A Formal Tower Defense Game With Uppaal

Trygve Myrland Tafjord

July 24, 2025

POLITECNICO
MILANO 1863

Contents

1 Introduction

This report details the formal modeling and verification of `UppaalTD`, a tower defense game, using the UPPAAL model checker. Our primary goal was to translate the game's informal rules, including enemy movement, turret targeting, and win/loss conditions, into a precise network of timed automata. The system is modeled through three core components: an Enemy Manager to control enemy spawning and movement, a Turret Manager to handle all defensive actions, and a Game Execution automaton to oversee the game state and determine its final outcome.

This report and the accompanying code focus exclusively on solving the Vanilla Version of the game, which involves a non-stochastic model with a fixed number of enemies and predefined parameters.

2 Component Description

2.1 Enemy Manager

The Enemy Manager automaton orchestrates the timed release and movement of all enemy units along the fixed path toward the Main tower. Only one clock is used to control all the enemies, this was a deliberate design choice to minimize the system state space and enhance verification efficiency. The enemy manager manages both circle and square enemies. This design choice allowed the group to use one clock for all enemy types which improves verification time and simplifies the synchronization between enemy movements. The disadvantage of this approach is that it makes the code more complex, and a lot of the logic requires the iteration through both enemy types, creating a more cumbersome codebase. There is an argument to be made for splitting the EnemyManager into one spawn manager and one move manager, as this would create two modules with smaller areas of responsibility, however as mentioned earlier the group opted for a more performance focused solution, and further more by merging the spawn manager and the move manager less synchronization is required.

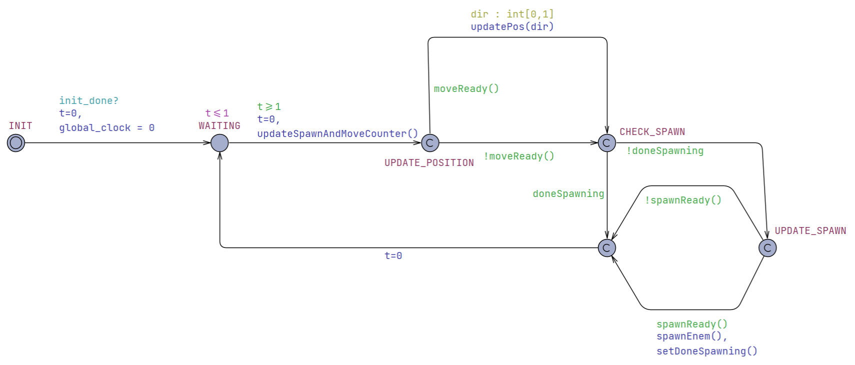

Figure 1: EnemyManager Automaton

The enemy managers has been modeled as follows:

1. **Spawning**
   For each clock cycle, the enemy manager updates *timeSinceLastSpawn* by one. This variable contains the time since the last spawn for both the square and circle enemy type. If *timeSinceLastSpawn* is higher than the spawn delay, a new enemy is spawned. This is done by setting the enemy health in the global enemy health variable. Once all enemies are spawned, the *timeSinceLastSpawn* is no longer updated.

2. **Movement**
   In parallel, for each clock cycle, every enemy type's timeSinceLastMove increments. Once it reaches the type's speed threshold, the automaton

advances every live unit of that type one grid cell along the defined path. Only live enemies are moved, and liveliness is checked by reading the global *enemy_health* variable.

3. **Path branching**
   At intersections, the system applies the externally supplied direction decision to choose the correct branch without altering its own timing logic. Direction is to be chosen randomly in intersection, and the edge select is utilized to achieve this.

4. **Tower engagement**
   Upon reaching the tower's cell, it withdraws the unit from further simulation by setting its HP to zero. Furthermore it decrements the tower's HP by the enemy's damage amount, and increments the global *num_dead_enem* variable.

2.2 Turret Manager

The TurretManager automaton regulates all defensive firing actions across heterogeneous turrets, weaving together initialization, cooldown management, target selection, and shooting into a single continuous cycle. Again, only one clock is used for all turrets. This design choice is crucial for reducing the statespace and providing a fast runtime for the verifications, especially checking for deadlocks.

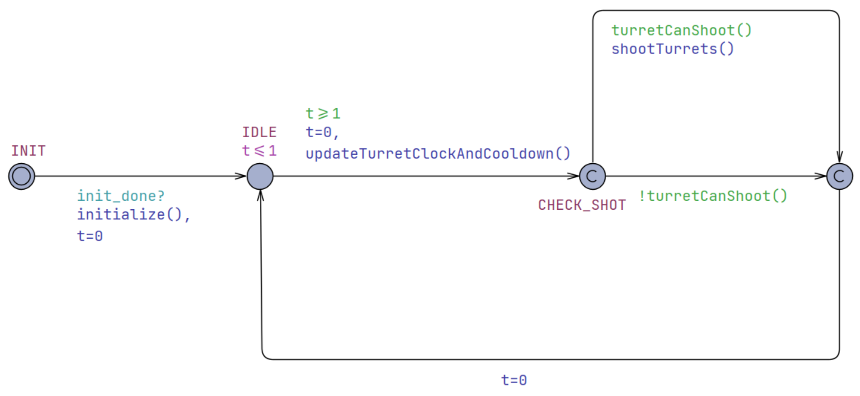

Figure 2: TurretManager Automaton

1. **Startup Synchronization.** Upon receiving the global "initialization done" signal, TurretManager resets every turret's cooldown state and the internal clock, ensuring all turrets begin ready to fire (though no shots are yet taken).

