28

Beyond the Kung-Fu Circle

A Flexible System for Managing NPC Attacks

Michael Dawe

28.1 Introduction

Action games featuring real-time combat have an interesting balance to strike when approaching difficulty and player challenge. Frequently, such games may have encounters featuring several adversaries in a group attacking the player, often with differing types of attacks. This can be desirable from a game design standpoint, for reasons ranging from total combat duration to narrative requests. However, inexperienced players may become overwhelmed by the sheer number of enemies attacking simultaneously, and so these games typically restrict opponents to attacking one at a time.

For *Kingdoms of Amalur: Reckoning*, the game design called for an authentic action game inside a traditional RPG setting, complete with a large number of enemies taking on the player at once. After evaluating several approaches of restricting attackers within a combat encounter, we used a technique capable of changing the number of attackers and even the types of attacks allowed against the player dynamically based on the attackers themselves and the player's chosen difficulty level.

28.2 The Kung-Fu Circle

Requiring that opponents attack the player one at a time is a technique known as the Kung-Fu Circle, named after classic scenes from martial arts movies in which the protagonist faces off against dozens of foes who launch their attacks one at a time. This is a simple algorithm to write and employ, and can make sense in games where focusing on a single opponent is desirable. However, this restriction may be too strict for a game with quick-flowing combat. To allow for a faster pace of combat and to give the player opportunities to use wider-affecting area attacks, it is advantageous to ease these restrictions and allow more than one enemy to attack at a time. A naïve approach might be to simply allow two or three enemies to attack simultaneously, but this will not account for different enemy types or the relative strength of their attacks. For *Reckoning*, we employed an approach known internally as the "Belgian AI" system (so-called for the iconic sketches of waffles used to describe the algorithm) to allow the combat team to design encounters utilizing a variety of creatures while always employing the same underlying rules to determine the number and types of creatures allowed to attack at the same time.

28.3 Belgian AI

At a high level, the Belgian AI algorithm is built around the idea of a grid carried around with every creature in the game. While every NPC had a grid for itself, in practice the player is the game entity we are most concerned about, so we will use the player as our example throughout this article. The grid is world-space aligned and centered on the player with eight empty slots for attacking creatures, much like a tic-tac-toe board with the player at the center. In addition to the physical location of those slots, the grid stores two variables: *grid capacity* and *attack capacity*. Grid capacity will work to place a limit on the number of creatures that can attack the player at once, while attack capacity will limit the number and types of attacks that they can use.

Every creature in the game is assigned a *grid weight*, which is the cost for that creature to be assigned a spot on someone's grid. The total grid weight of the creatures attacking a character must be less than that character's grid capacity. Similarly, every attack has an *attack weight*, and the total weight of all attacks being used against a character at any point in time must be less than that character's attack capacity.

The easiest way to show the impact of these variables is to describe an example of them in action. Let's look at a situation where an enemy soldier has become aware of the player and decides to launch an attack. Before doing so, the soldier needs to request a spot from which he can attack the player. For *Reckoning*, we kept all spatial reasoning out of the individual creatures and had a centralized AI responsible for handling the positioning of creatures in battle called the *stage manager*. Our soldier will therefore register a request to attack the player with the stage manager and wait to be assigned a spot. On its next update, the stage manager will process the request and compare the soldier's grid weight against the player's current grid capacity. For our example, we'll say that the soldier's grid weight is 4 and the player's grid capacity is 12. Since the grid weight of the soldier is less than the available grid capacity, the stage manager will assign the soldier the closest available grid position and reduce the player's available grid capacity to 8. The soldier now has

