7

But, It Worked on My Machine! How to Build Robust AI for Your Game

Sergio Ocio Barriales

7.1 Introduction

Every AI programmer has been through this stage at some point in their careers: you are young, full of energy, and keen to prove to everyone that you can make great things happen—and that you can do it fast! You finish your feature, test it on your map, and now it is in the game. Job done. Or is it?

Effective testing tries to ensure our AI meets design requirements and reacts and handles unforeseen situations properly. Experience helps you identify common pitfalls and errors. Seasoned AI developers can identify these and work with sets and categories of tests that try to catch problems, making their systems more robust. In this chapter, we show some of these strategies, analyze different scenarios, and provide tricks to help AI engineers approach AI testing with some inside knowledge.

7.2 Presenting Our Example

Debugging can be a very abstract topic, so, in order to make this chapter easier to read, we will present an example problem and use it to illustrate the different tests we can perform during the creation of an AI character. Let us use a sniper archetype, an enemy with increased accuracy when using their sniper rifle, which comes equipped with a scope and a laser sight. The sniper also carries a side pistol for close encounters. This role can be found in many action games, and they generally show similar behaviors. Our snipers will have three main states, *precombat*, *combat*, and *search*, as depicted in Figure 7.1.

Snipers can go into combat if they spot the enemy. Once there, they can go to search if the target is lost and, potentially, return to combat if the enemy is redetected. Snipers cannot go back to precombat after leaving this state.

7.2.1 Precombat

During precombat, our snipers will patrol between n points, stopping at each of them. There they sweep the area with their rifle. In this state, the sniper is completely unaware of the presence of any enemy.

7.2.2 Combat

When snipers are in combat, they have two options, depending on range to the enemy. At long distances, snipers will use their rifle to target and take down their enemy. In close quarters, snipers will revert to a side pistol and act like any other regular AI archetype in a cover shooter game.

7.2.3 Search

If the sniper loses line-of-sight to the enemy during combat, the sniper will go into the search state. In the search state, the sniper first confirms that the enemy is not at their last known position. During search, the sniper can sweep areas with his or her rifle and/or check the immediate region on foot if the enemy was last spotted nearby.

7.3 Basic Functionality Tests

After we implement a first version of our sniper, the first thing we have to verify is that the archetype works as described in the different design documents we have been working from. Please note that in a real-world scenario, we would not implement the whole enemy AI before we start testing it, but instead follow an iterative process. The tests described in this chapter—especially the basic functionality tests—should be applied to smaller parts as we develop our full feature. When we confirm that the sniper works as intended, our

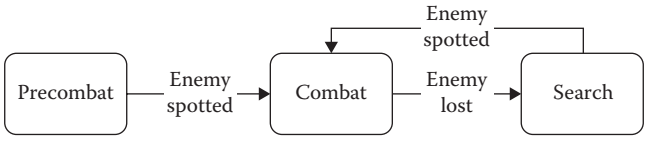

Figure 7.1

Base FSM that will control our AI behavior.

next job is to try and break the character in every possible way. This is the hardest part of testing an AI character, as every simple change can have unforeseen repercussions in other systems.

In the following sections, we will focus on the different parts that make up our AI agent, such as perception, reactions, and navigation, and will show some potential tests that are relevant for each case. While we will use the sniper to illustrate the process, our goal is to provide readers with enough information so they can identify similar problems on their own AI characters, and we can help them focus their testing efforts in areas that experience has shown us need special care. So let us get down to business!

7.4 Perception Stress Tests

When we talk about perception, we refer to all the systems that make the AI aware of its surroundings, and particularly those that allow it to detect other entities, such as the player. Traditionally, this includes the vision and hearing of the AI character, but it could potentially involve other senses, depending on the type of game and/or AI we are working on (e.g., smell tracking for dogs).

Let us focus on the vision system. For this, our sniper probably uses a few cones or boxes around their head that help them determine what enemies can be seen. As part of our basic functionality testing, we should have already checked that the player—or any other enemy—can indeed be detected, but we want to edge-case test this.

The sniper has two perception modes, depending on the weapon it is using: sniper rifle and side pistol, so we have to test them separately.

7.4.1 Rifle Perception Testing

Our sniper rifle has a laser sight, which is very common in games with this type of enemies. The laser is normally used as a way to telegraph both that a sniper is in the area and what the sniper is currently looking at.

