Evolutionary Optimization of Item Choices in Risk of Rain 2

Brian Herman (20095996)
CISC 455/851
Dr. Ting Hu

December 13, 2021

Abstract

As computing power increases, so does our ability to concur complex problems. Some of the most difficult problems include constraint satisfaction, planning, and optimization with large sets of variables or data. Generally, depending on the complexity of the problem these may fall under the category of np-complete, meaning that they cannot be solved in polynomial time. These problems require very long run times to solve in a straightforward way. Thus, we have developed a number of powerful algorithms to tackle such complex problems. One of these algorithms uses an evolutionary approach to develop an algorithm suited to the problem, these are called evolutionary algorithms. Evolutionary algorithms or EAs have a set of components that together model our own evolutionary process. These components are: initialization, evaluation, parent selection, offspring generation, mutation, and survivor selection. An individual in an EA may be represented by anything from binary strings to objects that suits the purpose of the program. Initialization is done over a given population size for the given representation of an individual. This is usually done such that each element of the individual is randomized. For certain problems a "smart initialization" could be used where elements of an individual are chosen based on a set of simple principles. Generally, this is unnecessary as EAs can generate reasonable solutions very early in run time. Once initialized the algorithm will move to the evaluation stage. In evaluation, each individual is given a fitness that represents how well it fits the required function. Once each individual has been assigned a fitness, the algorithm will then move to parent selection. In this component the algorithm will generally use the fitness assigned to each individual and choose "well fit" individuals to create offspring. In the offspring generation stage, offspring are created from one or more of the chosen parents. The offspring generation stage introduces large chances into the population, this can create changes for a large number of elements within an individual. Each offspring is mutated in the mutation stage based on its representation, this is generally done by changing a single, or small number of elements. Finally these offspring are evaluated and added back into the population. Finally there is a survivor selection phase in which some individuals will be removed from the population, generally due to age, low fitness, or a combination of both. Overall, we can use our own evolutionary process to model changes over many generations of individuals that can represent any kind of model. This is a very powerful algorithm to solve problems that can not be solved in a conventional way. The model that was solved in this report is a complex optimization task solved using an evolutionary algorithm.

1 Problem Description

Risk of Rain 2 is an action, rouge-like video game. In this game the player moves through stages, in each stage the player will fight re-spawning monsters to get gold which can then be spend to obtain items that increase the players abilities. As time and stages progress the monsters will become stronger and thus the player must be able to obtain enough items that will allow them to survive and progress to the next stage. Items are represented by 5 classes: white, green, red, orange and yellow. White items are the most common, on average the player can obtain a maximum of 5 white items per stage compared to the second most common green items, which the player will obtain on average a maximum of 2 per stage. This problem will focus on the optimization of which white items to choose for a given number of items.

List of white items

- Armor-Piercing Rounds
- Backup Magazine
- Bison Steak
- Crowbar
- Gasoline
- Lens-Maker's Glasses
- Medkit
- Monster Tooth
- Paul's Goat Hoof
- Personal Shield Generator
- Repulsion Armor Plate
- Soldier's Syringe
- Sticky Bomb
- Stun Grenade
- Topaz Brooch
- Repulsion Armor Plate
- Tougher Times
- Tri-tip Dagger

2 EA Design

This section will go over the components of the EA as listed in the abstract.

2.1 Representation

Each individual must represent a collection of items. The program will attempt to optimize the number of each item based on the constraints given. Each individual will be represented by a dictionary of all possible items, each value will be the number of the given item held by the individual.

2.2 Initialization

Using the given constraint, maximum number of items, the initialization will generate individuals up to the population size. Each individual is initialized with a dictionary of each item with values 0. The individual will make the given number of choices from the list of applicable items and for each item, add 1 to the respective value in its dictionary.

2.3 Evaluation

Evaluation is based on three components, item effect, simulation and component weight. The three major components being evaluated are, player damage and survivability, player damage against bosses, and player speed. Fitness is calculated as the sum of these components according to given weights. To understand the evaluation process some game knowledge is required.

