A Branch-and-Bound Approach for Optimizing NPC Housing in Pre-Hardmode Terraria

Faiq Azzam Nafidz - 13524003
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: faiq.azzam@gmail.com , 13524003@std.stei.itb.ac.id

***Abstract*—Terraria, an indie game developed by Re-Logic, is a 2D sandbox adventure game where the player can dig, build, explore, and fight within. One of the mechanics introduced in the game is NPC housing, where players can assign which friendly NPC lives in which houses. The location/biome the NPC resides in and its neighbors affect the happiness of said NPC, with NPCs that are happy provide certain bonuses in their interactions with the player. With the variety of NPCs and biomes in the game, there are lots of combinations of how to assign the housing of NPCs. This paper explores the optimization of NPC housing in the game by converting it into a programming problem and solves it using a branch and bound approach.**

Keywords—optimimization; branch and bound; terraria; housing; happiness mechanic

I. Introduction

Terraria is one of many successful indie games that has been released in the year 2011 [1]. The game is set in a 2D randomly generated world where players can dig, build, explore, and fight within it. The game is considered as one of the best-selling video games of all time, as of now, with 70 million copies sold (as of 2026) across different kinds of devices [2]. One of many achievements of this game is that it has received a Labor of Love award in 2021 Steam Awards [3]. Not only that, but Terraria also remains as one of the beloved games by the community, being the first ever game on Steam that reached more than 1 million reviews and still holds the "Overwhelmingly Positive" rating [4].

One of the aspects the game offers is to build things with resources gathered within the world the player is in. Building houses in the world can lead to friendly NPCs (non-player characters) to move in into the world. Those friendly NPCs can benefit the player in certain ways, such as giving the player quests, selling items, giving tips on how to advance, and others. Those benefits can be amplified when the NPC that gives those benefits is happy, with NPCs happiness affected by placement of their house in certain biome in the world and neighbors they are living with. Since there are a lot of NPCs and biomes in the game, there are countless ways to settle those NPCs in various biomes and adjust their neighbors.

That NPC happiness mechanic in the game creates a puzzle for players to determine which configuration of NPC housing is the optimal one to maximize benefits from their happiness. This kind of puzzle can be transformed into a programming problem where optimization is the focus. The goal is to optimize the configuration of NPC housing in terraria based on the biome the house in and the neighbor of NPC that resides within. The solution of that programming problem can be attained using a branch and bound approach as an example of the implementation of one of many algorithmic strategies in Terraria.

II. Theroretical Basis

A. Terraria and The Happiness Mechanic

Terraria is a game described by the developer as a game that blends the elements of classic action games with the sandbox-style creativity freedom [1]. The game features numerous biomes and friendly NPCs that can move in within the world. Biomes in Terraria are types of area within a Terraria world, each has its own characteristic of certain terrain blocks, walls, backgrounds, enemies, critters, music, and other features of the game [5]. On the other hand, friendly NPCs (or just NPCs as referred in the game's wiki page) are non-player characters that provide certain services to players, with most of them offers items in exchange of coins (in-game currency for trading with NPCs) [6]. Players can build houses in the world to attract certain NPCs into settling into the world. Those NPCs that can settle into a house that players build are called town NPCs, which will be the discussion of this paper.

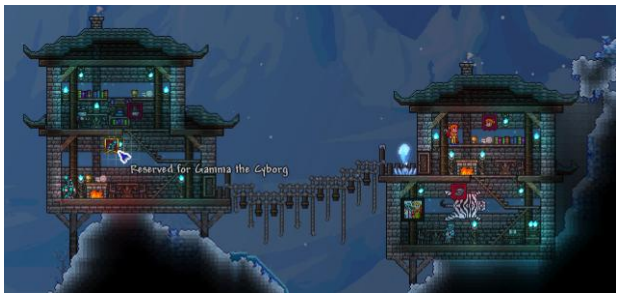

Fig. 1. Example of NPC housing in-game

Each town NPCs has unique living preferences. Meeting their living preferences can boost their happiness that in turn amplify the services given to players (in the form of discount on items purchases, higher prices on selling items, more rewards on

completing quests, etc.). There are several factors related to NPC housing that determine the happiness of NPCs: the crowdedness of the NPC's living location, the biome in which the house is located, and which specific NPCs are their neighbors; each with their own price modifier [6]. Onwards, NPC happiness is represented by the percentage of NPC's item prices that are being sold relative to their original prices, with the lower the percentage, the happier the NPC is.

