27

The Role of Time in Spatio-Temporal Reasoning

Three Examples from Tower Defense

Baylor Wetzel and Kyle Anderson

27.1 Introduction

In Akira Kurosawa's film *Seven Samurai*, a small group of defenders are hired to defend a small village from an invading army of bandits. Vastly outnumbered, their survival depended on mastering their environment. Humans are good at spatio-temporal reasoning. Understanding how objects flow through space over time is how we know whether we can cross the street when there's a car coming. It is also how samurai know where to place ambushes when defending a village.

Spatio-temporal reasoning is an important part of many decisions (in and out of games). It is how armies know where to place gun nests, prisons know where to place cameras, firemen know where to place firebreaks, cities know where to place highways, and malls know where to place stores. Much has been written about the spatial aspect of spatio-temporal reasoning. Identifying choke points, flanks, and avenues of approach is an important part of video game AI, and at this point is fairly well understood. Far less has

been written about time. Consequently, this chapter will focus specifically on the temporal aspect of spatio-temporal reasoning.

Temporal strategies are harder to visualize than spatial ones and, we believe, best explained by working through examples. Like Kurosawa's *Seven Samurai*, this chapter will focus on protecting a location from incoming enemies by identifying the best place to deploy our limited defenses. Defending a location is a common scenario in games that include any amount of strategic planning, such as tower defense games, strategy games, and "hold out until help arrives" missions in first-person shooters or action games. We will look at three examples. The first can be solved without directly thinking about time, whereas the second cannot be solved. The third shows how changing one's focus from space to time can produce new solutions.

27.2 Defend the Village

To keep our focus on the temporal part of spatio-temporal reasoning, our examples will use a tower defense game. This gives us difficult problems to solve but abstracts out the details not relevant to the topic.

For those unfamiliar with tower defense games, the genre grew out of real-time strategy games, with many players realizing they enjoyed designing the defenses around their base more than they liked attacking other players. The player is given a location to protect and one or more paths to that location. The player cannot directly attack the enemies (generically referred to as *creeps*) that will come down the path. Instead, they are given a set of objects (generically referred to as *towers*) that attack any enemy in range. Specifics vary by game—in some games the towers are soldiers with fast rifles or slow rocket launchers, in other games the towers are pits of acid, wizards with fire, or monkeys with darts. But what is always the same is that the gameplay revolves around placing towers strategically, in such a way that they prevent the enemies from getting past them.

Our examples come from *GopherTD*, a game we built to study how humans play strategy games. It is based on the popular tower defense game *Vector TD*. These are the details relevant to this chapter:

- There are 28 enemies (creeps) divided into two lines of 14 creeps each.
- Each line of creeps follows a fixed path through the map. Normally the two lines move side-by-side.
- Creeps move at a constant, fixed speed unless slowed by a slowing tower.
- Towers have a fixed rate of fire, do not run out of ammo, and never miss.
- There are several types of towers. To keep things simple, our examples will use only two types:
  - Attack towers attack the closest creep in range, staying on them until the creep dies or moves out of range, after which it moves to whichever creep is closest at that time. A creep must be hit multiple times before it is stopped.
  - Slowing towers attack the closest creep. Attacked creeps move at half speed for two seconds.
- The score is the number of creeps prevented from reaching the exit.

Figure 27.1 shows three examples of *GopherTD* levels.

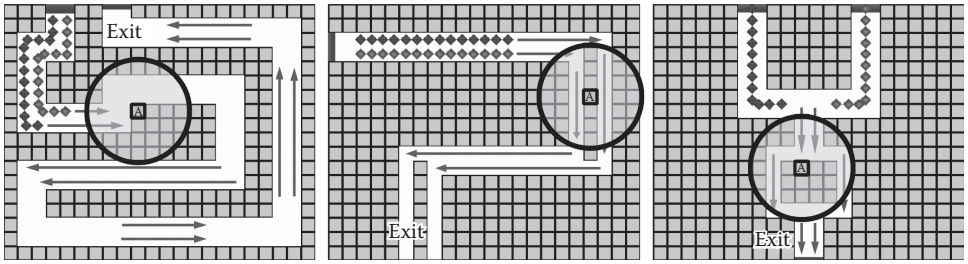


Figure 27.1

Maps from *GopherTD*. Creeps are shown at the entrance. Their goal is to make it to the spot marked Exit, representing the location you are trying to protect. The player's goal is to stop them by placing defenses (towers) along the path. Arrows show the direction the creeps move. A tower can be placed on any gray square. Circles show the range of the attack tower located at the cell marked A.

27.3 Example 1: U-Turns and the Maximum Usable Range Strategy

Our first example, while simple, illustrates some important concepts we should understand before tackling more complex problems.

