Modeling Bloons Tower Defense as a temporal two-dimensional knapsack problem with irregular shapes and side constraints: integer programming-based approaches

Maxence Delorme[(1)] and Jakub Malinowski[(2)]

(1) Department of Econometrics and Operations Research, Tilburg University, The Netherlands
(2) Department of Computer Science, ETH Zurich, Switzerland
Corresponding author m.delorme@tilburguniversity.edu

Abstract

In Tower Defense (TD) games, the objective is to defend a specific point on the game map from mobile units by constructing towers with offensive capabilities. In this work, we focus on Bloons Tower Defense (Bloons TD), one of the earliest and most prominent TD games. We show that the problem of finding tower configurations that win Bloons TD can be formulated as a temporal two-dimensional knapsack problem with irregular shapes and side constraints. We developed two ILP-based methods using the well-known dotted-board model: a greedy strategy, where tower configurations are determined one round at a time, and a multi-round strategy, where configurations for all rounds are determined simultaneously.

Our analysis highlights that certain aspects of the resulting optimization problem, such as defining an objective function aligned with winning Bloons TD and incorporating the temporal dimension arising from the game's multiple rounds, are crucial for finding effective tower configurations. While these aspects do not commonly arise in classical cutting and packing problems, we show how they can be successfully modeled for this TD game. The solutions generated by our approaches were able to win the game, in some cases outperforming human strategies. Overall, this work demonstrates the versatility of cutting and packing techniques, illustrating that models and methods initially developed for industrial applications can have practical relevance in very different domains. All our code, including the game simulation tool we developed, is freely available and can be used for future research in the TD genre.

Keywords: Tower defense; Two-dimensional knapsack problem; Nesting problem, Dotted-board model; Integer programming.

1 Introduction

Playing games is often considered a way to relax, although the degree of relaxation one experiences is frequently tied to the outcome of the game. Over time, the ludic dimension of certain games (especially video games) has evolved into a lucrative career path, with dedicated hobbyists becoming full-time gaming professionals (Taylor, 2012). The games that attract the greatest competitive interest are typically so complex that optimal strategies cannot be derived, even with access to high computing power. Driven both by the desire to win and by the rapid growth of the competitive gaming industry, the development of effective game strategies has emerged as a research topic in both Operations Research (Den Hertog and Hulshof, 2006; Álvarez-Miranda et al., 2018) and Reinforcement Learning (OpenAI et al., 2019), with some automated strategies even surpassing those of professional players.

Figure 1: Screenshot of Bloons TD.

One genre of video games that is popular among a broad audience is *Tower Defense* (TD). In a TD game, the player's objective is to defend a specific point on the map—typically a base or the end of a track—from mobile hostile units, commonly called *creeps*, by constructing towers using a limited resource (Avery et al., 2011). If the creeps reach the base, the player moves closer to losing the game (e.g., by losing lives). Creeps generally become stronger, more numerous, and more varied with each round, requiring the player to adopt increasingly complex strategies. Most TD games are won after a predetermined number of rounds, although some, such as the well-known "Bloons TD 6" (Kiwi, 2024), allow for an infinite number of rounds.

Decision-making in a TD game usually involves two components: constructing new towers and modifying existing towers. For the former, the player decides both the type of tower and the location where it will be constructed. For the latter, the player decides whether to upgrade or remove an existing tower. These decisions must adhere to the game rules, the most common ones being that the resources available for constructing or modifying towers are limited, that towers can only be placed at certain positions, and that they cannot overlap.

For the purpose of this work, we investigate a TD game called *Bloons Tower Defense* (Bloons TD), released in 2007 by Ninja Kiwi as a Flash internet game (Wiki, 2025), and the precursor to the Bloons TD series. A screenshot of the game taken from the wiki is shown in Figure 1. The point on the map to defend is located at the end of the track (at the top of the map), the creeps are the balloons advancing along the track, and the towers are monkeys, which can be placed anywhere except on (or very close to) the track. On the right side of the map, the available resources (money), the number of remaining lives (here, 40 out of 40), and the current round number (here, 3 out of 50) are displayed. The game is won if the player completes round 50 and lost if all 40 lives are depleted. Note that all rounds use the same map, and the game contains only this single map. The player starts with 650 money and gains more by damaging creeps, completing rounds, and selling existing towers. Money can be spent on constructing new towers or upgrading existing ones.

Among the important details of the game, we note that there are five tower types: *dart*, *tack*, *ice*, *bomb*, and *super monkey*. Each tower type has its own features, including its cost, its range (i.e., how far its projectiles travel), the effects its projectiles have on creeps (e.g., damage or slowing), its attack speed (i.e., how quickly projectiles are produced), and its footprint (no tower can be placed such that its footprint overlaps with another tower's footprint). Tower upgrades modify these features. Creeps move in a predetermined manner along the track from start to finish. Creeps also have a type, identified by a color in the game: *red*, *blue*, *green*, *yellow*, *white*, and *black*. The type of a creep determines its position in the hierarchy, and when damaged, it transforms into a creep lower in the hierarchy. For example, damaging a blue creep transforms it into a red creep, while damaging a red creep destroys it. The type of a creep

determines its speed and the amount of damage it deals to the player if it reaches the end of the track.

Whereas it is easy to become lost in the (non-exhaustive) game-specific details outlined above, we demonstrate in this work that the optimization problem associated with winning Bloons TD—which we refer to as *WBTDP* hereafter—is a temporal two-dimensional knapsack problem with irregular shapes and side constraints. This perspective allows us to formulate the problem using the dotted-board model (Toledo et al., 2013). Certain aspects, however, warrant further attention—for example, defining an objective function that aligns with winning the game, incorporating the temporal dimension arising from the game's 50 rounds, and accounting for the fact that the resource limit partially depends on the number of lives lost. Conversely, some game-specific characteristics, such as the fact that the tower footprints are circular and homogeneous, can be exploited to simplify the modeling. In the version of the game we study, we consider only two of the five available tower types (dart and super monkey), we do not allow selling towers, and we do not allow building towers during a round. Hence, while the solutions provided by our techniques can always be implemented in the game, there also exist feasible solutions that we do not consider.

The remainder of this paper is structured as follows. In Section 2, we review the relevant literature. Section 3 provides a formal description for WBTDP and presents several ILP-based approaches to solve it. In Section 4, we describe how we derived various model components, including the objective functions, which are designed as proxies for winning the game. Section 5 reports the results of comprehensive computational experiments assessing the proposed approaches. Finally, conclusions are drawn in Section 6.

2 Literature review

Although still somewhat unconventional, developing ILP-based approaches to assist in winning (video) games has been the focus of several research studies in recent years. For example, Den Hertog and Hulshof (2006) proposed an ILP model to determine the best play for a single round of the board game *Rummikub*, Ripamonti et al. (2017) described an ILP model for maximizing profit in the video game *Caesar IV*, Álvarez-Miranda et al. (2018) developed an ILP model to solve a routing problem arising in the mobile game *Pokémon GO*, and Lafond (2018) demonstrated that the problem of *speedrunning*—winning a video game in the shortest possible time—is closely related to well-known combinatorial optimization problems, such as the knapsack problem and the problem of finding a minimum-weight feedback arc set in a graph.

Regarding TD games, one of the earliest academic contributions is due to Avery et al. (2011), who, in addition to describing all common components of the genre, also highlighted certain optimization aspects inherent to these games, such as resource management and unit placement. Rummell (2011) and Tan et al. (2013) proposed evolutionary algorithms for winning TD games (a homemade game and a customized map in *Warcraft III*, respectively). Importantly, both studies used game simulations to evaluate the effectiveness of placing towers at specific locations. Dias et al. (2020) and Bergdahl et al. (2024) proposed deep reinforcement learning algorithms to win TD games (another homemade TD game and the well-known *Plants vs. Zombies* TD game). Whereas Dias et al. (2020) allowed their agent to make all decisions, Bergdahl et al. (2024) restricted their agent to choosing among four high-level strategies that were predefined by the authors. These two studies also relied on game simulations to compute the rewards used when training their agents. Focusing on the modeling aspect, Mazurova et al. (2020) proposed an Integer Linear Programming (ILP) model to determine the best tower placement given an associated value for each tower and each position, and also proposed a strategy to compute these values using game simulations. Although they prevented two towers from being placed at the same position, they did not account for tower footprints as they exist in Bloons TD. Moreover, their model did not incorporate the

constraints associated with the multi-round aspect of the game.

Several features of TD games are related to well-known combinatorial optimization problems. For example, Suttichaya (2017) drew parallels between certain components of TD games and the Hamiltonian path, knapsack, and art gallery problems (see **?** for the latter). In this work, we establish a connection between TD games and a particular type of two-dimensional *Cutting and Packing Problem* (CPP), motivated by the non-overlapping constraints associated with tower footprints. Two-dimensional CPPs have been extensively studied in the literature (see, e.g., the recent survey by Iori et al. 2021). Whereas a large portion of the literature has focused on the so-called "regular" case, in which both items and containers are rectangular, researchers have also investigated irregular packing problems (Leao et al., 2020), also called *nesting problems*, where this assumption no longer holds. The particular case of irregular packing in which all items are circles has also been studied (Hifi and M' Hallah, 2009).

Among the specific characteristics of the WBTDP, we note (i) that not all feasible solutions are equivalently good, (ii) that two feasible tower placements at consecutive rounds are not necessarily compatible within the same game, (iii) that towers cannot be built on (or very close to) the track along which creeps circulate, and (iv) that towers have a cost, which is ultimately bounded by a round-specific budget. Regarding the first point, and following the idea of Mazurova et al. (2020), it makes sense to associate a value with each tower/position pair and to define an objective function based on these values, resulting in a two-dimensional knapsack-type problem (Caprara and Monaci, 2004). Regarding the second point, this has been investigated in the CPP literature under the term "temporal", where the solution quality is measured over time (see, e.g., Clautiaux et al. 2021). A specificity of the WBTDP is that certain quality thresholds must be met at each round, so as not to lose the game. Regarding the third point, this resembles the concept of a defect (Yao et al., 2025) in the CPP literature, which is usually defined as a point that cannot accommodate any item. In the WBTDP, the track cannot accommodate any tower or tower footprint. Finally, the last point refers to a one-dimensional capacity (or budget) constraint, as observed by Mazurova et al. (2020). The WBTDP can therefore be seen as a temporal two-dimensional knapsack problem with irregular (or circular) shape and side-constraints associated with the budget and the track.

Given all the considerations above, a solution approach that seems particularly well-suited for the WBTDP is the dotted-board model, which was initially proposed by Toledo et al. (2013). In the dotted-board model, packing positions are discretized, and decision variables are associated with each item-position pair. Temporal and capacity constraints can be easily incorporated into the resulting ILP formulation, while the discretization of packing positions makes it possible to associate a profit with each item-position pair and to easily eliminate all positions located on (or too close to) the track. Note that the dotted-board model has been extensively investigated by the nesting community (see, e.g., Rodrigues and Toledo 2017; Sato et al. 2019; Guo et al. 2025).

3 Problem definition and integer programming-based approaches

In this section, we first introduce the mathematical notation and an ILP formulation to determine the best tower placement for a single round of the game, and then describe two frameworks to solve the WBTDP as a whole.