Fig. 2. Example of NPC shop interface in Terraria. The happiness percentage is displayed in the shop interface and directly affects item prices, reducing the Forest Pylon price from 10 gold originally to 8 gold 90 silver in this example.

On crowdedness factor, there are three possible conditions of how crowded the NPC housing is. The first condition, called Solitude Bonus with price modifier of 95%, is when an NPC lives with maximum of two other NPCs within 25 tiles and maximum of five NPCs within 120 tiles. Breaking the first constraint (more than two other NPCs within 25 tiles) led to the second condition called Crowded Penalty with price modifier of 105%, but only breaking the second constraint (more than five NPCs within 120 tiles led to the third neutral condition with price modifier 100%.

The biome of where an NPC lives also affects the happiness of said NPC. Each NPC has their own preferences for certain biomes, which are categorized as loved, liked, disliked, or hated; each with price modifier of 88%, 94%, 106%, or 112%.

Lastly, the specific neighbor of NPCs further affects the happiness of NPCs. An NPC is considered a neighbor of another NPC if said NPC is within 25 tiles of said another NPC. This means that there could be only 2 maximum neighbor NPC before the Crowded Penalty occurs, further complicates the assignment problem. Similar as the biome factor, each NPC has their own preferences for certain NPC, which are categorized as loved, liked, disliked, or hated; each with price modifier of 88%, 94%, 106%, or 112%.

This paper will only try to find the optimal NPC housing configuration in the pre-hardmode stage of Terraria (which is a stage before the player defeating a certain boss, the Wall of Flesh, that unlocks a lot more contents to the game). This is done to simplify the problem that is set as an example of how algorithmic strategies can be applied in the game. Below is the list of all pre-hardmode NPCs preferences towards biomes and neighbors (note that the Underground biome in the table includes Cavern and Underworld biomes also to minimize space, since they are always mentioned together) [6].

TABLE I. PRE-HARDMODE NPCS LIVING PREFERENCES

NPC	Biome Preference		Neighbor Preference	
	Likes	Dislikes	Likes	Dislikes
Guide	Forest	Ocean	Clothier	Painter (hates)
			Zoologist	
Merchant	Forest	Desert	Golfer	Angler (hates)
			Nurse	
Zoologist	Forest	Desert	Witch Doctor (loves)	Arms Dealer (hates)
			Golfer	Angler
Golfer	Forest	Underground	Angler (loves)	Merchant
			Painter	
			Zoologist	
Nurse	-	Snow biome	Arms Dealer (loves)	Zoologist (hates)
				Dryad
				Party Girl
Tavernkeep	-	Snow biome	Demolitionist (loves)	Dye Trader (hates)
			Goblin Tinkerer	Guide
Party Girl	-	Underground	Zoologist (loves)	Merchant
			Stylist	
Demolitionist	Underground	Ocean	Tavernkeep (loves)	Arms Dealer
			Mechanic	Goblin Tinkerer
Goblin Tinkerer	Underground	Jungle	Mechanic (loves)	Stylist (hates)
			Dye Trader	Clothier
Clothier	Underground	-		Mechanic (hates)
				Nurse
Dye Trader	Desert	Forest	Arms Dealer	
			Painter	
Arms Dealer	Desert	Snow biome	Nurse (loves)	Demolitionist (hates)
				Golfer
Dryad	Jungle	Desert	Witch Doctor	Golfer (hates)
				Angler
Painter	Jungle	Forest	Dryad (loves)	
			Party Girl	
Witch Doctor	Jungle	-	Dryad	Nurse
			Guide	
Stylist	Ocean	Snow biome	Dye Trader (loves)	Goblin Tinkerer (hates)
				Tavernkeep

NPC	Biome Preference		Neighbor Preference	
	Likes	Dislikes	Likes	Dislikes
Angler	Ocean	Desert	Demolitionist	Tavernkeep (hates)
			Party Girl	
Mechanic	Snow biome	Underground	Goblin Tinkerer (loves)	Clothier (hates)
				Arms Dealer

B. Combinatiorial Optimization Problem

Combinatorial optimization problem is an optimization problem that involves finding optimal solution from finite set of discrete solutions in a discrete space [7]. This kind of problem can be found in various activities such as scheduling, routing, assigning, decision-making, etc. A combinatorial optimization problem can be defined as a set of instances

$$C = (F, c) \quad (1)$$

where F is set of feasible solutions and c is cost function [7].