27.3.1 The Problem

Consider the leftmost map in Figure 27.1. If you had only one attack tower, where should you place it? Figure 27.2 shows five options, each representing a different type of spatial structure—a hallway, corner, U-turn, double-sided wall, and an interior corner. Which is the best position and why?

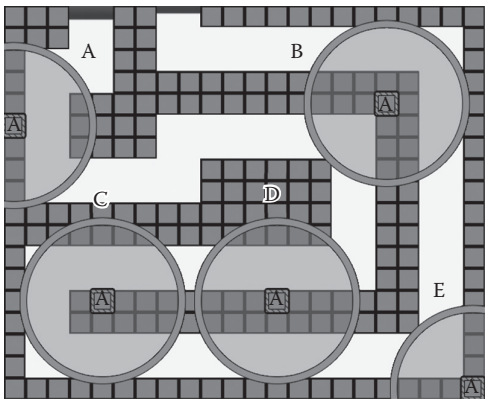


Figure 27.2

An attack tower placed at five different types of positions. (A) A wall. (B) A corner. (C) A U-turn. (D) A wall. The tower overlooks two hallways. (E) An interior corner.

We wanted to design an AI that thought like a human so we began by studying humans. We asked 59 people, a mixture of gamers and nongamers, where they would put the tower (among other things [Wetzel 2014]). The answer was nearly unanimous: C, the U-turn (only one person disagreed, choosing D, the hallway).

What makes the U-turn a better position than the others? People answered in different ways. Some said that it covers three hallways, whereas B and D only covered two and A and E covered one. Some said C covered everything positions such as D did plus more. Others said that C simply looked like it covered more area. Unsurprisingly, given how good humans are at visual estimation, they were correct: position C has a usable range of 30 (i.e., there are 30 path tiles the creeps move across that are inside the tower's range) as opposed to 29 at B, 26 at D, 16 at A, and 8 at E.

27.3.2 Discussion: Space-Time Equivalence

Each of the above explanations means essentially the same thing: maximize the tower's usable range, meaning place the tower where it covers the largest number of tiles that the creeps will walk over. We call this the *Maximum Usable Range* strategy and it is a common and obvious strategy.

Words say a lot about our thinking process. If you look at the reasons given for selecting position C you should notice that the language is almost entirely spatial. If this does not seem surprising, it might be because you believe the problem is a spatial problem. It is not. The goal is nonspatial: maximize the score, with one point earned for each creep destroyed. The action is spatial: place the tower on the map. The effectiveness of the action is temporal: since the tower fires at a fixed rate of speed doing a fixed amount of damage per shot, maximizing the tower's effectiveness means maximizing the amount of time it spends firing. Since the tower only fires when creeps are in range (a spatial property), maximizing the tower's effectiveness means maximizing the amount of time there are creeps in the tower's range.

As with many terrain reasoning problems, the problem is a mixture of spatio-temporal reasoning and nonspatio-temporal goals. Why, then, do people treat it like a spatial problem? One reason is that is easier for people to think about space than time. A large part of the human brain is dedicated to understanding space, something that is not true for time. If a spatio-temporal problem can be converted to a spatial one, it can leverage that hardware.

Two other reasons why people convert spatio-temporal problems to spatial ones are that it works well and that it makes sense. In many cases, space and time are proportional and linearly correlated. In this case, the more area inside a tower's range, the longer it takes the creeps to move through it and therefore the more time creeps are in range and the more time the tower has to fire at them. Increasing a tower's usable range (space) increases the amount of time it spends firing (time).

27.3.3 Discussion: Strategies and Affordances

We wrote AIs to solve each of the maps mentioned in this chapter. The best AIs perform as well as the best human players we studied (which is perhaps unsurprising given that we based them on those players). All of the AIs work in the same way: they are given a set of affordance-keyed strategies and query our Spatial Affordance Query System for places to apply their strategies. We will give examples later but first we need to define how we use the terms "affordance" and "strategy."

An *affordance* is something that affords (allows) an action. More importantly, it suggests an action. The classic example is a door handle (Norman 1988). A plate on a door says (to most people) "push me," whereas a handle says "pull me." In our example, the U-turn was an affordance that said (to most people) "place your tower here." Alternately, for those who measured usable range directly rather than using the U-turn as a proxy for maximum usable range, usable range is an affordance. There are many types of affordances. In the case of U-turns, the affordance is spatial geometry, whereas in the case of usable range, it is a variable to be maximized.