Each stage has two main sections. The first section is to fight the re-spawning monsters (once monsters are killed new monsters will replace them). Killing a monster will give gold which is used to gather items from chests that are scattered around the map. The second section is a boss fight. The boss fight consists of fighting against a "Champion" level monster as explained below. Once this boss is defeated the player will be able to activate the teleporter again and be sent to the next stage. There are three classes of monsters "Basic Monsters", "Minibosses", and "Champions". Basic monsters and minibosses may always spawn on the map near the player during both stages. As the difficulty increases the monsters chosen to spawn will become stronger and new monsters with different abilities will be introduced. During the boss fight, one or more champion level monsters will spawn that must be defeated to move to the next stage. This is a very simplified version of the essential game mechanics, more detail will be provided in the discussion.

The first component of the evaluation, the simulation, is done by comparing the damage and protection of the player based on the given items, to a monster. Firstly, all items that can be evaluated using a flat bonus to one of the players stats will be added. This includes items with probability events as an "effective change" in the players stats based on the average change. This is done to eliminate the probability within the fitness evaluation to avoid individuals getting higher fitness based on luck. After all items that effect player stats are added the simulation will begin. In the simulation the player will deal damage based on it's abilities and items to a monster with a set amount of health to evaluate the players damage. The monster will also periodically deal damage to the player to evaluate the players defence. The return from this simulation is the amount of time required for the player to kill the monster (get the monster health to 0). If the player dies during the simulation the return is the maximum amount of time (sys.maxsize). This component is used to test the change in players damage and survivability when compared to a single enemy.

The second component of the evaluation is based on the player's boss damage multiplier. The boss damage multiplier is based on a single item "Armor-Piercing Round". Other than this item the boss damage is the same as the regular damage. Based on the importance of the player's extra damage dealt to bosses, the importance of this item is quite low. Thus the return from this component is simply the number of this item held by the player. This will result in an evaluation that heavily relies on the players regular damage yet if a maximum can be found, will still be able to increase fitness by increasing boss damage.

The final component is the player's speed. Unfortunately speed was not able to be quantified as a means of defense in the simulation stage. In reality the player speed has two benefits. Increases in player speed allow for chests to be found faster by allowing the player to search the map faster. Overall this decreases the amount of time required per stage which means the player will be getting the same amount of items while the difficulty increases by a lower amount (since the time spent is less and difficulty is a factor of time). The player speed also allows for the ability to move away from incoming attacks and thus negate taking damage. Because of this, in the game speed is also a component of the players survivability. To take this into account the player speed is return from this component.

The final fitness if the sum of, the inverse time required returned by the first component, the boss damage multiplier returned by the second component, and 2 * the player speed returned by the third component. The coefficients of -1, 1, and 2 were found through testing and the relative importance of each component in the game.

2.4 Parent Selection

Parent selection is done by tournament selection with n=2 parents selected per loop. For each parent, the tournament will select a given number of individuals from the population such that each individual may be chosen more than once. From the tournament, the parent chosen will be the individual within the tournament selection with the highest fitness. Thus, 1 parent will be chosen per tournament and the resulting selection will be the winners of each tournament.

2.5 Crossover

Representation is an unordered dictionary. Thus crossover done, based on a given crossover rate, by individual element swap between two parents with a given swap rate. Each of two offspring is initialized as a copy of one respective parent. Then the crossover will go through

each offspring and for each item based on the swap rate given, either leave the value as the initial value from one parent or swap to the value from the second parent.

2.6 Mutation

Mutation is done by selecting two elements from an individual such that the first element chosen will be removed and the second element will be added. This is equivalent to a swap between items. Thus, the first element chosen must not have a value of 0 (there must be an element to swap).

2.7 Survivor Selection

Survivor selection is done my least fit removal from mu + lambda population. Offspring are added into the population then, based on the number of offspring, an equivalent number of least fit individuals are removed from the population.

3 EA Results

3.1 Output

Single run output final best solution:
{'Armor-Piercing Rounds': 0, 'Backup Magazine': 0, 'Bison Steak': 0, 'Crowbar': 1, 'Energy Drink': 8, 'Gasoline': 0, "Lens-Maker's Glasses": 10, 'Medkit': 0, 'Monster Tooth': 0, "Paul's Goat Hoof": 0, 'Personal Shield Generator': 0, 'Repulsion Armor Plate': 6, "Soldier's Syringe": 21, 'Sticky Bomb': 0, 'Stun Grenade': 3, 'Topaz Brooch': 0, 'Tougher Times': 0, 'Tri-tip Dagger': 1}
fitness: -94.9

