesirable.

If the map in Figure 7.1 had contained two NPCs, their locations would've been represented by two independent possibility maps. The AI could create the illusion of finely coordinated movement between them by taking account of both their maps when deciding when and where to instantiate them. For example, assuming that both NPCs had started at the same location but the second moved west out of sight of the player, then the possibility map for the first NPC is as shown in Figure 7.1d, and the map for the second would be as shown in Figure 7.1f. As the player turns to the right, it is clear that the AI could instantiate an NPC almost anywhere and, in particular, could instantiate both NPCs to create the illusion of a planned coordinated attack from behind and to the side, from the front and to the side, from the front and from behind, and both from the front, both from the side, and both from behind. Note that, although the AI is technically cheating by using possibility maps, it always has to behave in a way that is consistent with the player's observations and hence a skillful player can often prevent the AI from springing these kinds of opportunistic traps by careful observation.

Figure 7.2 shows a level with a resource item—a health pack that can be used by NPCs—represented by an H. If the player initially observes the NPC, as shown in Figure 7.2a, and the NPC moves out of sight to the north, the possibility map will evolve as shown in Figure 7.2b. When it becomes possible for the NPC to have reached the health pack, the AI can either decide that the NPC should pick the health pack up immediately, in which case the possibility map is reset with the only possible location of the NPC being the location of the health

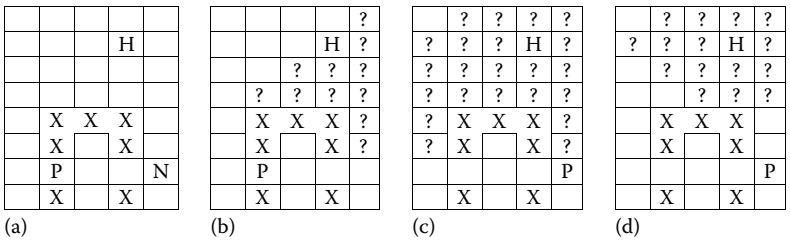

Figure 7.2

Possibility maps of an NPC's location with a health pack. (a) A possibility map with one health pack and a player facing an NPC. (b) The state of the map a few seconds after the NPC moves north out of sight of the player. (c) The state if the NPC did not pick up the health pack and the player moves three squares east. (d) The state if the NPC did pick up the health pack.

pack, or the AI can defer the decision. In order to defer the decision, the AI needs to fork the possibility map into two—one for the location of the NPC with the health pack and one for the location of the NPC without. The latter is simply a continuation of the original but the former is a new map with the only possible location of the NPC being the location of the pack.

If, as the NPC possibly reaches the health pack, the player moves east three squares, the possibility map for the NPC without the health pack, which is shown in Figure 7.2c, reveals that both melee and ranged weapon attacks are possible and the possibility map for the NPC with the health pack, which is shown in Figure 7.2d, shows that only a ranged weapon attack is possible. The AI therefore has the opportunity to choose between the lower health version of the NPC performing a melee or ranged weapon attack, the higher health version performing a ranged weapon attack, and keeping the NPC hidden. If the NPC is particularly effective in melee combat or ineffective at a distance, the benefits of deferring the decision about collecting the health pack are obvious.

Forking can be used to defer decisions in relation to a wide variety of actions—should an NPC pick up a key, unlock a door, flick a switch, buy a sword, etc.? It can, however, also result in a combinatorial explosion of possibility maps if the environment contains too many opportunities for action and the AI will usually have to make some decisions earlier than is strictly necessary simply to control the number of maps. Even deferring a decision for a short time, however, can be beneficial as the AI often has more information available to it when it comes to make the deferred decision than it would've had if it had not deferred the decision at all.

7.4 Updating Probability Maps

Like possibility maps, probability maps must also regularly be updated—this time to propagate probability information according to its propagation rules. The precise form of those rules depends on what the probability map represents: if it represents the location of an NPC, for example, the propagation rules would reflect the likelihood of the NPC moving from each location to every other. If an update results in nonzero probability in a state that would affect the player's observations, or the player makes an observation that determines a state that has nonzero probability, the AI must put the game into that state with the probability indicated by the probability map. If the game is put into the state, the probabilities of all mutually exclusive states are set to zero, but if the game is not put into the state, the probability of the state is set to zero. Normalization of the probability map must be maintained at all times—that is, the sum of the probabilities of all states must always be equal to one.

Consider, for example, a guard that's controlled by a state machine with the states patrolling from point A to point B, patrolling from point B to point A, eating, sleeping, and playing solitaire. Each of those states would be assigned a probability by the probability map and the map's propagation rules would provide probabilities for transitions between states and, perhaps, minimum times that the transitions should take. Table 7.1 gives some example transition probabilities, and Table 7.2 shows how the probability map evolves after the player sees the guard patrolling from point A to point B.

If the player entered the dining hall when the probability map had the values in the second to last row of Table 7.2, the AI would instantiate the guard in the eating state with probability 0.097. If the instantiation happened, the probability of the eating state would be set to one and the probabilities of all other states would be set to zero. If the instantiation did not happen, the probability of the eating state would be set to zero and the

Table 7.1 Transition Probabilities for a Guard

		Next State				
		A to B	B to A	Sleeping	Eating	Solitaire
Current state	A to B	0.00	1.00	0.00	0.00	0.00
	B to A	0.95	0.00	0.00	0.05	0.00
	Sleeping	0.04	0.00	0.95	0.01	0.00
	Eating	0.18	0.00	0.01	0.80	0.01
	Solitaire	0.50	0.00	0.50	0.00	0.00

Table 7.2 Probability Map for a Guard

Time	A to B	B to A	Sleeping	Eating	Solitaire
1	0.000	1.000	0.000	0.000	0.000
2	0.950	0.000	0.000	0.050	0.000
3	0.009	0.950	0.001	0.040	0.000
4	0.910	0.009	0.001	0.080	0.000
5	0.023	0.910	0.002	0.064	0.001
6	0.877	0.023	0.003	0.097	0.000
6 after observation	0.970	0.026	0.003	0.000	0.001

probabilities of the other states updated so that they still sum to one, as shown in the last row. In practice, state transition probabilities usually need to vary with time of day to produce realistic behavior. For example, the probabilities of transitioning from any state to the sleep state would be very high at night for a guard that was only working the dayshift.

