# Balanced Map Generation using Genetic Algorithms in the Siphon Board-game

Jonas Juhl Nielsen and Marco Scirea

Maersk Mc-Kinney Moller Institute, University of Southern Denmark,
msc@mmmi.sdu.dk

**Abstract.** This paper describes an evolutionary system for the generation of balance maps for board games. The system is designed to work with the original game Siphon, but works as a proof of concept for the usage of such systems to create maps for other board games as well. Four heuristics and a constraint, developed in collaboration with the game designer, are used to evaluate the generated boards, by analyzing properties such as: symmetry, distribution of resources, and points of interest. We show how the system is able to create diverse maps that are able to display balanced qualities.

## 1 Introduction

Board games are increasing in popularity, and successfully publishing a board game is becoming increasingly challenging [1]. Hence it is important to have a solid game design, so that this specific game will stand out of the other thousands that are developed. An important game design element in board games is balance. If a game is completely unbalanced, it gives an almost guaranteed victory for the same player every time, which makes it unpleasant to play. In a balanced game the initial player situation (e.g. the starting country in Risk) should not ensure victory or defeat. As an example of an advantage a player might have, consider having the first move, a balanced game should allow the player skills (and sometimes luck) to determine the progression of the game regardless of such advantages. This problem relates to all board games and is consequently an area that needs focus. Board games are becoming more and more advanced, which makes it complex to ensure balance. The incomplete information, randomness, large search space and branching factor makes it almost certain that the designer will miss substantial balance elements. When introducing procedural elements to the game – as the multiple scenarios for victory and dynamic map of Betrayal at House on the Hill – it becomes even harder to control the scope of all possible actions that the players can take. Artificial intelligence (AI) techniques can be employed to explore the game space and provide evidence that a certain rule in the game must be removed or revised. To acquire the balanced experience, play testing is essential. After the AI has generated various setups for the game, they can be tested by human players to evaluate if it still feels fun to play. [2]

This paper highlights the exploration of how to make a board game balanced using artificial intelligence techniques, specifically with procedural content generation (PCG) applied to board generation using a genetic algorithm (GA). This will be applied on a self-developed board game called Siphon. This study is based on a limited version of Siphon and concern itself with balancing game maps. In the future the project will consider other game play elements that also affect game balance. The simplified version of the game will be described in section 3. The genetic algorithm uses several heuristics in the fitness functions that describe different desirable features to create interesting and balanced maps.

## 2 Background

Video Games drags our attention and makes us able to sit in front of the screen for hours and hours, just to progress a virtual character in games like World of Warcraft. Some board games and card games can inspire our strategic minds, to find optimized ways to beat friends and family in our most favorite games. Sometimes we are killing the greater evil together to survive in the darkness. What is it with all these games that drags our attention? A part of the answer can be simplified to "A great game design". [3]

This project introduces the game design element "balance", with the focus at board games. To accomplish balance in board games, it is important first to locate the areas, which can be unbalanced and then figure out a plan to make it balanced and keep the fun in playing the game. By introducing AI in game design, it provides the opportunity to explore the game space much quicker than a human being would ever be able to do. They can be used to discover unbalanced strategies or even rules that are not covered in the rule-book.

### 2.1 Designing balance

Game Design is a huge area of study, which is about figuring the tools of play, the rules, the story plot and line, the possible strategies, etc. As this project focuses at the balance aspect of game design, we will look at internal, external and positional balance, where some good balanced games will be used as examples, to understand what a good design is.[4]

One of the primary concerns of a game designer is to generate a game with long replay ability which is caused by e.g. varied experiences and removing unfairness. Internal balance has the focus of eliminating false decisions and regulating dominant strategies. Eliminating false decisions means that every action that has been chosen can lead to a high scoring weight when combined with other actions, which means that they are all valid for the chosen strategy. This means that no choices are considered as a false decision. If a game implements false decision making, the experienced players will have a huge advantage because of his/her knowledge of the game and not the ability of outsmarting the other. The board game Stratego uses internal balance, which introduces the false decision making. It uses two frameworks that are made to equalize the output of different

game choices. Specifically, Stratego uses the intransitive relationship framework which means that there is always a counter (as also in Rock-Paper-Scissors) Stratego uses ranks to determine who beats who. The higher ranks beat the lower ranks with one exception: The highest rank is beaten by the lowest rank.