3.2 Discussion

The population with a set seed for random number generation will converge quickly (around 5000 of 10000 loops) after this small changes (below 1.0 difference) may still occur. Some runs with no random number seed will never find good solutions though this is less likely than converging to a single individual. For the purposes of this Project a single solution is a good result for optimization with the number of items given. If the items increases, for example adding green, red, orange, and yellow items, then there will likely be far more local maxima within the fitness range based on certain combinations of items. For an example of this, when the damage of the enemy increases in the evaluation, instead of getting items that heal the player instead the player will get more "Stun Grenades". Stun grenades have a chance to stun the enemy for 2 seconds every time the player lands a hit. The player also has a very large number of "Soldier's Syringe" which increases attack speed. Therefore the optimal solution is not to heal, but to rely on the high attack speed, meaning many hits per second, and the stun grenades in order to decrease the amount of times the player will be hit.

3.3 Full Explanation/Optimal Playstyle

Based on the most fit individual we can draw a number of conclusions about the use of each item.

- Armor-Piercing Rounds: Clearly with a value of 0, the weight of increasing boss damage is not very important, overall the player chooses to focus on regular damage
- Backup Magazine: The player has three abilities, primary ability can always be used, secondary ability can be used with a cooldown between uses, and similarly the "special ability" has a cooldown between uses. The backup magazine provides the player with an extra use of their secondary ability. Note that each individual use has its own cooldown, for example if the cooldown is 3 seconds and the player has 2 backup magazines, then after 3 seconds the player can use one secondary ability, after 6 seconds the player can use 2. Based on the output, it seems that the use of the secondary ability is not a major factor in total damage output. Thus it is better to focus on overall damage or the primary ability
- Bison Steak: the bison steak increases health by 25, for comparison the base health of the player is 110. Based on the output the use of bison steak is outweighed by the use of other defensive items such as "Repulsion Armor Plate" as seen below.

- Crowbar: the crowbar deals 75% more damage to enemies with 90% or higher of their maximum health. The use of a single crowbar indicates that there is a benefit to this increase but likely due to the restriction of the enemy having 90%+ health it is not beneficial to continue stacking past 1.

- Energy Drink: The player has two modes of travel, running and sprinting, with sprinting being significantly faster than running with base speeds for both. Energy drink increases the player's sprinting speed by 25%. The player should generally be sprinting around 80% - 90% of the time and thus a coefficient of 0.8 was included when applying the overall increase in speed. When compared to "Paul's Goat Hoof" which increases the players overall speed (running and sprinting) by 14% it seems that it is still always better to choose an energy drink.

- Gasoline: unfortunately the use of gasoline is purely for fighting multiple enemies and therefore a value of 0 makes sense with the current evaluation method.

- Lens-Maker's Glasses: these glasses increase the chance of a "critical strike", which doubles damage, by 10%. Clearly once the player has 10 of these, their "critical rate" is 100% and thus all attacks will be double damage. Therefore a value of 10 indicates that getting the maximum useful amount of this item is optimal.

- Medkit: The medkit heals the player by 20 + (5% per medkit) 2 seconds after taking a hit. Once again we see that the optimal method uses other defensive items.

- Monster Tooth: this is another item that cannot be evaluated with the current method as it requires a monster to die for the effect to occur.

- Paul's Goat Hoof: as described above this increases the overall speed of the player. Since the player is generally sprinting most of the time. It is more efficient to get energy drinks.

- Personal Shield Generator: this item gives the player "armor" based on their maximum health. Armor decreases the amount of incoming damage by the product of 100 / (100 + (amount of armor)). Once again the player opts for different defensive items

- Repulsion plate armor: this seems to be one of the main sources of defence for the player. Given the damage done by the enemy, repulsion plate armor decreases the damage by a flat 5 * number of item. Clearly based on the amount of damage the enemy is doing, this is a good item for defense. Note that in a real game the enemy damage will increase over time and this item becomes less and less effective.

- Soldier's Syringe: This seems to be the best item taking 21/50 slots. Soldier's syringes increase the player's attack speed by 15%. Attack speed increase, will increase the number of primary attacks the player can use within a given time (base attack speed is 6 shots per second). Clearly the main source of damage for the player is the primary ability and soldiers syringe is one of the best ways to increase overall damage.

- Sticky Bomb: sticky bomb gives a small chance to attach a bomb when hit that deals additional damage. Based on the value of 0, it is less efficient to get a sticky bomb compared to other items.