Probability maps provide the AI with information about the relative likelihoods of different game states and that allows it to distinguish between states that are almost certain and others that are theoretically possible but highly unlikely. In this sense, they are more powerful than possibility maps. Unfortunately, it can be difficult to come up with transition probabilities that produce sensible behavior and, if the transition probabilities vary with time, the probability map will never stabilize. This means that the AI will, in principle, need to continue to update it regardless of how long it is since it was last affected by the player. These problems can be avoided by combining probability and possibility maps using the method described in the next section.

7.5 Combining Possibility and Probability Maps

A useful approximation to a normal probability map can often be achieved by combining a static probability map and a possibility map. A static probability map assigns probabilities to states but the probabilities are not propagated across the map according to transition probabilities as they are in a normal probability map. Instead, they represent the likelihoods of observing states when no previous observations have been made and a possibility map is used to ensure that only states that are consistent with the player's observations can be instantiated.

For example, Table 7.3 shows three static probability maps for a guard, the first to be used during the working day, the second at mealtimes, and the third at night. If the player

Table 7.3 Static Probability Map for a Guard

Time	A to B	B to A	Sleeping	Eating	Solitaire
Working day	0.495	0.495	0.001	0.003	0.001
Mealtimes	0.010	0.010	0.000	0.970	0.010
Night	0.002	0.002	0.990	0.003	0.003

observes the games room, the AI checks to see whether the possibility map indicates that it's possible for the guard to be in there and, if it is, the AI instantiates the guard with the probability specified by the static probability map after it's been adjusted so that the sum of the probabilities of all possible states is one. For example, in the unlikely event that it's mealtime and the player has just observed that the guard is not in the dining hall, then the guard must either be patrolling from A to B, patrolling from B to A, or playing solitaire—hence, the probabilities of those states are all $0.010/(1 - 0.970) = 0.333$. If the player observes the games room, the AI would therefore instantiate the guard there with probability 0.333.

If the player left the area completely, it wouldn't take long for every state in the possibility map to be marked as possible, at which point, no further computation would be necessary until the player returned and made another relevant observation.

7.6 Factorizing Game State

The complete state of a game is an extremely complex multidimensional entity, and it is not realistic to expect to be able to create probability and possibility maps over the state in its entirety. Instead, it is necessary to factor the game state into independent elements and create multiple independent maps. Such elements must be independent in the sense that the state of one should not affect the probable or possible states of another. The most easily identifiable independent elements are usually the locations of NPCs, although they can become dependent when NPCs need to interact.

For example, if the guard in the earlier example had played poker rather than solitaire, then it would've been necessary for the AI to make sure that there was never only a single guard in the games room, thereby creating a dependency between the locations of guards. That problem could be solved by checking the probability and possibility maps of all guards when the player enters the games room to see if enough guards could be there for a game to be taking place and only instantiating them if that was the case. To guarantee that the AI always had a choice, however, it would need to make sure that no single guard ends up with the games room as his or her only possible location—something that could easily be achieved when deciding whether to instantiate guards elsewhere.

Shared resources create more serious dependencies. Consider, for example, a map with two NPCs and a health pack that can be taken by only one of them. As has already been described, as each NPC possibly reaches the location of the health pack, their possibility maps must fork and the game must maintain four possibility maps—two for each NPC, one to represent possible movement with the health pack and one without. When the AI decides to instantiate an NPC, it must decide whether it will be the version with the health pack or the one without and must remember that the health pack is with only one of the NPCs. The situation gets even more complicated if the health pack regenerates after a short time. In that case, it is possible for both NPCs to have picked it up but one

of them could only have done so after it had regenerated. This situation can be modeled with additional forking to account for the order in which the NPCs picked up the pack and to allow for its regeneration time.

Another example of a shared resource that creates dependencies is a lift. Lifts create dependencies because, when one NPC moves the lift, it affects its availability for the others in an extremely complex way—the amount of time that an NPC's possibility map takes to propagate from one floor to the next via the lift depends on the location of the lift at the time the NPC possibly reaches it, the times other NPCs possibly reached it, and where they possibly left it—and all of that depends on deferred decisions that have not yet been made. In principle, this problem can also be solved by forking but it's probably better and certainly simpler to use an approximation such as to ignore the dynamics of the lift altogether and just propagate possibility information between floors with a slight delay.

If a suitable factorization of the game state cannot be found, a joint map can be created to model the probability or possibility of the combined state of multiple elements. This is effectively what is being done by forking; when, in the earlier example, an additional possibility map was created when the NPC could've picked up the health pack, the AI was dynamically creating a possibility map for the combined states of the location of the NPC and its health. The problem with this approach is that the number of states in a joint map grows exponentially with the number of elements of game state that it represents and hence joint maps can be excessively large and unwieldy. Some factorization is therefore always necessary for the successful application of probability and possibility maps.

7.7 Conclusion

This chapter has described probability and possibility maps and shown how they can be used individually and in combination to produce level-of-detail effects and allow the AI to defer decisions to create the illusion of highly intelligent, coordinated, and carefully planned behavior. Respect for the player's observational history ensures that this is achieved without the player noticing any inconsistencies and provides a way for players to limit the options of the AI through their own careful planning and observation.