3.1 Mathematical notation and single-round ILP model

In the WBTDP, we are given a set of possible positions $\mathcal{P} = \{1, \ldots, P\}$ and a set of tower types $\mathcal{J} = \{1, \ldots, J\}$, each type $j \in \mathcal{J}$ having its own set of feasible construction positions $\mathcal{P}_j = \{1, \ldots, P_j\}$ and

its own set of possible upgrades $\mathcal{U}_j = \{1, \ldots, U_j\}$. The game consists of $\mathcal{L} = \{1, \ldots, 50\}$ rounds, each associated with a specific budget B_l. Each tower type $j \in \mathcal{J}$ with upgrade $u \in \mathcal{U}_j$ has a construction cost c_{ju} and a profit value v_{jup} when constructed at position $p \in \mathcal{P}_j$. Finally, we are given a set $\mathcal{I}$ of pairwise-incompatible tower placements, where (j, p, j', p') belongs to $\mathcal{I}$ if it is impossible to construct both a tower of type j at position p and a tower of type j' at position p' because their footprints would overlap, along with a set $\mathcal{D}$ of pairwise-compatible tower upgrades, where (j, u, u') belongs to $\mathcal{D}$ if it is possible to transition from upgrade $u \in \mathcal{U}_j$ of tower type $j \in \mathcal{J}$ to upgrade $u' \in \mathcal{U}_j$. Note that (j, u, u) always belongs to $\mathcal{D}$ (i.e., a tower can always remain at its current upgrade), and for any two distinct upgrades u and u' of a tower of type j, at most one of (j, u, u') and (j, u', u) belongs to $\mathcal{D}$ (i.e., there is a hierarchy among upgrades, and downgrading or moving between upgrades at the same hierarchical level is not possible).

Using binary decision variables x_{jup} taking the value 1 if a tower of type $j \in \mathcal{J}$ with upgrade $u \in \mathcal{U}_j$ is constructed at position $p \in \mathcal{P}_j$, and the value 0 otherwise, the problem of determining the best tower placement for a single round $l \in \mathcal{L}$ of the game can be defined as follows:

$$\max \quad \sum_{j \in \mathcal{J}} \sum_{u \in \mathcal{U}_j} \sum_{p \in \mathcal{P}_j} v_{jup} \cdot x_{jup} \tag{1}$$

$$\text{s.t.} \quad \sum_{u \in \mathcal{U}_j} x_{jup} + \sum_{u' \in \mathcal{U}_{j'}} x_{j'u'p'} \leq 1 \qquad (j, p, j', p') \in \mathcal{I}, \tag{2}$$

$$\sum_{j \in \mathcal{J}} \sum_{u \in \mathcal{U}_j} \sum_{p \in \mathcal{P}_j} c_{ju} \cdot x_{jup} \leq B_l, \tag{3}$$

$$x_{jup} \in \{0, 1\} \qquad j \in \mathcal{J}, u \in \mathcal{U}_j, p \in \mathcal{P}_j. \tag{4}$$

Objective function (1) maximizes the value of the constructed towers, constraints (2) prevent overlap between tower footprints, and constraint (3) ensures that the total tower cost does not exceed the budget of the round.

It is known that the pairwise incompatibility constraints (2) are both numerous and lead to a weak LP relaxation. Rodrigues and Toledo (2017) showed that these can be replaced by clique constraints, which can be obtained by solving a clique covering problem in a graph where each node (j, p) represents a tower type-position pair and each edge $\big((j, p), (j', p')\big)$ represents an overlap between two pairs (i.e., $(j, p, j', p') \in \mathcal{I}$). The objective is then to cover every edge of the graph with the minimum number of cliques. Once a set of covering cliques $\mathcal{Q}$ has been determined, where each $q \in \mathcal{Q}$ denotes a set of vertices, the following constraints are added:

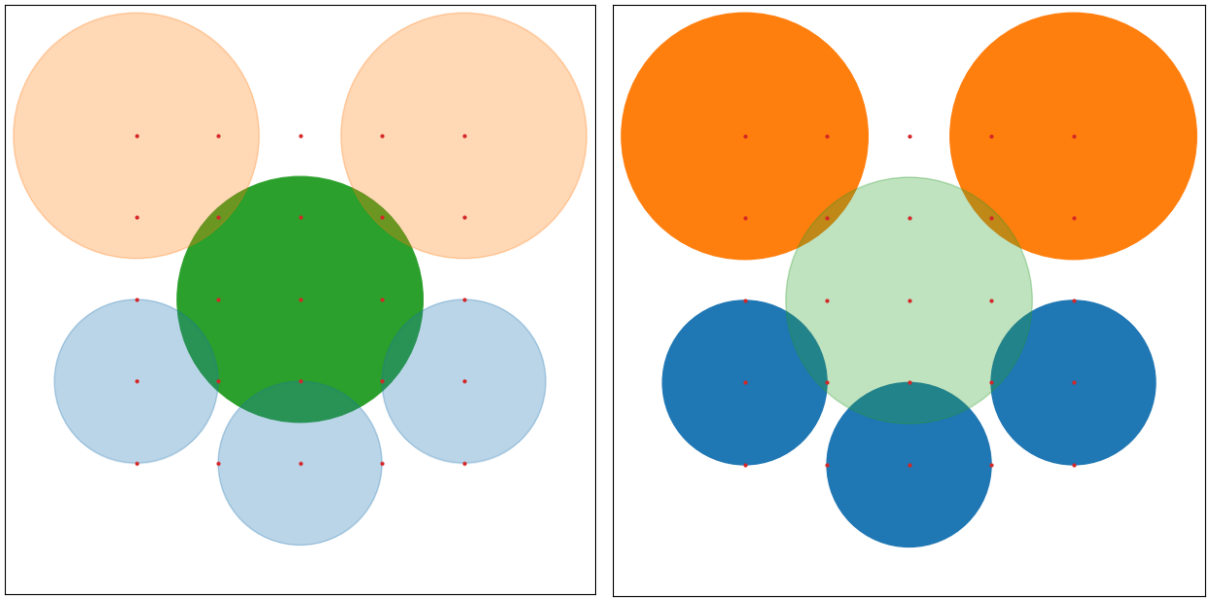
$$\sum_{(j,p) \in q} \sum_{u \in \mathcal{U}_j} x_{jup} \leq 1 \quad q \in \mathcal{Q}. \tag{5}$$

Whereas this approach is valid in the general case where items may have arbitrary shapes (including circles), solving a clique covering problem (which is $\mathcal{NP}$-hard) is challenging in practice, and the resulting number of no-overlap constraints remains large (although much smaller than $|\mathcal{I}|$). Instead, we try to exploit the fact that the game contains only two tower footprints, resulting in a small-large tower partition—small tower types are stored in $\mathcal{J}_1$, and large ones in $\mathcal{J}_2$—which allows us to derive the following no-overlap constraints:

$$\sum_{u \in \mathcal{U}_j} x_{jup} + \frac{1}{M_1} \sum_{\substack{(j,p,j',p') \in \mathcal{I} \\ j' \in \mathcal{J}_1}} \sum_{u' \in \mathcal{U}_{j'}} x_{j'u'p'} + \frac{1}{M_2} \sum_{\substack{(j,p,j'',p'') \in \mathcal{I} \\ j'' \in \mathcal{J}_2}} \sum_{u'' \in \mathcal{U}_{j''}} x_{j''u''p''} \leq 1 \quad j \in \mathcal{J}, p \in \mathcal{P}_j. \tag{6}$$

An illustrative example of these constraints is shown in Figure 2. In practice, constraints (6) state that if a tower of type j is constructed at position p, then no other tower of type j' can be constructed at position p' if $(j, p, j', p') \in \mathcal{I}$ (see the left part of the figure). If this is not the case, then the constraint is non-binding. Indeed, note that the fact that both (j, p, j', p') and (j, p, j'', p'') belong to $\mathcal{I}$ does not imply that (j', p', j'', p'') also belongs to $\mathcal{I}$. In fact, several variables $x_{j'u'p'}$ and $x_{j''u''p''}$ may take the value 1 simultaneously when x_{jup} takes the value 0 (see the right part of the figure). However, M_1 and M_2 need not be defined as arbitrarily large constants: smaller values can be determined by solving a co-clique-type problem, as described in Section 4. In fact, we even show that a separate pair of (M_1, M_2) values can be defined for constraints (6) depending on whether they apply to a tower type j that belongs to $\mathcal{J}_1$ or $\mathcal{J}_2$.

Figure 2: Illustration of the no-overlap constraints (6). Each dot represents a possible tower placement: (a) when a tower of type j is constructed at position p, towers with overlapping footprints cannot be built; (b) when a tower of type j is not constructed at position p, multiple towers with overlapping footprints may be built, provided that they are mutually compatible.



(a) Binding constraint

(b) Non-binding constraint

3.2 Frameworks for the WBTDP

To handle the temporal aspect of the WBTDP, two features must be addressed: the fact that a solution at a given round $l \in \mathcal{L}$ should be compatible with the solution from the previous round, and the fact that one should optimize over the objective values of all 50 rounds, with the additional constraint that each round-specific component must be sufficiently high so as not to lose the game.

3.2.1 A greedy framework

A first approach consists of optimizing each round separately while adding a set of constraints to ensure compatibility between the solutions of two consecutive rounds. If we denote by $\bar{x}^1_{jup}$ the value taken by variable x_{jup} after optimizing model (1)-(4) for round 1, the compatibility constraints for round 2 can be written as:

$$\sum_{u' \in \mathcal{U}_j : (j,u,u') \in \mathcal{D}} x_{ju'p} \geq \bar{x}^1_{jup} \quad j \in \mathcal{J}, u \in \mathcal{U}_j, p \in \mathcal{P}_j, \tag{7}$$

which ensure that if a tower of type j with upgrade u was constructed at position p in round 1, then the same tower (or one of its compatible upgrades) is also present in the solution for round 2. For subsequent rounds $l = 3, \ldots, 50$, $\bar{x}^1_{jup}$ is replaced by $\bar{x}^{l-1}_{jup}$. Note that this greedy framework does not necessarily have to follow the chronological order of the rounds. For example, after computing $\bar{x}^{50}_{jup}$, the value taken by variable x_{jup} after optimizing model (1)-(4) for round 50, the corresponding constraints for round 49 are:

$$x_{jup} \leq \sum_{u' \in \mathcal{U}_j : (j,u,u') \in \mathcal{D}} \bar{x}^{50}_{ju'p} \quad j \in \mathcal{J}, u \in \mathcal{U}_j, p \in \mathcal{P}_j, \tag{8}$$

which allow the construction of a tower of type j with upgrade u at position p in round 49 only if that same tower (or one of its compatible upgrades) is also present in the solution for round 50. For earlier rounds $l = 48, \ldots, 1$, $\bar{x}^{50}_{jup}$ is replaced by $\bar{x}^{l+1}_{jup}$. Finally, we note that this greedy approach can also start from any intermediate round $l \in \mathcal{L} \setminus \{1, 50\}$. In that case, one should solve model (1)-(4) with compatibility constraints of type (7) for rounds $l' = l + 1, \ldots 50$ and of type (8) for rounds $l' = l - 1, \ldots, 1$.