- Stun Grenade: stun grenade gives a chance per hit to stun the enemy for 2s. Coupled with the high attack speed from soldier's syringes it seems the optimal defensive strategy is to hold a few of these and stun enemies rather than heal from their damage.

- Topaz Brooch: once again this item requires killing a monster to activate making it non-effective in the evaluation

- Tougher Times: this gives a chance to block incoming attacks based on the hyperbolic equation 1 / (1 + 0.15 * number of tougher times). The final example of a defensive item that is not used

- Tri-Tip Dagger: this item gives a 10% chance to cause bleed on the enemy per hit causing damage over time. Interestingly the optimal number of these is only 1, it seems that this is an effective item but after a single stack, other items provide better average damage increase.

4 Comparison

4.1 Problems With Comparisons

Finding similar algorithms or programs to compare this to is quite difficult due to the difficulties with evaluating such a large number of items with such a complex evaluation method. Generally optimal strategies/items are found over the course of long periods of time through analysis of each item's stacking type (linear or hyperbolic). The accepted method is simply "theory crafting" in which experienced players will speculate and test different theories for viability. However effective this is at finding the best sets of items, it will likely never find the optimal values. The largest problem with evaluating item sets through algorithms is the inability to recreate game like situations for simulation of effectiveness. The best algorithms are simply a combination of evaluating the single item's effectiveness compared to other items, and the rate at which this is surpassed. This method will only work for a subsection of the total items, as calculating the effective difference of some items is situational. For example, one item excluded from this project is the "Focus Crystal". This item increases damage linearly by 20% per stack for enemies within 13m of the player. Calculating the effective damage increase of this would require you to be able to determine the average amount of time the player can spend within 13m of an enemy. This is based on a large number of factors such as: number of other enemies, increased danger with proximity to enemies, distance at which the monster spawns from the player. All these factors make it nearly impossible to find the average time spent within a proximity of 13m, making it impossible to calculate the effective damage increase of this item. Alternatively for example one can easily calculate the effective damage of an item such as "Lens-Makers Glasses" which increases crit rate by 10% per stack up to 100%. The damage increase of a critical strike is double normal damage. Therefore, effective increase in damage would be damage + (0.1 * number of glasses) * damage.

4.2 Method

The most obvious method is simply to take into account all the values of efficient changes for each item based on the number of stacks and stack type. Doing this, a value can be placed on each item based on category, attack or defense. The simplest item would be a Soldier's Syringe. This is because the % increase in damage can easily be calculated and does not require any probabilistic averages and the stack type is linear meaning the value of getting this item is always the same. This gives a value of damage increase of (damage per second by primary attack) * 1.15 for soldier's syringe. The value of this is constant for each stack as the stack type is linear. Alternatively, items with stack type of hyperbolic will be compared to find the stack count at which the increase diminishes. Using this calculation the value of an item such as Tougher Times is 22 before no stacks should be gained. The problem with this method is that it does not tell you necessarily what items work well together, just how good each item is individually. However, due to the multiplicative nature of damage increase (an increase of primary attack damage by double would increase the effectiveness of attack speed) all combinations of items can be compared to find the combinations with the highest overall increase. This allows you to see which items have the optimal combinations and the optimal strategy would be to reach the effective number of stacks for all hyperbolic items and then stack the single linear item with the highest score. Overall this method is generally the only approach used besides the initial "theory crafting".

5 Discussion

This section will discuss both the viability of an evolutionary algorithm as a solution to this problem as well as the problems found when implementing an algorithm.

5.1 Viability of Evolutionary Approach

Overall this problem is quite viable for an evolutionary approach. There is no good way to find optimal solutions using conventional methods given the complexity of the connections. Each item has value on its own, value when compared to other items, and value based on the number of stacks that effects both. For simple example from above, the effectiveness of stun grenades increases when attack speed increases. Overall this problem is quite complex to solve using conventional computation methods. This is the main reason that theory crafting has become such a large component of the game. Many players simply test out new theories in every game. Overall an evolutionary algorithm would likely be the best approach to finding optimal strategies. A final positive note, due to the nature of evolutionary algorithms to find local maxima, as an experiment one may use this fact to find different strategies and

play styles. At the end of the day, games are meant for fun and trying out lots of good strategies provides some diversity to the game play.