What is the length of this laser? In the real world, design will probably have decided this for us, so let us say it is 60 m. The sniper is looking around an area through his or her sights, and we decide to pop right in front of its laser at 59 m. But … nothing happens. Why?

The most common answer to this problem is that our vision range does not match the expectations set by the laser. Should we increase this range for the sniper? The problem is shown in Figure 7.2.

The danger of extending the vision cone is that we will not be only extending its length, but also its width, if we want to maintain the same FOV. This, of course, will depend on the actual implementation of our vision system, but let us assume that we use vision cones for simplicity. This vision range problem raises some more questions. For example, if the sniper is looking through his or her rifle's sight, should he or she actually be able to focus on things that are on the periphery of its cone?

After further testing and discussions with design, we could decide to modify the vision of the sniper and have a shorter cone and a box that surrounds the laser, as shown in Figure 7.3, and our problems seem to have gone away.

We could do some additional testing to make sure all the possible target distances work as expected, and we should also check that the sniper will not detect enemies that are not inside the areas defined by its vision model.

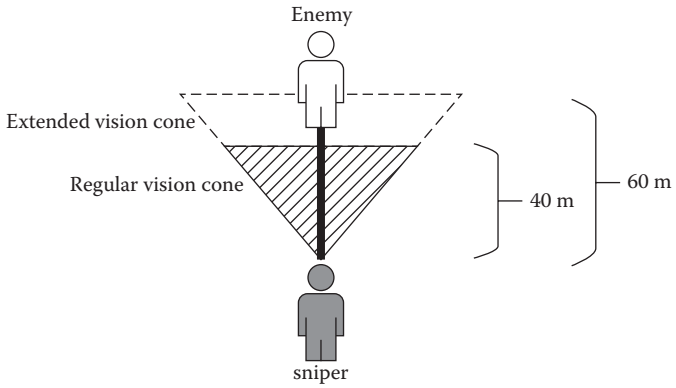

Figure 7.2

If the sniper's laser range is longer than their vision range, the enemy will not be detected.

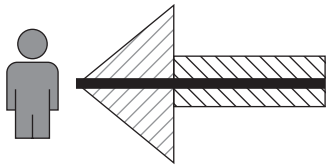

Figure 7.3

Upgraded vision model for the sniper.

7.4.2 Pistol Perception Testing

The sniper can be in a *regular guard* mode when using its pistol, so we retest regular perception. Everything looks ok, but suddenly we notice that the sniper detects things that are very far away. Why? Well, the answer in this case is that the new vision model we introduced to fix the perception problems we had with the rifle should not be used if the sniper is not looking through the rifle! We make the change, and everything looks good now. Perception works!

7.5 Reaction Stress Tests

A good AI character should be able to react to multiple different stimuli in plausible ways. Not reacting or playing a wrong reaction can break the immersion of the player very easily. Reaction tests look mainly for realization problems, that is, reactions that do not look good or that break the believability of our AI character. Here we focus on trying to break the AI reactions. Let us look at some reaction types in more detail.

7.5.1 Spotting the Enemy

The AI should be able to detect the enemy in any state, and we checked this by applying the tests presented in the previous section. What we are trying to double check now is that the sniper

plays an appropriate reaction behavior (e.g., barks and/or animations) when the detection happens. For this, we should try to test every possible combination of sniper state (i.e., sniper rifle equipped vs. pistol equipped) over a few distance ranges (e.g., long, medium, and short distance). At long/medium distances, most of our reactions will probably work pretty well: the AI will play a reaction animation and some barks, and a transition will happen. At short distances though, things start getting more interesting.

The first thing we have to test is if our animations make sense at close quarters. We probably should not play the same animation when the AI sees the enemy at 40 m, right in front of the AI, or if the enemy just bumped into the AI. A *surprise* behavior, probably a more exaggerated animation, should be used in the latter cases. Another potential problem is that the sniper could be showing slow reaction times when the rifle is equipped—as per our design, the sniper uses its pistol when in close proximity. Switching weapons after the reaction is complete can look robotic and will be slower, so the best solution probably involves adding some special case animations that bake the weapon swap into the reaction.

7.5.2 Shooting at the AI

We should also test that the AI is reacting when it is shot at. We need to check that the AI will always react to this—although the reaction can be subtler in full combat. We should also check that our different animations look believable depending on what weapon was used to hit the AI, how hard it was hit, what body part was affected, and so on. The system for these reactions can be more or less complex depending on the game.