3.2.2 A multi-round framework

A second, more involved approach consists of optimizing multiple rounds $\mathcal{L}' = \{1, \ldots, L'\}$ simultaneously. To do so, we now consider two round-specific parameters: α_l, which determines the relative importance of tower placement at round l compared with the other rounds in $\mathcal{L}'$, and T_l, which sets a minimum threshold for the quality of tower placement at round l. Adding an extra index to the binary decision variables such that x_{jupl} now takes the value 1 if a tower of type $j \in \mathcal{J}$ with upgrade $u \in \mathcal{U}_j$ is located at position $p \in \mathcal{P}_j$ in round $l \in \mathcal{L}'$, and the value 0 otherwise, determining the best tower placement across all L' rounds can be formulated as follows:

$$\max \quad \sum_{j \in \mathcal{J}} \sum_{u \in \mathcal{U}_j} \sum_{p \in \mathcal{P}_j} \sum_{l \in \mathcal{L}'} \alpha_l \cdot v_{jup} \cdot x_{jupl} \tag{9}$$

$$\text{s.t.} \quad \sum_{u \in \mathcal{U}_j} x_{jupl} + \sum_{u' \in \mathcal{U}_{j'}} x_{j'u'p'l} \leq 1 \qquad (j, p, j', p') \in \mathcal{I}, l \in \mathcal{L}', \tag{10}$$

$$\sum_{j \in \mathcal{J}} \sum_{u \in \mathcal{U}_j} \sum_{p \in \mathcal{P}_j} c_{ju} \cdot x_{jupl} \leq B_l \qquad l \in \mathcal{L}', \tag{11}$$

$$\sum_{j \in \mathcal{J}} \sum_{u \in \mathcal{U}_j} \sum_{p \in \mathcal{P}_j} v_{jup} \cdot x_{jupl} \geq T_l \qquad l \in \mathcal{L}', \tag{12}$$

$$x_{jupl} \leq \sum_{u' \in \mathcal{U}_j : (j,u,u') \in \mathcal{D}} \bar{x}^{l+1}_{ju'p} \qquad j \in \mathcal{J}, u \in \mathcal{U}_j, p \in \mathcal{P}_j, l \in \mathcal{L}' \setminus \{L'\}, \tag{13}$$

$$x_{jupl} \in \{0, 1\} \qquad j \in \mathcal{J}, u \in \mathcal{U}_j, p \in \mathcal{P}_j, l \in \mathcal{L}'. \tag{14}$$

Objective function (9) now maximizes a weighted value of the constructed towers across all considered rounds (although, from a game perspective, a tower is only constructed once). No-overlap constraints (10), budget constraints (11), and compatibility constraints (13) are adaptations of (2), (3), and (8), respectively, in which the round index has been added. Note that no-overlap constraints (10) can also be expressed with big-M coefficients, similarly to (6). Finally, constraints (12) ensure that the minimum round-specific quality thresholds are met. We describe in Section 5 how α_l, T_l, and $\mathcal{L}'$ were determined,

4 Deriving the model components from the game

In this section, we describe how various components that are necessary in the models described in the previous section were built so as to accurately replicate Bloon TD.

4.1 Game simulation

Although the ILP-based approaches described in the previous section do not require a game simulation to run, as was the case for the reinforcement learning approaches of Rummell (2011) and Tan et al. (2013), many sets and parameters must still be properly defined beforehand to ensure that the resulting solutions correspond to successful tower configuration plans that a player can implement in the actual game. To achieve this, we used a Bloons TD simulation initially proposed by de Oliveira (2023), which we modified to mimic the game behavior as closely as possible, while also returning the performance metrics and indicators necessary for our work. For example, we adjusted the creep speed and types, the projectile speed and disappearance conditions, as well as the tower targeting strategy, so that they exactly match those in the game (based on the information available in Wiki 2025; Blooncyclopedia 2025). For unavailable data, such as tower footprint size, map dimensions and shape, creep and projectile hit boxes, and creep spawn rate, adjustments were made based on additional estimates that were obtained by measuring elements in the original game using the image editor GIMP. We also added a run mode that produces no graphical output, allowing simulations to be executed as quickly as desired. The source code of our game simulator is publicly available at https://github.com/jjmal/BTD-Simulation. While we did our best to replicate the game as closely as possible, some discrepancies may still exist.

4.2 Determining $\mathcal{P}_j$ and $\mathcal{I}$

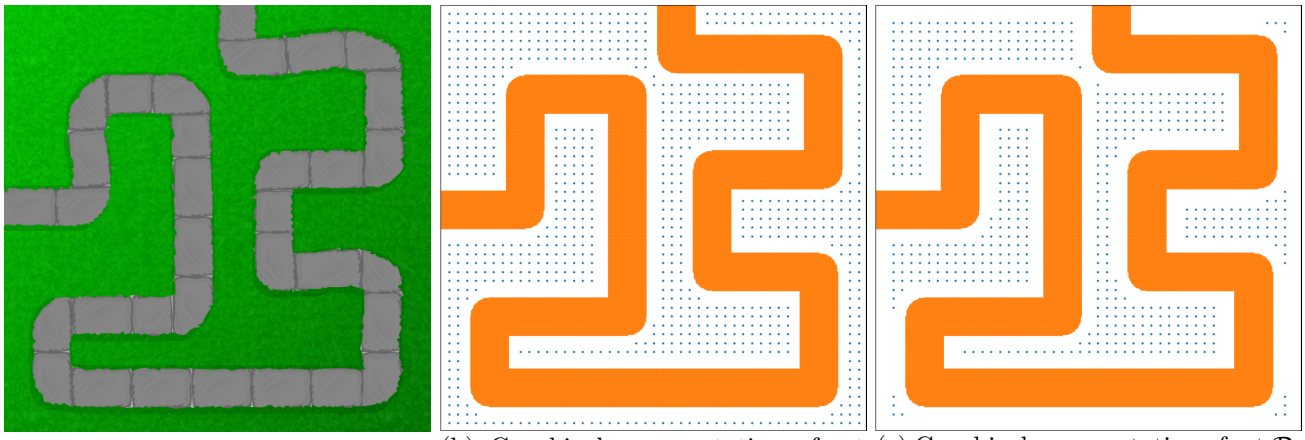
As the original game map (see the left part of Figure 3) is a square with a side length of 480 pixels, this corresponds to $P = 480 \times 480 = 230\,400$ potential positions where towers could be constructed (before accounting for map-related incompatibilities). Since the number of variables and constraints in our ILP-based approaches increases with the number of positions, and because it would be unrealistic to expect a human player to place towers with pixel-level precision, we discretized the grid by retaining only one position every 10 pixels, thereby reducing the total to 2304 candidate positions. After excluding locations whose tower footprints either overlap with the track or exceed the edge of the map, we obtained $P_j = 910$ feasible positions for small tower types $j \in \mathcal{J}_1$ (see the middle part of Figure 3) and $P_{j'} = 595$ positions for large tower types $j' \in \mathcal{J}_2$ (see the right part of Figure 3).

Regarding the computation of set $\mathcal{I}$, specialized geometric approaches from the two-dimensional CPP literature (Bennell and Oliveira, 2009), such as the no-fit polygon, can be used when items are irregularly shaped polygons. In the WBTDP, however, such techniques are unnecessary: since all tower footprints are circular, the footprint of a tower of type j (with footprint radius r) constructed at position p overlaps with that of a tower of type j' (with footprint radius r') constructed at position p' if the distance between p and p' is less than $r + r'$. Hence, for each tower type $j \in \mathcal{J}$ (with footprint radius r) and each position $p \in \mathcal{P}_j$, it suffices to consider two no-fit circles: one with radius $r + r_1$ within which no other small tower can be constructed, the other with radius $r + r_2$ within which no other large tower can be constructed, where r_1 and r_2 are the small and large tower footprint radii, respectively.

4.3 Determining M_1 and M_2

As described in Section 3, to determine the minimum values of M_1 and M_2 used in constraints (6), we must identify how many towers fit within a given radius. Let us first consider a small tower of type

Figure 3: Visualization of sets $\mathcal{P}_j$ on the game map.



(a) Original map of the game

(b) Graphical representation of set $\mathcal{P}_j$ for small tower types $\mathcal{J}_1$

(c) Graphical representation of set $\mathcal{P}_j$ for large tower types $\mathcal{J}_2$

$j \in \mathcal{J}_1$ with a footprint radius r_1 that is constructed at position $p \in \mathcal{P}_j$. Such a tower overlaps with every small tower built at a position p' such that the distance between p and p' is smaller than $2r_1$. Let us gather all these positions in the set $\mathcal{P}^1_{jp}$. Similarly, this tower also overlaps with every large tower built at a position p'' such that the distance between p and p'' is smaller than $r_1 + r_2$. Let us gather all these positions in the set $\mathcal{P}^2_{jp}$. In addition, let us gather in the set $\mathcal{I}^{11}_{jp}$ all pairwise-incompatible tower placements (p'_1, p'_2) $(p'_1, p'_2 \in \mathcal{P}^1_{jp})$ such that it is impossible to construct both a small tower at position p'_1 and a small tower at position p'_2 because their footprints would overlap. Sets $\mathcal{I}^{12}_{jp}$ and $\mathcal{I}^{22}_{jp}$ are built similarly. After introducing binary variables $y^1_{p'}$ taking the value 1 if a small tower is constructed at position $p' \in \mathcal{P}^1_{jp}$ and 0 otherwise, and $y^2_{p''}$ taking the value 1 if a large tower is constructed at position $p'' \in \mathcal{P}^2_{jp}$ and 0 otherwise, the problem of determining the maximum number of large towers that can be constructed at positions in $\mathcal{P}^2_{jp}$, given that exactly k small towers are built at positions in $\mathcal{P}^1_{jp}$ can be modeled as follows:

$$\max \quad \sum_{p'' \in \mathcal{P}^2_{jp}} y^2_{p''} \tag{15}$$

$$\text{s.t.} \quad \sum_{p' \in \mathcal{P}^1_{jp}} y^1_{p'} = k, \tag{16}$$

$$y^1_{p'_1} + y^1_{p'_2} \leq 1 \qquad p'_1 \in \mathcal{P}^1_{jp}, p'_2 \in \mathcal{P}^1_{jp} : (p'_1, p'_2) \in \mathcal{I}^{11}_{jp}, \tag{17}$$

$$y^1_{p'} + y^2_{p''} \leq 1 \qquad p' \in \mathcal{P}^1_{jp}, p'' \in \mathcal{P}^2_{jp} : (p', p'') \in \mathcal{I}^{12}_{jp}, \tag{18}$$

$$y^2_{p''_1} + y^2_{p''_2} \leq 1 \qquad p''_1 \in \mathcal{P}^2_{jp}, p''_2 \in \mathcal{P}^2_{jp} : (p''_1, p''_2) \in \mathcal{I}^{22}_{jp}, \tag{19}$$

$$y^1_{p'} \in \{0, 1\} \qquad p' \in \mathcal{P}^1_{jp}, \tag{20}$$

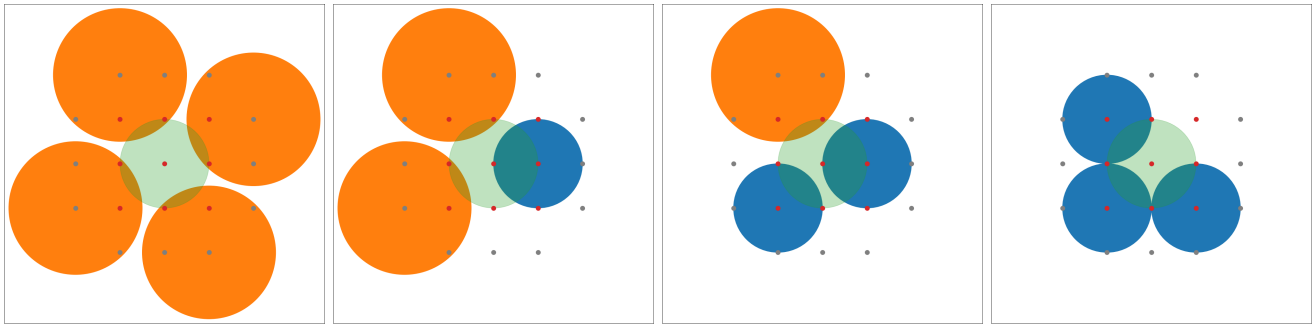
$$y^2_{p''} \in \{0, 1\} \qquad p'' \in \mathcal{P}^2_{jp}. \tag{21}$$

After solving model (15)-(21) for a small tower of type $j \in \mathcal{J}_1$ and a "central" position $p \in \mathcal{P}_j$ (i.e., a position for which $\mathcal{P}^1_{jp}$ and $\mathcal{P}^2_{jp}$ are as large as possible—or in other words, a position far enough from both the track and the map edges so that map-related incompatibilities have no effect), we obtain the following results:

- 4 large towers for $k = 0$ small towers (see the left-most part of Figure 4);

- 2 large towers for $k = 1$ small tower (see the center-left part of Figure 4);
- 1 large tower for $k = 2$ small towers (see the center-right part of Figure 4);
- 0 large towers for $k = 3$ or 4 small towers (see the right-most part of Figure 4).

