Cinematic Gameplay in Watchdogs 2: Pose Matching and AI Coordination

Christopher Dragert, Ph.D.

1 Introduction

In the original Watch Dogs game, there is a type of side mission called "Privacy Invasion". The idea fit perfectly into the motif of the game: Aiden Pearce, master hacker, would gain access to a camera, and observe people in domestic incidents, quirky slices of life, or even life or death situations. While narratively powerful, these side missions lacked gameplay, and were essentially motion-captured cut-scenes.

In Watch Dogs 2, our mandate was to dramatically improve these missions by adding gameplay. Now called "Invasion of Privacy" missions (IOPs), the player would be allowed to hack cameras, computers, and other electronics in the room. NPCs would then react to these events in a realistic fashion while advancing the narrative of the scene, with the aim of delivering a highly interactive experience. Early on, the decision was made to deliver IOPs with cinematic quality animation and sound – meaning each scene was to be fully motion captured with seamless branching. The final product was a compelling and highly engaging experience for the player.

Aiming for such a high quality-bar required addressing a number of significant technical challenges:

- How to make motion-captured NPC animations appear seamless in a branching scenario?
- To what extent can we support player control through NPC reactivity?
- How do we ensure the NPC reacts correctly through all player inputs?
- In scenes with multiple NPCs, how do we coordinate their reactions to be consistent with not just the scene, but also between group members?
- How do the NPCs maintain dialog flow and narrative consistency?

In this chapter, we will describe our solutions to each of these challenges, as well as the technical and design constraints imposed by our approach. The objective is to provide enough knowledge to enable the design and construction of your own custom cinematic gameplay system, with a set of constraints appropriate to your own game. To this end, we will also note other approaches that were not employed, along with their own technical and design trade-offs.

2 Overview: Invasion of Privacy Missions

This section describes the necessary background to understand IOP missions, as well as the important decision on how to use motion-capture animations. This enables us to properly dig into the technical challenges in the remainder of the chapter.

An Invasion of Privacy mission begins when player hacks into a junction box. The player's view is put into a camera in the scene, allowing them to view the contents of the

room and the NPCs within. The player can look around with the camera, profile the characters and hackable objects to learn more about them, and switch cameras to get a different view. Gameplay is advanced by hacking objects within the scene.

The 'Whistleblower' IOP, shown in Fig.1, features a man driven to suicide by a blackmail attempt. You can hack his phone in an attempt to connect him to help. However, if you only hack his phone, the people you contact will literally put him over the edge. If you instead hack his laptop you can find evidence of the blackmail, making it so that the next phone hack will connect him to a journalist and ultimately save his life.

Figure 1 The Whistleblower IOP, showing the hackable phone.

The behavior of each IOP was designed in detail in a mission design document. This was an essential step in communicating the flow, as well as spotting potential failure points. For instance, what is the correct behavior if the phone is hacked while the computer was downloading? In IOPs with heavy branching or multiple simultaneous options, putting the desired flow on paper was a vital step. A sample control flow for whistleblower is shown in Figure 2. The actual version includes much more detail, such as timing for enabling and disabling hacks, audio and visual effects, and other level design details.

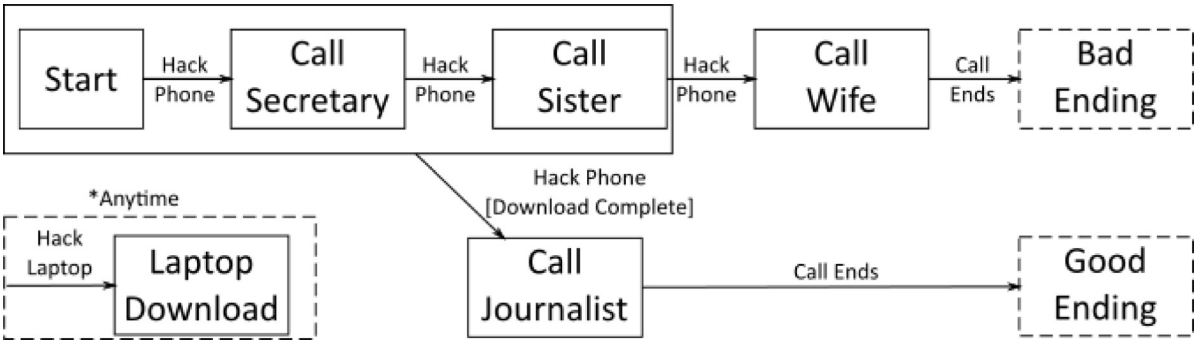

Figure 2 Control flow of the Whistleblower IOP.