7.5.3 Reacting to Nondamage Events

If we do not shoot directly at the AI, but start doing things such as shooting at the ground nearby, whistling or throwing a rock, our sniper needs to react properly to all these different stimuli. Depending on the type of event and how aggressive it is (e.g., loud gun bullet whiz vs. a light switch turned off), the sniper should react in different ways. The objective of our tests is making sure that the reactions follow the design and that they make sense. Some of the tests we could try are:

- Is the AI able to react to two events in a row?
- What if those events are of the same type? Are we playing the same reaction over and over? In this case, we may want to add some antiexploit code perhaps to escalate the AI reaction and potentially for transition into a different state.
- Can the AI handle a second event while it is still reacting to a previous one?
- Will the AI finish the current reaction before reacting to the second stimulus? Does this look broken?
- If the second event is of a lower priority (e.g., rock throw vs. bullet impact), it should probably ignore the second event.
- If, contrariwise, the second event is of a higher priority, is this new event handled right away?

- Is it realistic for the AI to be able to react to a second event while reacting to the first one?
- Is the AI reacting appropriately based on the distance to the event? For example, the sniper should probably look more agitated if it hears a gunshot right behind it than if the sound comes from 50 m away.

7.6 Targeting Stress Tests

Now that we have double checked that our AI can detect enemies and react to whatever they do, it is time to make sure it can handle being attacked by multiple targets or from different, unexpected directions properly. Depending on the game and the implementation, AIs can use the actual position of the enemy to select their behaviors, or they can keep a last known position (LKP) (Champandard 2009). LKP gets updated by the individual AI perception system or by perception shared with a group or squad of AIs. To illustrate these tests better, we will say our AI uses an LKP. We will not enter into details about how an LKP system works, and we will simplify some aspects for our examples.

With our tests, we want to validate two things: first, we need to know the LKP is being set appropriately when the player is detected and updated as necessary. Second, we need to know the LKP is propagated to other AIs at the right time.

7.6.1 LKP-Updating Tests

When the sniper detects an enemy, it will react and then go into combat, setting an LKP and updating it as long as line-of-sight remains unbroken. The best way to test that the LKP is being created or updated to a new position after a big event happens (such as the enemy being spotted by the AI) is teleporting an enemy to random positions around the AI and shooting a loud gun in the air (or using any other event that causes a transition into combat, based on the game's design): when an LKP is set, we move that enemy to another position, trigger the event again, and continue doing this for a few seconds, making sure the LKP is being updated accordingly.

To test the above in a normal gameplay scenario, we should probably make the enemy invincible, create an LKP, and start circling around an AI. It is possible that the AI can lose us during this process, which would indicate a problem. In real life, if someone is running in circles around ourselves, we will probably notice it—so should our AI! This loss can be caused by an LKP update that is too tightly coupled to the vision system. In order to fix this, we could add some time buffering so that vision has to be broken for a few seconds before the enemy is considered as lost. We could also consider the AI is actually seeing the enemy if the vision tests fail, but the enemy was visible before, and the AI can still ray-cast to them, which means they have not been able to occlude their position. We probably need a combination of both systems for best results.

We should also make sure that any detection HUD our game uses works as intended. Such systems communicate to players that they are in the process of being spotted, that they are fully detected, or that the AI has just lost them. Changes in the AI targeting code need to be cross-tested against the HUD as well.

7.6.2 LKP Propagation Tests

Stealth mechanics are becoming more important in mainstream action games, and thus everything that is related to perception, reaction, and player position communication

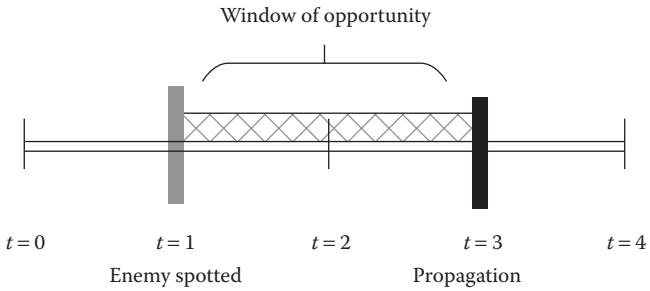

Figure 7.4

A window of opportunity allows enemies to deal with their AI spotters before the LKP is propagated.

should be thoroughly tested. LKP propagation is the typical system an individual AI uses to communicate to other AIs in an area or level that there is a threat, and where this threat is located. We need to double check that propagation is actually happening, but we also have to ensure LKPs will not get propagated during a reaction. The goal of this delay is to give the enemy a window of opportunity to eliminate the potential problem (e.g., by killing the detector) before their position is compromised after the detection, as shown in Figure 7.4.

