UPPAALTD: A FORMAL TOWER DEFENSE GAME

Formal Methods for Concurrent and Real-Time Systems Homework

ANDREA BELLANI (andrea1.bellani@mail.polimi.it)

July 9, 2025

ABSTRACT Uppaal is a tool for modeling, simulating and verifying real-time system as networks of timed automatons. In this report, we present our modeling and verification of the game UppaalTD (both vanilla and stochastic versions) in Uppaal. In Addition, we also present the most interesting modeling choices we discarded or adjusted in the final version of the model.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

1.1 Definitions

- **requirements** : the official specifications of UppaalTD's parameters and rules;
- **alive**, **targettable** or **shootable** enemy : the enemy has an health strictly greater than zero and is present on the map;
- **dismissed** enemy : the enemy is dead or it shot the MT and the following delay has expired;
- **configuration** : wave's composition and turrets configuration chosen (generalized definition from requirements one);
- **ending** of a wave : a wave is considered *ended* when each enemy is dismissed (i.e. it can't move or shoot anymore because it has already shoot or it was killed by turrets);
- **winning** configuration : in Vanilla version, a configuration is winning if, by setting it, a wave of 3 circles and 3 square can never defeat the MT;
- **location** : a location of an Uppaal template;
- **state** : the entire game's state in a certain time instant (i.e. each automaton's position and each variable's or clock's value);
- *Chebyshev distance* : given two generic n-dimensional points $x = (x_1 \dots x_n)$ and $y = (y_1 \dots y_n)$, their Chebyshev distance can be computed as $\max_{1 \leqslant i \leqslant n} \{|x_i - y_i|\}$.

1.2 Project development timeline

april 8th	First brief analysis of the requirements.
april 9th	Uppaal and homework presentation.
april 11th	First definition of channels and MT's template.
april 26th	First definition of enemy and turret template (still not synchronized) and shoot to MT modeled with `STMTCONTROLLER`.
may 1st	Complete definition of all templates and synchronization without `STMTCONTROLLER`.
may 15th	First definition of queries.
may 17th	Enemy compact version.
may 24th	Re-design of synchronization with clocks.
june 15th	First stochastic version.
june 24th	Started delivery procedures.
june 26th	Started report writing.
july 9th	Finished report writing.

Table 1: Project development timeline

1.3 Document structure

Main sections:

1. **Introduction** : sum up of our definitions for the terminologies used in the document and version history of our model;
2. **Model description** : description of the models and focus on the most critical modeling choices;
3. **Verification results** : detailed analysis of the queries verified;
4. **Analysis of selected configurations** : presentation and deeper analysis of the models behaviors for some interesting configurations;
5. **Conclusions** : resume of the conclusions obtained.

Appendixes:

A **Discarded choices** : presentation of the most interesting discarded design choices and the motivations behind their rejection.

1.4 Software and machines used

Usage	Software	Versioning
Modeling, simulation and verification	Uppaal	5.0.0
Report writing	TeXstudio	4.6.3
Version	Git	2.40.0.windows.1

Table 2: Software used

Each query was verified used two different machines with different performances that here we briefly describe:

Machine name	CPU	RAM	OS	Manufacturing year
Machine 1	AMD Ryzen 5 3500U (2.1 GHz)	8 GB (5,95 GB usable)	Windows 11 Home	2020
Machine 2	11th Gen Intel(R) Core(TM) i5-1135G7 (2.4 GHz)	24 GB (23,7 GB usable)	Windows 11 Pro	2021

Table 3: Machine specifications

This document was written over the template Arsclassica Article (https://www.latextemplates.com/template/arsclassica-article) with few adjustments by us.

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 one. 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 Entities modeling (for Vanilla version)

2.1.1 Map

There is no template that models the Map. Each enemy and each turret has a `Cell` constant/variable that represents its actual position on the map (`Cell` is simply a `struct` with two bounded integers):

- an enemy keeps its position updated while moving with the function `next`. If an enemy is outside of the map (because it has not spawned yet or it was killed) its position still has a "feasible" value but a flag prevents turrets to read it (see the following sections for more details);
- a turret can't change its position, it simply uses it to calculate its distance from enemies.

To be clear, red paths structures are "embedded" into the function `next`. Another possibility (implemented in some of the previous versions) would have been to define a `Cell` matrix (or vectors) instead of define the entire structure as a less-readable the sequence of if-else implemented by `next`. We preferred this solution since `next` is constant in time and space however, with a more complex map, this approach could result in difficult to maintain `next` functions.