The combinatorial optimization problem concept is connected to this paper because it is very similar to the NPC housing optimization problem. In fact, the NPC housing optimization problem can be viewed as a combinatorial optimization problem, where the objective is to determine an arrangement of NPCs in certain housing locations that maximizes overall NPC happiness. Since, in this problem, the number of NPCs and biomes in Terraria are finite, there are finite sets of discrete solutions to the problem that corresponds to the combinatorial optimization problem characteristics.

C. Branch and Bound

Branch and bound is an algorithmic strategy that can solve many programming problems, such as combinatorial optimization problems. The main idea of branch and bound is to search, systematically, the space of candidate solutions by organizing it as a tree, with each node representing a partial solution and each leaf representing complete solution. The algorithm operates through two kinds of processes that give it its name: branching and bounding [8].

Branch and bound can be viewed as an improvement over exhaustive search [8]. Instead of exploring every possible solution, the algorithm estimates whether a partial solution can still lead to an optimal answer. If it cannot, the corresponding search subtree with the root as the node that represents the partial solution is discarded, meaning the algorithm will not search further on that subtree. This process significantly reduces the number of nodes that need to be evaluated with the right estimation that is better the nearer the estimation is to the optimal solution.

In branch and bound, a node that has been generated but not yet expanded is called a live node. A live node at a time is selected to become the expanding node (E-node), which is the node currently being explored by the algorithm. The algorithm then expands the expanding node, which means generating all its possible children according to the problem constraints. Newly generated nodes become live nodes and can be selected as E-nodes in the future.

The search process is commonly represented by using a tree. The root node of the tree represents the initial state of the problem before any decision has been made. Internal nodes represent partial solutions, while leaf nodes correspond to complete solutions. Every path from the root to a leaf represents one feasible solution to the optimization problem.

The branching process partitions a problem into smaller subproblems. Suppose a node represents a partial assignment of NPCs to housing groups. Branching generates several child nodes, each corresponding to a different valid choice for the next housing group and biome assignment. By repeatedly branching, the algorithm eventually generates complete housing configurations.

The second component of the algorithm is bounding. For every generated node, a bound value is computed. In minimization problems, the bound represents an optimistic estimate of the minimum possible objective value that can still be achieved from the partial solution represented by the node. The bound must never overestimate the true optimal completion cost, otherwise optimal solutions could be incorrectly discarded.

Bounding enables pruning. Let UB denote the cost of the best complete solution found so far, often called the incumbent solution. If a node has a lower bound LB such that $LB \geq UB$ then no solution inside that node's subtree can improve the incumbent solution. Therefore, the entire subtree can be safely pruned without affecting the correctness of the algorithm. This pruning mechanism is the primary reason branch and bound is often significantly faster than brute-force enumeration.

The effectiveness of a branch and bound algorithm really depends on the quality of the bounding function. A tighter bound (higher lower-bound and lower upper-bound) allows more subtrees to be pruned earlier, reducing both memory usage and execution time. Conversely, a loose bound, while guarantees completeness, may force the algorithm to explore a much larger portion of the search tree, increasing process time on average.

In the context of NPC housing optimization, branch and bound is suitable because each partial housing configuration can be evaluated independently, and the computation on optimistic estimates of the remaining NPC happiness scores is straightforward. These estimates allow large portions of the search space to be discarded even before complete housing assignments are generated, making exact optimization feasible despite the large number of possible configurations.

III. Implementation

A. Problem Statement

Let $N = \{n_1, n_2, ..., n_k\}$ be the set of town NPCs available, and let $B = \{b_1, b_2, ..., b_l\}$ be the set of available biomes that are considered distinct from each other by NPCs. The goal is to partition N into disjoint groups $G_1, G_2, ..., G_m$ where each group G_i has size at most three ($|G_i| \leq 3$), and to assign each group G_i a biome $\beta_i \in B$. This group represents a set of NPCs that are considered neighbor/nearby of each other's (in a 25-tile distance

from each other). The decision to set each group to have at maximum 3 members in it (which means 3 NPCs at most in one group) is to satisfy the best crowdedness condition that leads to a Solitude Bonus, which states that at maximum there could be only two other NPCs within a 25-tile area from an NPC that in turn means a group in 25-tile radius should only have at maximum 3 NPCs in it.