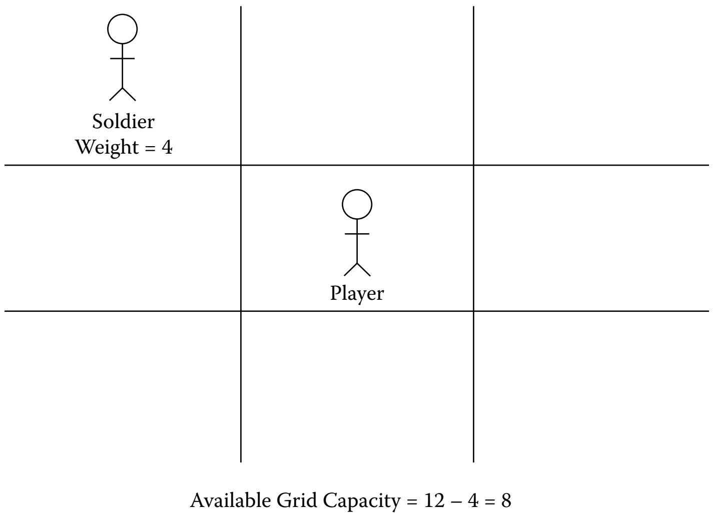

Figure 28.1

A soldier is granted permission to attack the player and takes up position on the grid, reducing the player's available grid capacity.

permission to approach the player and launch an attack. Figure 28.1 shows an example of what the grid might look like at this point.

Next, suppose a troll notices the player. Similarly to the soldier, the troll must request of the stage manager a position on the grid, but since trolls are larger and more powerful than mere humans a troll's grid weight is 8, double that of our ordinary soldier. Since that's still within the player's available grid capacity, the stage manager can assign a slot to the troll and reduce the player's available grid capacity to 0. Now when another soldier comes along (Figure 28.2), he can request a spot to attack the player, but the stage manager will not assign him a slot, since his grid weight is larger than the player's available grid capacity. This second soldier cannot approach the player and must wait outside the grid area.

Attack capacity and weight work similarly, but for individual attacks. So far in our example, both a troll and a soldier have been granted permission to attack the player, but they haven't picked out an attack to use yet. Suppose the player's attack capacity is 10. The troll may have two attacks to pick from: a strong charge attack with an attack weight of 6, and a weaker club attack with a weight of 4. Since the player's attack capacity is 10, the troll can pick the charge attack and inform the stage manager, which reduces the player's attack capacity to 4. Now the soldier picks from his attacks: a lunge with cost 5 or a sword swing with cost 3. Since the player's current attack capacity is 4, he's unable to use the lunge attack, so he'll have to settle for the sword swing this time.

28.3.1 Grid Sectors, Inner Circles, and Outer Circles

Though the algorithm often refers to grid positions, it is more natural to think of the grid slots as defining positions equidistant from the player, forming a circle some arbitrary distance away. For Reckoning, we further subdivided the grid into inner and outer circles, which we called the "attack" and "approach" circles. The radius of the attack circle

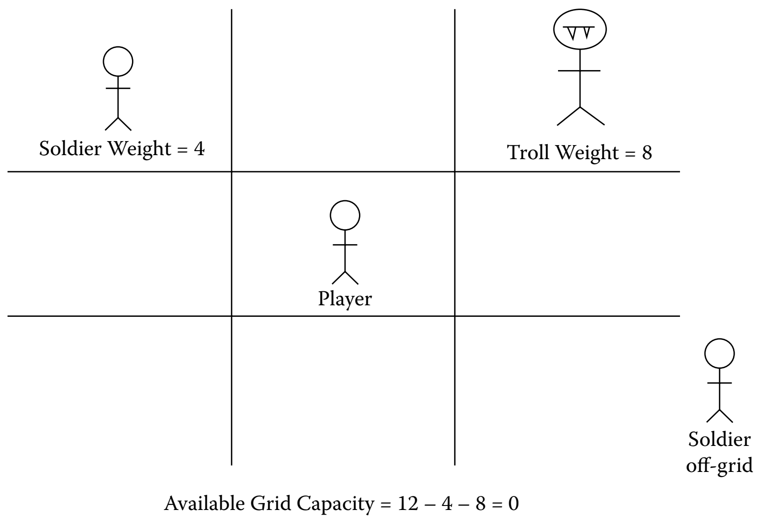

Figure 28.2

A troll is also granted permission to attack, bringing the grid capacity down to 0. The second soldier is not allowed to approach the player to attack.