7.7 Unreachability Stress Tests

Once LKPs are being set and updated correctly, we have to go a step forward in our tests and start checking how the AI behaves if the enemy was spotted in a location the AI cannot reach. We also need to check how the AI behaves if the enemy was spotted in a location the AI cannot see. What we are looking for is, at the very least, a believable behavior.

If the AI can see an unreachable enemy, we want the AI to move to a good position that still gives it line-of-sight on the enemy, and try to shoot from there. With this in mind, our first set of tests should be able to answer these questions:

- Is the AI stuck if the LKP is unreachable?
- Does the AI go into combat if it is shot at from an unreachable location?
- Is the AI staying in combat if the LKP is visible but unreachable?

The AI should also look good if the unreachable position is not visible. In this case, we probably want the AI to hold its position and get ready to defend itself from enemies trying to take advantage of the situation. After making sure the AI is not stuck in this situation either, we could perform some extra tests, such as:

- Is the AI aiming through walls directly at the LKP? We should avoid this, and systems like the Tactical Environment Awareness System (Walsh 2015) could help us decide what locations the AIs should be targeting.
- Is the AI aiming at a sensible target?

- Is the AI stuck in perpetual combat if the enemy is never redetected and the LKP stays unreachable and invisible? This is a question that needs to be answered by game design, but we probably want to time out in the LKP that allow our AI to recover from this situation.
- If there is a position the AI could move to gain LOS on the enemy, is the AI moving to it?

7.8 Navigation Stress Tests

An AI character in a cover shooter action game normally spends all its life doing one of the two things: playing animations and navigating from point to point. Bad navigation can suspend immersion quickly, so spending some extra time making sure it looks nice and debugged will pay off.

Going back to the sniper example, our design says that, in precombat, the sniper patrols from point A to B stops at the destination and does a sweep between a set of points associated to this vantage point. So let us go and follow a sniper and see what it does.

The sniper is at point A, finishes scanning and starts moving toward point B, walking with its rifle in its hands. Our two-hand walk cycles are probably tuned for smaller weapons—again, depending on the game and our animation budget—so the way the sniper moves could look a bit cartoony. An option to fix this is having the sniper stow the rifle before moving to a new point and either moving with pistol in hand or unarmed. Our job is to find the best-looking results with the tools and resources we have at hand.

Another thing to test is navigation during combat. We have a few candidates for buggy or undesirable scenarios here:

- Can the AI run and take cover properly with the currently equipped weapon?
- Can the AI cancel a move if the enemy moves and gets in the way?
- When/how do we detect this?
- Does the AI choose a new destination and never stop moving? Does it just stop in place and hold position? Does it play some sort of reaction animation before deciding to move to a new point?
- If the AI navigates near an enemy, will the AI look at the enemy?
- Should it strafe looking at the enemy? Should it aim at the enemy?
- Is the AI moving at the appropriate speed?
- Is the AI using traversals? (e.g., ladders, or being able to jump over gaps or obstacles)
- If there are two doors to enter a room, and there is another AI using one of the doors already, does the AI under test choose the other door?
- Can the AI open doors in its path?
- What does the AI do if it ends up outside the limits of the navigation mesh?
- If the game has different levels of detail, is the AI navigation properly on low LOD?

We should also try to do reaction testing when the AI is navigating to try and see if we can break it. Focusing on transition points is probably the best way to do this. For example:

- If the AI is entering a cover position and we expose the cover, how does it react? Is the behavior to enter the cover broken?
- If the AI is using a traversal in precombat and we shoot a single shot at it, does the traversal animation end? Does the AI fall and end up off-navmesh?

7.9 Weapon-Handling Stress Tests

When our AI can use multiple weapons and switch between them at any point, just as our sniper does between rifle and pistol, we need to make sure swapping weapons is not broken. Particularly, we want to try and avoid ending up with an improperly unarmed AI. Let us go back to our sniper. We made some changes in the previous section to allow it to navigate without a gun in precombat, so now we have three possible scenarios: the sniper is holding a rifle, the sniper is using a pistol, and the sniper is unarmed.