For a group $G_i = \{n_a, n_b, ...\}$ assigned to biome β_i, the happiness score of each member NPC n_a is computed as a product of applicable price modifiers:

$$H(n_a, \beta_i, G_i) = M\_biome(n_a, \beta_i) \times M\_crowd(|G_i|) \times \prod_{n_k \in G_i, n_k \neq n_a} M\_neighbor(n_a, n_k) \quad (2)$$

where M_biome(n_a, β_i) is the biome price modifier for NPC n_a in biome β_i (0.88 for loved, 0.94 for liked, 1.00 for neutral, 1.06 for disliked, or 1.12 for hated), M_crowd($|G_i|$) is 0.95 when $|G_i| \leq 3$ (satisfying the Solitude Bonus condition), or 1.05 otherwise, and M_neighbor(n_a, n_k) is the neighbor price modifier between n_a and n_k (0.88 for loved, 0.94 for liked, 1.00 for neutral, 1.06 for disliked, or 1.12 for hated). The total cost of a group is the sum of happiness scores of all its members:

$$cost(G_i, \beta_i) = \sum_{n_a \in G_i} H(n_a, \beta_i, G_i) \quad (3)$$

Because the grouping rules that are set make a housing group that are fully independent of each other (one group doesn't affect other group's happiness), the total cost of a complete assignment can be obtained by summing every group's cost:

$$TotalCost(S) = \sum_i cost(G_i, \beta_i) \quad (4)$$

with $S = \{(G_1, \beta_1), (G_2, \beta_2), \ldots, (G_m, \beta_m)\}$ represents solution of the problem.

The NPC housing optimization objective is to find a partition and biome assignment that minimizes the value of TotalCost.

B. Search Tree Structure

A search tree that represents the solution space is made to search for the optimal solution that is done by the branch and bound algorithm. Each node in the tree that is not on the leaf represents a partial solution, which is a set of housing groups that have been formed so far at certain level (the level where the nodes resides in the tree corresponds to how many groups has been formed, with root node considered as level 0). The root node represents the empty partial solution, where all 18 NPCs are unassigned. A leaf node represents a complete solution, where all 18 NPCs each have been assigned to certain group.

Branching is performed at each node that is not a leaf, by assigning one, two, or three NPCs that have not been assigned to a group and assigning that group a biome. This creates multiple child nodes that each corresponds to a different choice of a group made and biome assigned to said group. The cost accumulated at each node is the sum of costs of all groups formed so far, and the remaining cost is yet to be determined.

Each NPC is given an index value based on the order on the preferences table on Table I. This is done so, when forming a group, the earlier member of the group can be set to be the NPC that has the lowest index out of the rest (index of the first member is the lowest, second is lower than the third, and the third is the highest indexed NPC). This prevents the creation of multiple groups with the same set of members but each with different members ordering that can lead to redundant nodes on the tree.

Two paths in the tree, from root to a certain node, are considered the same if the groups that are made until the certain node are the same, even with different ordering of creation. Searching on those two nodes lead to the same generated subtree, making it the second search redundant. To prevent that, every group creation must consist of the first member of the group being the lowest indexed NPC that has not been assigned to a group. This will make the group creation order to be sorted by the first group member from the lowest indexed NPC to the highest.

