Greedy Algorithms Applications in Videogame Itemization

Introduction

MOBA games have become one of the most popular videogame genres, having multiple esports leagues with salaries and popularity levels that rival traditional sports.

One of the key, and most undervalued aspects of the game is itemization, typically, players use tools such as u.gg to obtain statistical data of the game and decide which are the optimal builds for a character.

Another approach is mathematically analyzing each build of interest and testing it using the game's practice tool, while this seems better than only using statistical data it is lacking efficiency, testing multiple builds can take a lot of time which coaches and analysts could be using in other tasks.

The objective of this report is showing how a greedy algorithm applied to League of Legends can be used along with statistical data to optimize itemization for a tank, maximizing the amount of time alive. The first section describes how the relevant items for the model work in the game, the second explains the structure behind the code, the third section discusses possible expansions and applications of the algorithm in the game and the fourth one sums up the conclusions of this project.

Index

Items

In League of legends, each non consumable item has stats, the most relevant of them for this algorithm are the following:

HP: Health Points

Armor: Reduce a percentage of the physical damage according to the formula $\frac{1}{100+Armor}$

Attack Damage: Gives more damage to basic attacks, also, the damage of some abilities scales with it.

Attack Speed: Increases the number of attacks per second of the character.

Lethality: reduces in a certain number the armor of the enemy.

Armor penetration: Reduces the armor of the enemy in percentage.

Damage amplifiers: Increase the damage if a certain condition is met.

Damage Reduction: certain items gave percentual damage reduction, for example, Plated Steelcaps, which reduces in 12% the damage from basic attacks.

Some of these stats increase according to the character level, in the next sections the assumptions in terms of this stats are explained.

For the tank the relevant stats are HP and armor, the others are relevant for the enemy attacking the tank.

There are also 3 relevant types of items for the model:

Mythic items: Only one can be bought, they give different effects to the user depending on the number of legendary items built.

Legendary items: Items not classified as mythic items that neither are boots.

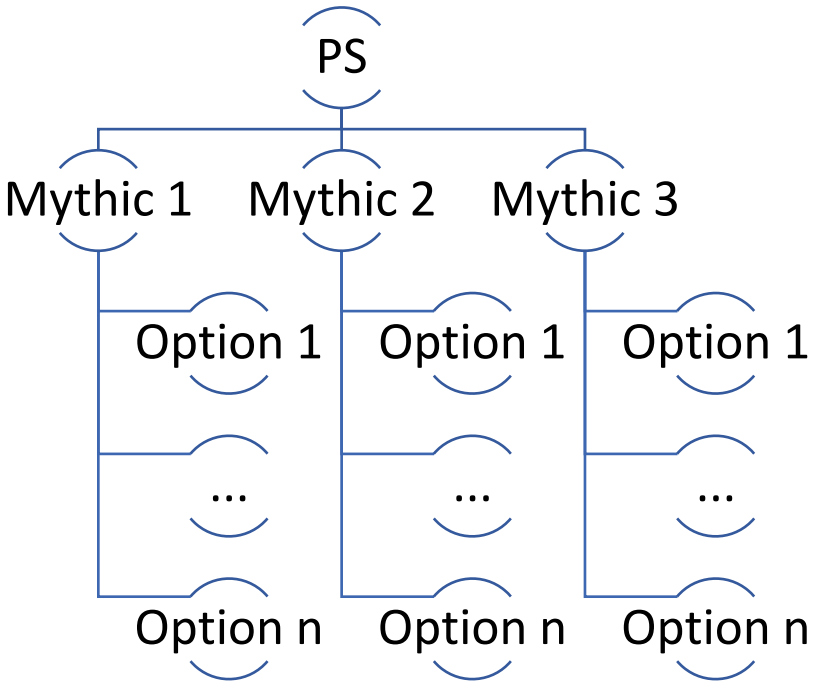
Boots: They give move speed and some additional stats to the character.

The Model
Click Here to see the code.

A few assumptions were made for this model:

- The tank has the average HP and armor for his class at level 13.
- The enemy is an attack damage carry and it has the stats of Aphelios (a character of League of Legends). It only uses basic attacks to damage the tank.
- This "enemy" also has the added stats from the items "Berserker's Greaves", "Immortal Shieldbow" and Lord Dominik's Regards.
- The first item for the tank is always Plated Steelcaps, since it is de facto the best boot against basic attacks.
- For the Mythic item there are three options which are statistically the best performing Mythic items for this class.
- The next item is the one determined using the algorithm.

The structure of the algorithm can be explained using the following tree:



The first node symbolizes Plated Steelcaps, the next three are the Mythic items the player can choose. For each of the nodes of the mythic items there are several nodes, each node symbolizes the subsequent purchase of a legendary item.