2.1.2 Main Tower (MT)

There is no template to model the Main Tower, it is simply modeled as a variable that enemies decrement in the shot. Originally, an MT template was designed (see the appendix for more details), it was removed in an attempt of optimizing the model, since we understood (after reasoning more on other more crucial design choices) that a so simple and "passive" entity like the MT does not really need to be implemented with a dedicated template.

Decoupling enemy and turret in two templates is necessary to model in a proper and clear way the interleaving between these entities but MT neither needs to trigger other components nor has any non-deterministic behavior. The only one centralized aspect that a dedicated template could have implemented was controlling that MT's life is not 0 before decreasing it, but this can be easily moved in enemy's template (before an enemy shoots, it checks that MT's life is not 0) without loss of readability.

2.1.3 Enemy

Let's explain the design choices beyond each enemy's behavior.

Spawn of an enemy
To model the fact that circles spawn "every x time units" and squares spawn "every y time units" we simply added a parameter `spawningTime` to enemy template. This parameter is simply a delay that must first elapse before an enemy can spawn on the map. By setting it (when we instantiate enemy processes) to `id`×`s` (where `s` is the spawning time defined for that kind of enemy in the requirements), the enemy with id 0 will spawn in the first time unit, the one with id `i` with a slack of `s` time units and so on. Note that, as we explain in the communication section, enemy ids must not overlap each other, therefore, if N enemies are present in the wave, and M of them are circles:

	circle(0)	...	circle(M-1)	square(M)	...	square(N-1)
id:	0	...	M-1	M	...	N-1
`spawningTime`:	0	...	2×(M-1)	0	...	3×(N-1-M)

Table 4: `spawningTime` assignment (using requirements spawning times)

Initialization of an enemy (this paragraph is only to group all the aspects relative to the enemy that will be further explained)
Once an enemy is spawned, it has to:

- initialize its record in the `shoot_table` and update the counter of targetable enemies;
- reset the `trip_time` clock;
- set `chosenPath`.

Move of an enemy After the speed delay has expired, an enemy has to make a move. In fact, a move is simply an update of the enemy's position with the function `next` and a consequent update of the `shoot_table` record (see the communication section).
The very interesting design choice behind the move of an enemy is how a (deterministic, since Uppaal `random` is not available in symbolic simulation) function like `next` can model the non-deterministic next cell choice an enemy takes in a red paths junction. Simply, the choices an enemy will take in the junctions is non-deterministically determined "a priori" when an enemy spawns ()by calling `initialize` with different parameters): We realized that since during the game there is no event that may change the probability that

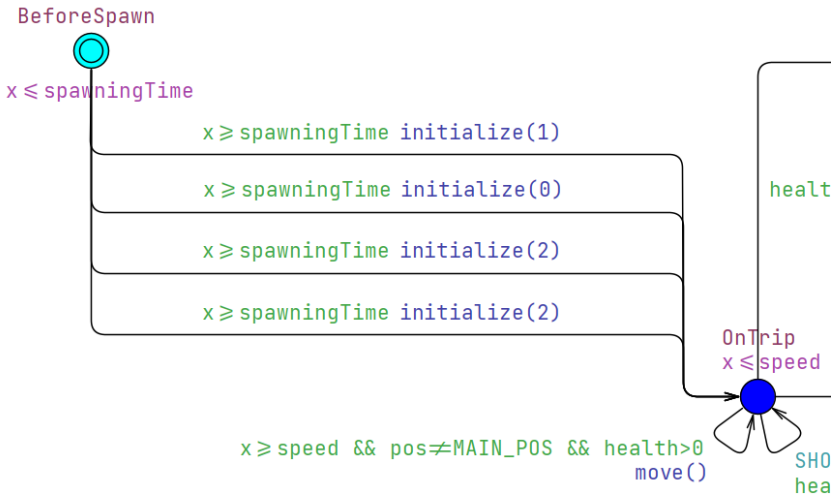

Figure 1: Close-up of how "a priori" non-deterministic path choice is implemented in the enemy template

an enemy takes a certain choice in a junction rather than another one, without loss of generality, these non-deterministic choices can be determined all at once by the time the enemy spawns.
This simple intuition really improved the readability of the enemy template (in the project development timeline we called this new version *compact enemy*) however, it can be convenient only if there are few possible choices.
Note that even if there exist three possible paths, it would be incorrect to model only one transition per-path. These transitions must model the same probability of taking any "sequence of choices". In other words, the probability of taking the choice "under" in the junction (7,4) has a 0.5 of probability to be taken, the other 0.5 is the probability of choosing "up" and then there is a 0.5 probability for each choice in the junction (10,7). If we want to determine these choices "a priori", it is not correct to model any possible path with a single transition, because it would lead the probability of taking any path to $\frac{1}{N}$ (where N is the number of possible paths to the MT).