2.1 Animation and Motion Capture

One of the major decisions during development was on the use of performance capture ('mocap' for short). Two options were considered: the first option was to aim for cinematic quality by using mocap for the entire scene, including all branches; the second option was to take a more systemic approach by employing existing walk cycles and object interaction animations, supplemented by a limited amount of new mocap for narrative moments. Table 1 summarizes the pros and cons of performance capture when compared with a more systemic approach.

Table 1 Pros and Cons of Performance Capture

Pros	Cons
+ Highest narrative fidelity + Audio and body language match + Ability to perfectly match reactions to the scene + Close-ups	- Have to pose match animations - Frozen in time: once mocapped, the narrative and flow become fixed - Must pre-plan all branches and combinations early in development - Cost

For a highly narrative piece of gameplay, much of the story is told through NPC dialog and reactions. The most direct way to get the NPCs to have the precise reactions and animations required would be to mocap exactly what is needed. Systemic animations will always be a less exact fit, which could constrain the narrative by limiting what the NPCs can express. For example, if a character is becoming frustrated, a fully mocapped scene could have a reaction that leads perfectly into a frustrated NPC stomping across the room. A systemic animation may not have a frustrated stomp and would have to settle for a standard walk cycle, impairing narrative fidelity.

While we knew that systemic animations would be the easiest and cheapest option, there was one compelling reason that lead us to go the other way: in many IOPs, the narrative suited having a camera placed immediately in front of the NPC for a close-up shot. While it is possible to fake lip-sync for dialog, it is immediately obvious at that distance. Also, the lack of facial animation is something that is only acceptable when the NPC is further away from the camera. We found that even generic body movements that read fine at a distance can fail to hold up when the NPC is too close, looking robotic and unnatural. The choice then became to use performance capture for the entire scene, or to do a systemic scene with no NPCs close to the camera, which would be a massive constraint on narrative.

There was one significant drawback to full mocap. Once the mocap data is received, the animators need a chunk of time to clean up the raw data. This forces us to mocap early in the development cycle, which has the effect of locking in narrative choices without necessarily having a full view of the game narrative. As the game narratives evolves naturally throughout production, making changes to a mocapped IOP is nearly impossible, while if the scene was systemic, it would be fairly trivial. Also, there is a significant financial cost to mocap, but that is primarily a production decision.

Ultimately, quality won out and the decision was made to fully motion capture all IOPs. While this created significant challenges, tackling these allowed us to achieve an excellent outcome. Among other things, this meant that each and every IOP had to be planned out in exacting detail in order to capture all possible branches and combinations, and not miss any shots.

3 NPC Behavioral Structure

With something as high quality as an IOP mission, and especially considering that it was to be fully mocapped, it was vital to give a clear structure to IOPs. This provides a clear design space while ensuring that the AI can actually express the designed behaviors. Equally important was that it provides constraints that prevent exploding the branching state-space or complicating the pose matching.

A consideration of the context can inform us of what a reasonable person might do in an IOP. A hack on an object will trigger a context sensitive action. In an IOP, hacking an electric device usually activates or deactivates it (e.g., turn out the lights, turn on audio speakers, etc.). Other more complex electronics will present a different and highly contextual action (e.g., access computer, upload virus, etc.). So what would our reasonable person think if a hack like this took place? The first reaction might be surprise or mild annoyance: "That's weird, what's wrong with my lights?" The NPC might choose to investigate or attempt to turn their lights back on. If it happens again, they might be inclined to assume that either the device is broken, or someone is messing with them. Yet another hack might lead to anger or frustration, cause them to give up on the clearly malfunctioning device, and so on.

Following from this thought experiment, we were able to create our structure, which we called *emotional escalation*. It provided the framework that we used for all IOPs. The core concept is that each hack increases the emotional intensity of the NPCs in the scene. If a hack annoyed a character in the scene, subsequent hacks would make the character increasingly irritated and eventually angry.

In the 'Always On' IOP, a teenage girl is dancing in her room, and the player can turn off the lights and change the music. This interrupts and annoys her, and she races to fix the music and/or lights before resuming her dance. The first hack slightly annoys the NPC, the second makes her angry, and the final one causes her to have a meltdown. Figure 3 shows the emotional escalation. Note how the escalation works – if you hack the lights, and then the music, she can play both investigate reactions. However, if you hack music twice, then the NPC is emotionally escalated. If you then hack the lights, she will play the lights escalated reaction, completely skipping the lights investigate reaction. Emotional escalation is one way.