Another framework is the transitive relationship, which uses a cost-benefit curve to compare objects. If the benefit is higher than the cost, the object will be categorized as overpowered and if it is lower as under-powered. An example of a game that uses this, is Magic the Gathering. It is a card game, where each card has a cost to use and has an effect when it is used. Ticket to Ride is also an example of a game that is very popular and uses the transitive framework. Players are rewarded for selecting and completing tickets. The harder the ticket is, the more rewarding it is to complete.

Another kind of balancing is the external balance. This is the primary focus of this project, since it encompasses the designers' choices of how the game is built (e.g. being the starting player or the positions of resources gives a player huge advantage). Symmetry in games are often appealing, since humans feel comfortable by seeking patterns. It is one of the easiest ways of making balance, since it gives the same position and options for each player. The reason not to make everything symmetric is that it can appear less interesting and subtle imbalance can easily be overseen. An example of a mirror symmetric game is Chess. The Fritz database, which contains more than eight million chess games, shows that the average score is 0.55 points, where 1 is the white winning, 0 is loosing and 0.5 is a draw. This shows that the starting player, using the white pieces, has an advantage in chess. Asymmetrical game play often provides more interesting scenarios, but it is harder to make balanced. Also, it might be harder for new players to discover a good action in a given situation, which will widen the skill level between newcomers and experienced players. The art is finding a combination between the symmetry and asymmetry to have an interesting but clear design for all kind of players and still have a balanced game.

### 2.2 Using AI for balance

A previous study [5] uses AI-based play testing on the board game Ticket to Ride, to detect loop holes and unbalanced strategies. The AIs plays the game using different evolutionary algorithms and evaluating each play through. By playing the game thousands of times with different maps, some strategies will be evaluated high. They might as well discover failure cases where the agents found game states that are not covered in the rules. These are the areas in the game design that needs focus. A high evaluation might be close to an optimal strategy, which might be overpowered. Evolutionary algorithm is the super-set of the genetic algorithm, which is used for Siphon (see section 4).

The popular board game Settlers of Catan has multiple options of setting up the game board when playing. One is randomization which can provide very


**Fig. 1.** Example of tiles and heroes cards in Siphon. Each tile can have up to 6 runes which can be used by heroes standing on them to activate abilities.

unbalanced boards for some players. Another way is following the setup that is displayed in the rule-book.[6]A third way is using "BetterSettlers".[1]

Civilization is a famous video game, which uses random map generation every time you play a new custom map. A study [7] describes how Civilization uses a real world map to discover types of environment by color, to gain the types of resources. After this the information from the map, (latitude and longitude) are used to generate the maps. At last genetic algorithms calculates the initial position, where the fitness function takes fairness and exploration into account.

Another study [8] uses a GA for generating new board games, by using simple and existing ones: Checkers, Tic-Tac-Toe and Reversi. Finding the game elements that they all have in common, creates the search-space for the algorithm and makes it able to generate 144 new games. Adding some nontraditional game elements, increases the search-space to 5616 games. After generating the games, they are played hundreds of times with a simulator. The games are then evaluated with the fitness function based on diversity and balance. By doing this, new and balanced board games are created, where the rules are defined by the AI. This is an example of how powerful GAs can be.

## 3 Game Design of Siphon

In this report, a simplified version of a self-developed board game called Siphon are used to demonstrate how GAs can be used to balance a board game.