Dismissal of an enemy Once an enemy has to leave the map, it simply updates properly the `shoot_table` record, the counter of the targetable enemies and the counter of left enemies.
A match is considered ended if the counter of left enemies is zero (i.e. all enemies spawned and leaved the map). Once the enemy is dismissed, it goes in a location where it waits that the match ends and then goes in an endless self-loop, this choice is in fact more crucial than it might seem:

- it prevents the system deadlock once the match is ended;
- putting a self-loop with no guard would of course prevent deadlock but it would possibly lead the starvation of other enemies (an enemy may arrive in the location and then self-looping indefinitely so other enemy could not move/shoot anymore). With the guard, this situation may happen only once all enemies are dismissed (and so they can't "starve" anymore).

4.4.1 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.

4.4.2 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]

4.4.3 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*.

4.4.4 Modeling of Enemy

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

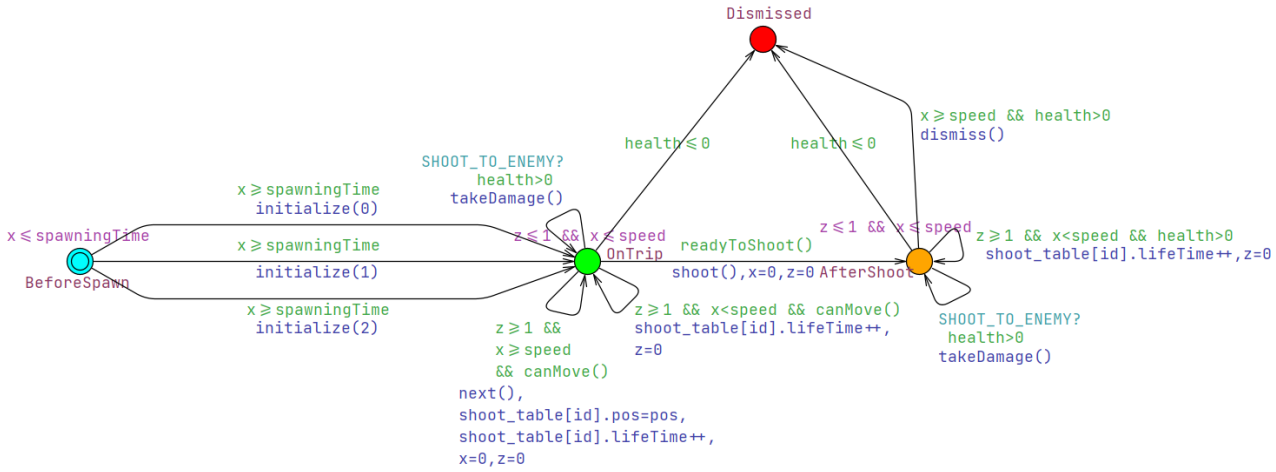

Figure 2: Enemy template

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.

4.4.5 *Modeling of Turret*

A turret has a pretty intuitive behavior:

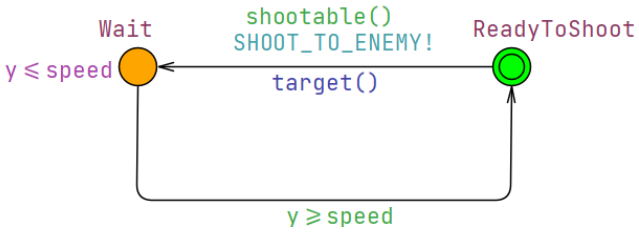

Figure 3: Turret template

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:

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 $\Theta(MAX_{ENEMIES})$, 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`.

4.5 Stochastic version's model

4.5.1 *Modifications from Vanilla model*

Stochastic model is not drastically different from the Vanilla one:

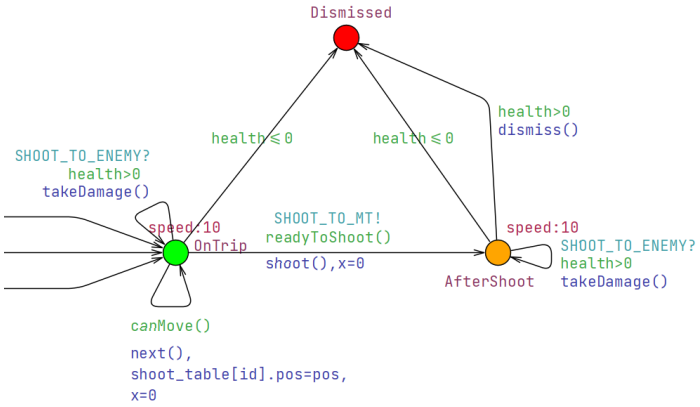

Figure 4: Enemy stochastic template

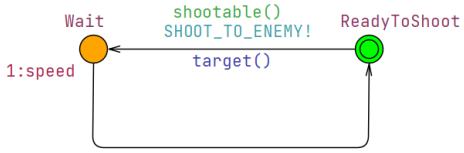

Figure 5: Turret stochastic template

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.

5 GAME CONFIGURATION

5.1 Vanilla version

5.1.1 Configuration 1 : The unlucky side

5.1.2 Configuration 2 : Sniper's inefficiency

5.1.3 Configuration 3 :

5.2 Stochastic version

5.2.1 Configuration 1 :

5.2.2 Configuration 2 :

5.2.3 Configuration 3 :

6 VERIFICATION RESULTS

6.1 Queries for Vanilla version

6.1.1 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:

6.1.2 Queries with turrets

6.2 Queries for Stochastic version

6.3 Consideration on efficiency

6.3.1 Starting spawning instant

6.3.2 Why urgent channels

6.3.3 Templates parameters

7 CONCLUSIONS

A DISCARDED CHOICES

We wanted to write this additional section mainly to better clarify the reasons behind our final design choices. Nonetheless, we would like to clarify that our design choices are what we believed to be more efficient and adequate for our interpretation of the game, so some of the discarded can possibly be the best ones in other contexts or with different requirements and also for this reason we wanted to state them in the report.

A.1 MT template

Originally, a template for MT was designed. It was, for its simplicity, the very first one to be designed (`decDamage` decreases MT's life by the value set in a global variable from the firing enemy):

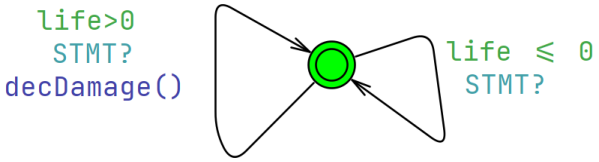

Figure 6: Original MT's template

A.2 Hard-coded enemies paths

The very first enemy version used the concept of `next` but there was no idea on how to model the non-deterministic moves (unless using function `random`, which is not available in symbolic simulation). The only idea was to hard-code vectors of cells representing each straight red path on the map and enemies template would have chosen 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"):

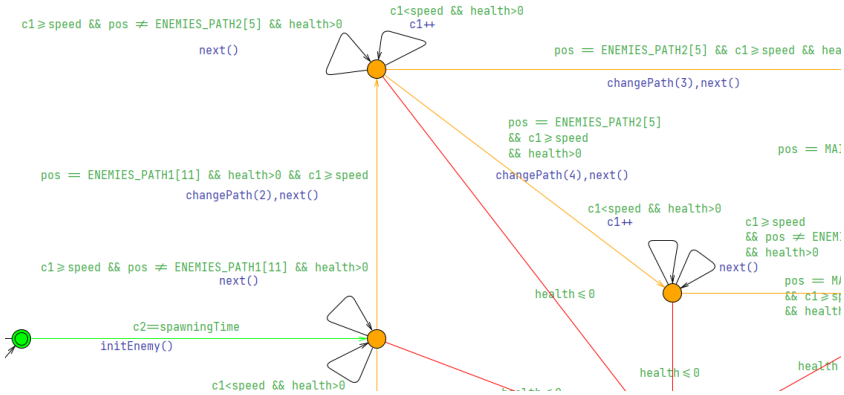

Figure 7: Original non-deterministic path choices example

A.3 Quadratic enemies scanning strategy

The very first turret version used to look for enemies to shoot in this way: for each k from 1 to the `range`, scan `SHOOT_TABLE` to find all the available enemies at exactly k cells of distance; then, choose the best iso-distant available enemy based on the requirements criteria. The worst-case asymptotic complexity of this procedure was $\Theta(k \times MAX\_ENEMIES)$ which was significantly worse than the one of the final version which is $\Theta(MAX\_ENEMIES)$ especially in the average case where there is no enemy inside the shooting range.