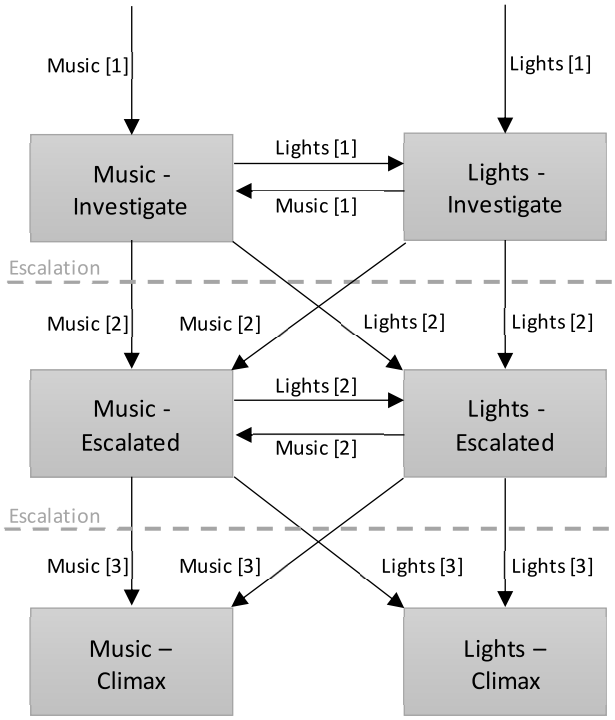

Figure 3 Emotional escalation for NPC behavior in the 'Always On' IOP.

3.1 NPC Reactions

When the player chooses to perform a hack, the hacked device would react as defined in the narrative. This often included some audio or visual component, such as a screen going to static, or an alarm going off, etc. In the interest of making this natural, the NPC would only ever react when it was realistic for them to perceive the device reaction. An NPC in headphones shouldn't hear background noise, and NPCs should never see visual hack outcomes when they occur behind them.

When an NPC does perceive that a hack has occurred, what would their reaction consist of? At a low level, a reaction usually consisted of a simple cycle: React -> Restore -> Resume:

- **React**: A reaction would always be preceded by a small timer to simulate human reaction time (e.g., 0.2 – 0.5s, depending on NPC activity and urgency of hack [Rabin 15]). After that window, the NPC would play their reaction animation.
- **Restore (optional)**: The NPC would attempt to repair the state of the scene. Is an alarm going off? Go to the alarm or control panel to turn it off. Did the lights go out? Try to turn them back on. This often triggered NPC movement.
- **Resume:** With the situation either under control or written off as a lost cause, the NPC would then resume their default behavior. There is no requirement that they necessarily return to their previous default. Thinking in terms of state, the reaction cycle can either be a self-loop that resumes previous activity, or it can be

a state change that moves the NPC to a new default.

To be clear, most reaction cycles are triggered in response to player action. However, some reactions are preset based on the scene. In the 'Always On' IOP, the idle look has the girl perform her dance in full two times. If nothing interrupts the dance, she completes it, and a final outro is triggered.

4 Branching and Reactivity

Our target on the gameplay side was to allow the player to branch the scene at any point - full interactivity. While a noble goal, it proved to be impossible for several reasons, which shall be illustrated through an example. Imagine the player has triggered a hack that has caused the NPC to start moving as part of their react-restore-resume cycle. During movement, the player hacks again and the NPC wants to start a new cycle. What should happen in this case? Since the IOP is not systemically animated, we can't just play a reaction at any point. Further complicating this, it is simply not realistic to mocap an arbitrary number of reactions and short walks so that we can interrupt our NPC at every step along the way. The sheer volume of mocap coverage required makes it prohibitive in terms of both time and money.

Figure 4 The 'Always On' IOP, showing the music hack.

The alternative is to engage in some subterfuge. Given the human reaction times for both the human player and the NPC in the scene, we can delay and defer reactions for as much as 0.5s before players start to notice. If it makes sense for the NPC to be distracted, we can delay even for as much as a second or two (so long as there is some other form of feedback such as a sound or visual effect to confirm to the player that their hack took effect). This delay gave us the window we needed to preconfigure starts and stops.

Each action that the NPC takes is decomposed into a series of short animations on the order of 0.5 – 2 seconds in length, where the end pose of one animation is the start pose of the next. By chunking out each animation in this fashion, we've essentially discretized our reaction space. Now, we can defer reactions to the start of the next animation, and in each instance, we know the exact pose of the NPC. By mocapping reactions from that start pose, we can eliminate foot-sliding, bad hip positions, and other situations that lead to bad blends. This approach allows the player to initiate hacks at any point, provides reactivity on the scale of the animation segment lengths, but provides well-defined branch positions. Indeed, this pose-matching approach formed the cornerstone of our animation approach.

