POLITECNICO
MILANO 1863

UppaalTD: a Formal Tower Defense Game

Authors
Leila Shekofteh, Juan Marco Patagoc, Francesco Marchi

Instructors
Prof. Pierluigi Sanpietro, Andrea Manini

Homework Project

A.Y 2024-2025

Abstract

The purpose of this report is to show how we can create a formal representation of a simple problem, how to exploit it to formally verify its properties and how we can extract information on the problem using formal results. The problem in question is a *Tower Defense Game*, a strategy game in which the player aims to protect valuable assets from an enemy invasion by strategically setting up defences.

1 Model Description

The high level description of the game is as follow. The model has 4 main components:

Main Tower The strategic asset to be defended, characterized by *life points*. **The game is considered to be lost if its life points drop to 0 or below**.

Turrets Our main defences. **Three type of turrets are available**: Basic Turrets, Cannon Turrets and Sniper Turrets, each with a different attack value, rate of fire and range.

Enemies Our assailants. Two type of enemies may appear:

- Circles: light and speedy enemies meant to overwhelm us;
- Squares: slower, tankier enemies that can withstand more punishment.

Map Our battle ground. A 16 x 8 grid where enemies move following a given path and where turrets can be placed on specific cells.

2 Upaal Model

2.1 Map

The map is represented internally as a 16 x 8 two-dimensional array of integers, *PATH* cells denote where the enemy will move (bifurcation points must be explicitly declared). The map itself can be modified, provided there is a path from the cell (0,0) that leads to the main tower. This solution is a trade-off between ease of use and scalability.
The Map Template doesn't represent the game map per se, its purpose is instead to be a **middle man between other components by handling messages**. Starting from its *Idle* state, the automata can receive two types of messages:

- *movement?*: an **enemy on the map moved**, the automaton notices and sends a message through the *turret_notify!* channel, **informing all turrets that an enemy moved** and they should check whether or not the enemy can be targeted;
- *enemy_leave?*: **an enemy was killed or it attacked the main tower and left**, in any case the automaton checks if the conditions to end the game are satisfied. If the conditions are satisfied the automaton sends a message to every component through the *end_all!* **informing that the game has ended**.

2.2 Turret

Turrets are **characterized by a set of parameters** (attack, rate of fire, range) that can be easily changed. Turrets are placed by specifying a parameter in each of the turret state arrays (atks[], rofs[], ranges[]) which share indexes with an array of positions (a locally declared struct) indicating the position of each turret. As such, number of turrets and their position can be easily changed provided they don't overlap with enemy paths (note that the cells in the grid map occupied by *TURRET* don't actually affect anything in the internal functions of the game, it's simply a visual cue).
The template features four main states:

- *idle*: the **turret is idle**, it can receive a message through the *turret_notify?* channel, this tells the turret that an enemy has moved so it must scan for suitable targets and move to the *ready_to_fire* state.

- *ready_to_fire*: in this state the turret can either go back to the *idle* state or to the *reload*. In the first case the turret found **no available targets**, in the second a **valid target was found and as such the turret fired, applying damage**. The internal clock is reset.

- *reload*: the **turret has fired**, it now must wait a time equal to its rate of fire before going back to the *idle* state. In the transition between states the turret scans for suitable targets and, **if they are found, will immediately apply damage, otherwise the turret will go back to the idle state** as there is no point in continuously scanning for targets unless a movement is detected.
  In the stochastic version the reload time is not fixed, **the state has a rate of exponential** equal to $\lambda_{turret} = 1/\mathit{fireSpeed}$, meaning that, once the guard is satisfied, the probability of leaving the state after a time t is equal to $Pr = 1 - e^{-\lambda t}$.

- *End*: **end state**, the game has ended.

Note that from all states a message through the *end_all?* channel can be received, **meaning that the game has ended** and so the template must move to the *End* state.

2.3 Enemies

The system keeps track of the enemy states **through a set of arrays that contain their position, health and the amount of time the enemy has been alive for**; an additional array keeps track of the status of enemies (whether or not they left the map or they have been destroyed). All parameters related to enemies are stored in state arrays that functions can freely access **instead of being directly passed to the templates**. Furthermore, enemies that still need to spawn sit still in cell (0,0) and **considered *not alive*, once the spawning criteria are met the system changes their status as *alive***.

The system two different templates for each of the two enemy types, the structure is the same. Their states are:

- *Spawn*: the **initial state**, the automaton sits in this state until the spawning criteria are met. When the criteria are satisfied the transition is triggered, coordinates are initialized, the enemy is flagged as *alive* and the coordinates of the next cell in the enemy's path is computed. Additionally, the system synchronizes on the movement channel.