It is important to point out that affordances are contextually defined—whether something is an affordance depends on a number of things including one's goal, capabilities, and other relevant factors. In our example, several things must be true for position C to be a U-turn. First, the creeps must move all the way around it. If the creeps came from two different directions and met in the middle, as they do in the rightmost map in Figure 27.1, the space might still be shaped like a U-turn but it does not afford the expected behavior of creeps moving around the U-turn. Second, C is only a U-turn if the attack tower's range is large enough to cover all three sides. If the tower's range were significantly wider, position D would also be a U-turn, whereas if it were taller, position B would be a U-turn (i.e., it would cover the U-turn that is down and to the left). Third, we only notice the U-turn because the action it affords is relevant to our Maximum Usable Range strategy. If we had different goals, the U-turn might stop affording us actions we care about. The important point for the AI developer is this: You cannot identify an affordance without knowing how you intend to use it. In this example, you not only need to know the map geography, you need to know the path the creeps take and the range of the tower to be placed.

In our experience, a *strategy* tends to be a simple action designed to achieve a single, concrete goal. Strategies exploit or otherwise rely upon an affordance. A player only considers using a U-turn strategy when the map contains U-turn affordances (i.e., areas that can be used as a U-turn by the strategy). Likewise, a player that does not know the U-turn strategy does not notice the afforded U-turns on the map.

Strategies are, in our view, small things. Most problems require the use of multiple strategies, which we refer to alternately as *strategy sets* or a person's *play-* or *problem solving-style*. In this example, where we only have one tower and wish to place it at the spot where it has the most usable range, a single strategy is sufficient. We will see more complicated playstyles in the next examples.

The implementation for the maximum usable range agent is given in Listing 27.1. It is only a few lines: Define the strategy then ask the solver to solve it. Defining a placement strategy involves specifying a *PlacementDecision*, which consists of three parts: *placement object*, *placement relationship*, and *anchor* (Wetzel 2014). For this AI, the strategy is to place the tower (placement object) on a spot that has the maximal (placement relationship) value for the relative property `PhysicalPathCellsInRange` (anchor). Relative properties can vary in magnitude (e.g., usable range), whereas an absolute property does not (e.g., a location either is or is not a corner). The relative property used by this AI is usable range, which it wants to maximize. The anchor could be a spatial feature but more often it is an affordance, as it is here. Other examples of strategies include placing an attack tower's range (placement object) so that it overlaps (placement relationship) another tower's range (anchor) or placing a slowing tower so that the exit to its range

Listing 27.1. The Maximum Usable Range agent.

```
Solution AI::getSolution(MapModel map)
{
    SolutionRequest request = new SolutionRequest(map);
    GroupToPlace group = new GroupToPlace();
    request.groups. Add(group);
    group.towers. Add(new AttackTower());
    Strategy strategy =
        new PlaceTowerOnRelativePropertyStrategy(
            MapPropertyOrdinal. PhysicalPathCellsInRange,
            StrategyOperator. Maximum));
    group.placementStrategies. Add(strategy);
    return Solver.getSolution(request);
}
```

(placement object) is in front of (placement relationship) the start to an attack tower's range (anchor) (Wetzel 2014).

The solver code (Listing 27.2) is equally simple: Find the affordances on the map and execute the strategy for each affordance. In this case, the AI asked the solver to place

Listing 27.2. The Solver creates a solution (list of tower placements) by instantiating the strategy for each group of towers. In practice, this means identifying affordances on the map and matching towers to those using each tower's strategy, which is a `PlacementDecision` specifying the affordance to use.

```
static Solution Solver::getSolution(SolutionRequest request)
{
  Solution solution = new Solution();
  solution.map = MapAnalyzer.getMapAnalysis(request.map);
  foreach (GroupToPlace group in request.groups)
  {
    foreach (Tower t in group.towers)
    {
      foreach (Strategy s in group.strategies)
      {
        List<GridPoint> candidates = s.getPositions(t, map);
        if (candidates. Count > 0)
        {
          GridPoint p = group.tieBreaker.get(candidates);
          solution.add(t, p);
          break;
        }
      }
    }
  }
  return solution;
}
```

one attack tower with the strategy "place the tower at the position where it gets the most usable range." The solver asks the MapAnalyzer for all the affordances on the given map for the specified tower (`getMapAnalysis`), compares the strategies to the affordances and places the tower on the best match (i.e., where the tower's usable range is maximized) then returns where the highest usable range for an attack tower is on the given map and returns a list of positions and the tower to place on each.

For this code to work a few things have to be true. You must be able to explicitly define your strategies (in Listing 27.1, the Maximum Usable Range strategy is an instantiation of the more general `PlaceTowerOnRelativePropertyStrategy`). Second, you must be able to define your affordances. Third, strategies need to be keyed to affordances, or at least, the computer needs to know how to apply a strategy given an affordance (in this case, the tower is placed on the spot where the affordance is detected). Fourth, you need to be able to identify the affordances on the map. We do this through our Spatial Affordance Query System, SAQS. Although the AI and solver code are quite short, SAQS is not and is therefore outside of the scope of this chapter. We believe, however, that by walking through examples, it will be clear how to identify the ones you need.