4.1 Pose Matching

To seamlessly stitch together animations, the ideal case is for a motion capture clip to start in a pose that matches the end-pose of the previous animation. This eliminates the need for a blend. Depending on the branches taken by the player, different animations can lead to the same position. Furthermore, different animations can play once an NPC is in a given animation. Since we don't know a priori the branches selected by the player, we potentially have to stitch together any of the leading and following animations at a given position. To do this, we introduce the concept of a shared *base pose*. As a rule, all animations leading to a given position must end in the base pose, and all animations starting from a position must begin in that same base pose. Figure 5 helps to illustrate the concept of a base pose.

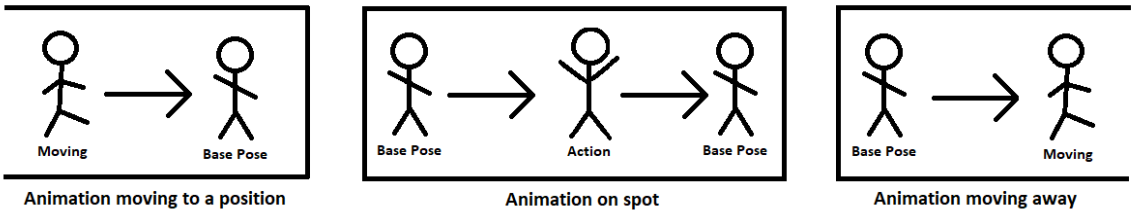

Figure 5 Base pose animation stitching.

Base poses are deliberately not neutral poses, and should not be thought of as such. Figure 6 shows the base pose for the NPC when attempting to fix the music in the 'Always On' IOP. Note that she is turned towards the remote on the table, and has her arm extended. This position was determined organically during mocap. The actor was instructed merely to move over and grab the remote – the arm extension came naturally. Usage of an organic base pose allows the actor to move in a way that makes sense to them, and dramatically improves performance quality. Once the base pose was chosen, the hack reactions at that position were captured using that pose as the starting pose. Similarly, all animations ending at that position were captured by having the actor end their motion in the base pose. This flowed well since the actor usually had to perform an action with the remote after the reaction, or else had just set it down and was retracting her arm.

Figure 6 The base pose when fixing the music in the 'Always On' IOP.

4.2 Reactions

Most IOPs start with the NPC in some kind of idle loop. NPCs could be sitting on the couch watching TV, dancing in place, running or a treadmill, and so on. This is a problem for our base pose solution. There's no realistic way to chunk out a long idle loop into a set of 1-2s animations that all start and end with the same base pose, while having it look at all natural.

The solution here was to take a more standard gameplay approach and employ blends. This was set up by directing performers to limit foot and hip movement during idles. Any reactions that could occur from idle would use that same foot and hip positioning, thereby preventing foot sliding or hip twisting. Basically, we have an alternate base pose, but only for the lower body, and it's consistent through the entire idle.

The upper body is in an unknown pose at the time of branching, and so we ensured all reactions from idle involved lots of upper body movement. It is generally safe to blend from an unknown arm and head position into an exaggerated or sudden arm swing. The big movement makes it hard to notice subtle irregularities in shoulder or head movement introduced by the blend. Through a combination of partial base poses from idles, and full base poses during reaction cycles, our pose-matching approach allowed for excellent reactivity during IOPs while still maintaining cinematic quality animations.

5 Statefulness in IOPs

Imagine an NPC picks up an object, and then as luck would have it, the player immediately triggers a branch. If we allow the branch, then we need to have an animation that includes the object. However, if the player had hacked a moment earlier, then the NPC would not have picked up the object and so an animation without the object is also needed. The end result is that the IOP has just been forked, and if the player keeps hacking, then the entire

rest of the IOP needs to handle the fork. The net effect is that the animation requirement would double, along with the mocap and animation workload. Indeed, the amount of animation required grows exponentially with the number of forks possible, and so it was imperative that we limit forking.

The brute force approach would have been to simply disallow statefulness in IOPs. However, this was too limiting for design and narrative. Instead, we found three useful solutions that allowed the usage of limited statefulness:

- *All Roads Lead to Rome*: If an NPC undergoes a state change in one path, then all possible paths need to do the same. This funnels the IOP back to a consistent state and resolves the fork. Figure 7 below shows the 'Child's Play' IOP were the NPC is wearing a headset. If the player hacks the headset, the NPC will remove it and a fork will be created. At this point, all other hacks will cause the NPC to end the call and remove the headset anyway, resolving the fork.