A.4 Locks

At the beginning, the first way of synchronizing entities was thought in a classical lock-unlock manner: Once an enemy reaches the MT (for the MT template please see the proper section of the appendix):

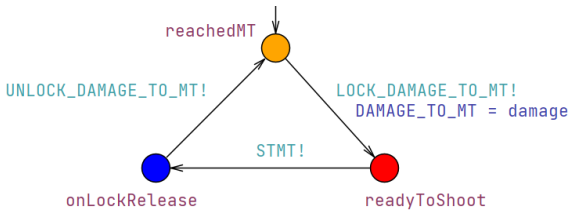

Figure 8: Close-up of the enemy locking STMT channel

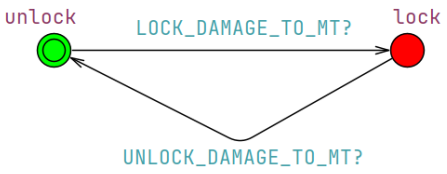

Figure 9: STMTCONTROLLER template

1. the enemy sends a message to STMTCONTROLLER and waits for its reply (more precisely, Uppaal chooses non-deterministically which one of the ready enemies can send the message to the controller);
2. once the controller replies the enemy places in the shared variable the damage for the MT;
3. the enemy shoots to the MT (i.e. sends a message over STMT);
4. the enemy sends a message to STMTCONTROLLER to release the "lock".

We understand that this solution is:

- deadlock-free: soon or later an acquired lock will be released and soon or later a lock request will be accepted;
- not starvation-free: since it is not guaranteed that any enemy that requests a lock will eventually obtain it.

We removed this concept since we understood that a single transition that both changes the global variable and performs the shoot would have produced the same behavior, since, provided that this transition is not synchronized with other enemies, only one of them can perform it in a time instant, therefore there is no possibility that an enemy places the damage for the MT into the shared variable and before sending the shooting message another enemy changes the variable and (or not) shoots to the MT (which would clearly create an undesired behavior).

A.5 Lifetime counter

As we have seen, turrets understand that an enemy is present on the map for a shorter amount of time by looking to its spawning. This idea lets enemies to not keep a counter updated to tell the turrets that they are present on the map for tot. time units. However, this idea is relatively new in the history of the model, since at the beginning, the lifetime of an enemy used to be update in each time unit by the enemy:

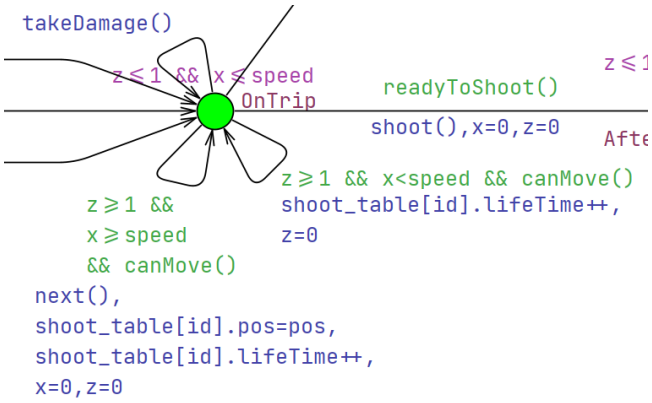

Figure 10: Close-up of of the lifetime counter of an enemy

With clock z at each time unit the lifetime counter would have been updated.

Note that this solution was also used to verify that enemies would have reached the MT spot in no more than MAX_PATH_LENGTH $\times$ *speed* time units.

REFERENCES

[1] P. San Pietro A. Manini. "The Uppaal Tool". 2005. URL: https://webeep.polimi.it.

[2] K. G. Larsen G. Behrmann A. David. "A Tutorial on Uppaal 4.0". In: (2006). URL: https://uppaal.org/texts/new-tutorial.pdf.

[3] I. Lee. *UPPAAL tutorial*. 2009. URL: https://www.seas.upenn.edu/~lee/09cis480/lec-part-4-uppaal-input.pdf.

[4] L. Lestingi. "Statistical Model Checking and Uppaal SMC". 2021. URL: https://webeep.polimi.it.

[5] F. Corradini M. Bernardo. *Formal Methods for the Design of Real-Time systems*. 2004. URL: https://link.springer.com/book/10.1007/b110123.

[6] P. San Pietro. "Lecture slides of FORMAL METHODS FOR CONCURRENT AND REAL-TIME SYSTEMS course". 2025. URL: https://webeep.polimi.it.

[7] A. Maggiolo Schettini. *Veryfing Real–Time Systems - The Uppaal Model Checker*. 2007. URL: http://groups.di.unipi.it/~maggiolo/Lucidi_TA/VerifyingTA-Uppaal.