27.4 Example 2: Spatial Symmetry and the Differential Slowing Strategy

It is often enough to convert a spatio-temporal problem to a spatial one and solve that, but not always. In this example, we consider a situation where we need to think explicitly about time.

27.4.1 The Problem

The map in Figure 27.3 has an interesting property: It is essentially featureless. It has no switchbacks or U-turns or any usable spatial features. The only positions where a tower could reach the creeps on both paths are on the thin strip of wall between the two. The positions on the wall are all essentially identical, with a usable range of 14, less than half that on the U-turn in Example 1 and not nearly enough to stop the invading creeps.

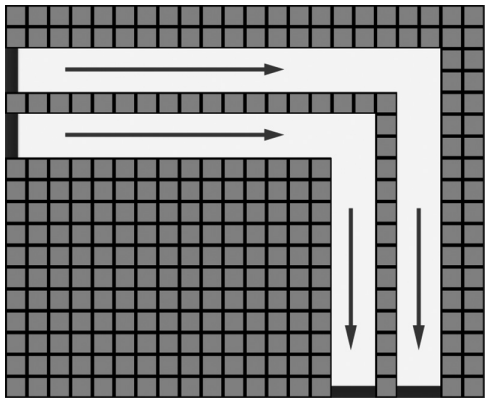


Figure 27.3

The map no left turns.

27.4.2 Discussion: Temporal Asymmetry and the Path Gap Affordance

In Example 1 we said that, although these problems were temporal, not spatial, it did not matter because space is often a good proxy for time. This is a problem where that is not true.

The spatial geometry is mirrored across the diagonal axis. This makes the map symmetric, but only spatially. When it comes to time, the map is strongly asymmetric. We can measure this by measuring where an object is at a given point in time. Pick a position along the path and consider which line of creeps will reach it first. Before the corner, the answer is neither, both lines will arrive at the same time. After the corner, however, the creeps on the inside (lower, left) path will reach a given position before those on the outer one. That is because the outer path is longer, taking longer to go around the corner (Figure 27.4).

We use the term *path gap* to refer to the difference between when a group on one path reaches a position versus when a group on a second path reaches it. Once you know the concept exists, it is easy to see and you will see path gap on every map where you are trying to solve a similar problem (an affordance only exists when it is relevant to one of your goals; if you do not have that goal, the concept stops making sense). If the concept suggests an action, a strategy to exploit it, then it is an affordance. Path gap is certainly an affordance.

We said earlier that our goal is to maximize the amount of time a tower spends firing, which we do by maximizing the amount of time that creeps are in range. Note that this does not refer to the amount of time that a single creep is in range, it is the amount of time that any creep is in range. With that in mind, consider Figure 27.4. A tower placed on the top half of the map will be active for the amount of time it takes a line of creeps to move through it. It does not matter which line since both move through at the same time. A line of creeps is 14 creeps long so the tower is active for as long as it takes for 14 creeps to move through it.

Now consider the same tower placed on the lower part of the map. The outer path is six tiles longer so the second line of creeps does not reach the tower until six tiles worth of creeps has already entered its range. The tower is therefore active for the amount of time it takes the first line of creeps to move through the tower's range, as in the positions along the top of the map, plus the time it takes for the last six tiles worth of creeps from the second line to move through. If we say that a creep is one tile wide, the tower's active

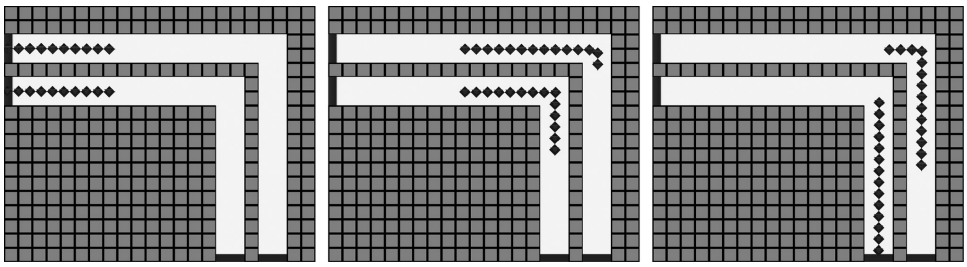


Figure 27.4

As they round the corner, the creeps on the outer path fall behind.