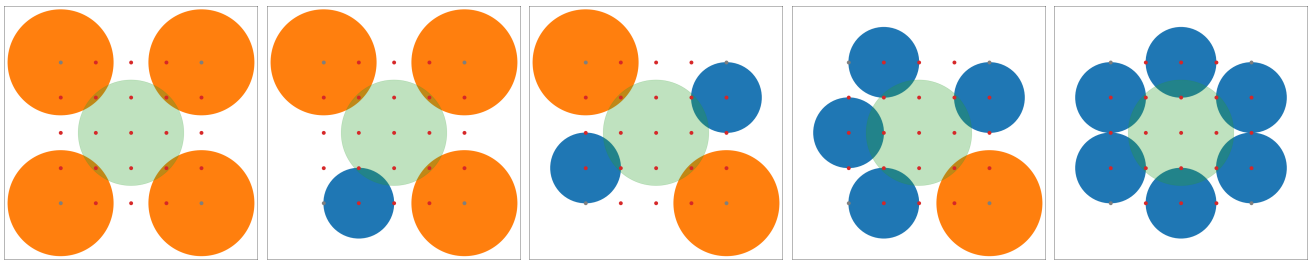
Figure 4: Visualization of the solutions obtained from model (15)-(21) for a small tower of type $j \in \mathcal{J}_1$. Gray dots indicate positions that are valid only for large towers, whereas red dots indicate positions that are valid for both small and large towers.



(a) $k = 0$ small towers. (b) $k = 1$ small towers. (c) $k = 2$ small towers. (d) $k = 3$ or 4 small towers.

Hence, $M_1 = M_2 = 4$ are valid values for the no-overlap constraint (6) associated with the considered j and p. One could repeat the process for every other position in $\mathcal{P}_j$, potentially leading to slightly lower M_1 and M_2 values for positions that are near the track or the map edges. However, as this would result in numerous additional computations, and since the values $M_1 = M_2 = 4$ seem sufficiently small while also being valid for all positions in $\mathcal{P}_j$, we did not do so. Using a similar process for a large tower of type $j \in \mathcal{J}_2$ and a central position $p \in \mathcal{P}_j$, we obtained $M_1 = 6$ and $M_2 = 4$ (see Figure 5).

Figure 5: Visualization of the solutions obtained from model (15)-(21) for a large tower of type $j \in \mathcal{J}_2$. Gray dots indicate positions that are valid only for large towers, whereas red dots indicate positions that are valid for both small and large towers.



(a) $k = 0$ small towers. (b) $k = 1$ small towers. (c) $k = 2$ or 3 small towers. (d) $k = 4$ small towers. (e) $k = 5$ or 6 small towers.

4.4 Determining v_{jup}

This is undoubtedly the most important yet challenging component of the WBTDP: how can we estimate v_{jup}, the impact of constructing a tower of type j with upgrade u at position p? We developed five strategies to compute these values—the first three based on geometrical concepts, and the last two relying on our game simulator.

Our first strategy, **S1**, focuses solely on the number of points covered by the range of a tower. For a given tower type $j \in \mathcal{J}$ and upgrade $u \in \mathcal{U}_j$, we compute the number of pixels along the centerline of the track that fall within the tower's range if it is constructed at position $p \in \mathcal{P}_j$. For the dart tower

without any upgrades, we directly set v_{jup} to this value. For the (much more powerful) super monkey tower without any upgrades, we multiply this value by 14.5 to obtain v_{jup}, since its attack speed is 14.5 times higher than that of a dart tower. For upgrades that increase the so-called *piercing effect* of the projectiles—essentially allowing them to damage two creeps before vanishing instead of one—we multiply this value by 1.75 to reasonably account for the upgrade's effect, since most but not all projectiles will eventually damage two creeps. Finally, for upgrades that increase the range of the tower, we do not apply any multiplier, as more positions naturally fall within the tower's upgraded range.

In our second strategy, **S2**, we also compute the number of pixels along the centerline of the track that fall within a tower's range, but we now apply a coefficient to each pixel that depends on its distance from the tower. Intuitively, a projectile launched from a distant tower is less likely to hit a creep than one fired from a nearby tower: while the projectile is in flight, the creep may move out of range or be destroyed by another projectile. Given an Euclidean distance d between a position $p \in \mathcal{P}_j$ and a track pixel, we compute the corresponding coefficient based on the inverse of the distance as

$$\frac{\frac{1}{d} - \frac{1}{240}}{\frac{1}{30} - \frac{1}{240}} \cdot (1 - \alpha) + \alpha$$

where 30 and 240 are the minimum and maximum possible values for d, respectively, and α is a parameter that ensures the coefficient lies within $[\alpha, 1]$. After computing the weighted sum, the 1.75 and 14.5 multipliers corresponding to tower types and upgrades are applied.

In our third strategy, **S3**, we again compute the number of pixels along the centerline of the track that fall within a tower's range and apply a pixel-dependent coefficient. However, in this case, the coefficient is determined by *the shooting angle*— the angle formed between the direction of a projectile launched by the tower and the segment of the track to which the pixel belongs (see the left part of Figure 6) Intuitively, a projectile fired from a tower positioned perpendicular to the track (middle part of Figure 6) is less likely to hit a creep than one fired from a tower parallel to the track (right part of Figure 6): in the former case, the projectile has only a short window of opportunity to reach the creep, whereas in the latter, it remains on the creep's path for a longer duration. Given a shooting angle a between a position $p \in \mathcal{P}_j$ and a track pixel, we compute the corresponding coefficient as

$$|\cos(a)| \cdot (1 - \alpha) + \alpha$$

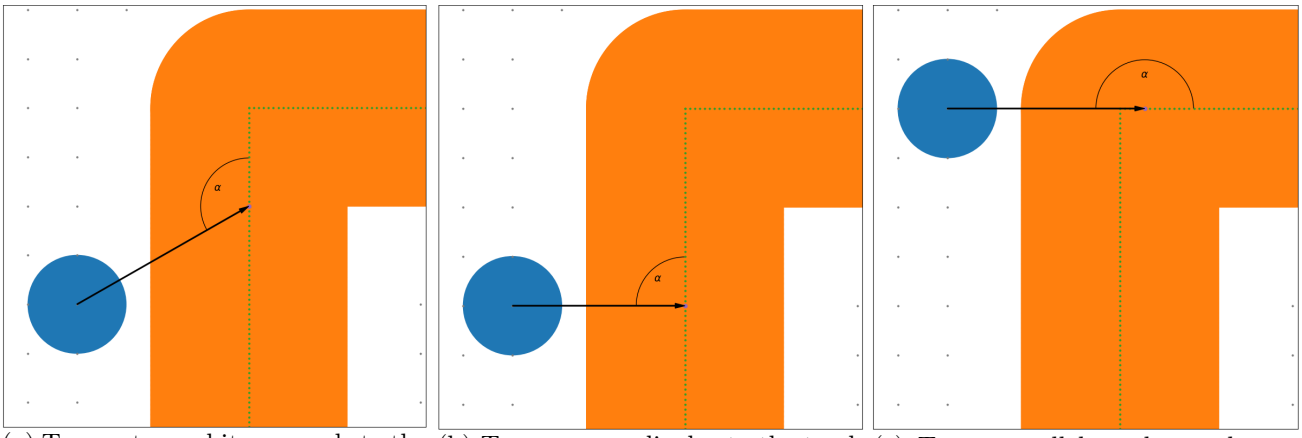
where $\cos(a)$ is the cosine function, which maps an angle a to a value in the range $[-1, 1]$, and α is a parameter that ensures the coefficient lies within $[\alpha, 1]$. After computing the weighted sum, the 1.75 and 14.5 multipliers corresponding to tower types and upgrades are applied.

Finally, our last two strategies, **S4** and **S5**, determine the value of constructing a given tower of type $j \in \mathcal{J}$ and upgrade $u \in \mathcal{U}_j$ at position $p \in \mathcal{P}_j$ by running a single round of the game in our simulation, where only that tower is placed at the given position, and counting the number of times creeps are damaged. The fourth strategy computes this in a simulated round 25, whereas the fifth strategy does so in a simulated round 50.

4.5 Determining B_l

The round-specific budget B_l available at a given round $l \in \mathcal{L}$ is defined as a pre-determined value $\bar{B}_l$ minus the number of lives lost in previous rounds. While $\bar{B}_l$ is known (e.g., in the Wiki 2025), the latter depends on the effectiveness of tower placements in prior rounds. Adjustments to B_l are straightforward in the greedy framework when rounds are considered chronologically. However, this is not the case when

Figure 6: Visualization of the shooting angle α for three tower-pixel pairs.



(a) Tower at an arbitrary angle to the track segment.

(b) Tower perpendicular to the track segment.

(c) Tower parallel to the track segment.

starting from an intermediate round or when using the multi-round framework.

As a result, we considered three scenarios: optimistic (**SC1**), pessimistic (**SC2**), and realistic (**SC3**). In the optimistic scenario, we assume that no lives are lost, so $B_l = \bar{B}_l$ for all rounds $l \in \mathcal{L}$. In the pessimistic scenario, we assume that all lives but one are lost at round 2, with $B_1 = \bar{B}_1$ and $B_l = \bar{B}_l - 39$ for all rounds $l \in \mathcal{L} \setminus \{1\}$. In the realistic scenario, we assume that lives are gradually lost until round 14 (the hardest in our experiments), with $B_1 = \bar{B}_1$, $B_l = \bar{B}_l - 3 \cdot (l-1)$ for $l = 2, \ldots, 14$, and $B_{l'} = \bar{B}_{l'} - 39$ for $l' = 15, \ldots, 50$.