5.2 Problems With Implementation

Above all the positives of an evolutionary approach looms the ever complex problem of evaluation. At the beginning of this project I had assumed that a majority of time would be spent on the evaluation. The initial idea was to simulate entire games. The method was to have individuals represented by an ordered list of items. As the simulation continues the player would gain items based on the individual's list, where the fitness would be the number of stages the player can complete. Unknown to me at the start, the method that the game uses to spawn both interactable items and monsters is incredibly complex and not very well documented. Most methods are explained on the game's wiki however the complexity of the description for each component varies greatly. To understand many components I had to ask members of the game's community who have been working with the code for years. Eventually I came to a basic understanding of the way that this was done. Unfortunately, understanding the methods was not the same as being able to implement them without matching the convolution of the original implementation. Completing this simulation would likely have taken longer than the project time period. Once realizing this I had decided to instead simulate the average interactable and monster spawns per stage over time. Doing this would give me two things, the average interactable spawns which tell me how much gold the player requires to open the chests, and the average health and damage of the monsters as well as the amount of gold they will give. Knowing this I would be able to use the current damage and health stats for the player to determine the amount of time it would take to kill enough monsters to open all the chests. After this I would have to determine the amount of time the player would require for the boss fight. As you can tell this is getting more and more convoluted because at this point I would require the equivalent to an entire small version of the game to run in efficient time and an AI algorithm that would be able to deal and take damage based on the environment. It was at this point, about 45 hours into coding that I realized this was not actually possible, and if it was it would take far longer than I have (some of this code was left in the commented out sections in both globals.py and evaluation.py to show the complexity of the problem, please do not attempt to use any of this code).

The next step was to simplify. Coming to the conclusion that my evaluation method would not be plausible a new method needed to be developed. This method would greatly simplify the evaluation into it's major components. These were: single target damage, multi target damage, speed, and boss damage. Each component would need to be evaluated and added to the fitness by some weight coefficient. For example, in a real game multi target damage is the most important as it allows you to deal more damage overall to the most amount of targets. Following this, speed is the second most important as it allows you to complete stages faster which positively effects the player level to enemy level ratio. The method of evaluation for single target damage would be as it is in the final implementation. The player would be set against a set incoming damage and a large health bank. The player would be required to stay alive and get the enemy health to 0 in the shortest amount of time possible. The multi target evaluation would set enemies at varied distances from each other, the player would kill the enemies one by one from closest to farthest allowing for the damage to crossover from one enemy to the others within the area of effect for the attacks. To explain this a little further, some items will deal damage to enemies within a certain range of a killed enemy. The third evaluation would be very similar to the first, single target evaluation. The only difference in this evaluation is the inclusion of armor-piercing rounds. This item is the only one that effects bosses only, however the increase in damage is substantial (20% per stack linearly). Finally the speed of the character would be evaluated as a flat number. Each of these four evaluations would be multiplied by coefficients, accounting for effective change in overall performance, and summed. Now we come to the final problem, the multi-target damage evaluation has a handful of non-calculable factors. This includes, the incoming damage from multiple enemies over range and the ability to dodge damage based on the number of enemies. This is a factor that is directly affected by the player's mechanical skill (how good they are at the game) and any attempt to recreate this with a static program becomes impossible as the average incoming damage is not calculable.

Finally I find myself at the ultimate disappointment in coding an optimization problem. That being, any optimal solution will not suit the real problem, only the simulated problem. At this point I am forced to accept that the solutions found will not be applicable. Worse than this, I only had less than 2 days left to implement the evaluation method. Faced with this, it was decided to entirely scrap the multi-target damage evaluation and continue with a simplified version of the boss evaluation method. In the end the algorithm is quite reasonable for the simulation provided. It is able to find an optimal solution for

most runs, given a random seed of 10 it will always find this solution as a large subset of the population. Clearly I would have preferred to get an optimal and applicable solution but given the complexity of the evaluation I would have to settle. Overall I learned far more about these algorithms than can be shown in the final algorithm. This project was a great opportunity to face an incredibly difficult task that required research and understanding of the implementation of another parties codes. I was fortunate that the initial problem was able to be converted into a possible algorithm. At the end of the day I would not choose to do another project as the problems that I faced were somewhat new, or at least this level of complexity of a problem. It has also taught me that I do not want to work with such convoluted evaluation methods again with time constraints.

In conclusion, the algorithm had many faults and required multiple revisions teaching something new at every step. The final version of the code is efficient at determining an optimal set of values that fits the evaluation method. Unfortunately the project has only taught me about evolutionary algorithms and simulations and not provided me with the optimal strategy to be good at games.