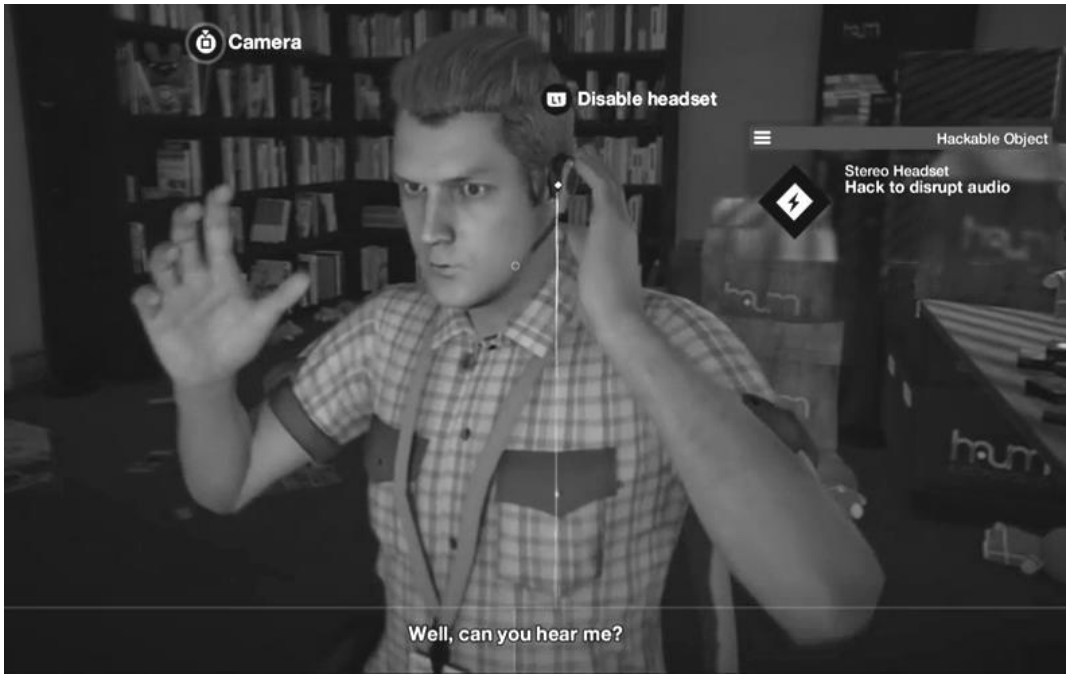

Figure 7 NPC With a Stateful Headset in the 'Child's Play' IOP.

- *Quick Like a Bunny*: The NPC changes state, acts quickly, then goes back to the original state. No Branching is possible during these brief, stateful periods. In the 'Always On' IOP, the dancing girl NPC picks up the remote, hits a button, and sets in back down, all in less than a second. The stateful action is allowed to occur, but no fork is created.
- *Noise? What Noise?:* This solution limits the scope of NPC reactions while the state change is active. The narrative is designed so that during a stateful hack, it makes sense for the NPCs to ignore that stimuli. Forks are only expensive when the NPCs react, since it is the reaction that requires animation support. This means that statefulness of this type can be employed without restriction.

As a rule, outro animations were not interruptible. Once an end state of the scene was reached, all player hacks were disabled or ignored. This was vital as it allowed for full narrative expression in the final state of the scene. Also, pose-matching was no longer necessary. Statefulness was also allowed at this point, since branching was no longer possible.

6 NPC Coordination

A major piece of the puzzle was ensuring that all NPCs could react in a coordinated fashion. Since the aim was cinematic quality, we needed to have highly realistic and coordinated reactions. This includes group synchronized reactions such as a gunshot causing all NPCs to react simultaneously, and NPC to NPC reactions where one NPC reacts specifically to the action of another NPC. The group coordination system must therefore be sophisticated enough to provide perfect synchronization when required, while allowing for desynchronizations and recovery.

The behavior of each NPC can be concisely described as an idle loop that is interrupted by a reaction-cycle branch. After the branch, the NPC either returns to the previous idle loop, advances to a new idle loop, or moves to an uninterruptible outro. If we consider branches to be triggered by generic events, then we can abstract away the different event types (player triggered, NPC triggered, timer based, etc.).