5 Computational experiments

In this section, we first empirically evaluate the effectiveness of the five v_{jup} values introduced in Section 4, then assess the performance of the solution methods presented in Section 3, and finally conduct experiments with other game instances. Our optimization algorithms (available for download at https://github.com/mdelorme2/Modeling-Bloons-Tower-Defense-as-a-temporal-two-dimensional-knapsack-problem) were implemented in `C++`, while our simulations were implemented in Python. All computational tests for which computation time was reported were executed on a virtual machine `AMD EPYC-Rome Processor` with 2.00 GHz and 64 GB of RAM memory, running under `Ubuntu 20`. The ILP models were solved using `Gurobi 11.0.2` and, unless stated otherwise, all used pairwise no-overlap constraints. A single core was used for all tests, and the barrier algorithm was used to solve the root nodes of the models.

One performance measure used to compare our WBTDP solutions is how well they perform relative to human strategies. Two recommended human strategies were retrieved from the Wiki (2025). The first strategy (**HUM1**) involves placing only Dart Towers without upgrades, while the second (more complex) strategy (**HUM2**) places towers of various types and upgrades according to specific instructions. Since these strategies were not defined with a pixel-level precision, we reimplemented them as accurately as possible. For the sake of fairness, we therefore tested two modes for our ILP-based approaches: one in which only Dart Towers without upgrades are allowed (**MD1**), and one without any restrictions (**MD2**).

5.1 Evaluating the v_{jup}

To evaluate the effectiveness of the five strategies introduced in Section 4 for estimating v_{jup}, we measured the correlation coefficient between the value of a tower configuration in a given round (computed as the

sum of v_{jup} values associated with the towers in that configuration) and the number of lives remaining after running the simulation for that round using the same tower configuration.

To do so, we first computed the best tower configuration for a given round l using model (1)-(4), disregarding the temporal aspect. Once an optimal configuration for the round was found, say with value $\bar{z}_l$, we solved the model again 19 times, once for each $k = 2, \ldots, 20$, each time adding the constraint

$$\sum_{j \in \mathcal{J}} \sum_{u \in \mathcal{U}_j} \sum_{p \in \mathcal{P}_j} v_{jup} \cdot x_{jup} \leq \frac{k}{20} \cdot \bar{z}_l, \tag{22}$$

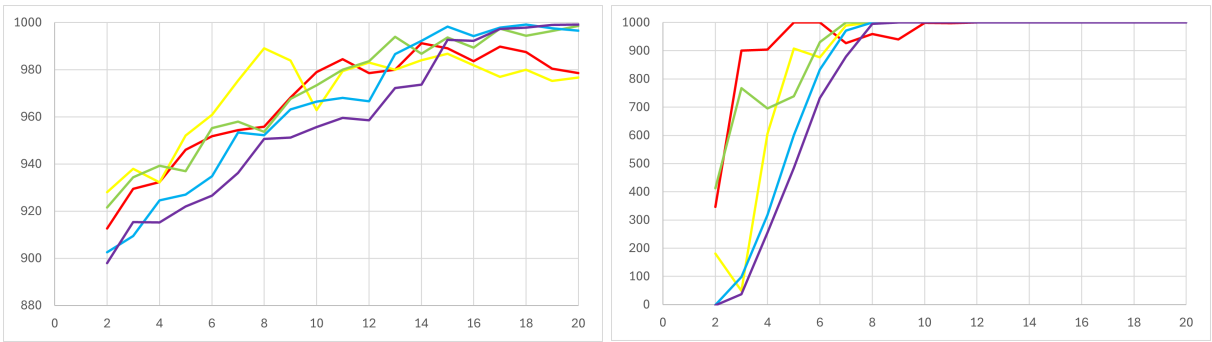
to enforce some diversity among the obtained solutions. For each run, we used a 60-second time limit for the solver and requested the best five solutions using the parameters `PoolSolutions = 5` and `PoolSearchMode = 2`. For a given round and a given strategy, we therefore obtained $19 \times 5 = 95$ data points. Note that our simulator allows running a single round of the game and setting an arbitrary number of lives. In total, we tested the five strategies, two values of α (0 and 0.5) for the second and third strategies, and three rounds (14, 25, and 50) for each mode. The number of lives was artificially increased to 1000 in the simulation because the original limit of 40 did not allow for a clear distinction between poor and very poor tower configurations. The correlation coefficient are reported in Table 1.

Table 1: Correlation coefficient between the tower configuration values and the number of lives remaining

Round	Mode **MD1**							Mode **MD2**						
	S1	**S2**		**S3**		**S4**	**S5**	**S1**	**S2**		**S3**		**S4**	**S5**
		$\alpha = 0$	$\alpha = 0.5$	$\alpha = 0$	$\alpha = 0.5$				$\alpha = 0$	$\alpha = 0.5$	$\alpha = 0$	$\alpha = 0.5$		
14	0.86	0.68	0.75	0.93	0.8	0.95	0.97	0.79	0.78	0.8	0.86	0.84	0.96	0.96
25	0.75	0.66	0.72	0.68	0.76	0.82	0.88	0.23	0.7	0.51	0.44	0.34	0.82	0.86
50	0.69	0.6	0.65	0.65	0.66	0.76	0.79	0.42	0.64	0.5	0.5	0.46	0.73	0.77

Whereas the correlation is positive in all tested cases for each of the five strategies, it is clear that **S5** is the best proxy, with **S4** being a close second. There is no clear dominance among the other strategies, nor any indication of the best α value. To provide better intuition for the interpretation of these correlation values, we display in Figure 7 the number of lives remaining as a function of k for each strategy (with $\alpha = 0$ for **S2** and **S3**) in two rounds and modes: one with a very high correlation (round 14, mode **MD1**, shown on the left) and one with a very low correlation (round 50, mode **MD2**, shown on the right).

Figure 7: Number of lives remaining (y-axis) versus k (x-axis). **S1** is shown in red, **S2** in yellow, **S3** in green, **S4** in blue, and **S5** in purple.



(a) Round 14 in mode **MD1**

(b) Round 50 in mode **MD2**

Regarding round 14 in mode **MD1**, it is clear that strategies **S4** and **S5** provide the best tower configurations. These are the only two strategies resulting in less than a 1-life loss on average when

$k = 20$, followed closely by **S3**, which loses 1.4 lives on average. As k decreases to 0, we observe a consistent decrease in the number of lives remaining for **S4** and **S5**, but not for the other strategies. In fact, **S1** peaks at $k = 14$ and **S2** peaks at $k = 8$. The story is different for round 50 in mode **MD2**. Here, all strategies result in zero lives lost on average when $k = 20$, and this remains true as long as k stays above 12. For lower k values, the performance of strategies **S4** and **S5** decreases dramatically, whereas it remains relatively high for the other strategies (although often below $1000 - 40 = 960$, thus resulting in a game loss). In this case, the higher correlation coefficient is driven by the number of remaining lives at low k values, which is significantly lower for **S4** and **S5**. We conclude from these experiments that strategy **S5** is the most effective for computing the v_{jup} values since higher such values are correlated with a greater number of lives remaining (and hence, with winning the game).

5.2 Evaluating the greedy framework

After running the greedy framework described in Section 3 for the five strategies (with both α values), in the two modes, starting from round $l = 1$, and under the three scenarios, we ran a complete game simulation for each of the resulting tower configuration plans and recorded: (i) whether the plan was feasible (i.e., the resource loss resulting from the number of lives lost at previous rounds did not make any of the planned tower configurations infeasible, column "fea."), (ii) if the game was feasible, the round at which the game terminated (column 'l.r.', value "W" indicates that the game was won), and (iii) if the game was feasible, the number of lives remaining at the end of round 50 (column "liv.", value "0" indicates that the game was lost). These metrics are reported in Table 2. For the sake of comparison, we note that human strategy **HUM1** won the game with 8 lives remaining at the end of round 50, whereas **HUM2** lost the game at round 20.

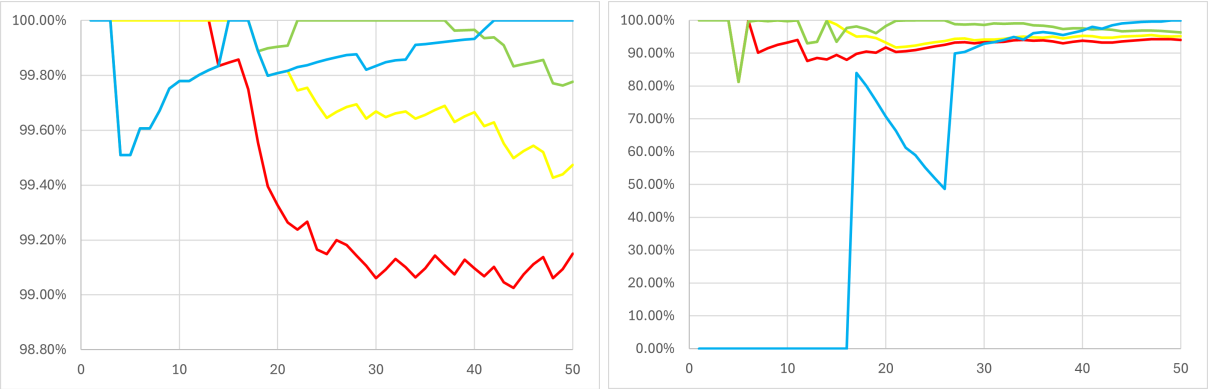
Table 2: Results of the game simulation for each tested tower configuration plan

MD	SC	S1			S2						S3						S4			S5		
					$\alpha = 0$			$\alpha = 0.5$			$\alpha = 0$			$\alpha = 0.5$								
		fea.	l.r.	liv.	fea.	l.r.	liv.	fea.	l.r.	liv.	fea.	l.r.	liv.	fea.	l.r.	liv.	fea.	l.r.	liv.	fea.	l.r.	liv.
MD1	SC1	✓	7	0	✓	7	0	✓	7	0	✗	-	-	✓	7	0	✗	-	-	✓	W	33
	SC2	✓	7	0	✓	7	0	✓	7	0	✓	9	0	✓	7	0	✓	W	18	✓	W	24
	SC3	✓	7	0	✓	7	0	✓	7	0	✓	11	0	✓	7	0	✓	W	27	✓	W	33
MD2	SC1	✓	7	0	✗	-	-	✗	-	-	✗	-	-	✓	7	0	✗	-	-	✗	-	-
	SC2	✓	7	0	✓	6	0	✓	7	0	✓	12	0	✓	7	0	✓	W	9	✓	W	13
	SC3	✓	7	0	✗	-	-	✓	8	0	✓	12	0	✓	7	0	✓	W	28	✓	W	33

These results lead to the same conclusions as those drawn in the previous section: **S4** and **S5** are clearly the best proxies, with the latter performing slightly better, while the other strategies are not competitive. It is also worth noting that the optimistic scenario **SC1** often results in unfeasible tower configuration plans, particularly in mode **MD2**. This occurs far less frequently in the realistic scenario **SC3**, and never (by design) in the pessimistic scenario **SC2**. Although tower configuration plans generated under **SC2** did not lead to better outcomes than those produced under **SC1** or **SC3** in this set of experiments, we observed in other runs that this was not always the case.

We are now interested in the loss associated with using a greedy approach. To this end, we compare in Figure 8 the objective value reached by the greedy framework at each round in the two modes (**MD1** on the left and **MD2** on the right) for strategy **S5** with an upper bound obtained by running model (1)-(4) without the constraints related to the temporal aspect of the problem. We tested four starting rounds for the greedy framework—1 (in red), 14 (in yellow), 25 (in green), and 50 (in blue)—in the optimistic scenario **SC1** and computed the ratio between the associated configuration values and the upper bound.

