6

Preventing Animation Twinning Using a Simple Blackboard

Michael Dawe

6.1 Introduction

Arguably, the most important aspect of an artificial agent in a game is the player perception of that agent. An agent acting intelligently can break the illusion of intelligence by looking awkward, silly, or robotic to the player. When a realistic game has many similar characters, as is frequently the case in open-world games, it can become quite likely for two or more characters visible to the player to begin playing the same animation simultaneously, a problem known as animation twinning. However, avoiding animation twinning is a relatively easy problem to solve, even for large character populations.

6.2 Animation Twinning

Animation twinning occurs when two agents play back the same animation at a close enough time so as to appear indistinguishable. It is easily noticeable to a player and breaks the illusion that a game is simulating actual persons, since real people never exhibit the exact same motions in real-world situations. Even when two agents have different appearances, the same animations playing on the two identical skeletons are quickly picked out as artificial by the human eye. While ideally agents would have a wide variety of

animations to play, realistically, the available number is constrained by budgets and animator time. Additionally, if a game puts a large population of characters on screen, console loading times and memory overhead may become a factor in loading many different animations to be played back.

One practical solution is to introduce a delay in animation playback, as the same animation played back on two identical skeletons with such a delay looks much more natural. This solution is quick to implement and requires minimal animator time to ensure adequate differentiation among the animations a character can play in a particular situation. While locomotion animations probably cannot be delayed for gameplay reasons, most other ambient animations, such as eating a bagel or talking on a cell phone, will benefit from a short delay to prevent characters from looking mechanical and breaking the illusion of intelligence.

6.3 Animation Blackboard

Assuming a large population of agents (say, 30 or more being simulated at once), it is impractical for a single agent to query the animations of every other agent before playing an animation of its own. However, a single data structure available to all agents, such as a blackboard [Isla 02], can be used to store the relevant information in a manner that can be quickly checked for duplicates. When an agent needs to play an animation, the blackboard can be checked to ensure that no other character is playing the same animation within whatever time period is specified. Once an agent has permission, the blackboard is updated with the animation the agent has chosen to play.

The `AnimBlackboardEntry` definition in Listing 6.1 shows the information stored about every animation playing. First, each animation needs a unique identifier to be checked against. Practically, this can be any hashed value based off of the animation, carefully selected to avoid hash collisions. Next, the game time that the animation started playing at is stored, so we can allow characters to play the same animation after whatever amount of delay is appropriate. The world location where the agent is standing is useful in situations where characters must play the same animation due to many characters being on screen with a small number of animations to choose from, as players will have more difficulty noticing similar animations being played when they are separated in space. Finally, it is useful to separate animations into two categories: those which can easily be delayed for a significant amount of time (reading a book, eating a sandwich) and those which must not be delayed for very long (reacting to gunfire or an explosion). Agents that fail to react quickly to important stimuli will look just as odd as those that play identical animations.

Listing 6.1. `AnimBlackboardEntry` definitions.

```
enum AnimPriority {NORMAL, REACTION};

class AnimBlackboardEntry {
  Hash animId;
  unsigned int time;
  Vector3 worldLocation;
  AnimPriority priority;
};
```

Listing 6.2. `AnimBlackboard` definition.

```
class AnimBlackboard {
  AnimBlackboardEntry blackboard[32];  //Sized for population
  size_t front, back;//Treat array as a circular queue
  unsigned int priorityTimes[];//One per AnimPriority
};
```

Listing 6.3. `CanPlayAnim` function pseudocode.

```
bool AnimBlackboard::CanPlayAnim(Hash anim, Vector3 location) {
  For each entry
    If entry is too old, move the front of the queue up
      If entry isn't our anim, skip it
      If entry is our anim:
        If (now - entry.time) < min delay for entry.priority
          return false
  //Add the anim to the blackboard here, or do it afterwards
  return true
}
```

The `AnimBlackboard` definition in Listing 6.2 defines an array that will be used as a circular queue of `AnimBlackboardEntry` items. The size of the array will depend on the maximum number of agents needing to be animated simultaneously. Finally, the `priorityTimes` array stores the delay necessary before playing an identical animation.

When an agent needs to check if playing a particular animation is permitted, it can call `CanPlayAnim` as defined in Listing 6.3. Every call to `CanPlayAnim` walks from the front of the queue to the back. If the current entry is older than the maximum delay we care to enforce, the front of the queue can be pushed up to that entry, since it will never need to be checked against again. If an entry is found that we need to consider based on the time it was played, we compare the identifier of the animations and, if they match, return false if the time since that animation was played is within the delay period. Otherwise, the animation is allowed. Distance can also be considered here, if it was included in the `AnimBlackboardEntry` definition by allowing animations that would have been disallowed if they are far enough apart. If allowing animations based on distance, every entry in the blackboard must be checked. Once the animation has started to play, the agent must add it to the blackboard.

6.4 Delay Times and Animation Considerations

The amount of time animations should be delayed is subjective, but good results were found with a 1 s delay for normal animations and 300 ms for reaction animations. In general, the delay for normal animations is less critical than the delay for reaction animations, and the delay amount needs careful examination and adjustment. Too small of a delay (under 200 ms) produces indistinguishable differences, while too long of a delay (over 500 ms) results in characters not reacting quickly enough.

Also important in discussing the delay amounts is how the animations themselves can affect the delay needed, again impacting reaction animations more significantly. If the first half-second of a particular reaction animation has very little character movement, a delay less than that will still produce characters looking identical for a period of time. Work with the project animators to ensure enough variation in the first section of the animation and ensure consistency across a reasonable delay period, then experiment and adjust the timing based on what looks best.

While including the location an animation plays at is optional, it proved useful in situations where a population of over 20 characters all needed to play reaction animations simultaneously to get up from park benches and picnic tables in order to react to a threat. Despite having a number of animations to play considerably less than the population, the location information helped stop characters that were near to each other on screen from playing similar animations, which improved the overall appearance of the scene.

6.5 Conclusion

Characters playing simultaneous animations can look awkward, but it's simple to create a solution to avoid exact animation duplicates and performs well for large populations. While increasing the available number of animations for characters to play will always help reduce delays in playback, a short amount of time spent implementing this blackboard system will produce a marked difference in the appearance of the game, even with a relatively limited number of animations to choose from.

Reference

[Isla 02] Isla, D. and Blumberg, B. 2002. Blackboard architectures. In *AI Game Programming Wisdom*, ed. S. Rabin, pp. 333–344. Hingham, MA: Charles River Media.