In the code, the stats of each item were loaded using a list with the notation [Item name, effective HP, Armor].

```
Mythic = [["Sunfire Aegis",350+50,35], ["Frostfire Gauntlet",350+100,25], ["Turbo Chemtank",350+50 ,25 ]]

#Legendary items, excluding edge cases (sterak's and randuin)
Legendary = [["Chempunk Chainsword",250,0],["Dead Man's Plate",300,45],["Knight's Vow",400,0],["Redemption",200,0],["Warmog's armor",800,0],
            ["Demonic Embrace",450,0],["Titanic Hydra",500,0],["Thornmail",350,60],["Zeke's Convergence",250,25]]

#Sterak's
steraks = ["Sterak's Gage",400,0]
#Randuin's
randuin = ["Randuin's Omen",250,80]
#Frozen Heart
frozen = ["Frozen Heart",0,80]
#Plated Steelcaps Armor
PS_arm = 20
```

For each combination of items, the HP and effective armor are calculated, which makes it easy to calculate the amount of damage per second the enemy does to the tank and the time the tank lives.

```
HP = 1615 + Mythic[i][1] + Legendary[j][1]
#effective armor as (base + items - lethality) * (1-armor pen percentage from LDR)
EffArmor = (83.5 + PS_arm +Mythic[i][2]+Legendary[j][2]-4.5 )*0.65
#damage amp from cut down and dominik (same formula for both)
cd_dr_dmg_increase = ((HP-1494)/1494-0.1)*1/9+0.05
#since aphelios has both it is applied multiplicatively
#0.88 multiplier from plated steelcaps passive
damage_per_second = 277* 1.63 *(1+cd_dr_dmg_increase)*(1+cd_dr_dmg_increase)*(100/(100+EffArmor)) * 0.88
time = HP/damage_per_second
if time > max_non_edge:
  max_non_edge = time
  max_items_non_edge[0] = ["Plated Steelcaps", Mythic[i][0], Legendary[j][0]]
```

If the time the character can tank damage is higher than the previous best combination, the time is saved as a float and the items built as a list.

Each of the items with special effects were treated as edge cases:

<u>Sterak's Cage</u>

This item gives the character a shield that lasts 4 seconds. If the shield is bigger than the amount of damage the ADC does in 4 seconds, the effective HP the item gives to the tank is the damage done by the ADC in those 4 seconds, however, if the shield is less than that damage, it means the effective HP added is the shield itself.

```
shield = 100+0.4*HP
if shield > damage_per_second*4:
  HP = HP + damage_per_second*4
else:
  HP = HP + shield
time = HP/damage_per_second
if time > max_st_edge:
  max_st_edge = time
  max_items_st_edge[0] = ["Plated Steelcaps", Mythic[i][0], steraks[0]]
```

Randuin's Omen

This item reduces the critical strike chance and the attack damage of the enemy ADC for 4 seconds, to model this the damage per second of the ADC while the item is active is calculated. If the tank survives those 4 seconds, the time alive is calculated as 4 seconds plus the time till the rest of the HP goes to 0. Also, it reduces the damage from basic attacks, which is included in the model as the Damage_reduction variable.

```
AA_r_active_dmg = 213*0.9*0.2 + 213*0.9*1.75*0.8
#so the dps while randuin's is active is
dps_active = AA_r_active_dmg * 1.63 *(1+cd_dr_dmg_increase)*(1+cd_dr_dmg_increase)*(100/(100+EffArmor)) * 0.88 - Damage_reduction
time = HP/dps_active
#if the tank lives more than those 4 seconds the time is 4 + time for remaining health
if time>4:
  time = 4 + (HP-dps_active*4)/damage_per_second
if time > max_r_edge:
  max_r_edge = time
  max_items_r_edge[0] = ["Plated Steelcaps", Mythic[i][0], randuin[0]]
```

Frozen Heart

This item reduces the incoming attack damage by a flat amount, it also reduces the attack speed of the ADC. Because of this, the calculations are the ones in the previous edge case, excluding the effect of the active and adding the attack speed reduction.

According to the model, the best item combination is:

```
The best item combination is:
[['Plated Steelcaps', 'Sunfire Aegis', 'Frozen Heart']]
With a time alive of 14.151774536026275 seconds
```

Expanding the Model

There are several ways to expand the model. If the optimal build against an enemy team with hybrid damage is needed, a second enemy can be added to the model, adding the average magic damage per second to the code, thus providing the best build against both times of damage. Against enemies with true damage a flat amount of damage per second can be added to the calculations of the time alive. It can also be a variable if the true damage scales depending on the HP of the tank.

If the gold needed for the optimal build is too high, therefore, delaying the power spike of the tank, it is possible to add a gold restraint to the model, it's also possible to calculate the gold efficiency of each item and adding a restraint for this variable in the model.

A similar algorithm can be used to calculate the optimal build for a damage carry, minimizing the time the enemy character lives, if there are items built because of its active effects, like Zhonya's hourglass, it's possible to "fix" the item in the algorithm, hence calculating only the branches for that respective item.

Conclusions

As shown in this report, the applications of greedy algorithms in itemization are huge, with enough time, it is possible to create a bigger model, for example to get the optimal build against a certain team comp. It's also an extremely efficient process, the algorithm took less than 1 second to output the best item combination, while calculating the 36 item combinations in game would have taken more than 1 hour.

Players making itemization mistakes in professional matches evidences the necessity of having an analyst automating the itemization process, it's also worth noting that teams like Cloud 9 already have an esport data science department, which can help with this task.