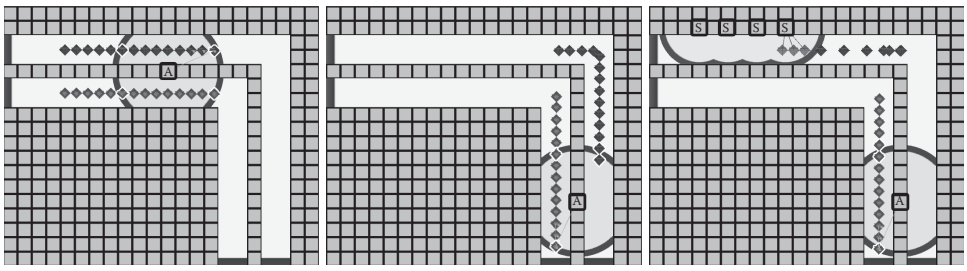
time on the bottom part of the map grows from the amount of time it takes 14 creeps to move through to the amount of time it takes 20 creeps to move through. The physical space in range did not change but the amount of time the tower spends firing did; the map is spatially symmetric but temporally asymmetric and the positions on it are spatially equivalent but temporally different. We have grown time without growing space. We call this the *Exploit Path Gap* strategy.

27.4.3 Discussion: Forcing Affordances and the Differential Slowing Strategy

Path Gap is a spatial affordance in the sense that it is based on the length of a path. As such, it can only be used on maps where the geometry supports it. The key idea, however, is not the length of the paths, it is the difference between when two groups arrive at a location. That gap in arrival times is an affordance we can exploit. Once we know this is valuable property, we not only become more sensitive to other features that can create it, we can consider ways to create it ourselves, even on maps where the geometry does not support it.

In Section 27.2 we mentioned that there are two types of towers, attack and slowing. We did not use the slowing towers in Example 1 because it did not affect the strategy we used (maximizing usable range) or the way we thought about space. Here, where we want to exploit the difference in arrival times, slowing towers give us an interesting option. We could use the slowing towers to slow the creeps while they are in the attack tower's range, a good and straightforward strategy, but we can do better.

In the *Differential Slowing* strategy, slowing towers are placed where they only slow one group, causing it to fall behind the other and effectively creating a path gap. Using our slowing towers, we can slow one line of creeps and not the other, thus causing one group to fall behind the other. In this example, we can combine Differential Slowing with the Exploit Path Gap strategy to create a much larger gap (obviously, both strategies should target the same group).In practice, using both Exploit Path Gap, which exploits a temporal affordance caused by a spatial feature, and Differential Slowing, which creates a temporal affordance, fully separates the two groups, causing them to go through the tower's range at different times (Figure 27.5). This doubles the amount of time the tower spends firing.



(a) (b) (c)

Figure 27.5

Three strategies. (a) *Maximum Usable Range,* tower placed before the corner. (b) *Exploit Path Gap*. (c) *Differential Slowing*.

27.4.4 Implementing the Differential Slowing Strategy

In our study, once players recognized the path gap affordance and figured out the Differential Slowing strategy, they seemed to have little trouble successfully applying it, even on maps they previously struggled on. In contrast, this was the most difficult AI for us to implement. Because of that, we feel it is valuable to spend a little time explaining how we did it.

SAQS keeps an influence map of each affordance it tracks. These maps can be used as filters or weighted and combined. The resulting map is a desirability map. Our AI uses a solver which picks the highest value spot on a desirability map.

For the differential slowing AI we need to combine a few strategies. The attack tower's placement depends on three affordances: usable range, path gap and coverage balance (a ratio of how much of the tower's usable range covers one path versus another; if the spot with the best usable range does not allow the tower to attack both paths, the enemies approaching from the second path will get through). The desirability map is a weighted combination of these three. The slowing towers want to maximize their usable range but only on positions that are *single path overwatch* positions. A single path overwatch is a position that affords the tower the ability to attack one path but not another. The map of single path overwatch positions is a filter. The solver only applies the maximum usable range strategy to those positions returned by the single path overwatch filter.

Creeps must be separated before they are attacked; there is no point in slowing them after they have passed the tower. To achieve this, the map must be divided into two zones, the slowing zone and the kill zone, and the slowing zone must be placed before the kill zone *temporally* (i.e., we do not care where on the map it happens spatially but the creeps must pass through it before they enter the kill zone). In order to determine where *before* is, we need a way to measure the flow of creeps over time through the space. We used a *path* feature (not affordance; the path does not directly or immediately suggest an action and it is an invariant feature of the map rather than being dependent on our goals) with the property *step number*, which is simply how many steps it is from the start of the path.

Dividing the map into zones is not as easy as it might sound. At its most basic we just need a split point—slowing towers go before this point, attack towers after—but determining that split point is not straight forward. The slowing towers want the path-specific single path overwatch positions with the highest usable range. The offense tower wants the position with the highest (weighted) combination of path gap, coverage balance and usable range. If we are lucky, the best kill zone positions are "later" than the best slowing zone positions but we need to handle the situations where they are not. If the best kill zone position is at the start of the map (which it is not here but is on many other maps), you either have to abandon the strategy or pick an inferior kill zone—but not too inferior, or the strategy is not worth it.

