Application of Knapsack Problem in Procedurally Generated Game Levels

Muhamad Ihsan (13511049)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13511049@std.stei.itb.ac.id

***Abstract*—Video games are very good entertainment materials.They're good for spending free time, and therefore must be fun, refreshing, and of course exciting. One of the most important aspects in an exciting game is the concept of levels. Levels make players want to play continuously and won't bore them. Levels make the game harder each time the player progresses, making the player plays on and on. Moreover, with the addition of randoming aspect on level generations, the game becomes more challenging, just by adding some simple surprise factors. Of course, the levels won't be fully randomized, there are a lot of factors to be taken into account, such as difficulty, player's choice, etc. Those factors make randoming a level a not-so-simple problem. This is where Knapsack Problem comes in. The solution to Knapsack Problem offers a good way to generate a randomized level, while still considering the factors mentioned before.**

***Index Terms*—Games, Level Generations, Randoms, Knapsack Problems**

I. INTRODUCTION

Nowadays, everybody knows what video games are. Many even loves them, because video games are so fun and entertaining. What people see from a video game can differ so much, someone may like it for its stunning graphics, others for its heartwarming story. And many others love the game solely because of its exciting gameplay.

What is a gameplay? Gameplay is the way the game is played. Gameplay defines the game itself, if not, why else would people call it a 'Gameplay'? Gameplay is the core of the game, while the graphics, the music, the story, all of them are just additions.

Gameplay itself, is a very broad topic. First of all, there's game genre, which is some kind of the big theme of the game. A game's genre can be adventure, puzzle, racing, you name it. And even if there are two games with the same genre, they can still differ greatly to the point of making people questioning 'where's the similarity between these games?'.

Different though as they might be, games can't be far from the concept of levels. Levels don't have to differ by difficulty, which has become a common misconception. Levels don't have to get harder each time the player progresses, although the majority of the games out now follow that guidelines. Still, levels have to be different from one another. The difference can be in the form of theme, or maybe a slight change in gameplay.

There are two general types of level design based on how they are generated, static-level and randomly-generated-level.

Static-level is where the levels in the game are statically defined by the game creator. The characteristics of this level design is that the game surely follow a strict storyline plot and a fixed asset allocation for each of the level inside it. This level design is usually implemented on a big serious game with heavy story and detailed graphic. Or it can be implemented on small games also, but one thing is certain, the game creator already predefined all the possible routes from the game's start to finish.

Randomly-generated-level is where the levels in the game are randomly generated each time a certain milestone is passed, for example after finishing a level. This level design allows near-infinite combination of levels to be faced by the players. It is usually implemented on small-scale games, where replayability is highly valued, and this design offers it. Replayability makes a game stay exciting even after numerous playthroughs, because the levels are different each time the player starts the game anew. A game genre that consistently applies this concept is called **roguelike**.

Although extremely hard, it is not impossible to make a game with randomly-generated-levels contains a definitive story. It will put constraints on the level generations and cause generating a perfectly suitable level seems a bit harder. It will put another factor to be considered in the game's generation algorithm.

It's not like there are no consideration factors to begin with. Factors such as player's level, player's choice, level theme, etc. have always been there. These factors will no doubt make calculation a bit harder while still keeping the

levels randomed.

The main solution for the level generation problem is to apply the Knapsack Problem. The main idea of Knapsack Problem is to fit as many goods as possible inside the bag so that it wouldn't overweight and we get the most profitable combination of goods.

Similar to the Knapsack Problem, in level generation problem, the level (bag) must include game contents (goods) as optimally as possible while not passing the content limit (not overweight). And there are also a few factors to be considered when attempting to solve with the Knapsack Problem, which will be discussed later.

II. Related Theories

A. Game Level Design

Game Level Design is one of the disciplines of game development revolving around the creation of video game levels, stages, missions, etc. Level design is both an artistic and technical process.

There are two primary purposes to be fulfilled by Level Design, which is providing players with a goal, and providing players with enjoyable play experience. Good level design strives to produce quality gameplay, provide an immersive experience, and sometimes, especially in story-based games, to advance the storyline.

Maps' design can significantly impact the gameplay. For example, the gameplay may be shifted towards a platformer (by careful placement of platforms) or a puzzle game (by extensive use of buttons, keys, and doors). Some FPS maps may be designed to prevent sniping by not including any long hallways, while other maps may allow for a mix of sniping and closer combat.