C. Lower Bound Computation

The computation of the lower bound at any given node is done by estimating the minimum possible cost for each remaining unassigned NPC, then summing those estimates. For an unassigned NPC n_i, the lower bound contribution is the lowest happiness score that n_i could achieve by having their most preferred biome, their most preferred unassigned NPCs as neighbors (that gives the lowest price modifier for n_i), and the Solitude Bonus crowd modifier of 0.95. The equation to get that lower bound contribution is as follows.

$$LB\_contrib(n_i, remaining) = \min_{b_j \in B}(\text{M\_biome}(n_i, b_j)) \times 0.95 \times \min_{n_j \in remaining, n_j \neq n_i}(M\_neighbor(n_i, n_j)) \quad (5)$$

This lower bound contribution calculation is considered optimistic because it assumes that every NPC that remains can all be placed with their best possible neighbor in their preferred biome, which is unlikely to be achievable in a single feasible partition (since each NPC can only be in one group). With it, the lower bound of a node can be calculated by summing the accumulated cost with the sum of every lower bound contribution for every remaining NPCs not yet assigned to a group. The equation to calculate the lower bound for a node is as follows.

$$LB(node) = TotalCost(node) + \sum_{n_i \in remaining} LB\_contrib(n_i, remaining) \quad (6)$$

It is a valid lower bound because of the optimistic nature of the calculation so that the actual cost of any completion from a certain node is at least as large as the lower bound estimates. When LB(node) value is greater than or equal to the upper bound, which is the cost of the best complete solution so far, the node is pruned and its subtree is not explored by the algorithm.

D. Initial Upper Bound Computation

When dealing with a big solution space such as this problem offers, it's not enough to just initialize the upper bound to be "infinite" (higher than any number possible). Upper bound with infinite value at the start of the calculation delays early pruning until a solution is found, because pruning is done for nodes with LB(node) greater than the upper bound (which is impossible since upper bound has value greater than any number possible). To improve early pruning in the calculation, a definitive upper bound is needed to optimize the process.

One way to obtain an upper bound is to find a solution that is considered near optimal; using a greedy method that does not guarantee an optimal solution. The simplest greedy strategy to find a near optimal solution is to just assign each NPCs into a group of one member with preferred biome of said NPC assigned to the group. With this, the computation doesn't involve any of the complicated neighbor preferences, so it is just a straightforward summing each NPCs happiness with preferred biome modifier and the Solitude Bonus modifier. The equation is as follows.

$$init\_UB = \sum_{n_i \in N} \min_{b_j \in B}(M\_biome(n_i, b_j)) \times 0.95 \quad (7)$$

E. Creating Problem Instance

Since this paper explores optimizing NPC housing in Terraria at the pre-hardmode stage, the set that represents the NPCs to be assigned is $N = \{n_1, n_2, \ldots, n_{18}\}$ where n_i represents an NPC that is listed in Table I at *i*-th order from the beginning (N = {Guide, Merchant, …, Mechanic}).

There are six biomes that are considered distinct from each other by the NPCs in pre-hardmode, according to Table I. That means, the set that represents available biomes that can be assigned to a group of is B = {Forest, Underground, Desert, Jungle, Ocean, Snow}. Note that the Underground biome in the set includes Cavern and Underworld biomes, since they are always mentioned together.

Each NPC preference data is also needed for the calculation of optimal NPC housing configuration. The data can be found at Table I. The biome preference data can be represented as a function that returns the price modifier as below.

$$M\_biome(n_i, b_j) = \begin{cases} 0.88, & n_i \ loves \ b_j \\ 0.94, & n_i \ likes \ b_j \\ 1.06, & n_i \ dislikes \ b_j \\ 1.12, & n_i \ hates \ b_j \\ 1, & otherwise \end{cases} \quad (8)$$

The neighbor preference data can also be presented as a similar function as below.

$$M\_neighbor(n_i, n_j) = \begin{cases} 0.88, & n_i \ loves \ n_j \\ 0.94, & n_i \ likes \ n_j \\ 1.06, & n_i \ dislikes \ n_j \\ 1.12, & n_i \ hates \ n_j \\ 1, & otherwise \end{cases} \quad (9)$$

A solution to this is in the form of $S = \{(G_1, \beta_1), (G_2, \beta_2), \ldots, (G_m, \beta_m)\}$ where $G_i = \{n \mid n \in N\}$, $|G_i| \leq 3$, $\cup_i G_i = N$, and $\cap_I G_i = \emptyset$, also $\beta_i \in B$.