6.1 Group Behaviors

The group behavior system employed grew from the group behaviors pioneered by Martin Walsh for *Splinter Cell: Blacklist* [Walsh 15], with significant contributions from Sergio Ocio. The power of the system comes from having a highly modular, flexible structure, allowing us to modify and instantiate variations of the group behavior at run-time based on the game-state.

There are two levels to the group behavior system – the block level and the behavior level. The behavior level drives NPCs to perform specific actions – move here, play this animation, and other behavioral primitives. Each NPC in a behavior has a *behavior track*, which is connected series of actions that dictates how the NPC will act. Actions are connected acyclically and, similar to a petri-net, an action can only start when all inputs have been fired. An input to the next action can be fired when the current action starts or finishes, and each firing can be delayed with a customizable timer. This allows us to set up complex scenarios such as “face that NPC 0.5-0.7s after he starts talking”, then “play your response 0.8-1.3s after he completes, and at the same time, begin an animation”. When all actions are complete, the behavior is complete. A sample behavior is shown in Figure 8.

In IOPs, all animations are fully mocapped and pose-matched. A typical behavior track for a reaction would look like this: wait 0.3s then start a reaction animation; when that animation completes, immediately play a move; when the move completes, play a restore animation; when the restore completes, immediately play the return move.

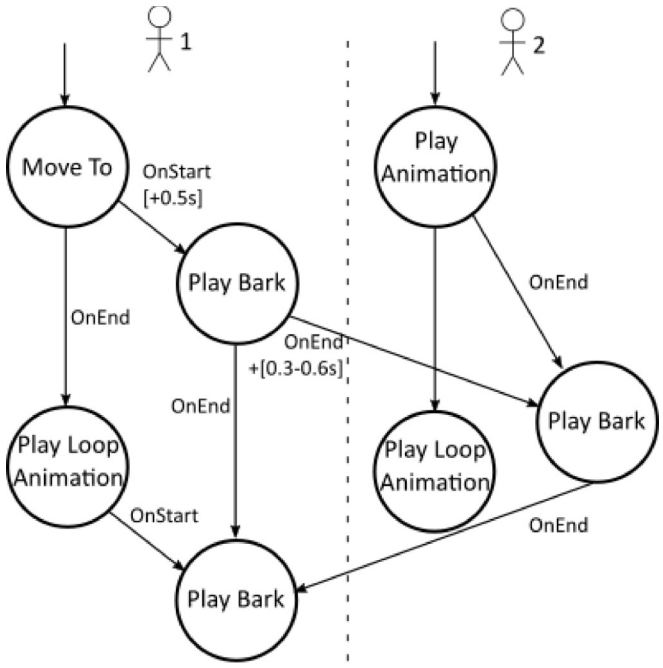

Figure 8 A behavior with two tracks, showing how actions can connect, cross tracks, and can connect with different timings.

The higher level is the *strategy* level. Strategies are composed of a set of connected strategy blocks, that each point to a set of behaviors that could potentially fill that block. Strategy blocks are connected in a simple fashion, where completion of the behavior in one block allows the next block to start. If multiple blocks precede a single block, then all of the blocks must complete before the next block starts. Similar to actions, blocks are connected acyclically. If multiple NPCs are in a block, then their behavior is synchronized by action connections that cross tracks. When a block completes, the strategy will move the NPCs into their next block or blocks, then start the contained behaviors. Figure 9 shows an example group behavior with two participants. There we can see how the NPCs can switch between synchronized blocks and independent blocks.

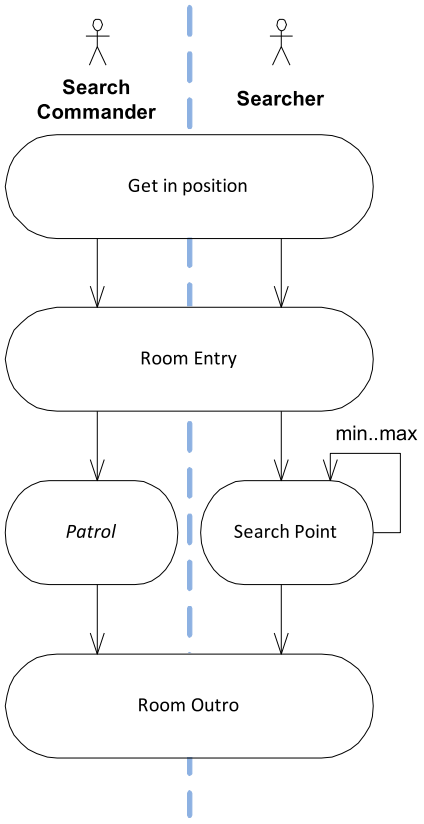

Figure 9 A sample group behavior strategy.