is defined by the minimum and maximum distances for melee attacks. When a creature first passes the grid capacity check and is granted permission to approach the player, it will move to stand inside the approach circle. When it is granted permission to perform a particular attack (such as the troll's charge or the soldier's sword swing), it will move into the attack circle to do so. Meanwhile, characters that have not yet passed the grid capacity check (such as the second soldier in our previous example) will stand outside the approach circle, awaiting their chance to step in. This helps the player to determine which creatures are immediate melee threats. Figure 28.3 shows what it might look like.

28.4 Behavior Integration

To fully take advantage of the grid system, creatures in *Reckoning* had a series of specific common behaviors to enforce positioning and attack rules. This also increased behavior reuse, as any creature that could use melee attacks would share these common behavior sets.

First, any creature that was within the distance defined by the outer circle but without permission to attack had to leave the circle as quickly as possible. This helped make it clear to the player which creatures were attacking. Additionally, it prevented monsters from getting in the way of each other when making their actual attacks.

Creatures waiting on the outside of the grid but not assigned grid slots were not given permission to approach, but would instead be given the location of a slot on the grid that was not currently occupied. The creatures would attempt to stand in front of their given slots, even if they did not have permission to attack, which made for natural flanking behavior without any creature explicitly needing to know about any other creature in combat.

Finally, creatures would relinquish control of their grid slot back to the stage manager immediately after launching an attack. Thus, the stage manager could queue requests for

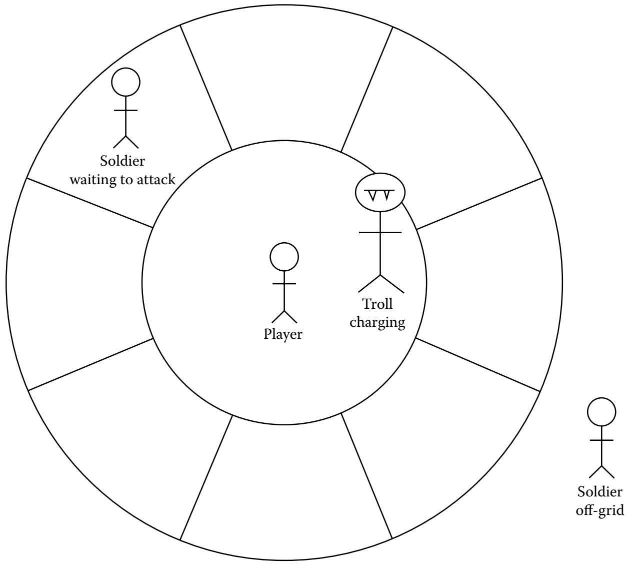

Figure 28.3

The inner- and outer-circle approach. Creatures assigned a grid slot move to their assigned section on the outer circle, but only creatures launching attacks are allowed inside the innermost circle. Any creature not assigned a slot must wait outside the outer circle.

attack slots and rotate the creatures allowed to attack throughout a combat encounter. This kept creatures moving into and out of immediate melee range of the player and ensured no single creature could monopolize attack opportunities. In the case of encounters with a small number of creatures, any creature that gave up its position after making an attack could immediately be assigned a slot again.

28.5 Scaling Difficulty

Grid capacity and attack capacity work together to limit both the number and types of creatures and attacks that can be thrown at the player at any given time. One advantage of this system is the immediate difficulty scaling that can come out of changing the player's grid capacity and attack capacity, all without changing the carefully balanced individual grid and attack weight set for each creature. As the player increased the difficulty in *Reckoning*, we scaled up the grid and attack capacities accordingly. The change to grid capacity allowed more creatures to surround the player during combat, while the change to attack capacity allowed more of those creatures to attack simultaneously and to use more powerful attacks as well.