To determine whether the strategy makes sense, let us set some minimum requirements (values were chosen after a few iterations of play testing). A slowing zone must have at least four single path overwatch positions (again, overwatching the desired path) with a 45% or greater usable range ration (i.e., at least 45% of the tower's range covers the desired path). The kill zone must have at least one position of some value, say four path gap, 70% usable range ratio, and 60% coverage balance.

We generate a set of candidate split points by using a sliding space-time window. Start at the beginning of time and space (the path start) and move along it, scoring the adjacent positions. In practice, we grab the path of interest, start at the first position on it and ask SAQS for all positions within a slowing tower's range of it. Those that match we store on a map and the count of matches we store on the path. We then move to the next step on the path and repeat. The same process is done for the kill zone but moving backward through time (i.e., start at the end of the path).

Once the sliding window maps are built you can find the split points for the slowing and kill zones where the strategy's minimum needs are met. If the borders cross, the strategy is not going to work and the AI picks a different one (the recognition of affordances on the map triggers strategies that the AI knows; if multiple strategies activate, each can be tested to see which are the most effective). Once found, there are three regions: the minimum areas for the slowing and kill zones, and the unclaimed area between them. Our AI places attack towers first since the quality of attack tower positions often vary greatly, whereas all of the slowing zone positions are guaranteed to let a tower slow one group and not the other. The attack tower placement evaluates positions in the combined kill zone and unclaimed area. Once the towers are placed, the rest of the area is added to the slowing zone and the differential slowing towers are placed.

27.5 Example 3: Quantifying Space-Time and the Attack Window Separation Strategy

Our goal (maximize the amount of time a tower spends firing) is temporal and our action (place an object on a map) is spatial. If space and time are linearly correlated, we can measure everything spatially knowing that when we increase space we also increase time. If space and time can be manipulated independently, as they were with Differential Slowing, we need a way to measure time and a way to convert between space and time. In this section we consider such a metric, as well as its implications for the original U-turn example.

27.5.1 AL: A Metric for Measuring Space-Time

Our work uses a unified space-time metric called *al* (for agent length). It represents the amount of time it takes an agent (in this case, the creep) to cross their body length. It allows us to measure items both spatially and temporally and convert between the two.

We are going to work several examples so we need to set some values:

- A creep is 1 tile wide
- A creep can move 1 tile per second
- An attack tower fires 10 times a second

These are not the exact values from the game but they make the math easier to follow. For the same reason, we are also going to pretend in these examples that all values are whole numbers.

Let us start with a simple example. Suppose we say that a tower has an active time of 6al. We can convert this to a spatial measure and say that the path through the tower's range is six tiles long. We can convert this to a temporal measure and say that the tower is active for six

seconds. We can convert this to a nonspatio-temporal measure and say that it fires 60 times and does 60*x* damage, where *x* is the amount of damage per shot. If the tower covers four path tiles at a second position, its size at that position is 4al, it will be active for four seconds and fire 40 times doing 40*x* damage. Placed at the first position, the tower does 50% more damage.

In this case, time and space are linearly correlated and we can find the best spot by just maximizing space, meaning we get no value from the al metric. To see where this is not true, we need to talk about attack windows.

27.5.2 Attack Windows: Spatial and Temporal, Unified and Separated, Agent and Group

In the previous example we said "If the tower covers four path tiles at a second position, *its* size is 4al." We did not say what "it" was. *It* refers to the tower's attack window. An attack window is the opportunity an object has to attack. There are several ways to measure them. A *spatial attack window* is a contiguous block of space where the tower can attack. In Figure 27.2, the tower on the U-turn (C) has one large spatial attack window, whereas the tower in the hallway (D) has two small spatial attack windows. A *temporal attack window* is the period of time the tower has to attack. A tower will have as many temporal attack windows as spatial ones under two conditions. First, the group only crosses through the space once. If a tower has one spatial attack window but a group passes through it multiple times, it is the same as if it had multiple attack windows since the tower gets to attack on multiple occasions. If a group crosses the same space multiple times, the number of temporal windows can be larger than the number of spatial windows. Second, if a tower has two spatial attack windows but a group can be in both at the same time, it is a single temporal window—since, from the tower's point of view, there is no pause between attacks, it is all the same time period. If the group is long enough or the spatial attack windows close enough that a group can be in both at once, the tower can have fewer temporal attack windows than spatial ones. Whether temporal windows are unified or separated is important, as will be shown in the next section.

An attack window's *temporal agent size* is the temporal size of the window for a single agent, meaning it is the amount of time it takes one creep to make it through the attack window. Its *temporal group size* is the size of the window for a group of agents moving through it. This is the one to pay attention to as it is not necessarily correlated with space. The reason for this is that, from a functional perspective (e.g., from the perspective of a tower attacking), groups do not move through time the way individuals do.

