UPPAALTD: A FORMAL TOWER DEFENSE GAME

Formal Methods for Concurrent and Real-Time Systems Homework

ANDREA BELLANI

July 4, 2025

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

1 INTRODUCTION

1.1 Definitions, acronyms and abbreviations

1.2 Model's version history

1.3 Document structure

2 MODEL DESCRIPTION

We modeled both the Vanilla and the Stochastic version of the game. In particular, we firstly modeled the Vanilla version and then proceeded to modify it to model the stochastic requirements. In this section we provide a detailed explanation of both versions and how the Stochastic one was obtained from the Vanilla. We clarify that our main purpose in this section is to clarify and support our design choices rather then explain the technical aspect underlying the Uppaal code, which can be read from the templates comments.

2.1 Vanilla version's model

2.1.1 How game's state is modeled

We refer as *game's state* to the global state of match. It is represented by the two **`int`**:

- `left_enemies` : in each instant, it represents the number of enemies that have not died yet;
- `targetable_enemies` : in each instant, it represents the number of enemies that are currently active on the map (already spawned and alive).

The match is considered *ended* if (and only if) `left_enemies` is 0. In fact, `targetable_enemies` is designed only to facilitate the process turrets use to identify enemies to shoot.

2.1.2 How communication between automatons is modeled

We can identify from the game's description two kind of "possible messages" sent between automatons:

1. an enemy shoots to the MT;
2. a turret shoots to an enemy.

We decided to model only "2" with a real message sent over a channel. In particular, since the MT is modeled simply as an **`int`** variable (see the *Modeling of MT* section for more details), in order to shoot to it, an enemy simply decrements the variable of an amount equal to its damage.[NON SPIEGATO IL CONCETTO DELLA MUTUA ESCLUSIONE]

Since there is no way in Uppaal to send a message to a precise entity in a deterministic way, a pretty easy solution we found was:

1. a turret places into the `target_record` variable the id of the targeted enemy with the relative damage and sends a message over the broadcast channel `SHOOT_TO_ENEMY`;
2. each enemy that receives the message checks that the id of the target corresponds to its id and only in this case decreases its life.

[SPIEGARE PROSSIMAMENTE LA QUESTIONE DELL'URGENT]

2.1.3 Modeling of MT

As mentioned in the previous section, MT is simply modeled as an **`int`** variable that represents MT's life. It is initially set to value defined in the requirements. For the reasons why the MT is not modeled with a template, please see the dedicated section in *Discarded Choices*.

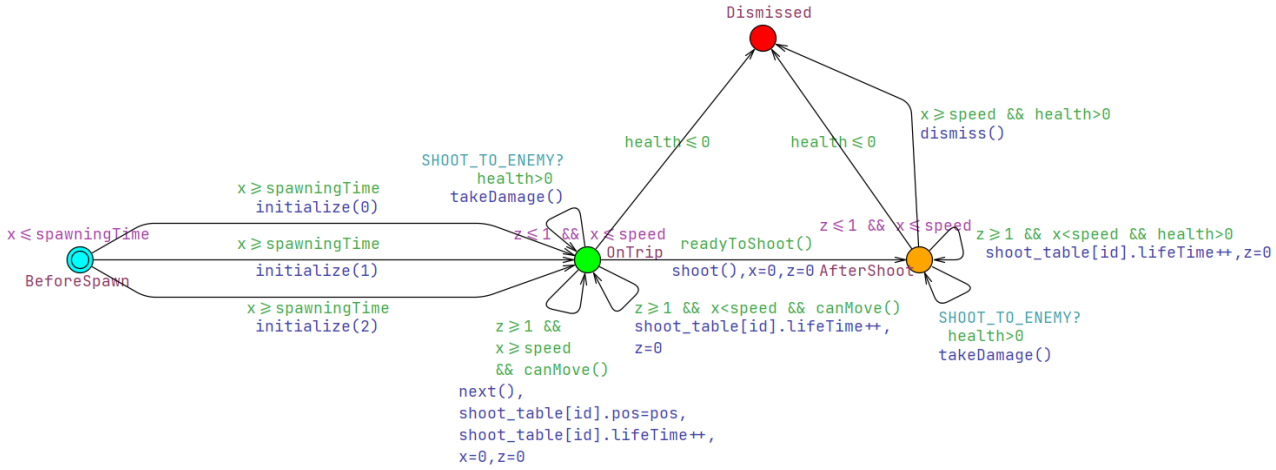

Figure 1: Enemy template

2.1.4 *Modeling of Enemy*

Enemy's template is without a doubt the most complex one:

Instead of describing each state, we would rather describe how each required feature of an enemy is implemented:

- concept of *spawn* : requirements state that circles should spawn at regular time intervals each other, and analogue for squares. We decided to define `spawningTime` as a parameter for an enemy, which determines the spawning timer for the enemy. So, in order to satisfy the requirements, the enemy `i`-th enemy for each kind should have this parameter set at $i * S$ (where S is the spawning time defined in the requirements for that kind of enemy);
- how enemies move on the map : once an alive enemy is ready to move and it has not arrived at the MT spot, it calls the function `next()` which simply sets its variable `pos` to the number of the next cell from the one where it was;
- how non-deterministic path choice is implemented : since in a symbolic simulation, `next` can only have a deterministic execution, the non-determinism in the next cell choice in proximity of a junction is implemented simply by non-deterministically choose a number from 0 to 2 as soon the enemy spawns (i.e. the parameter passed to `initialize`). Then, if the enemy is in a junction, `next` chooses the next cell accordingly to this number (please see the dedicated section *Discarded choices* for more details on why this design choice was preferred);
- how speed delay is implemented : simply, a clock `x` is used to model the speed delay timer;
- how an enemy responses to a shoot : on shoot, the enemy simply decreases its life (provided that it was the target) and its life goes to zero (or beyond), `takeDamage` calls `dismiss` to set its unavailability in the shoot table and decrease the global counters. Note that the fact that `takeDamage` decrements the life counter and in case calls `dismiss` is more appropriate than always splitting the calling of `takeDamage` to the calling of `dismiss`, since it may lead having one or more time instants where a dead enemy is dead but still considered available because `takeDamage` did not update the global information;
- providing all information for the shoot table : the update of the shoot table from an enemy is quite trivial, apart for the field `lifeTime`, which has to be updated at each time instant (`z` clock is needed for triggering an update of it at every time instant);
- shoot to the MT : once an enemy reached the MT spot it simply decrements the MT life of its damage and after a delay of `speed` (where it can still be shot) is dismissed.

2.1.5 *Modeling of Turret*

A turret has a pretty intuitive behavior:

Once it is ready, it looks for targetable enemies (available enemies inside its shooting are) by scanning `SHOOT_TABLE` with `shootable()`. If there is at least one, it uses `target()` to select and shoot the best one according to the requirements criteria:

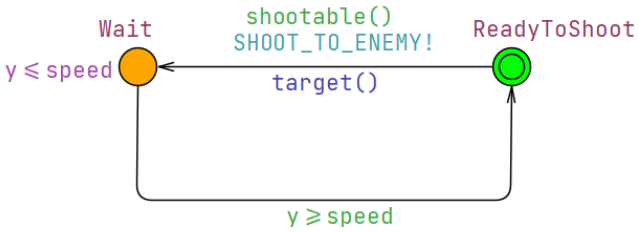

Figure 2: Turret template

1. `SHOOT_TABLE` is scanned until an available enemy inside the shooting area is found and there are still available enemies in the next positions of `SHOOT_TABLE` (`targetable_enemies` tracks the number of currently available enemies);
2. the found enemy is considered as the best target;
3. the scan of the remaining elements of `SHOOT_TABLE` is performed until the number of available enemies found is equal to `targetable_enemies`;
4. at the end of the scan, if an enemy to target is found, the procedure `shoot` is called, which simply sets into `target_record` the id of the targeted enemy and the relative damage (and also resets the clock `y` for the delay count).

This, perhaps not intuitive way of scanning enemies guarantees and asymptotic complexity $\Theta(\text{MAX}_\text{E}\text{NEMIES})$, which fits our application.

If the shoot succeeded, the turret stays in `Wait` until the delay expires and it can target and shoot again. Note that if all enemies are died, turrets will be in deadlock once they come back in `ReadyToShoot`.