F. Program Implementation

To translate the problem into code, data structures need to be created first. Each NPC is represented by a record called NPC, containing NPC name, biome preference array, and neighbor preference array with each array stores the price modifier based on the M_biome and M_neighbor function above. A group is represented by a record called Group containing mask (an identifier unique to a group, used to iterate every possible group easily), biome that is an enum, and cost that represents the happiness value in price modifier.

```
struct NPC {
    string name;
    double biome_mods[BIOME_COUNT];
    double neighbor_mods[18];
};
struct Group {
    uint32_t mask;
    Biome biome;
    double cost;
};
```

Biomes are represented as an enum that contains 6 different biomes according to the set B in the problem instance.

At the start, the program loads all NPC and their preference information and converts it into numerical modifiers according to the set N in problem instance and function at (8) and (9). After that, all values that needed to be cached/stored are computed: mapping of mask to index of possible group, cost of every possible group, and the lower bound table that stores the lower bound of every possible node. These are done to optimize the code so that there are no redundant calculations. The upper bound is also calculated after the caching process completed.

The root node of the search tree is created as an empty NPC housing configuration, represented as a mask (number that represents the node). The root node contains an empty list of groups, zero accumulated cost, and the complete set of NPCs marked as unassigned. The branch-and-bound algorithm then repeatedly performs branching and bounding operations on the tree from the root node until either all nodes have been explored or all remaining nodes have been pruned. When the algorithm visits a leaf node, a node with complete housing configuration, its total cost is compared against the current solution, which the cost is the upper bound. If the cost of the newly found solution is lower than the cost of the current solution, the current solution is updated to match the newly found solution.

The final output of the program should be the housing configuration with the minimum total cost together with the corresponding biome assignments and search statistics.

Full code implementation can be downloaded in the link: https://drive.google.com/drive/folders/15mKWzAdeOo0mVSMw4gNxnoSvVcf5tSuL?usp=drive_link

G. Result

The program outputs as follows.

```
================ OPTIMAL CONFIGURATION ================
Group 1:
  Biome: Forest
  NPCs : Guide, Zoologist, Golfer
  Group Cost: 2.5183

Group 2:
  Biome: Forest
  NPCs : Merchant
  Group Cost: 0.8930

Group 3:
  Biome: Desert
  NPCs : Nurse, Dye Trader, Arms Dealer
  Group Cost: 2.4613

Group 4:
  Biome: Underground
  NPCs : Tavernkeep, Demolitionist
  Group Cost: 1.6218

Group 5:
  Biome: Ocean
  NPCs : Party Girl, Stylist, Angler
  Group Cost: 2.6254

Group 6:
  Biome: Snow
  NPCs : Goblin Tinkerer, Mechanic
  Group Cost: 1.6218

Group 7:
  Biome: Underground
  NPCs : Clothier
  Group Cost: 0.8930

Group 8:
  Biome: Jungle
  NPCs : Dryad, Painter, Witch Doctor
  Group Cost: 2.4647

------------------------------------------------------
Total Happiness Cost = 15.0993
Nodes Generated      = 4722025
Nodes Pruned         = 4704912
Execution Time       = 69 ms
======================================================
```

This means, an optimal solution is found in a reasonable time and is better than the greedy solution calculated as the upper bound. The optimal solution can be written as S = {({Guide, Zoologist, Golfer}, Forest), ({Merchant}, Forest), ({Nurse, Dye Trader, Arms Dealer}, Desert), ({Tavernkeep, Demolitionist}, Underground), ({Party Girl, Stylist, Angler}, Ocean), ({Goblin Tinkerer, Mechanic}, Snow), ({Clothier}, Underground), ({Dryad, Painter, Witch Doctor}, Jungle)}.

IV. Conclusion

This paper has explored the NPC housing optimization problem in Terraria as an example of a combinatorial optimization problem and solved it using a branch and bound approach. The happiness mechanic in Terraria that is affected by biome and neighbor preferences of each NPC, plus the crowdedness of each group of NPC housing, creates a kind of puzzle for players to tackle on how to optimize the NPC housing. Not only that, but the happiness mechanic in Terraria also being an example of a combinatorial optimization problem that can be converted into a programmable problem and solved using algorithmic strategies such as branch and bound.