Gimmick maps are sometimes created to explore selected features of gameplay, such as sniping or fist fighting. While they are briefly useful to level designers and interesting to experienced players, they are usually not included in final list of levels of the game because of their limited replay value.

Levels are generally constructed with flow control in mind, that is directing the player towards the goal of the level and preventing confusion and idling. This can be accomplished by various means.

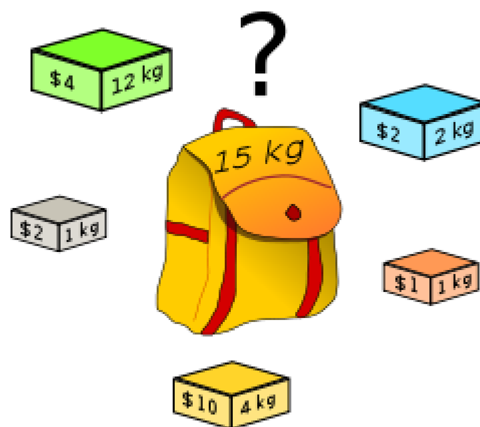
B. Roguelike

The roguelike is a sub-genre of role-playing video games, most often characterized by random level generation and permanent death. The first game to use this concept is called *Rogue*, a 1980 ASCII graphical game, hence the name roguelike.

The gameplay elements characterizing the roguelike genre were explicitly defined at the International Roguelike Development Conference 2008, named the so-called "Berlin Interpretation". Included in the Berlin Interpretation are :

- Roguelike games randomly generate dungeon levels, though they may include static levels as well. Generated layouts typically incorporate rooms connected by corridors, some of which may be preset to a degree (e.g., monster lairs or treasuries). Open areas or natural features, like rivers, may also occur.
- The identity of magical items varies across games. Newly discovered objects only offer a vague physical description that is randomized between games, with purposes and capabilities left unstated. For example, a "bubbly" potion might heal wounds one game, then poison the player character in the next. Items are often subject to alteration, acquiring specific traits, such as a curse, or direct player modification.
- The combat system is turn-based instead of real-time. Gameplay is usually step-based, where player actions are performed serially and take a variable measure of in-game time to complete. Game processes (e.g., monster movement and interaction, progressive effects such as poisoning or starvation) advance based on the passage of time dictated by these actions.
- Most are single-player games. On multi-user systems, leaderboards are often shared between players. Some roguelikes allow traces of former player characters to appear in later game sessions in the form of ghosts or grave markings.
- Roguelikes traditionally implement permadeath. Once a character dies, the player must begin a new game. A "save game" feature will only provide suspension of gameplay and not a limitlessly recoverable state; the stored session is deleted upon resumption or character death.

C. Knapsack Problem



The knapsack problem or rucksack problem is a problem in combinatorial optimization: Given a set of items, each with a mass and a value, determine the number

of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.

There are some variations to the Knapsack Problem, with the most common being the **1-0 Knapsack Problem**. The problem restricts the number of item to zero or one. There are other variations of the Knapsack Problem, such as **Bounded Knapsack Problem, Unbounded Knapsack Problem, and Fractional Knapsack Problem**.

i. 1-0 Knapsack Problem

Mathematically the 0-1-knapsack problem can be formulated as follows:

- Maximize $\sum_{i=1}^{n} v_i x_i$ subject to $\sum_{i=1}^{n} w_i x_i \leqslant W, \qquad x_i \in \{0,1\}$

where n is the number of items, $z_1 - z_n$ are the items, and each z_i has a value of v_i and weight of w_i. The maximum weight of the knapsack is W.

There are several ways to solve this problem. This problem can be solved by Greedy Algorithm (although the result probably won't be optimum), or it can be solved by Dynamic Programming also.

The Greedy variant involves putting the goods into knapsack starting from the most valuable, or from the lightest. The Greedy variant is so unreliable that it can actually be classified as an approximation algorithm.

The Dynamic Programming variant involves a recursive calculation of m[n,W] where the recursive function of m[i,w] is defined as follows :

- $m[i,\,w] = m[i-1,\,w]$ if $w_i > w$ (the new item is more than the current weight limit)
- $m[i,\,w] = \max(m[i-1,\,w],\, m[i-1, w-w_i] + v_i)$ if $w_i \leqslant w$.

This problem can even be solved by Brute Force (as well as a lot of other problems), although it will consumes too much time and very inefficient.

ii. Fractional Knapsack Problem

This problem is a variant to the previously stated 1-0 Knapsack Problem, but this time, the value of the goods can be between 0 and 1 (can be decimals), so this problem can't be solved by Brute Force method, because the possibility is unlimited.