- *prepare_to_move*: committed state needed to distinguish between the cases where: **a normal move must be performed**, **the enemy has reached a bifurcation and must act accordingly or the enemy is in front of the Main Tower**. The outgoing transitions also update the enemy's health.

- *Move* and *Move_bifurcation*: after an amount of time equal to the speed value of the enemy, the outgoing transition is taken and the position of the enemy in the global state array is updated. In the case of a bifurcation, the automaton **non-deterministically chooses one of the two possible paths**. **Another possibility in this state is that the enemy died**, and as such the transition to the *leave* state must be taken.
  Additionally, **the stochastic version adds a rate of exponential** to the state equal to $\lambda_{enemy} = speed/10$.

- *Decide*: **the automaton checks whether or not the enemy is still alive or not**. If still alive the transition to *prepare_to_move* must be taken, computing the next position in the meantime.

- *prepare_to_attack*: the enemy is right in front of the Main Tower, **one final delay must elapse before the enemy con move on top of the Main Tower and damage**. The enemy can **still die** in this state.

- *Attack*: the **enemy has completed the attack**, after a delay equal to its speed it leaves the map.
  In the stochastic version this **state also has the rate of exponential**.

- *Leave*: the **enemy has left the map**, all transition to this state flag the enemy as dead as it can no longer be targeted by any turret.
  As all the other states where an enemy moves, **the stochastic version has a the same rate of exponential**.

Note that, as for turrets, every state can synchronize on the *end_all?* channel if the game is over.
One last thing to note is the *getNextPositions()* function, responsible for computing the enemy path. The function uses heuristics and the internal map representation to correctly guide enemies through the map.

The chosen solution is very easy to work with, distinct templates for each enemy type **rules out the need to pass parameters within templates**, it also **rules out the need to have long arrays with repeated constants for each individual enemy** which is not a concern for the vanilla version where the number of enemies is a manageable 6, but has proven very useful for the stochastic version where the enemy count reaches upwards of 700. There is, however, no free lunch: this scheme would not handle the addition of more enemy types very well, multiple additional templates would need to be created.

3 Analysis and Results of the Vanilla Version

The specification required us to verify a set of properties.

3.1 Property I

The first property, to be verified without turrets, is as follows:

The game never reaches a deadlock state.

The property was verified using the following query:

$$A\Diamond\, M.GameOver$$

The query **doesn't actually verifies the absence of deadlocks** as a deadlock state will eventually be reached, **the GameOver state** (the system would go on forever otherwise). Because of this the query simply **checks that ALL paths EVENTUALLY reach the *GameOver* state**.
The verifier yields a **positive result**: this means that **the only way for the system to be in a deadlock state is to be in the game over state**, in other words the game has ended correctly.

3.2 Property II

The second property, to be verified without turrets, is as follows:

All enemies can reach the Main Tower.

The property was verified using the following query:

$$\begin{aligned} A\Diamond\,(&C(0).\text{ReachMainTower} \wedge S(0).\text{ReachMainTower} \wedge \\ &C(1).\text{ReachMainTower} \wedge S(1).\text{ReachMainTower} \wedge \\ &C(2).\text{ReachMainTower} \wedge S(2).\text{ReachMainTower}\ ) \end{aligned}$$

This **property was verified by setting the health points of the main tower to 1000**, enemies would simply destroy the tower otherwise, denying other enemies the possibility of reaching it and thus making the verification fail. The query simply checks that all paths eventually lead to a state where every enemy has the boolean *ReachMainTower* set to true.
The **property is satisfied**, this means that **eventually all enemies reach the Main Tower**.

3.3 Property III

The third property, to be verified without turrets, is as follows:

> *Circles reach the Main Tower in no more than $n \cdot c$ time units, where n is the length of the longest path from the start of the Map to the Main Tower, and c is the speed of the Circles.*

The property was verified using the following query:

$$
\begin{aligned}
A\Diamond\, ((&\text{globalClock} \leq \text{LONGEST\_PATH} \cdot \text{CIRCLE\_SPEED} \\
&+ \text{CIRCLE\_SPAWN\_INTERVAL} \wedge C(0).\text{Attack}) \\
\vee\, (&\text{globalClock} \leq \text{LONGEST\_PATH} \cdot \text{CIRCLE\_SPEED} \\
&+ 2 \cdot \text{CIRCLE\_SPAWN\_INTERVAL} \wedge C(1).\text{Attack}) \\
\vee\, (&\text{globalClock} \leq \text{LONGEST\_PATH} \cdot \text{CIRCLE\_SPEED} \\
&+ 3 \cdot \text{CIRCLE\_SPAWN\_INTERVAL} \wedge C(2).\text{Attack}))
\end{aligned}
$$