It has been found that the optimal solution to said problem is achievable through the means of programming with minimal code and execution time in mere milliseconds. The speed of the program is further increased by precomputing calculations that will be used in the program to avoid redundant calculations as a form of code optimization. The optimal solution to the problem is as follows. S = {({Guide, Zoologist, Golfer}, Forest), ({Merchant}, Forest), ({Nurse, Dye Trader, Arms Dealer}, Desert), ({Tavernkeep, Demolitionist}, Underground), ({Party Girl, Stylist, Angler}, Ocean), ({Goblin Tinkerer, Mechanic}, Snow), ({Clothier}, Underground), ({Dryad, Painter, Witch Doctor}, Jungle)} with S = {(G_1, β_1), (G_2, β_2), …, (G_m, β_m)} as the solution, G_i as a group of NPC that lives nearby and considered neighbors to each other, and β_i as the biome the group resides in. This optimal solution can be a guide to players that want to maximize the happiness mechanic benefits in Terraria at the pre-hardmode stage that minimizes price modifiers as low as possible.

For future work, this problem can be solved by using different algorithm strategies that are applicable as a comparison to the branch and bound algorithm. Additionally, the decision to make every group of NPCs has three members maximum can be removed to expand the search even further, but it is not advised to use this paper's strategy as it will create an even bigger space solution that couldn't be searched through in a reasonable time. Another further analysis and implementation can be done by weighing each NPC with a constant or other means that represents the importance of maximizing the NPC's happiness, because some NPCs in Terraria is subjectively favored to maximize their happiness (NPCs most used by the player such as Goblin Tinkerer, Nurse, Arms Dealer, etc.) and some other NPCs do not significantly benefit from happiness optimization (NPCs that don't sell items or monetary services such as Guide or that is rarely used by the player such as Clothier, Dye Trader, Stylist, etc.).

Acknowledgment

The author would like to express his gratitude first and foremost to Allah for providing the strength, motivation, courage, and inspiration to complete this paper. The author also expresses his gratitude to family and friends that has been supporting the author from the beginning of the author's college life, some even before that. Finally, the author expresses his gratitude to the author's teacher, Mr. Rinaldi, that has granted the author the knowledge that is used as a basis for this paper for an entire semester in Strategi Algoritma course in college.

REFERENCES

[1] Re-Logic, "Terraria," Steam, 2011. [Online]. Available: https://store.steampowered.com/app/105600/Terraria/ (Accessed: June 18, 2026)

[2] Re-Logic, " Celebrating 15 Years of Terraria!" Steam, 2026. [Online]. Available: https://store.steampowered.com/news/app/105600/view/662735945909403986 (Accessed: June 18, 2026)

[3] Re-Logic, "Terraria Wins the 2021 Labor of Love Steam Award!" Steam, 2022. Available: https://store.steampowered.com/news/app/105600/view/3131689217797601169 (Accessed: June 18, 2026)

[4] Xueyang, "Terraria to Become the First “Overwhelmingly Positive” Game on Steam with Over 1 Million Reviews," Superpixel, 2022. [Online]. Available: https://www.superpixel.com/article/194835/terraria-become-first-overwhelmingly-positive-game-steam-over-1-million-reviews (Accessed: June 18, 2026)

[5] Terraria Wiki, "Biomes," Wiki.gg, 2026. [Online]. Available: https://terraria.wiki.gg/wiki/Biomes (Accessed: June 18, 2026)

[6] Terraria Wiki, "NPCs," Wiki.gg, 2026. [Online]. Available: https://terraria.wiki.gg/wiki/NPCs (Accessed: June 18, 2026)

[7] E. Kaya, B. Gorkemli, B. Akay, and D. Karaboga, “A Review on The Studies Employing Artificial Bee Colony Algorithm to Solve Combinatorial Optimization Problems, Engineering Applications of Artificial Intelligence, Volume 115, 2022. [Online]. Available: https://doi.org/10.1016/j.engappai.2022.105311 (Accessed: June 18, 2026)

[8] Rinaldi, N. U. Maulidevi, M. L. Khodra, "Algoritma Branch and Bound (Bagian 1)," IF2211 Strategi Algoritma, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung, 2026. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2025-2026/17-Algoritma-Branch-and-Bound-(2026)-Bagian1.pdf (Accessed: June 19, 2026)

APPENDIX

Program implementation (source code and executable): https://drive.google.com/drive/folders/15mKWzAdeOo0mVSMw4gNxnoSvVcf5tSuL?usp=drive_link

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 19 Juni 2026

Faiq Azzam Nafidz–13524003