To solve Fractional Knapsack Problem, a Greedy Algorithm alone is enough because it can be solved by a Greedy-By-Density algorithm, where density is the value of the goods divided by the goods' weight.

D. Procedurally Generated

Procedurally generated is a term for a game level designing where the level itself is generated not from a static source, but it's generated on-the-fly instead. The levels in the game are created by means of algorithm. Historically the first games created with this concept are very memory-constrained, and this method is used to solve the memory problem.

The levels in the game are created with the help of Pseudorandom Number Generators. The generated numbers then will be used as a part of an algorithm with predefined seed values to create a very vast game world that appeared premade.

Most of the games nowadays have each of their detailed part of the map designed from the start, so the map of the world is static and of course make it heavily impacted by storyline. The example of these are games from Assassin's Creed series and games from Grand Theft Auto series.

One of the most fenomenal game that used procedural generation until now is Minecraft. Minecraft is well known for its features, including randomly generated world at the start of the game, while still including many aspects of adventures, explorations, and a lot of features that make a game interesting.

Minecraft, one of the best example of a game with procedurally generated world. At the beginning of the game, the world is randomly generated, each tiles, each part of the world is.

E. Pseudorandom Number Generator

A pseudorandom number generator (PRNG), also known as a deterministic random bit generator (DRBG), is an algorithm for generating a sequence of numbers that approximates the properties of random numbers. The sequence is not truly random in that it is completely determined by a relatively small set of initial values, called the PRNG's state, which includes a truly random seed.

Although sequences that are closer to truly random can

be generated using hardware random number generators, pseudorandom numbers are important in practice for their speed in number generation and their reproducibility. And its properties fit the purpose of procedural generation concept.

III. Implementation

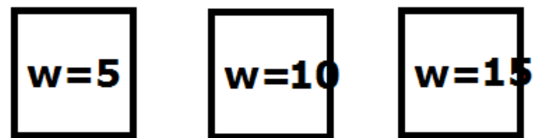
Basically the problem in implementing Knapsack Problem into the level generation problem is the inclusion of the random numbers generated by the Pseudorandom Number Generators and how can the number fit in the Knapsack Problem.

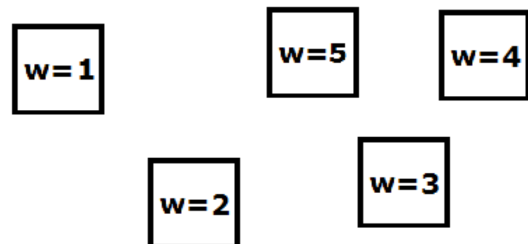
Associating the level generation problem with the Knapsack Problem is not too hard. First of all, we must identify the goods to be put into the bag (the candidates).

The candidates are usually found in the form of fragments of a map, each with their own value and weight. The candidates can also be in the form of enemies, or obstacles. But the main thing is, the value of the candidates are what makes the game full of features, and the weight of the candidates are the contributing points to the difficulty of a level.

For comparison, we must maximize the weight of the candidates so that the difficulty of the level doesn't differ too much. In fact, it is obligatory to make the total weight of the candidates equal to the predetermined weight of the level (that is the weight of the knapsack in the Knapsack Problem).

What makes the Knapsack Problem sometimes differ between the total weight of the goods and the capacity of the bag in the optimum solution is the fact that the weights of the goods differ greatly from one another. Suppose the weights are 5, 10, and 15, then the only possible combinations of total combined weights of the goods are 5, 10, 15, 20, 25, and 30. For knapsack with capacity 27,28, or 29, the result will be the same as a knapsack with a capacity of 25.

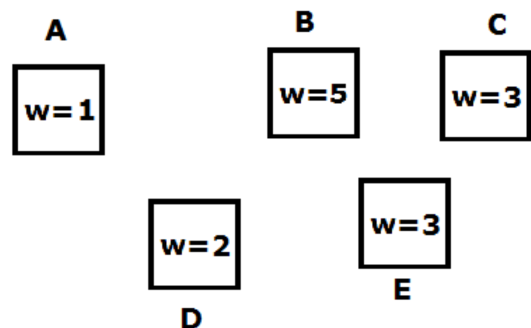




The upper part of the picture contains fewer number of goods, and therefore the total combination is fewer. So basically, the more goods (candidates) there are, the more diverse the combination of goods possible.