Siphon is a competitive two-player game. Each player has 5 heroes, which they have picked in turns from a hero pool. The heroes are summoned in an arena which is a board consisting of 37 hexagon tiles, placed as a rectangle. A tile has up to four different rune types and can have up to six of these in any

[1] BetterSettlers (www.bettersettlers.com), uses an algorithm that provides the fairest distribution of starting setup


**Fig. 2.** The hero cards can rotate over a tile, and need to be align the correct runes to activate abilities.

combination. The runes count as resources. The system creates balanced boards by deciding where to place these runes. See Figure 1.

A hero has various abilities and has two split runes on each side of the card. These are the same types that can appear on the tiles. To cast abilities, it is a requirement that the runes appearing on the abilities are met. See Figure 1. This means that the split runes displayed on the hero cards, must combine a full rune on the board tiles, by rotating the hero card. See the rotating hero card on Figure 2. Since the heroes can both move around on the tiles and rotate, and each hero are different, the branching factor is huge since there is many different actions available. It would be very difficult for a designer to find a board that is balanced just by trial-and-error, and the odds of finding a balanced board are minimal. As such the game present a good framework for the usage of AI techniques for balancing.

## 4 Methods

The algorithm that we employed to balance Siphon is a genetic algorithm (GA), which is a subset of evolutionary algorithms. [9] GAs are inspired by evolution theory, which is the process of natural selection and survival of the fittest. These algorithms are used to solve problems where"brute force" algorithms would take too long. GAs generate solutions to optimize and search for problems that uses bio-inspired operators. These operators are ***selection***, ***mutation*** and ***crossover***.

Before going in depth with the before-mentioned operators, Darwinian evolution theory will be explained[10]. He uses three elements that describes the natural selection that occurs in our nature.

**I.Heredity** - The children of an evolution need to receive the properties of their parents. If a creature reproduces, it needs to pass down its traits to the next generation.

**II.Variation** - The traits that are present in the population need to vary, so that all creatures in it reproduce a variety of traits. If this was not the case, all children will possess the same genome as their ancestors.

**III.Selection** - The fittest creatures of a population need to be able to pass down their genes, so that the stronger survives. This is the evolution and is commonly referred as "survival of the fittest". A fit creature does not necessarily mean "physically fit" but is based on its likelihood of being good at a specific task. The selection operator uses this principle.

In GAs the creatures of a population are often referred to as "individuals", where the traits of the creature are the "genes". In practice, the individuals are the solutions we want to assess and the genes are the characteristics that define a solution. From the Darwinian principle of variation populations are usually initialized randomly. After creation, each individual of the population is evaluated by using an evaluation (fitness) function. The fitness values are then used by the ***selection operator***. The operator is used to determine which individuals are going to be chosen to become the parents for the next generation. In this implementation all individuals of a population have been evaluated the most fit individuals are placed in a mating pool. For the reproduction step, a simple ***one-point crossover operator*** is used. This crossover operator consists of: i) choosing two random individuals from the mating pool (parents), ii) picking a random index to split the genomes, iii) creating two new individuals (offspring) by combining the first part of the genome of one parent with the second part of the other (and vice-versa). After the crossover step, it is usually necessary to use a ***mutation operator***. This operator is used to introduce random changes in the genes of the offspring, that would otherwise have only a combination of the genes of its parents. The mutation operator can be seen as a random local search mechanism, as it should never result in very big variations in the genome. As GAs contain many stochastic elements to the optimization process, there is a chance for evolution to get stuck in a local optimum, mutation used to limit this shortcoming of the algorithm family. An example of when this problem might appear occurs when the starting population does not have the genes that are required to get to the optimal fitness score. With mutation, it is possible to change a gene that is not a part of the ancestors and access the genes that allow for the possibility of acquiring the optimal fitness score.