Figure 8: Greedy framework performance for starting rounds 1 (red), 14 (yellow), 25 (green), and 50 (blue), relative to the upper bound.



(a) Ratios in mode **MD1**

(b) Ratios in mode **MD2**

Regarding mode **MD1**, it is clear that using a greedy approach does not lead to any significant loss, as the resulting tower configurations always have values at most 1% below the theoretical maximum for all rounds and all tested starting rounds. Given the importance of successfully completing the early rounds with as many lives remaining as possible to win the game, using starting round 14 or 25 appears to be the best option. Running a complete game simulation with the tower configuration plans resulting from these starting rounds under the pessimistic scenario **SC2** resulted in winning the game with 27 and 25 lives remaining, respectively (compared to 24 lives for starting round 1). Regarding mode **MD2**, the results differ significantly, with gaps reaching 20% in at least one round for each of the four tested starting rounds. The greedy framework starting at round 50 reaches 0% at round 1 because it is bound to construct an expensive tower at round 17 and must therefore save resources beforehand. Obviously, the resulting tower configuration plan cannot win the game, as all 40 lives would be lost in the early rounds. Among the other three starting rounds, it seems that, once again, those using starting rounds 14 and 25 obtain the best performance. Running a complete game simulation with the tower configuration plans resulting from these starting rounds under the pessimistic scenario **SC2** resulted in winning the game with 36 lives remaining in both cases (compared to 13 lives for starting round 1).

These experiments also indicate that strategy **S5** is the most effective for computing v_{jup}, as it is the one that, when implemented in the game simulation, wins the game with the highest number of remaining lives. We further showed that considering the optimistic or realistic scenarios could result in infeasible tower configuration plans, whereas the pessimistic scenario, although often outperformed, does not always produce worse plans when running the game simulation. It is also interesting to note that our best-performing tower configuration plan outperforms both human strategies, although we emphasize that these strategies were not re-implemented by game experts.

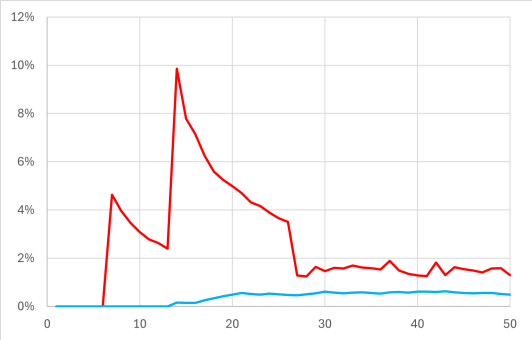
5.3 Evaluating the multi-round framework

While we observed that the greedy framework reached solutions whose values were very close to a valid upper bound for **MD1** and relatively close for **MD2**, we now investigate whether better solutions can be achieved with the multi-round framework.

In our first set of experiments, we solve model (9)-(14) for each round $l \in \mathcal{L}$ as follows: (i) we set $\mathcal{L}' = \{1, \ldots, l\}$, (ii) we set $\alpha_{l'} = 0$ for all $l' \in \mathcal{L}' \setminus \{l\}$ and $\alpha_l = 1$, and (iii) we set $T_{l'} = \bar{z}_{l'}$ for all $l' \in \mathcal{L}' \setminus \{l\}$ and $T_l = 0$, where $\bar{z}_{l'}$ is the optimal solution value of the model when solved for round l'. Informally, this can be viewed as the counterpart of the greedy framework starting from round 1 in

which only the solution values obtained in prior rounds are fixed, not the solutions themselves (i.e., the tower configurations). For example, if an optimal solution for round 1 constructs dart towers with no upgrades at positions 374 and 445 for a total value of 10 200, then the greedy framework ensures that dart towers (possibly upgraded) are also placed at positions 374 and 445 in round 2. In contrast, the multi-round framework re-computes tower configurations for both rounds 1 and 2, maximizing the total value of the configuration in round 2 while ensuring that the total value of the configuration in round 1 is at its maximum, 10 200, and that the two configurations are compatible. A 3600s time limit was imposed for every round, and for rounds $l \in \mathcal{L} \setminus \{1\}$, the solution from the previous round $l-1$ was used as a warm start to the solver to reduce computation time. We report in Figure 9 the relative improvement achieved by the above-described multi-round framework over the greedy framework, computed as $\frac{z_{\text{multi-round}} - z_{\text{greedy}}}{z_{\text{greedy}}}$ for each round in the two modes (**MD1** in blue and **MD2** in red) for strategy **S5** under the pessimistic scenario **SC2**.

Figure 9: Relative improvement resulting from the use of the multi-round framework over the greedy framework for modes **MD1** (blue) and **MD2** (red).



Interestingly, we observe that better solutions can be obtained using the multi-round framework, although the improvement is relatively minor for mode **MD1**. The resulting tower configuration plan achieved a win in the game simulation, with 31 lives remaining at the end of round 50 (compared to 24 for the greedy framework). For mode **MD2**, the improvements are more substantial, particularly for intermediate rounds 7-26. The resulting tower configuration plan also secured a win in the game simulation, with 25 lives remaining at the end of round 50 (compared to 13 for the greedy framework).

We highlight that, whereas solving model (9)-(14) for all 50 rounds took less than 150s in total for mode **MD1**, solving the model for mode **MD2** was more computationally challenging, which is why we imposed a 3600s time limit for each round. We note that, beyond round 19, no solutions could be certified as optimal; however, the observed optimality gap was reasonably low (usually around 5% and always below 8%). The total computation time for mode **MD2** was 32 hours. We also tried to solve the model using the big-M no-overlap constraints, but these turned out to increase the computation time. For mode **MD1**, the total time was 590.7s (the resulting tower configuration plan, while achieving the same objective value, left 15 lives remaining at the end of round 50). For mode **MD2**, the total time was 32.4 hours (the resulting tower configuration plan, while achieving a slightly lower objective value, left 30 lives remaining at the end of round 50).

Finally, we solved an adaptation of model (9)-(14) in which we maximized the lowest effectiveness ratio across all rounds, where the effectiveness ratio for a given round $l \in \mathcal{L}$ is computed as the total value of the tower configuration at round l divided by the upper bound obtained by solving model (1)-(4) without the temporal constraints. We report in Table 3 several model-related metrics—time (column "t(s)"), objective value ("obj."), number of variables, constraints, and non-zero elements ("# var.", "# cons.", and "# nz.", respectively)—along with the outcome of the game simulation for the resulting tower

configuration plan in the last three columns, for each of the two modes, three scenarios, and two versions of the no-overlap constraints. A 360 000s time limit was imposed for each of the resulting twelve runs.

Table 3: Model-related metrics and outcomes of the resulting tower configuration plans

No-overlap constraints	MD	SC	Model-related metrics					Simulation results		
			t(s)	obj.	# var.	# cons.	# nz.	fea.	l.r.	liv.
pairwise	**MD1**	**SC1**	19.1	0.9967	91001	230790	551880	✗	-	-
		SC2	17.4	0.9967	91001	230790	551880	✓	W	30
		SC3	18.2	0.9967	91001	230790	551880	✓	W	29
	MD2	**SC1**	360 000	0.9755	316751	1171665	3080510	✓	W	40
		SC2	73 101	0.9702	316751	1171665	3080510	✓	W	36
		SC3	133 104	0.9702	316751	1171665	3080510	✓	W	40
big-M	**MD1**	**SC1**	142.2	0.9967	91001	135690	597380	✗	-	-
		SC2	150.3	0.9967	91001	135690	597380	✓	W	15
		SC3	135.4	0.9967	91001	135690	597380	✓	W	31
	MD2	**SC1**	360 000	0.9771	316751	358115	3155760	✗	-	-
		SC2	114 515	0.9702	316751	358115	3155760	✓	W	40
		SC3	164 202	0.9702	316751	358115	3155760	✓	W	36

Interestingly, we observe that the minimum deviation across all rounds is extremely low for mode **MD1** (around 0.33%), while it is substantially higher for mode **MD2** (between 2 and 3%). Although the decrease in the number of variables is non-negligible, using big-M no-overlap constraints, here again, reduces model performance compared to their pairwise counterparts (except for mode **MD2** under scenario **SC1**, where neither strategies could find an optimal solution within the time limit, but the solution obtained with big-M constraints had a higher value than that obtained with pairwise constraints, even though it led to worse results in the simulation phase). Further investigation showed that the solver's preprocessing is simply more effective (i.e., removes more constraints) when pairwise no-overlap constraints are used, possibly converting them into clique constraints of type (5). Regarding the game simulation outcomes, they appear to be slightly better than those of the greedy framework presented in Table 2, with three distinct "perfect" solutions (i.e., tower configuration plans that win the game without losing any lives) found for the first time.

Overall, these experiments highlight that, although **S5** is the best of the tested strategies, tower configuration plans with the same **S5** score can produce different outcomes in the game simulation, and configurations with a lower score may sometimes perform slightly better in practice than those with a higher score. The results also show that, while using big-M no-overlap constraints significantly reduces the number of constraints in the model, using pairwise no-overlap constraints is empirically more effective, even when the big-M coefficients are fine-tuned.

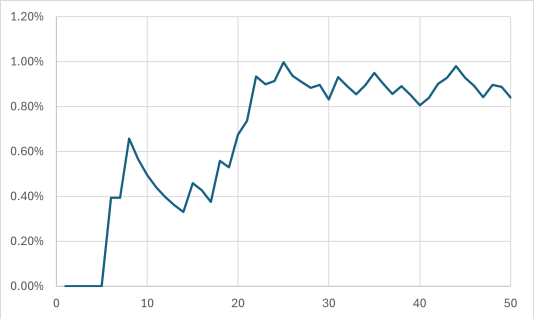
5.4 Experimenting with other game instances

Whereas the experiments in previous sections offered interesting insights related to the WBTDP, we must recognize that these were all based on a single game instance (albeit with various v_{jup} coefficients). In this last section, we modify this instance to derive additional findings.

First, we assess whether better results could be achieved by using a finer discretization. We previously allowed each tower to be constructed every 10 pixels, and we now assess the effect of reducing this spacing to 5 or 2 pixels. We report in Figure 10 the relative improvement in the multi-round framework (with the parameters described at the beginning of Section 5.3 and pairwise no-overlap constraints) achieved by using granularity 5 instead of granularity 10 for each round in mode **MD1** for strategy **S5** under the

pessimistic scenario **SC2**.

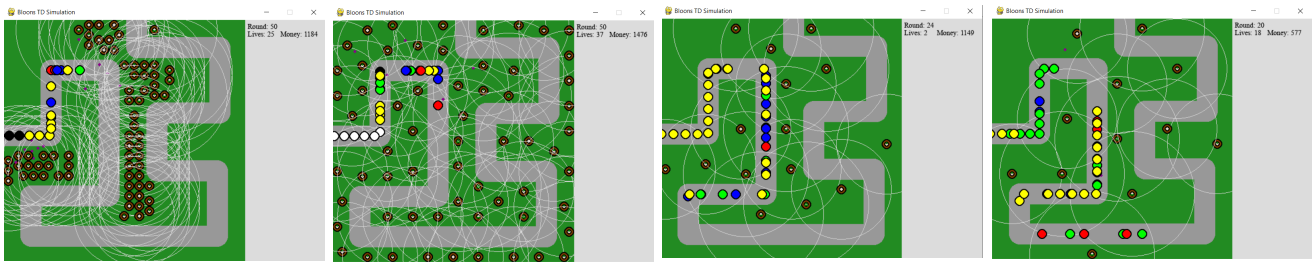
Figure 10: Relative improvement in the multi-round framework achieved by using granularity 5 instead of granularity 10.