As before, the property was **verified by setting the health points of the main tower to 1000**. The query checks that all paths eventually lead to a state where at least one circle is in state *Attack* and the *globalClock* is less than $25 \cdot 1 + (\text{id} + 1) \cdot 2$.
The condition adjusts for different spawning times, **reason why we check that at least one enemy satisfies it (if one satisfies it then all other enemies also must satisfy it)**.
The **property is satisfied**, this, along with property II, **guarantees that all circles not only always reach the turret but they also reach it in the correct amount of time**.

3.4 Property IV

The fourth property, to be verified without turrets, is as follows:

> *Squares reach the Main Tower in no more than $n \cdot s$ time units, where n is the length of the longest path from the start of the Map to the Main Tower, and s is the speed of the Squares.*

The property was verified using the following query:

$$
\begin{aligned}
A\Diamond\, ((&\text{globalClock} \leq \text{LONGEST\_PATH} \cdot \text{SQUARE\_SPEED} \\
&+ \text{SQUARE\_SPAWN\_INTERVAL} \wedge S(0).\text{Attack}) \\
\vee\, (&\text{globalClock} \leq \text{LONGEST\_PATH} \cdot \text{SQUARE\_SPEED} \\
&+ 2 \cdot \text{SQUARE\_SPAWN\_INTERVAL} \wedge S(1).\text{Attack}) \\
\vee\, (&\text{globalClock} \leq \text{LONGEST\_PATH} \cdot \text{SQUARE\_SPEED} \\
&+ 3 \cdot \text{SQUARE\_SPAWN\_INTERVAL} \wedge S(2).\text{Attack}))
\end{aligned}
$$

Just as in property II and III, the property was **verified by setting the health points of the main tower to 1000**. The query checks that all paths eventually lead to a state where at least one circle is in state *Attack* and the *globalClock* is less than $25 \cdot 2 + (\text{id} + 3) \cdot 2$.
The condition adjusts for different spawning times, **reason why we check that at least one enemy satisfies it (if one satisfies it then all other enemies also must satisfy**

it).
The **property is satisfied**, this, along with property II and III, **guarantees that all enemies not only always reach the turret but they also reach it in the correct amount of time**.

3.5 Property V

The fifth property, to be verified without turrets, is as follows:

> *All enemies never leave the red path.*

The property was verified using the following query:

$$
\begin{aligned}
A\Box\ (&S(0).\text{Spawn} \vee (S(0).\text{next}[0][0] \neq -1 \wedge S(0).\text{next}[0][1] \neq -1) \\
&\wedge S(1).\text{Spawn} \vee (S(1).\text{next}[0][0] \neq -1 \wedge S(1).\text{next}[0][1] \neq -1) \\
&\wedge S(2).\text{Spawn} \vee (S(2).\text{next}[0][0] \neq -1 \wedge S(2).\text{next}[0][1] \neq -1) \\
&\wedge C(0).\text{Spawn} \vee (C(0).\text{next}[0][0] \neq -1 \wedge C(0).\text{next}[0][1] \neq -1) \\
&\wedge C(1).\text{Spawn} \vee (C(1).\text{next}[0][0] \neq -1 \wedge C(1).\text{next}[0][1] \neq -1) \\
&\wedge C(2).\text{Spawn} \vee (C(2).\text{next}[0][0] \neq -1 \wedge C(2).\text{next}[0][1] \neq -1))
\end{aligned}
$$

Simply checks that for all paths it is always true that every enemy is either in the *spawn* state or the coordinates of their next position is different than (-1,-1) (which represent invalid coordinates). The verifier yields a **positive result**.
Properties I through V guarantee the correct functionality of the base game.

3.6 Property VI

Property number six asks us to verify whether the standard configuration is winning or losing. The query used is simple:

$$E\Diamond\ \text{mainTowerHP} \leq 0$$

Simply checks if a path where the condition is evaluated exists. **If it does exist the game was lost, if not the game was won**.
The verifier **yields a negative result**, meaning that the **game was won**. This is not surprising, as the configuration used is very powerful: snipers cover almost the entire map, taking out big chunks of health for every shot; the basic turret with its high rate of fire softens up targets that reach the main bifurcation; cannons deal the finishing blow.

3.7 Property VII

Property number seven asks us to verify that in the previous configuration the game never goes into a deadlock state.
The query used is the same for property I and it yields a **positive result**.
This result, paired with the results from properties I through V, guarantees the correct execution of the game.