### 4.1 Fitness function

When creating a population, all individuals have to be evaluated in order to determine the "fitness" of the solutions they represent. Fitness functions are usually very domain-dependent and are in fact the driving force of the evolutionary process. z These heuristics have been defined in collaboration with the designers of the game (one of which is an author of this paper). One constraint and four heuristics have been defined. The constraint has to be fulfilled before an individual is evaluated on the other heuristics. In practice, that means that all generated boards **have to** satisfy the constraint, but might satisfy in different

amounts (possibly not optimally) the other heuristics. The fitness scoring are as follows:

$$f(sol) = \begin{cases} Constr(sol) & \text{if } Constr(sol) < 1 \\ w_1 Even(sol) + w_2 Symm(sol) + & \\ \quad w_3 Rune(sol) + w_4 Mono(sol) & \text{if } Constr(sol) = 1 \end{cases}$$

The possible fitness values are between $[0, 2]$. While the constraint has not been met this value is restricted between $[0, 1]$, effectively forcing evolution to first create individuals that satisfy constraints. Each of the other four heuristics is assigned a weight. A heuristic with a higher weight will be prioritized over the others. In Siphon, an individual is a complete board setup and the genes are the tiles.

**The constraint heuristic** ($Constr$) is based on the sums for each type of rune that is allocated on the hero cards. The percentage of these sums are then calculated from the total amount of runes on the hero cards. An individual (a board setup) is then evaluated with a higher score the closer it gets to these percentages by placing the runes on the tiles.

**The even-distribution heuristic** ($Even$) is evaluating how evenly distributed runes are all over the board (e.g. a board with x tiles and x runes of a type has 1 of these runes on each tile).

**The symmetry heuristic** ($Symm$) is based on symmetry, as that is often a quality that is desirable for balance. The designer felt it was too static to aim for complete mirror symmetry (which is also a trivial problem that does not require evolution to solve), so it was decided that a potentially imperfect diagonal symmetry was more desirable. With the weight parameter, it is possible to adjust the amount of symmetry, since the GA usually does not get a perfect score in this metric.

**The rune-count heuristic** ($Rune$) is evaluating that the number of runes on the tiles are below four, with the exception of the the center tiles, which have six. This heuristic is used to create dominant tiles, which generates high risk - high reward areas on the board.

**The mono-rune heuristic** ($Mono$) is aiming toward having tiles with the same rune type. This heuristic's purpose is to create interesting territories (biomes) which might high rewards, so that the players have an incentive to want to capture it, which generates more action in the game.

Considering all the different heuristics that are part of the fitness function, individuals might never get to a perfect score, since satisfying a condition might break another. Nonetheless, the GA is able to create boards that have a close to perfect (if not optimal) score. These individuals are still valid candidates for a balanced board, and can be play tested by human players and compared to evaluate, which one that should be the chosen one. This might require many candidates, but the GA are able to decrease the number of candidates significantly. Figure 3 shows a board generated by the GA, with a fitness score of

1.92. This can be compared with other generated boards and get play tested by human players.

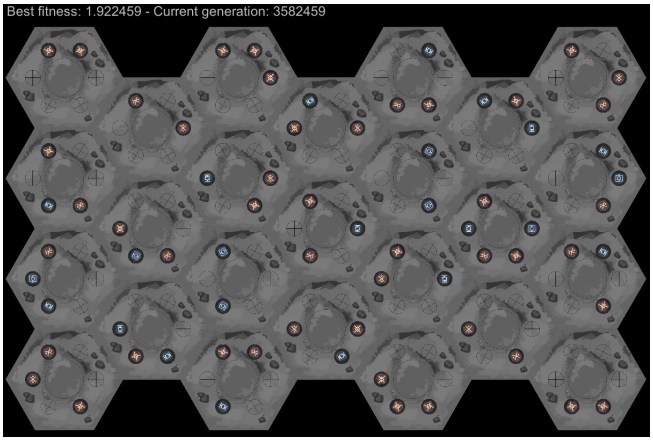


**Fig. 3.** An example of a generated map with fitness quite close to the optimal value (1.92).

## 5 Analysis

There are many parameters of the GA that can be adjusted. These include: the population size, the mutation rate, heuristic weights, and heuristic-specific parameters. By finding the right adjustment, it might be possible to get closer to optimal solutions, but finding these can be difficult and require a large amount of testing. In this section, the results from testing the GA used in Siphon are displayed and described.