Finally, a tower's *active time* is the amount of time the tower spends firing. This is equivalent to the combined temporal group size of all attack windows.

A tower can have multiple temporal group attack windows. The size of each window is the amount of time it takes a group to move through the space. This is equivalent to the time it takes the first member of the group to make it through, the time it takes the last member of the group to make it through and the difference between the first creep exits the range and the last one enters. The time it takes for the first creep to make it through the range plus the time it takes for the last creep to enter the range is just the line length so the temporal group size is:

```
temporalGroupSize = lineLength + temporalAgentSize – 1
```

(the –1 is to avoid counting the last creep twice).

It is worth noting that a tower will fire for as long as there is a creep in range. It does not matter how many there are; the tower does not fire faster with ten enemies than with two. For this reason, the density of enemies does not matter. If two lines of enemies pass through a tower's range at different times, the tower gets two opportunities to attack, but if two lines move through at the same time, the tower only gets one. In the first example we talked about usable range. This turns out to be unimportant. What matters is the path length through the window. To use an example from a different genre, imagine you are in a first-person shooter and have the option of camping over an alleyway or in front of a bank of windows. In the alleyway you can see an enemy running the length of the alley, which takes five seconds. In front of the building you can see five windows. There are five people inside, all of whom cross in front of the window at the exact same, spending one second—the same second—in their respective windows. Even if both positions look over the same amount of space, the amount of time you have to attack (five seconds in the alley, one in front of the windows) is different.

27.5.3 The U-Turn Problem, Revisited

Let us revisit the question asked in Example 1—what is the best position to place an attack tower in Figure 27.2. 58 of 59 people we studied said position C, the U-turn. An attack tower at C has one spatial and one temporal attack window. Its usable range is 30, the length of the inner path is 13 and the outer path 16. Since the temporal agent size is the time it takes one creep to follow the path through the tower, the temporal agent size is the same as the longest path, 16al. There are 28 creeps in two lines of 14. The outer line of creeps takes a longer path, with each corner adding 2 tiles, causing the last four creeps of the outer line to still be in range once the inner line has left (see Figure 27.6). The line length is therefore 18. The data are summarized in Table 27.1. The temporal group size of the U-turn at position C is:

```
temporalGroupSize = lineLength + temporalAgentSize – 1
temporalGroupSize = 18al + 16al – 1al = 33al.
```

An attack tower placed on the U-Turn has an active time of 33al, meaning it has 33 seconds to fire 330 shots to stop 28 creeps.

One of 59 players chose position D, an unremarkable position in a hallway that overlooks two halls. It has two spatial attack windows. They are far enough apart that no creep will still be in the first when the first creeps enters the second. As with the U-turn, because the outer line of creeps passes two corners before entering the second attack window, the line length changes from 14 in the first window to 18 in the second. The data are summarized in Table 27.1. The size of the temporal group attack windows are:

```
Hall AW1 TGS = 14al + 7al – 1al = 20al
Hall AW2 TGS = 18al + 6al – 1al = 23al
Combined Hall TGS               = 43al
```

An attack tower placed in the hallway far enough away from the U-turn that the attack windows are independent (which we call the *Attack Window Separation* strategy) has an active time of 43al, meaning it has 43 seconds to fire 430 shots to stop 28 creeps.

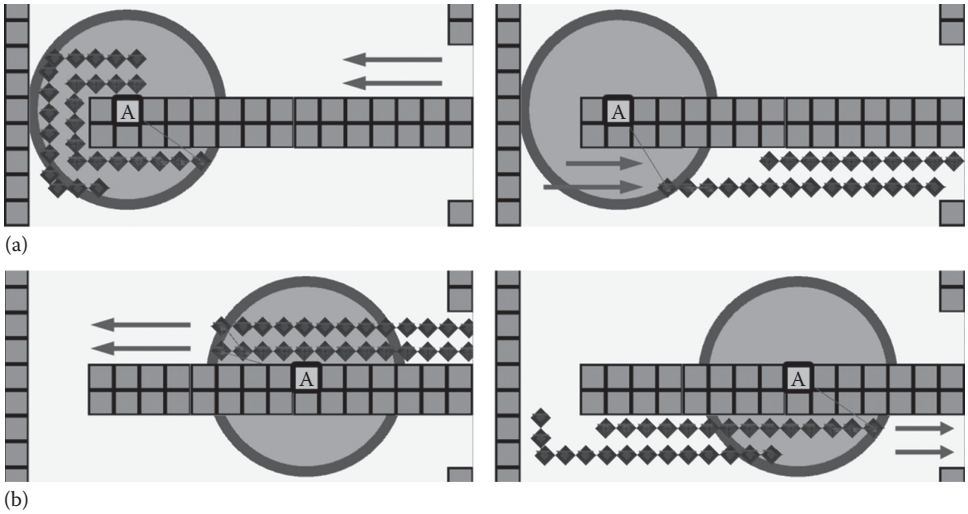


Figure 27.6

Two lines of agents moving through across a U-turn. (a) Creeps moving through the range of a tower covering a U-turn. (b) Moving through the range of a tower covering a hallway.