3.8 Property VIII

We were required to provide two different configurations.

3.8.1 Configuration II: No Reinforcements

Multiple battles have worn your resources thin and reinforcements are not coming, all you have left are some cannons and a sniper.
The configuration consists of only 4 turrets:

- Sniper in cell $(2, 3)$;
- Cannon in cell $(5, 5)$;
- Cannon in cell $(8, 2)$;
- Cannon in cell $(8, 6)$;

The **result is a loss**. Unfortunately all units at our disposal have long cooldowns between each shot and the final steps in the path are completely defenceless meaning that unless enemies are immediately destroyed they are free to go to the main tower.

3.8.2 Configuration III: Battle of Annihilation

Annihilation is a military strategy in which an attacking army seeks to entirely destroy the military capacity of the opposing army in a single decisive blow. The key idea is trying to destroy enemies as soon as possible.
The configuration is as follows:

- Basic in cell $(2, 3)$;
- Basic in cell $(5, 5)$;
- Cannon in cell $(8, 2)$;
- Cannon in cell $(8, 6)$;
- Sniper in cell $(11, 5)$;
- Sniper in cell $(14, 2)$;
- Sniper in cell $(14, 6)$;

The property is not satisfied meaning that this is a **winning configuration**. Once again, not a surprising result: **basic turrets with their high rate of fire take out a big chunk health from enemies leading them in range of cannons and sniper**. The annihilation strategy is typically used in situations where time is against us, which is exactly our case.

4 Analysis and Results of the Stochastic Version

The following simulations were performed using the following parameters:

- $\varepsilon = 0,05$ - The probability uncertainty
- $\pm\alpha = 0,05$ - The probability of false positives

We decided on these values because of the relatively high complexity of the system and because of the **trade-off between time required and precision not being favourable**.

4.1 Property I

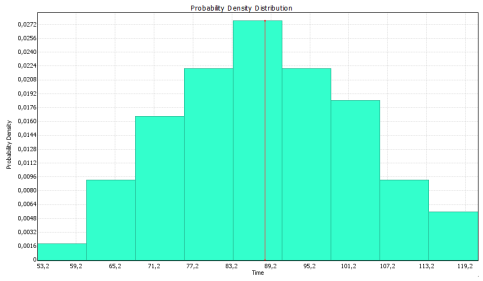
The first property, to be verified with turrets, is as follows:

> *Simulate the system for a maximum timeout of 200 time units and estimate how long the Main Tower remains undamaged in the configuration provided in the query (VI) in the previous section.*

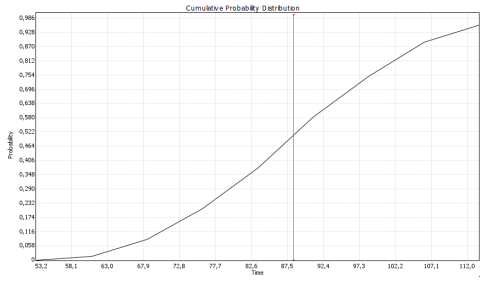
The results we obtain from 100 simulations show that the **earliest time in which the Main Tower takes damage is around 60**, while the **latest time is around 115**. This is expected: the probability distribution used is the *exponential distribution*, which has an expected value equal to $\frac{1}{\lambda}$ (where λ is the rate of exponential). This means that ***on average*** squares will move after $\frac{1}{0,3} = 3,33$ time units, while circles will take $\frac{1}{0,1} = 10$ time units; so by assuming that turrets will not be present and by taking squares, which are faster, and multiplying the time required for each step for the numbers of steps required to reach the tower, we obtain $3,33 \cdot 25 = 83,25$. **We can hypothesize that on average a square will take 83.25 time units to reach the tower (assuming a turret doesn't destroy it sooner)**.
We proved this initial hypothesis with the following query:

$$\Pr[\leq 200]\,(\Diamond\, \text{mainTowerHP} \neq 0)$$

The query yielded the following results



(a) Probability Density Distribution



(b) Cumulative Probability Distribution

Figure 1: The tower is guaranteed to take damage after a certain time.

The mean is equal to 88.25, which is very close to the initial hypothesis. This means that **on average the turrets hold out enemies for just 5 time units**.