### 5.1 Effect of population size

Figure 4 and 4 display the best fitness and the average fitness in a population over generations for two data sets. The results will be used to analyze what to optimize and to see after how many generations the GA stops finding new solutions. The only parameter that will be changed is the population size, which are displayed on the graphs. The static parameters are set to: Mutation Rate: 0.2, Heuristic Weights ($w_1, w_2, w_3, w_4$): 0.25, 0.25, 0.25, 0.25

The best fitness curve are the same for both data sets and reach a plateau around 7-8000 generations. The two average fitness from the data sets are instead quite different. The data set with the population parameter set to 400, rarely satisfies the constraints and therefore has an average fitness lower than 1 (the score indicating the constraint satisfaction). The other data-set rises after there

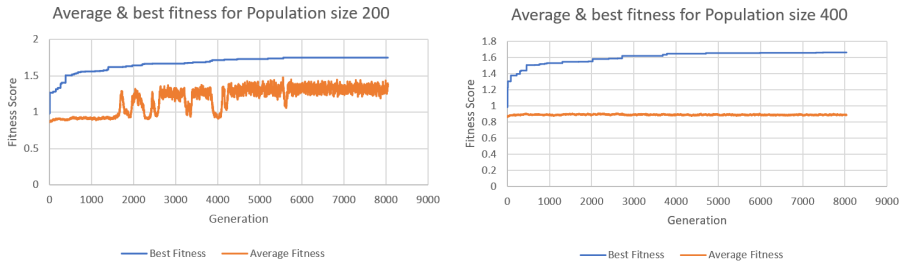


**Fig. 4.** Best and average fitness in the setup with 200 and 400 individuals in the population

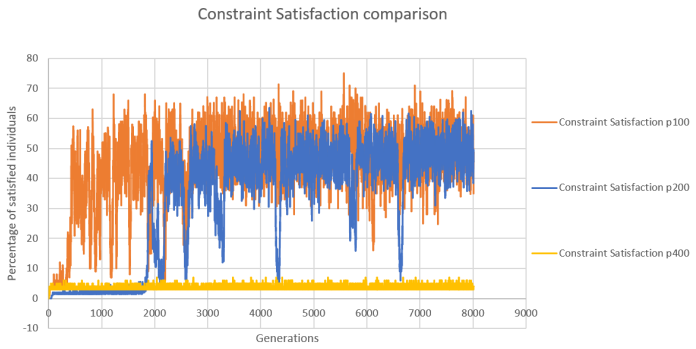


**Fig. 5.** Amount of individuals that satisfy the constraint as evolution progresses.

are enough individuals that satisfy the constraint in the mating pool. That is why there is a delay of the average fitness.

Figure 5 shows the amount of individuals in a population which satisfy the constraint over generations. Three data sets are used with same static parameters previously defined. The population parameter are set to 100, 200 and 400 for the data sets. It appears that for this problem a relatively low population size (around 150) makes it easier for the GA to satisfy the constraint (see Table 1). When the mating pool starts having individuals which satisfy the constraint, it will be move likely that following generations will as well.

### 5.2 Individual heuristic effect

A population size of 150 will be a static parameter for the next data set. As previously tested, the fitness score rarely gets above $1,8$. This could be caused by the heuristics working against each other, or an optimal solution might not exist. Table 2 compares the maximum fitness scores found by running evolution when removing each heuristic one at a time. The ones that are still active are assigned identical weights.