The connection between strategy blocks and behaviors is abstract, made through a *behavior category*. A block points to a behavior category, while behaviors are members of behavior categories. When a block is started, a matching behavior is selected to fill that block. Every behavior category must have a default behavior, which has no preconditions and therefore always matches. Other matching behaviors are called specializations and must have preconditions for the game state where the specialization is appropriate. In an IOP, a typical precondition would be the emotional state of the AI – "If the NPC is emotionally escalated, play the escalated specialization instead of the default". When the preconditions are met, the specialization is always selected over the default. In cases where multiple specializations exist, by construction we ensured that the preconditions are mutually exclusive. This rule was sufficient for IOPs, though it is easy to imagine other contexts that would benefit from a specialization priority system.

6.1 Branching in a Group Behavior

For a group behavior to branch, we need a new block that defines the branch behavior. The process involves creating a new branch block, inserting it dynamically into the strategy and creating appropriate connections to return when the branch is complete. A typical strategy specifies the list of events that can cause a branch, along with the associated behavior for each event. In IOPs, the events were usually things like 'camera hacked', 'timer expired',

and so on. Like blocks in the main strategy, a branch block points to a set of behaviors with one default and possibly others with preconditions. In the 'Always On' IOP, there was a single event for 'Music Hacked', and the preconditions would indicate the emotional escalation to ensure the appropriate reaction was selected.

Branches occur immediately when the event is received, so some is required at the behavior track level to ensure continuity. The behavior tracks in the branch block begin with special actions called a *container action*. At branch time, the current action of each NPC is taken out of the behavior track and placed into the container action in the branch. The settings of the container action define when to stop the current action. Typical options are to:

- Halt the current action immediately, most often when an NPC is hit and needs to perform an immediate hit reaction
- Halt the current action after a small timer, usually to add in reaction time delay
- Halt the current action after a synchronization event. An example of this would be when an NPC is unaware of an event, an only becomes aware when the other NPC yells at him. This would be accomplished through the use of an OnEnd event with a timer to the end of the PlayBark Action.

Once the container action is resolved, the behavior will play out in regular fashion. Upon completion the branch completes, the interrupted action is returned to its original behavior, where it is restarted, resumed, or discarded as appropriate. This allows us to resume behaviors part way through if desired.

Tracking the game state required the group behavior system to have a simple blackboard system. For example, the emotional escalation system used in many IOPs was tracked on this blackboard. In 'Always On', this allowed us to determine if a hack should cause an irritated reaction, an escalated reaction, or trigger the climax. A music hack would always trigger the music branch, but the information on the blackboard would tell us which music reaction was appropriate based on the emotional state. The branch block would then specialize to the game-state by substituting in the appropriate behavior.

Placing NPCs in single-participant blocks afforded us precise control in the timing of how individual NPCs react to different hacks. A great example of this comes from the 'Bad Publicity' IOP. In this scene, the main character is sitting at his computer playing an intense action game while bragging through his stream about illegal activities. He is listening to loud music through headphones and is wholly unaware of the world around him. Through a series of hacks, the player can call the police. The scene ends when the police burst in and arrest the main character, as shown in Fig. 10.

Figure 10 Police arrest the main character in the 'Bad Publicity' IOP.

Structurally, the main character is in his own behavior block, reacting only to the hacks that affect the game he is playing. This means that the main character and the police are completely independent. When the ending is triggered, we can have the police enter the premises without affecting the main NPCs behavior. Only when the police announce their presence does the NPC react with them. At this point, synchronization is required and so we branch the police and main character into the same finale block. But, he doesn't actually terminate his behavior when the room entrance block is started. Instead, he has a short hold on his previous behavior, and only begins his reaction after 30 frames when the door is kicked in. His outro animation skips the first 30 frames to account for the gap. When he finally begins his reaction, the animations are correctly synchronized, and so the cops can proceed to do a synched takedown animation and put the main character in handcuffs.

7 Narrative Expression

In a typical cinematic, expositional dialog is a core tool in expressing narrative. The exact ordering of the scene (including speech interruptions) is planned out in the script. Since there are branches, the writer can easily ensure that all the important narrative beats are hit.