While, as expected, better solutions can be obtained using a finer granularity, the improvement is relatively minor. In fact, while the resulting tower configuration plan still achieved a win in the game simulation, it did so with only 14 lives remaining at the end of round 50 (compared to 31 for granularity 10), once more showcasing that minor improvements on the optimization side do not necessarily translate into improvements on the simulation side. In addition, we highlight that the computation time increased dramatically when using a finer granularity: from less than 150s with granularity 10 to more than 5 hours with granularity 5. In fact, the increase was such that we could not run the framework for mode **MD2** or for granularity 2. We also attempted using big-M no-overlap constraints, but those did not help.

Second, we assess whether enforcing a form of tower dispersion (which tends to be a common strategy in TD games, but is currently neither encouraged by the objective function nor enforced by a set of constraints) could be beneficial. To that end, we artificially increased the footprint of every tower to three levels: small, medium, and large increase. Note that this increase affects only incompatibilities between two towers, and not those between a tower and the track or the map edges. We display in Figure 11 the tower configurations reached in the last round of the game simulation for the multi-round framework (with the parameters described at the beginning of Section 5.3 and pairwise no-overlap constraints) in mode **MD2** for strategy **S5** under the pessimistic scenario **SC2** for each level of tower-footprint increase.

Figure 11: Tower configurations obtained by the multi-round framework for different levels of tower-footprint increase: none (win with 25 lives), small (win with 37 lives), medium (loss at round 24), and large (loss at round 20).

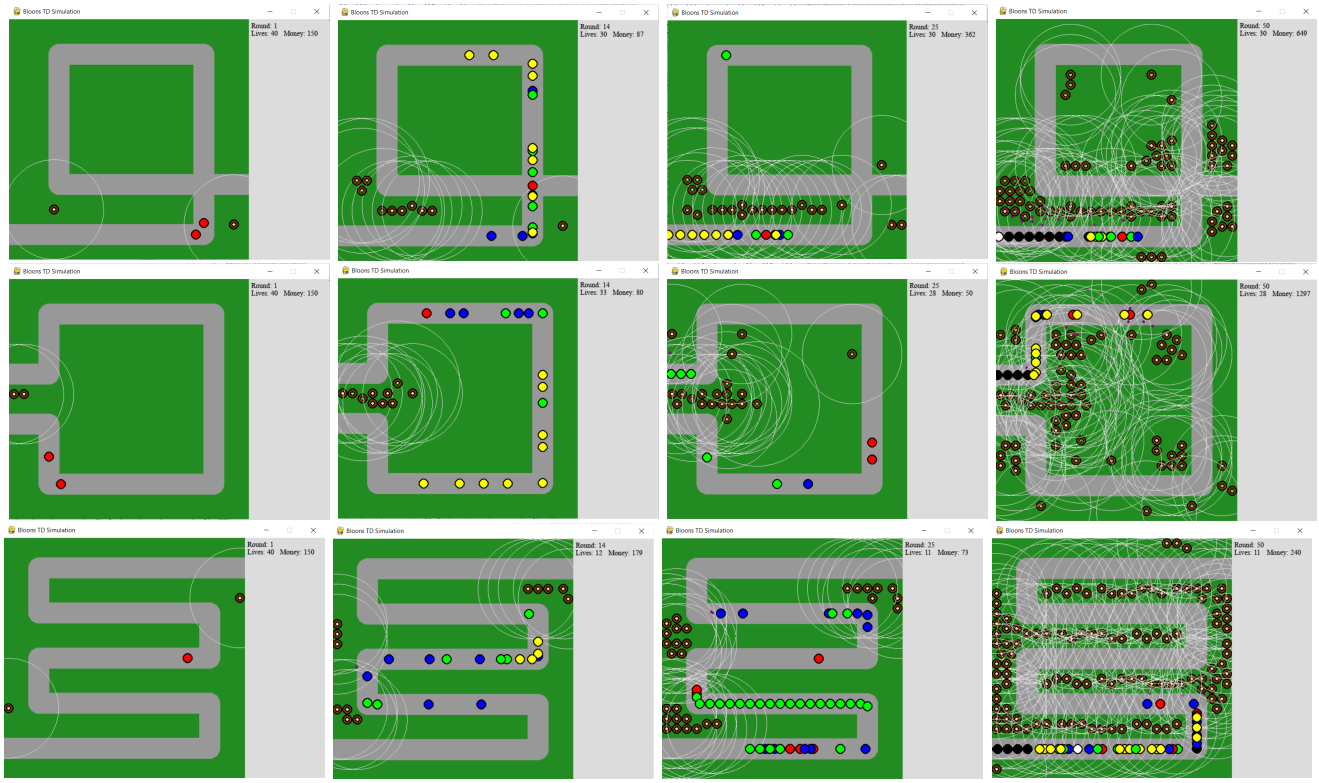


First, it is clear that when tower dispersion is neither encouraged nor enforced, the towers tend to be built in tight clusters (see the left-most figure). Enforcing a small amount of dispersion (middle-left figure) seems to be beneficial, as the number of lives remaining at the end of round 50 increases from 25 to 37. Increasing the dispersion requirement further (middle-right and right-most figures) has a considerably negative impact, ultimately leading to a game loss. Note that, in the figures, there were still available positions to construct towers, meaning that the issue was not that all positions were filled, but rather that the tower configuration plan was not effective enough. The same experiments were also made in mode

MD1, but there the limited number of available positions quickly became an issue, even for the smaller tower-footprint increase.

Finally, we wanted to asses whether the techniques we developed were not overfitting the single map available in the game, but instead could extend to new customized maps. We therefore developed three new game maps and integrated them into our game simulator: one in the shape of a loop, one in the shape of a circle, and one in the shape of a snake. We then solved each of these maps using the greedy framework in mode **MD2** for strategy **S5** under the pessimistic scenario **SC2**. We report in Figure 12 the tower configurations obtained at rounds 1, 14, 25, and 50 for each of the three maps. As the tower configuration plan for the snake map resulted in a loss in mode **MD2**, we report instead the plan for mode **MD1**, which resulted in a win.

Figure 12: Tower configurations obtained by the greedy framework for the "LoopMap" (mode **MD2**), the "CircleMap" (mode **MD2**), and the "SnakeMap" (mode **MD1**) at rounds 1, 14, 25, and 50.



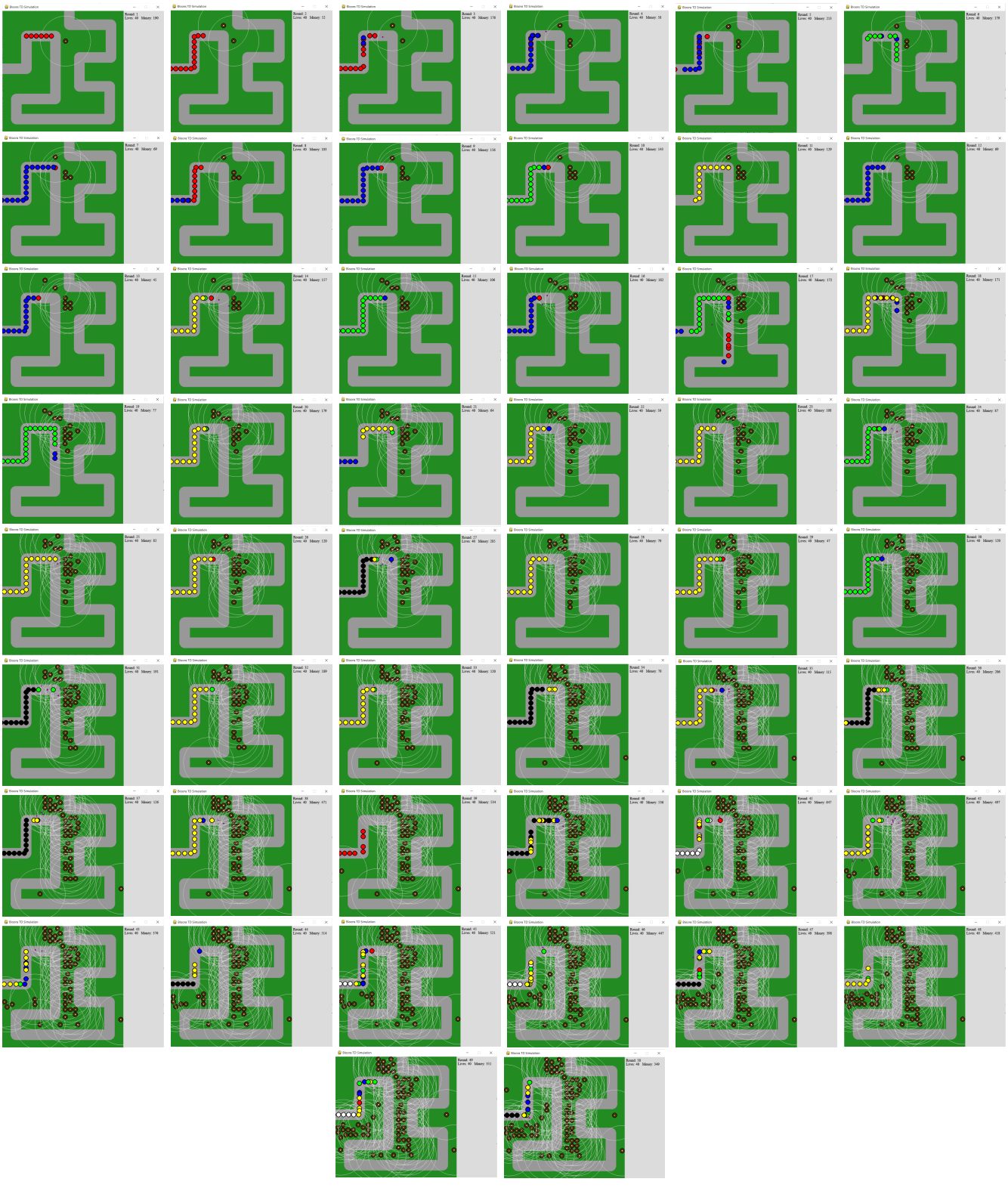
We note that a win could be secured by the tower configuration plans produced by our approaches for each of the three newly added maps, although not with all lives remaining and not in mode **MD2** for the snake map. Still, these results demonstrate the versatility of our approaches.

6 Conclusions

We formulated the problem of winning the game Bloons TD as a temporal two-dimensional knapsack problem with irregular shapes and side constraints, and we developed a game simulator as well as two solution frameworks that generate tower configuration plans which, when implemented in the game simulator, result in a win. An example of such a winning tower configuration plan that completes round 50 with all lives remaining is shown in Figure 13.