4.2 Property II

The second property, to be verified with turrets, is as follows:

> *Simulate the system for a maximum timeout of 200 time units and estimate how long the Main Tower remains undamaged in the configuration provided in the query (VI) in the previous section.*

The probability was computed using the following query:

$$\Pr[\leq 200]\,(\Box\,\text{mainTowerHP} > 0)$$

which yielded $\Pr \in [0.687667\,,\,0.783396]$ with a confidence of 95% over 253 runs. This is not surprising as we saw that **in the vanilla version configuration I was a winning configuration**.
However by performing additional simulations with a higher time out we can observe that the estimation of the probability of winning converges to $\simeq 5\%$. This suggests that **200 time units may not be enough time for every enemy to actually reach the tower**.

4.3 Property III

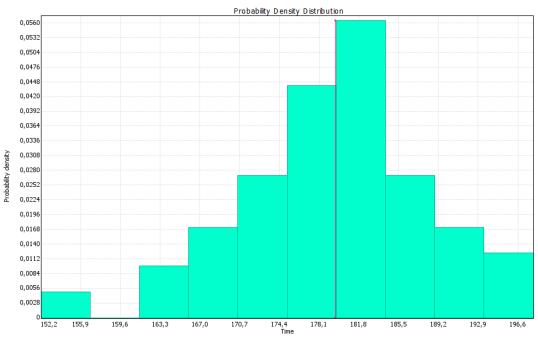
The third property asks us to perform simulations using two other configurations of our choice.

4.3.1 Configuration II: Snipers Nest

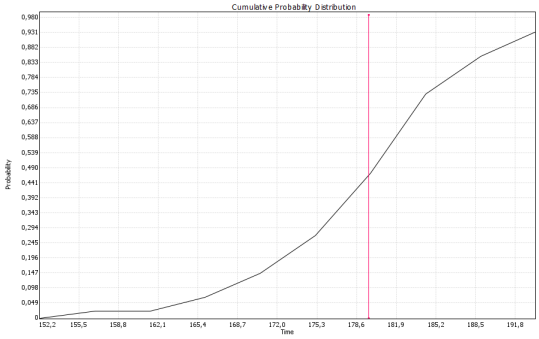
This configuration is composed of six sniper turrets.
We tested the probability of the tower being destroyed with the following query:

$$\Pr[\leq 200]\,(\Diamond\,\text{mainTowerHP} \leq 0)$$

which yielded Pr $\in$ [0.938982 , 0.999716] with a confidence of 95% over 88 runs.



(a) Probability Density Distribution

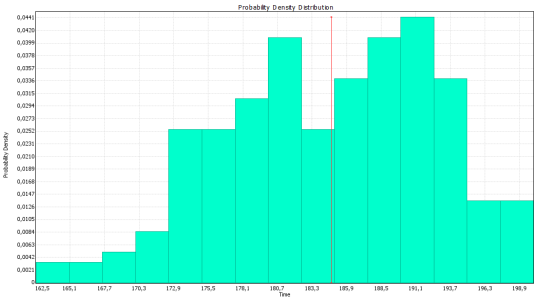


(b) Cumulative Probability Distribution

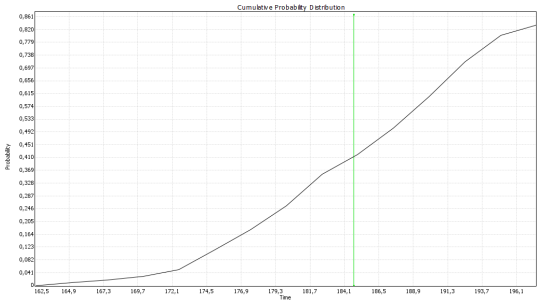
This configuration is similar to the one used in property II of the vanilla version, **which we saw was a losing configuration**. This explains **why the probability of losing is so high**. Furthermore we can observe a steeper incline in the cumulative probability distribution as time goes on, followed by a more gentle incline, this could be **a conseguence of the long cooldown on the sniper turrets**: in the beginning every turret is ready to fire, managing to hold out enemies, but as more turrets go into a waiting state, the ability to handle enemies decreases.
Furthermore we observe that by increasing the timeout, the probability converges to the same value we have found before: **this suggests that losing is inevitable, it is merely a matter of *when* the game ends**.

4.3.2 Configuration III: Artillery Barrage

The configuration is composed by 5 cannons and the query used was the same as before. The query yielded a probability of Pr $\in$ [0.818759 , 0.908975] with a confidence of 95% over 205 runs.



(a) Probability Density Distribution



(b) Cumulative Probability Distribution

The **incline is more linear**, suggesting that, while the higher rate of fire certainly helped

defending the tower, their **lower damage was simply not enough to handle all the enemies leading to the tower being overwhelmed**. Once again not a surprising result, seeing as a configuration like this would easily lead to a loss in the deterministic version of the system.