The situation is different in IOPs – the writer can no longer make strong assumptions about ordering and narrative flow. This doesn't obviate the need for a strong narrative, and so we needed to come up with a narrative structure that was resilient to branching. For dialog flow to make sense, the writers needed to know when certain beats were hit. The simplest solution is that if an important line gets interrupted, replay after the interruption to ensure narrative flow. In a cinematic quality scene, hearing the same line with the same intonation feels very artificial and robotic. Another option would be to skip to the next line, but this risks losing the context if an important narrative beat is skipped. Instead, we had to take a more fine-grained approach that directly encapsulates narrative beats.

Consider the following lines from the 'Haum Intruder' IOP where one NPC is trying

to make a sale:

```
COLE: Prices are going up, Grizz.
COLE: This is exclusive material I'm providing.
```

The purpose of this beat is to establish that Cole is taking an aggressive bargaining position. The remainder of the scene involves Grizz trying to get the price lower. If the line about prices going up is skipped, then the narrative becomes muddled, while the second line is flavorful but doesn't set the scene. Here, we introduce the concept of a *Point of No Return* (PONR). Each dialog block was given a PONR, set no more than 1.5 seconds after the beginning of the block:

```
COLE: Prices are going up (PONR), Grizz.
COLE: This is exclusive material I'm providing.
```

For this to work, the main narrative beat of the dialog block had to be front loaded and placed *before* the PONR. This forced the narrative team to rewrite part of the script to meet that requirement, but with that done, the functionality becomes straight-forward. If a branch is triggered before the PONR, it is safe for the NPC to repeat the entire dialog – since the PONR is at most 1.5s into the line, the player has not heard enough of the dialog to make repetition feel artificial. If the branch is after the PONR, skip the remainder of that beat – the structure of the writing has ensured that the player has heard the important narrative information. The entire conversation is then broken up into a series of consecutive, interruptible narrative beats with PONRs placed in each one. This results in setups like the following, where the high price is front-loaded to ensure narrative delivery:

```
COLE: I could get four K for this kid. (PONR)
COLE: There's an Exec in the valley who doesn't nickel
and dime me.
GRIZZ: Look, I'm just, feeling squeezed, but I'm
definitely interested.
```

After interruptions, we needed to smooth out the return from the reaction to the main dialog. In this case, the restore step in the reaction cycle was actually restoring the dialog flow. Each reaction ended with a rejoin that was pose matched to the default dialog pose. Included in this was a generic rejoin dialog:

```
COLE: Lost my fucking train of thought...
GRIZZ: You were talking about bankrupting me...
COLE: Yeah, yeah, yeah...
```

Interactions that were not tied to NPCs provided an alternate vehicle for narrative. In the 'Condemned' IOP, there was a phone that could be hacked to play a message. Once triggered, this would play the message in its entirety, regardless of the other hacks that were taking place.

Of course, the sledgehammer approach was to prevent branching entirely. It restricts gameplay and was always the choice of last resort, but by using this sparingly, it allowed certain critical sections of the narrative to be expressed in an uninterruptable format. Mechanically, it meant either locking out the player from triggering a branch, or having the NPCs ignore hacks during that time. This was rarely done, as it violated our philosophy of being as permissive as possible with respect to hack timing.

8 Conclusion

Cinematic gameplay is highly constraining. The cost associated with performance capture makes it very challenging to come up with a large set of animations. Through the use of pose-matching techniques, we were able to create the illusion of freedom, and ensure that all branches and reactions were seamless. Depending on the exact requirements in your game, it might be possible to use some of the same techniques to create a similar illusion.

As we have discussed throughout the paper, many of the difficulties we faced could be eliminated through the use of systemic animations. However, this greatly impacts the animation and overall quality. Trying to develop a high-quality system with systemic animations would certainly be an interesting challenge.

In either case, a way to manage group interactions is a requirement. The group behavior system that we employed certainly offers one solution to this problem. By maintaining a highly flexible block structure with individual behavior tracks, we suffer some overhead costs, but this is more than compensated for by the incredible amount of fine control over group interactions. Qualitatively, we found it much less taxing to use a top-down approach and reason at the group level, rather than trying to make proper group interactions an emergent property of individual NPCs.

9 References

[Rabin 15] Rabin, S. 2015. Agent Reaction Time. In *Game AI Pro 2: Collected Wisdom of Game AI Professionals*, ed. S. Rabin 31-34. Boca Raton, FL: A K Peters/CRC Press.

[Walsh 15] Walsh, M, 2015. Modeling Perception and Awareness in Tom Clancy's Splinter Cell Blacklist. In *Game AI Pro 2: Collected Wisdom of Game AI Professionals*, ed. Steve Rabin 313–326. Boca Raton, FL: A K Peters/CRC Press.