28.6 Additional Concerns

While the Belgian AI is straightforward enough, care needs to be taken to ensure that it works well with a fast-paced action combat game. Additionally, there may be other design goals concerning creatures and their attacks that the system can be adapted to.

28.6.1 Maintaining Grid Assignments in the World

As mentioned previously the grid is world-oriented, so slots on the grid move around with the player. Sometimes, this would lend itself to situations where the slot that creatures had been assigned would no longer be the closest slot to them. This would frequently occur when the player rolled out of the way of an attack towards other creatures.

To solve this, we gave the stage manager full control over the assignment of the slots on the grid. On any given frame, it could decide which creatures to assign to slots in order to make the best use of the player's grid capacity. In the case of an empty grid, creatures could be assigned the slot closest to their current position. If creatures were already on the grid, the stage manager could assign a slot that was not the closest, or perhaps shift assignments of creatures already granted slots to make room for the newcomers. Creatures would never remember their grid slot assignments and were responsible for checking with the stage manager to get their current slot assignments every frame.

Creatures that had previously been determined to be in the best position to attack could also suddenly not be the best choice to attack the player. For a concrete example, suppose the player had encountered the troll and soldier as before, with a second soldier waiting to attack on the outside of the grid. If the player suddenly moves towards the second soldier who is currently without permission to attack, it would be best if the stage manager could move permission from the first soldier to the second to take advantage of the second soldier's new position.

To solve this, the stage manager could "steal" slot assignments from one creature to give to another or otherwise reassign creatures as it saw fit. In practice, this was tweaked such that it tended to leave creatures with permission to attack alone, unless the assigned creature was outside of the attack grid while some other creature was within the space of the grid, in which case the creature assignment would shift. Shifting could also occur in order to move attacking creatures to different positions on the grid. This shifting would attempt to reduce the total travel time for all creatures to their assigned places in combat. Further, creatures that were actively launching an attack would "lock" their assignment with the stage manager, so they wouldn't lose their slot during an attack and accidentally overload the player's attack capacity. Such an algorithm might look like the following:

```
Attacking list = all attacking creatures
For each creature in Attacking list
    Find closest slot for creature
    If closest slot is locked, continue
    Assign closest slot to creature
    Remove creature from Attacking list
    If closest slot was already assigned,
    Remove assignment from that creature
```

28.6.2 Further Restrictions on Enemy Attacks

While the attack capacity restricts the total number of attacks that can be launched at the same time, it doesn't prevent many creatures from launching the same type of attack simultaneously, especially at higher difficulty levels where the attack capacity is increased, and especially in encounters with many of the same type of creature. In order to help ensure that a variety of attacks occur in a given combat encounter, *Reckoning* also imposed cooldowns on individual, creature-wide, and global bases to prevent creatures from launching too many of the same attacks within a certain window of each other. This helped tremendously in preserving game balance at the most difficult setting, as even with an increased attack capacity, particular enemies could be prevented from using too many area-of-effect type attacks at once.

28.7 Conclusion

Managing game difficulty is an area worthy of serious consideration and research. While it's important to not overwhelm a player learning the game, flexible systems are imperative in order to provide challenging gameplay for more experienced players. Simple approaches, like the grid-based one presented here, can help empower game designers to create different and interesting encounters that can scale for a multitude of difficulty levels simply by adjusting a small set of initial values.

While the implementation of this algorithm is simple enough, its real strength lies in how the positioning logic is centralized to the stage manager. Simply having behaviors to move characters out from slots they aren't assigned to can work to get characters to flank and avoid each other, but it's easy to see how this algorithm could easily be paired with an influence map technique to help characters avoid each other and move around the battlefield more naturally. Other techniques could similarly be combined with the idea of a battle grid to tailor the technique to the needs of an individual game while maintaining the flexibility of adaptive difficulty levels.