2. **Cooldown Monitoring.** It then enters a steady loop: first scanning all turrets and re-arming any whose cooldown timer has completed, then, if no turret is ready, incrementing the cooldown timers of all still-cooling turrets, waiting one time unit, and repeating this check.

3. **Target Scanning.** As soon as at least one turret is re-armed, TurretManager suspends cooldown updates and, for each ready turret, inspects every live enemy within its firing radius. It ranks potential targets using a strict hierarchy. First, turrets prioritize attacking Squares over Circles. Then, for the chosen enemy type, it selects the enemy closest to the turret. If multiple enemies are at the same distance, the one that has been on the map for the shortest time is targeted to pick the best candidate.

4. **Firing Action.** In one atomic step, each ready turret:

   - Shoots its selected target, subtracting its damage from that enemy's health.
   - Immediately re-enters cooldown (resetting its timer).
   - If an enemy's health drops to zero, records the kill and removes the unit from play.

5. **Repeat Loop.** After firing, TurretManager resets its internal clock and returns to the cooldown-monitoring phase, enforcing the configured inter-shot delay. This cycle—monitor → scan → fire → reset, continues until the game concludes.

2.3 Game Execution

The Game Execution section oversees start-up and end-game detection, modeling its behavior in three stages:

1. **Initialization** at launch it runs an init routine that resets all enemies health and position to zero as well as assigning its shape.

   then there are two checks

2. **Victory check** If the total number of dead circles and squares equal the predetermined number of enemies and the Main tower is not dead then the game must be won.

3. **Loss** In parallel it watches the remaining life of the Main Tower. If those ever fall to zero the game ends up in a loss state.

2.4 Channels

The model utilizes a single broadcast channel, init_done, to perform an initial handshake that ensures all automata have initialized their local parameters before the simulation begins.

Beyond this initial handshake, further channels for synchronization were found to be unnecessary for the model's core logic. The primary interaction between the EnemyManager and TurretManager templates occurs through shared global variables, notably enemy_health and enemy_pos. The access pattern for these variables is well-defined: one automaton is the designated writer, while the other acts as a reader.

This design choice is justified because the system operates on a time-driven, rather than event-driven, basis. State changes, such as an enemy taking damage or moving to a new cell, do not require an immediate, synchronized action from another automaton. Instead, each manager polls the status of these shared variables at discrete time intervals (every one time unit, as implemented). Consequently, the use of shared variables without explicit channel synchronization is a deliberate and efficient modeling choice that avoids unnecessary complexity.

3 Results

3.1 Game Configurations

The game-configurations used in the Vanilla-version are summarized in the following table. Overall all the queries were able to run in a reasonable time.

Table 1: Game Configurations

Configuration	**# of Turrets**	**Turret Types and Positions**	**Result**
Vanilla	7	1x Basic at (5,5) 4x Cannon at (8,2), (8,6), (14,2), (14,6) 2x Sniper at (2,3), (11,5)	Win
Alternative 1	7	7x Basic at (5,5), (8,2), (8,6), (14,2), (14,6), (2,3), (11,5)	Win
Alternative 2	7	7x Sniper at (5,5), (8,2), (8,6), (14,2), (14,6), (2,3), (11,5)	Win
Alternative 3	4	4x Basic at (5,5), (8,2), (8,6), (14,2)	Win
Alternative 4	2	2x Basic at (5,5), (8,2)	Loss

The verification results are presented in the following table.

Table 2: Verification Results for the Vanilla Version

Property Description	**Result**	**Time to Verify**
Properties Verified Without Turrets		
The game never reaches a deadlock state.	`Verified`	0.021s
All enemies can eventually reach the Main Tower.	`Verified`	0.011s
Circles reach the Main Tower within the calculated maximum time.	`Verified`	0.018s
Squares reach the Main Tower within the calculated maximum time.	`Verified`	0.018s
Enemies always remain on the designated red path.	`Verified`	1.219s
Properties Verified With Turrets		
The specified "Vanilla" configuration is a winning configuration.	`Verified`	0.489s
The game, with the "Vanilla" turret configuration, never deadlocks.	`Verified`	2.677s
The "Alternative 1" configuration (7 Basic turrets) is winning.	`Verified`	0.387s
The "Alternative 2" configuration (7 Sniper turrets) is winning.	`Verified`	0.433s
The "Alternative 3" configuration (4 Basic turrets) is a winning configuration.	`Verified`	0.293s
The "Alternative 4" configuration (2 Basic turrets) is a losing configuration.	`Verified`	0.239s