2.2 Stochastic version's model

2.2.1 Modifications from Vanilla model

Stochastic model is not drastically different from the Vanilla one:

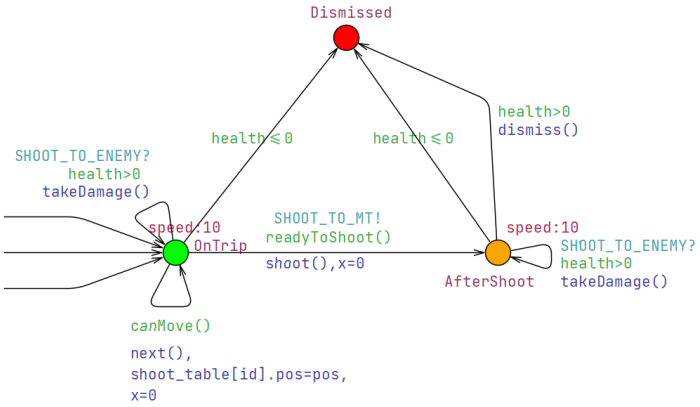

Figure 3: Enemy stochastic template

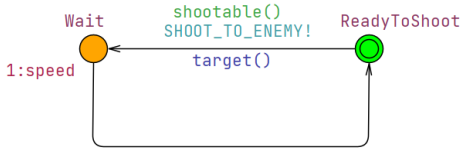

Figure 4: Turret stochastic template

For what strictly concerns the templates:

- enemy template :
  - speed delay so no more modeled as a delay but as a exponential rate that determines the probability of coming out from the state. Since while enemy is approaching the MT the only available transition (provided that the enemy is alive) is the one that calls `next`, the probability of moving to the next cell is determined by the exponential rate `speed:10`. This is analogue also in `AfterShoot`;
  - even if MT is still modeled as a variable and not as a template, an enemy that wants to shoot sends a message over the channel `SHOOT_TO_MT`, which simply is a `urgent broadcast` channel useful to force enemies to "not wait" before shoot when they can, otherwise also the probability of shooting to the MT at an instant would have been modeled, but requirements never ask for it, so it is better to model any outgoing transition that may change the game dynamics as `urgent` if no probabilistic

delay should affect it (transition to `Dimissed` if `health`$\leqslant$`0` does not really affects game dynamics since the enemy has already died);

- instead of keep counting time instants to increment their `lifeTime`, enemies have their lifeTime information already determined by the instant when they spawned of the map. This simplification could also have been used in the Vanilla version but there are queries that specifically argue on how many time instants have already passed by the time an enemy arrives at the MT spot;

- turret template :
  - turrets firing speed is no more modeled as a delay but as an exponential rate of `1:speed` (same reasoning of the enemy's speed delay modeling);
  - since the concept of lifetime was replaced with the entering (time instant when an enemy spawns), also turrets have to decide which enemy to shoot by using the concept of entering. Saying that younger enemies should be prioritized is equal to saying that enemies spawned later are prioritized, therefore, without loss of generality, we can use the entering information instead of lifetime.

3 GAME CONFIGURATION

3.1 Vanilla version

3.1.1 Configuration 1 : The unlucky side

3.1.2 Configuration 2 : Sniper's inefficiency

3.1.3 Configuration 3 :

3.2 Stochastic version

3.2.1 Configuration 1 :

3.2.2 Configuration 2 :

3.2.3 Configuration 3 :

4 VERIFICATION RESULTS

4.1 Queries for Vanilla version

4.1.1 Queries without turrets

Since these query must be verified without turrets, we can assume that no enemy will die in any time instant. We interpreted the queries given as (in romans numbers, the corresponded requirements queries):

I. [Q11] game is in deadlock state if and only if the match is ended;

II. [Q12] all enemies can reach the MT;

III, IV. [Q13] enemies reach the MT in no more than `MAX_PATH_LENGTH`$\times s$ time units (where s is speed's enemy);

V. [Q14] an enemy must always be in a valid cell.

Let's now analyze how we translated these natural-languages statements in Uppaal queries:

- [Q11] game is in deadlock state if and only if the match is ended:

```
A[]((deadlock imply matchEnded()) and (matchEnded() imply deadlock))
```

- [Q12] all enemies can reach the MT:
- [Q13] enemies reach the MT in no more than `MAX_PATH_LENGTH`$\times s$ time units (where s is speed's enemy):
- [Q14] an enemy must always be in a valid cell:

5 DISCARDED CHOICES

We wanted to write this additional section mainly to better clarify our design choices however, we want to clarify that our design choices are what we believed to be more efficient and adequate for our interpretation of the game. Some of the discarded can possibly be the best ones in other contexts or with other needs, also for these reasons we wanted to state them in the report.

5.1 MT template

Originally, a template for MT was designed. It was, for its simplicity, the very first designed template. We removed it in an attempt of optimizing the model, since we understood (after reasoning more on the more important design choices) that a "passive" entity like the MT does not really need to be implemented with a dedicated template (`decDamage` decreases MT's life of the value set in a global variable from the firing enemy):

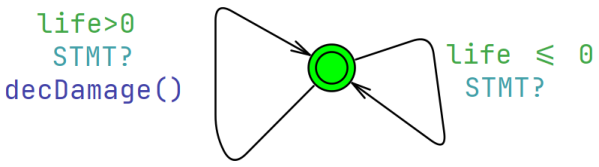

Figure 5: Original MT's template

Dividing enemy and turret in two templates is necessary to model in a proper and clear way the interleaving between these entities but MT never needs to trigger other components or to implement non-deterministic choices. The only one centralized behavior a dedicated template could have implemented was controlling that the MT's life is not 0 before decreasing it, but this can be easily moved in enemy template (before an enemy shoots, it checks that MT's life is not 0) without loss of readability. Of course, different requirements may have lead us to different design choices.

5.2 Hard-coded enemies paths

The really first enemy version in fact used the concept of `next()` but there was no idea on how to model the non-deterministic path choices (unless using `random()`, which is not available in symbolic simulations). The only idea was to hard-code cells vectors representing each straight red path on the map and enemies template would choose between them non-deterministically with transitions (once an enemy arrives in the last cell of a path, then it will start to follow non-deterministically one of the "next paths").

Then, we realized that since during the match there is event that can change the probability that a enemy takes a certain choice in a junction, without loss of generality path choices are always chosen by the time each enemy is spawned. This simple intuition really improved the readability of the enemy template.

5.3 Quadratic enemies scanning strategy

The very first turret version used to look for targetable enemies in this way: for each k from 1 to the shooting range, scan `SHOOT_TABLE` to find all the available enemies at exactly k cells of distance from the tower; then, choose the best iso-distant available enemy based on the criteria given in the requirements. The asymptotic complexity of this procedure is $\Theta(k \times MAX_{ENEMIES})$ which is significantly worse than the final version.

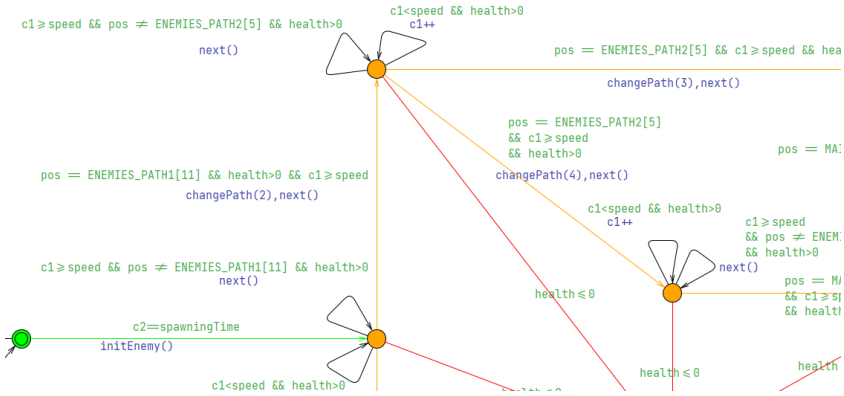

Figure 6: Original non-deterministic path choices example

5.4 Locks

[PRIMA O METTERLO]

6 CONCLUSIONS