**Table 1.** Population Size Effect

| **Population Size** | **Fitness score** |
|---|---|
| 100 | 1.72 |
| 150 | 1.755 |
| 200 | 1.75 |
| 400 | 1.655 |
| 800 | 1.58 |

**Table 2.** Heuristic Effect

| **Parameters** | **Values** |
|---|---|
| Population Size | 150 |
| Mutation | 0.2 |
| Even-Distribution | 20 |
| Symmetry | 10 |
| Rune-count | 40 |
| Mono-rune | 30 |

By removing single heuristics one by one and evaluating the scores we wanted to investigate which heuristic are most difficult to satisfy. As the table shows, the symmetry heuristic are the most difficult, since the fitness function evaluates the score highest when it is removed. With these results the designer can consider lowering the weight of the more difficult heuristics or readjust the heuristic algorithm.

## 6 Conclusion

The purpose of this project was to explore the balancing of board games using a genetic algorithm. A simplified version of the self-developed game Siphon was used as a case-study. It presents a map that has a big impact on how the game plays out and consequently that is where the algorithm has been applied. A genetic algorithm uses a fitness function to evaluate how "good" a board is. The performance is dependent on the heuristics chosen, and as such these were developed together with the designer of Siphon. The heuristics were defined by analyzing and trying different strategies that seems to make an interesting game.

To provide a proper evaluation of the system, we plan to conduct an experiment where players will be asked to play generated boards, evaluate them, and compare them with human-designed ones. One of the advantages of using GAs for these generative tasks is that, given the stochastic elements of the technique, it allows for the generation of diverse maps, which still fulfill (to some degree) the requirements set by the heuristics. The heuristics do not necessarily need to be perfectly satisfied in order to make a balanced and interesting map. Another possible improvement to the system would be to replace the heuristic with a simulation: to evaluate if a map is balanced we can simulate a number of games and analyze the results (e.g. how many times did player one win? How

long did the games last in average?). A possible candidate for the implementing the player controller is Monte Carlo Tree Search, which is domain independent (does not need heuristics). While this paper focuses on creating balanced maps, it would be interesting in future work to reverse this concept to create very unbalanced (unfair) maps. This could be used to create extra challenge for the players, or as a game mechanic if you assume the players would be able to act on the map layout. Since the algorithm is entirely based on the heuristics, they can be modified to any needs. For example, as wars are rarely balanced, a use case could be having the enemy part of the map as static, so that the system could create optimal positioning of resources for the player/user. That said, we believe that this system would be of more use in creating training simulations than optimizing positions for real-world usage.

To conclude, this paper presents an exploration of the usage of GAs for the generation of balanced board-game maps. The results, while preliminary, seem to show potential for creating diverse and interesting maps.

## References


1. s. Adkins, S.: The 2017-2022 global game-based learing market. Serious Play Conference (2017)
2. Yannakakis, G.N., Togelius, J.: Artificial Intelligence and Games. Springer (2017)
3. Fullerton, T.: Game design workshop: a playcentric approach to creating innovative games. CRC press (2008)
4. Harkey, A.: Balance. (2014)
5. de Mesentier Silva, F., Lee, S., Togelius, J., Nealen, A.: Ai-based playtesting of contemporary board games. In: Proceedings of the 12th International Conference on the Foundations of Digital Games, ACM (2017) 13
6. Teuber, K.: The settlers of catan - game rules and almanac (1995)
7. Barros, G.A., Togelius, J.: Balanced civilization map generation based on open data. In: Evolutionary Computation (CEC), 2015 IEEE Congress on, IEEE (2015) 1482–1489
8. Hom, V., Marks, J.: Automatic design of balanced board games. In: Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE). (2007) 25–30
9. Mitchell, M.: An Introduction to Genetic Algorithms. MIT Press, Cambridge, MA, USA (1998)
10. Shiffman, D.: The nature of code. Free Software Foundation (2012) Accessed 12/04/2018, http://natureofcode.com/book/chapter-9-the-evolution-of-code/.