The simplest tests are just to check basic functionality, that is, make sure weapon transitions work as expected when the AI is not disrupted. However, the most interesting ones require trying to break the weapon swaps. For our particular example, we would like to test the following:

- The sniper is about to move from point A to B in precombat, so it starts stowing the rifle. If an enemy shoots it midway through the animation: does the AI play the hit reaction? Does it drop the rifle? Does it stow it at all? Do we let the animation finish?
- If the sniper does not have a weapon and gets shot, does it equip a weapon after reacting? Which one? Also, as we mentioned before, we should probably have reaction animations that bake in the equipping of a weapon to make reactions snappier.
- If the sniper has just reached point B and starts grabbing its rifle, what does it do if it gets shot?
- If the sniper detects an enemy nearby and starts stowing the rifle in order to draw its pistol and the enemy shoots the AI, what happens?
- Consider when the sniper gives up after the enemy runs away in combat. The AI tries to go into search, putting its pistol away and trying to equip the rifle. What does the AI do if it gets shot in the middle of the process?

Other interesting tests have to do with weapon reloading. The main questions we want to answer in this case are, if the AI gets shot in the middle of a reload:

- Is the animation played completely? Is the animation interrupted?
- Is the weapon reloaded at all?

7.10 Exotic Elements Stress Tests

Some of the different systems that we use to build our characters can be shared between different archetypes or reused among different types of characters with some minor modifications or tweaks. However, other parts are inheritably specific to certain characters, like, for example, the sniper rifle. These are the exotic elements that define what makes our

character different from others, and thus we need to put extra work in making sure they work well and look polished. Defining these tests in advance can be difficult, due to the uniqueness of some of these components but the main goal, as it has been the trend in this chapter, is testing and making sure the feature cannot be broken.

As an example, for the sniper rifle's laser, we probably would like to test the following:

- Is the laser coming out the right location on the weapon?
- Is the laser following the weapon when it moves? Is the laser moving independently from the weapon?
- If it is, is the deviation noticeable?
- Is there a reason why the laser is allowed to move independently?
- Does the laser look good in every situation? For example, what should we do with the laser when the sniper is stowing the weapon? Should it be turned off? Same question for the sniper playing a reaction to a big hit—is the movement of the laser acceptable?

7.11 Group and Timing Stress Tests

At this point, our AI should be robust, fun, and believable when it operates as an individual unit. But what happens if we start using our character in conjunction with other characters of the same and/or different types? Tests that we could want to perform are:

- If a group of AIs receive an event, are all of them reacting to the event in perfect synchrony? We may want to add some small, random delays to the reactions to smooth things.
- Are different types of AI working well together? Is there any type that is breaking behaviors for others? For example, our sniper can see from longer distances if the enemy is still at the LKP, potentially put the other AIs in the level into search before they have been able to get past their reaction behaviors. We may want to control these interactions if we consider they are affecting the game negatively. For example, in the case of the ultrafast LKP validation sniper, we can prevent snipers from validating LKPs if there is any other nonsniper unit in the level that can reach the enemy position.

7.12 Conclusion

Building good character AI requires going the extra mile. Making sure our characters follow the design of the game is the first step, but long hours of work tweaking and testing should follow our initial implementation. Having a good eye to find problems or knowing how to polish and take features to the next level is an important skill that every AI developer should have, and it is a skill we all keep improving with time and practice, discussions, and advice.

Readers may have noticed all the tests presented in this chapter follow some progression. We start by testing if things are correct if left undisturbed, and increasingly generate problems and poke the AI in different ways to check how the system handles these interruptions: How does the AI react when it is interrupted? Like, for example, the AI getting shot at while it is swapping weapons. What happens if we ask it to do two

different things at the same time? For instance, when the AI is reacting to a small event and receives a second one. Alongside, we continuously are on the lookout for unintended consequences—such as snipers detecting players at extended range after we modify their vision boxes—and always ensuring that every system, even if it is not directly connected to our change, is still functioning as expected.

It is our hope that readers have understood the problem of testing character AI thoroughly and that the techniques and examples presented in this chapter have served as a starting point to approach the testing challenges they may face in their own work.

References

Champandard, A. J. Visualizing the player's Last Known Position in Splinter Cell Conviction. http://aigamedev.com/insider/discussion/last-known-position/(accessed May 15, 2016).

Walsh, M. 2015. Modelling perception and awareness in Tom Clancy's Splinter Cell Blacklist. In *Game AI Pro 2: Collected Wisdom of Game AI Professionals*, ed. S. Rabin. Boca Raton, FL: A K Peters/CRC Press, p. 313.