Table 27.1 Spatial data for positions in Figure 27.2

	(C) U-Turn	(D) Hallway
Usable range	30 tiles	26 tiles
Path length	13 inside	AW1: 7 inside/6 outside
	16 outside	AW2: 6 inside/4 outside
AW agent size	16al	AW1: 7al
		AW2: 6al
Line length	18al	AW1: 14al
		AW2: 18al

In most respects, the U-turn is a better position. An attack tower placed there has 15% more usable range, a 23% longer path, a 23% larger attack window temporal agent size (+23%) and 12% longer average line length (+12%). Despite this, the same tower placed in the hallway will active for 30% longer (43 seconds vs. 33) and do 30% more damage.

27.5.4 Attack Window Decay

Because it has a 30% larger combined temporal attack window, it is tempting to say that a hallway tower it is 30% better than a U-turn tower. It is not. In the game, an attack tower placed on the U-turn stops 14 creeps, whereas the same tower in the hall stops 20, 43% more. If the tower only fires 30% more often and therefore only does 30% more damage, why does it achieve a 43% higher score?

We said that the attack window temporal agent size is the amount of time it takes for one agent to walk through a tower's range and is the amount of time the tower has to attack

the agent, which can also be phrased as the amount of time the agent is in danger. We need to amend that a little.

The U-turn in the last example had an attack window temporal agent size of 16, meaning it took an agent 16al to move through it and the tower had 16al of space-time to attack it. Suppose the tower kills the agent in 2al. If the enemies are back to back, a second enemy came into range 1al later and has moved another 1al along the path. During this time the tower has been busy with the first enemy, meaning the second enemy has not been attacked or hurt yet. By the time the tower is free to attack it, the second enemy only has 15al it needs to cross, and the tower only has 15al to attack him or her. The temporal attack window has shrunk. By the time the tower stops the second enemy, the third enemy is 2al into the region and the tower only has 14al left to stop him or her. Assuming enough enemies relative to window size and how quickly the tower can dispatch enemies, the tower eventually runs out of time to destroy enemies and can only wound them, which in many applications, from stopping zombies to damaging incoming tanks, earns you no points since the agent still makes it to the area you are defending and is able to do damage.

When an agent is holding a tower's attention, he or she is buffering for the other agents, keeping them from taking damage—in many games, when players do this it is referred to as "tanking." Once the tower finishes with the agent it must play catch up. Since the tower has a fixed rate of fire, catching up is impossible as long as there are enemies in range.

How does a tower catch up? It needs time. It needs to process the queue it has before starting on new work—which it can do if the attack window is divided into parts and it can process each one independently. This is exactly what happens when the tower is placed in the hallway and exactly what goes wrong when it is on the U-turn. It is also why we said earlier that two spatially independent attack windows are a single temporal attack window if enemies can enter the second attack window while the tower is busy with enemies in the first one—the tower does not have a chance to catch up and the temporal attack window continues to decay. The tower needs a breather so that the attack window size can reset.

As a note, we call the strategy we are using here *Attack Window Separation*, and the affordance that triggers it is, of course, *attack window separation*, a numeric measure of the distance between the end of one spatial attack and the start of the next (as before, you need to know how time flows through space, which for us meant path step number). If you know how long the enemy line length is, you could ask the SAQS for all positions where the attack window separation is enough to temporally decouple the windows. As with differential slowing, you can use slowing towers placed between the attack windows to stretch the length as necessary, creating attack window separation affordances that enable an attack window separation strategy on maps where it was not possible before. Because implementing this has no new ideas that were not covered in the previous sections Differential Slowing, we will not walk through the implementation here. We do hope, however, that this and the earlier examples let you appreciate the value of a good spatial affordance query system.

27.6 Conclusion

We know we live in a world of objects moving through space over time and to deal with it we need to do spatio-temporal reasoning, but much of what we do is actually spatial reasoning, allowing space to be a proxy for time. Often our thoughts are spatial, our reasoning

is spatial, our strategies are spatial, and the affordances we recognize are spatial. This is not a bad thing and we can often perform quite well this way, but if we can bring ourselves to focus on time—to recognize temporal affordances and craft temporal strategies—we can sometimes do much better.

References

Norman, D. 1988. *The Design of Everyday Things*, New York.

Wetzel, B. 2014. Representation and reasoning for complex spatio-temporal problems: From humans to software agents. PhD diss., University of Minnesota.