Imagine if the goods are aplenty, and their weights are 1,2,3,4,5,6,7,8,9,10, then surely more combination will appear and the potential of each level will be maximized.

But it raises yet another problem. Will all the levels with the same weight similar? If all the goods in the candidates have different weights, then the exact problem will happen. But if there are several candidates with different value and content, but of similar weight, then the problem can be avoided.



With the W capacity of 11 as example, the combination can be ABDE or ABCD even though C and E have equal weight.

The random factor can be implemented now that with a predetermined weight condition already met, there are still several possibilities to be chosen from. Thus the general problem is solved.

Now, there are other factors to be considered, such as in several levels, there are some game fragments that are essential to the level and have to be included no matter

what.

But it's never a problem. We only need to take out these essential fragments from the candidates pool and substract the total weight of the combined essential fragments from the capacity (W). The rest can be calculated normally and the desired result will appear. Of course the random factor would have diminished, but it's the consequences of the inclusion of the essential fragments, not the drawback of using this Knapsack Problem as a solution.

We've talked a lot about the effects of the weight in the level generation, but we haven't talked a single time about the effect of the value of the candidates. Why need value calculation when the weight calculation itself is enough?

In simple examples, we have talked about the algorithm in a very small scale, where the number of goods is countable by fingers. Now imagine if the number of goods is numbered to a hundred, or even thousands. We're going to need another parameter to measure, which is the value of the goods.

Imagine if there are a lot of fragments with the same weight (let's say 20-30 of them). And the fragments are varied by their values, which means the better the value of the fragment, the more fun the game will be.

So basically the value of the candidates serves as a mean to prioritize some fragments over another, just so to make the level more interesting. And it is done through the very definition of the Knapsack problem itself.

But what if some candidates can't be made into hundreds of fragments? It is possible to make a hundred fragments of the map asset containing combination of trees, rivers, soil, etc. but it is very exhausting to make a hundred different enemies that encounterable as a part of the level.

Making a lot of different enemies is very hard, because it will need a lot of drawing, a lot of balance, and many more. It's basically making a lot of new things just to be randomized by the game's system itself. Luckily, most of the times enemies will have levels and other countable parameters, and it works well for our favor.

We can make the enemies differ a lot by applying the Fractional Knapsack Problem. The weight will still be weight, but the value can be changed into the enemies' level or health, or attack point, whichever preferred.

Through the application of Fractional Knapsack Problem, we can find the solution even though there are only a couple of enemies available, because the solution of Fractional Knapsack is always optimum.

The example of a randomly-generated level taken from the 2013 video game Rogue Legacy. The room is generated randomly, as well as the enemies found inside the room. At the top right of the screen there's a map showing the player's current location. The map is generated randomly as well because each room is.

IV. CONCLUSION

Procedurally generated levels in video games is one of many level designs available. It's good to be implemented on small-scale games such as indie games that doesn't care a lot about storyline, but needs a good factor of replayability.

One of the ways to implement it is by the means of Knapsack Problem Application. Included in the Knapsack Problem are the weights and values of the candidates which make the Knapsack Problem similar to the level generation problem.

Here's how it works. Firstly, identify all the candidates, usually in the forms of fragments of the game (such as enemies, obstacles, rooms, etc.). After that, fits as many candidates as possible into the level (the knapsack) while still under the capacity (not overweight). This can be achieved through Greedy Algorithm or Dynamic Programming. The result will be the randomed, balanced level, as a desired result.

REFERENCES

[1] Hatfield, Tom (2013-01-29). "Rise Of The Roguelikes: A Genre Evolves". Gamespy. Retrieved 2013-04-24

[2] Berlin Interpretation (definition of a "Roguelike") from RogueBasin, a Roguelike development wiki

[3] Garey, Michael R.; David S. Johnson (1979). Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman. ISBN 0-7167-1045-5. A6: MP9, pg.247.

[4] Kellerer, Hans; Pferschy, Ulrich; Pisinger, David (2004). Knapsack Problems. Springer. doi:10.1007/978-3-540-24777-7. ISBN 3-540-40286-1. MR 2161720.

[5] Martello, Silvano; Toth, Paolo (1990). Knapsack problems: Algorithms and computer interpretations. Wiley-Interscience. ISBN 0-471-92420-2. MR 1086874.

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, 20 Desember 2013

Muhamad Ihsan (13511049)