From an optimization perspective, we found that defining an objective function aligned with winning Bloons TD was particularly challenging; the only suitable ones were those derived from the game simulator. Even then, the resulting proxy function was still imperfect, as two tower configuration plans with the

Figure 13: Tower configuration plan that completes round 50 with all lives remaining.



same objective value could lead to different outcomes in the game simulation. We also observed that modeling no-overlap constraints (a well-studied issue in two-dimensional cutting and packing problems) was a challenge in this problem as well, with big-M constraints underperforming compared to their pairwise counterparts, even when the big-M coefficients are fine-tuned. Overall, computation time was a significant issue in several aspects of this work. Finding optimal solutions, even when using a high granularity for tower positioning, could take more than an hour in some cases. The game simulator, while allowing for accelerated gameplay, also required substantial time when used to compute the value of constructing each tower at each position, especially for lower-granularity instances.

As future research directions, we believe that designing better objective functions is essential for achieving stronger results (e.g., ensuring that all optimal solutions win the game without losing any lives). From the configuration in Figure 13, we observe that towers tend to be placed in clusters, a feature that is not penalized by the model but is not necessarily ideal in the game, since multiple projectiles may target the same creep, making some of them ineffective when fewer projectiles would have sufficed to eliminate that creep. While we showed that enforcing a certain level of dispersion through additional constraints could be beneficial, incorporating interaction costs among towers could also mitigate this issue, though it would naturally make the optimization problem more difficult to solve, as it would introduce a quadratic component into the objective function (quadratic knapsack problems being notoriously more difficult than their classical counterparts). We also observed that certain rounds (especially those in the early to mid-game) were much harder than others (typically those in the late game), which could motivate the use of round-specific coefficients in the multi-round framework. We also believe that using clique constraints, as done by Rodrigues and Toledo (2017), could improve the computational performance of the proposed methods. It would also be interesting to integrate the elements of the game we did not consider: the other three tower types (which are available in our game simulator, but are more difficult to associate with a value, as these towers have special effects such as slowing the creeps, which is only beneficial in the presence of towers that can damage them), the option of selling towers (not available in our game simulator, but easily integrable into our optimization frameworks), and the option of constructing towers during a round (also not available in our game simulator and very difficult to incorporate into our optimization frameworks as it would introduce many intermediate sub-rounds within each round). Finally, different variations of the game could also be studied. While our simulator incorporates new maps, well-known challenges include finding a winning tower configuration plan that uses the minimum amount of money and completing the game as quickly as possible.

Acknowledgments

This work used the Dutch national e-infrastructure with the support of the SURF Cooperative using grant no. EINF-12272. We also thank Martijn Schoot Uiterkamp for useful suggestions.

References

Akçay, F., Delorme, M., 2026. Solving the strip packing problem with a decomposition framework and a generic solver: Implementation, tuning, and reinforcement-learning-based hybridization. *Computers & Operations Research* 185, 107276.

Álvarez-Miranda, E., Luipersbeck, M., Sinnl, M., 2018. Gotta (efficiently) catch them all: Pokémon go meets orienteering problems. *European Journal of Operational Research* 265, 779–794.

Anderson, S., Falconer, R., 2025. Emergence of player tactics by expert-guided machine learning: An industry tower defence case study. *Entertainment Computing* 54, 100963.

Avery, P., Togelius, J., Alistar, E., Van Leeuwen, R., 2011. Computational intelligence and tower defence games. In *2011 IEEE Congress of Evolutionary Computation (CEC)*, IEEE, pp. 1084–1091.

Bell, J., Griffis, S., Cunningham III, W., Eberlan, J., 2011. Location optimization of strategic alert sites for homeland defense. *Omega* 39, 151–158.

Bennell, J., Oliveira, J., 2008. The geometry of nesting problems: A tutorial. *European journal of operational research* 2, 397–415.

Bennell, J., Oliveira, J., 2009. A tutorial in irregular shape packing problems. *Journal of the Operational Research Society* 60, S93–S105.

Bergdahl, J., Sestini, A., Gisslén, L., 2024. Reinforcement learning for high-level strategic control in tower defense games. In *2024 IEEE Conference on Games (CoG)*, pp. 1–8.

Blooncyclopedia, 2025. Bloons tower defense (game). https://www.bloonswiki.com/Bloons_Tower_Defense_(game), accessed 2025-02-22.

Bortfeldt, A., Winter, T., 2009. A genetic algorithm for the two-dimensional knapsack problem with rectangular pieces. *International Transactions in Operational Research* 16, 685–713.

Burke, E., Kendall, G., Whitwell, G., 2004. A new placement heuristic for the orthogonal stock-cutting problem. *Operations Research* 52, 655–671.

Cacchiani, V., Iori, M., Locatelli, A., Martello, S., 2022. Knapsack problems—an overview of recent advances. part II: Multiple, multidimensional, and quadratic knapsack problems. *Computers & Operations Research* 143, 105693.

Caprara, A., Monaci, M., 2004. On the two-dimensional knapsack problem. *Operations Research Letters* 32, 5–14.

Chvátal, V., 1975. A combinatorial theorem in plane geometry. *Journal of Combinatorial Theory, Series B* 18, 39–41.

Cid-Garcia, N., Rios-Solis, Y., 2021. Exact solutions for the 2d-strip packing problem using the positions-and-covering methodology. *Plos one* 16, e0245267.

Clautiaux, F., Detienne, B., Guillot, G., 2021. An iterative dynamic programming approach for the temporal knapsack problem. *European Journal of Operational Research* 293, 442–456.

Cordeau, J.F., Furini, F., Ljubić, I., 2019. Benders decomposition for very large scale partial set covering and maximal covering location problems. *European Journal of Operational Research* 275, 882–896.

Costa, M., Gomes, A., Oliveira, J., 2009. Heuristic approaches to large-scale periodic packing of irregular shapes on a rectangular sheet. *European Journal of Operational Research* 192, 29–40.

Côté, J.F., Dell'Amico, M., Iori, M., 2014. Combinatorial Benders' cuts for the strip packing problem. *Operations Research* 62, 643–661.

Den Hertog, D., Hulshof, P., 2006. Solving Rummikub problems by integer linear programming. *The Computer Journal* 49, 665–669.

Dias, A., Foleiss, J., Lopes, R., 2020. Reinforcement learning in tower defense. In *International Conference on Videogame Sciences and Arts*, pp. 127–139.

Guo, B., Wang, Y., Wang, Z., Huang, S., Xu, Z., Huang, C., Peng, Q., 2025. Irregular multi-plate packing based on the dotted board model. *Computers & Industrial Engineering* 209, 111439.

Hifi, M., M' Hallah, R., 2009. A literature review on circle and sphere packing problems: Models and methodologies. *Advances in Operations Research* 289, 150624.

Iori, M., de Lima, V., Martello, S., Miyazawa, F., Monaci, M., 2021. Exact solution techniques for two-dimensional cutting and packing. *European Journal of Operational Research* 289, 399–415.

Kenmochi, M., Imamichi, T., Nonobe, K., Yagiura, M., Nagamochi, H., 2009. Exact algorithms for the two-dimensional strip packing problem with and without rotations. *European Journal of Operational Research* 198, 73–83.

Kiwi, N., 2024. Bloons TD 6. https://ninjakiwi.com/Games/Mobile/Bloons-TD-6.html.

Kraner, V., Fister Jr, I., Brezočnik, L., 2021. Procedural content generation of custom tower defense game using genetic algorithms. In *International Conference "New Technologies, Development and Applications*, pp. 493–503.

Lafond, M., 2018. The complexity of speedrunning video games. In *9th International Conference on Fun with Algorithms (FUN 2018)*, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, pp. 27:1–27:19.

Lastra Díaz, J., Ortuño, M., 2026. A parallel branch-and-bound-and-check algorithm for nesting,

Leao, A., Toledo, F., Oliveira, J., Carravilla, M., Alvarez-Valdés, R., 2020. Irregular packing problems: A review of mathematical models. *European Journal of Operational Research* 282, 803–822.

Liu, T., Geng, J., Sun, X., Si, X., Zhang, H., Lu, Y., 2026. Multi-objective deployment planning of anti-UAV detection system at airport. *International Journal of Machine Learning and Cybernetics* 17, 88.

Mazurova, O., Samantsov, O., Topchii, O., Shirokopetleva, M., 2020. A study of optimization models for creation of artificial intelligence for the computer game in the tower defense genre. In *2020 IEEE International Conference on Problems of Infocommunications. Science and Technology (PIC S&T)*, pp. 491–496.

Mundim, L., Andretta, M., de Queiroz, T., 2017. A biased random key genetic algorithm for open dimension nesting problems using no-fit raster. *Expert Systems with Applications* 81, 358–371.

de Oliveira, G., 2023. Tower defense system – inspirado em bloons. https://github.com/ShadowHeinzZ/bloons-td-1, accessed 2025-03-15.

Oliveira, Ó., Gamboa, D., Silva, E., 2023. An introduction to the two-dimensional rectangular cutting and packing problem. *International Transactions in Operational Research* 30, 3238–3266.

OpenAI, Berner, C., Brockman, G., Chan, B., Cheung, V., Debiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R., Gray, S., Olsson, C., Pachocki, J., Petrov, M., de Oliveira Pinto, H.P., Raiman, J., Salimans, T., Schlatter, J., Schneider, J., Sidor, S., Sutskever, I., Tang, J., Wolski, F., Zhang, S., 2019. Dota 2 with Large Scale Deep Reinforcement Learning. Technical Report 1912.06680, OpenAI.

Ripamonti, L., Trubian, M., Maggiorini, D., Previti, S., 2017. Video games and operations research: Two synergic partners? *Computers in Entertainment* 16, 1–12.

Rodrigues, M., Toledo, F., 2017. A clique covering mip model for the irregular strip packing problem. *Computers & Operations Research* 87, 221–234.

Rummell, P., 2011. Adaptive ai to play tower defense game. In *2011 16th International Conference on Computer Games (CGAMES)*, pp. 38–40.

Sato, A., Martins, T., Gomes, A., Tsuzuki, M., 2019. Raster penetration map applied to the irregular packing problem. *European Journal of Operational Research* 279, 657–671.

de Souza Queiroz, L., Andretta, M., 2022. A branch-and-cut algorithm for the irregular strip packing problem with uncertain demands. *International Transactions in Operational Research* 29, 3486–3513.

Sutoyo, R., Winata, D., Oliviani, K., Supriyadi, D., 2015. Dynamic difficulty adjustment in tower defence. *Procedia Computer Science* 59, 435–444.

Suttichaya, V., 2017. Desktop tower defense is NP-hard. In *Trends in Artificial Intelligence: PRICAI 2016 Workshops*, pp. 19–25.

Tan, T., Yong, Y., Chin, K., Teo, J., Alfred, R., 2013. Automated evaluation for ai controllers in tower defense game using genetic algorithm. In *International Multi-Conference on Artificial Intelligence Technology*, pp. 135–146.

Taylor, T.L., 2012. *Raising the stakes: E-sports and the professionalization of computer gaming*. The MIT Press.

Toledo, F., Carravilla, M., Ribeiro, C., Oliveira, J., Gomes, A., 2013. The dotted-board model: a new MIP model for nesting irregular shapes. *International Journal of Production Economics* 145, 478–487.

Wiki, B., 2025. Bloons td wiki. https://bloons.fandom.com/wiki/Bloons_Wiki.

Yao, S., Zhang, H., Wei, L., Liu, Q., 2025. An exact approach for the two-dimensional strip packing problem with defects. *Computers & Industrial Engineering* 200, 110866.