Evolving Maps and Decks for Ticket to Ride

Fernando de Mesentier Silva
New York University
Brooklyn, NY
fernandomsilva@nyu.edu

Scott Lee
New York University
Brooklyn, NY
sl3998@nyu.edu

Julian Togelius
New York University
Brooklyn, NY
togelius@nyu.edu

Andy Nealen
New York University
Brooklyn, NY
nealen@nyu.edu

ABSTRACT

We present a search-based approach to generating boards and decks of cards for the game Ticket to Ride. Our evolutionary algorithm searches for boards that allow for a well-shaped game arc, and for decks that promote an equal distribution of desirability for cities. We show examples of two boards generated by our algorithm and compare our results to those of the actual components of the game. Our approach creates game content that is specifically designed towards metrics that can affect gameplay in an impactful way.

CCS CONCEPTS

• **Mathematics of computing → Evolutionary algorithms**; • **Applied computing → Computer games**;

KEYWORDS

Procedural Content Generation, Board Games, Evolutionary Algorithm

ACM Reference Format:
Fernando de Mesentier Silva, Scott Lee, Julian Togelius, and Andy Nealen. 2018. Evolving Maps and Decks for Ticket to Ride. In *Foundations of Digital Games 2018 (FDG18), August 7–10, 2018, Malmö, Sweden.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3235765.3235813

1 INTRODUCTION

Many popular board games eventually spawn some variant, expansion or derivative work which modifies the original in some way. In some cases, these changes are insignificant or purely aesthetic as is the case in Monopoly: Electronic Banking Edition [8]. Other variants add or modify aspects of play in complex ways, such as in Pandemic: Reign of Cthulhu [18]. Players are often found to prefer one variant over another, often citing balance or a more enjoyable game arc.

While the differences between variants of the same board game are typically well-defined on the mechanics level, it is not obvious how they differ on a dynamics level, and how to measure it. Balance is difficult to bind to a single number, as is game flow. Traditional playtesting offers a way to measure these metrics for games, but it can be expensive, time-consuming, and ultimately imprecise. It may be possible to automate the playtesting process, but that presents its own set of challenges.

Certain aspects of a game experience can be expressed as a set of numerical properties and factors, upon which a designer could tune and optimize. While the correlation between these factors and perceived quality is debatable, a systematic way to maximize and minimize selected properties of a game at will is nonetheless useful.

A method to automatically generate quality content would be a powerful tool for designers to have. This work tackles the challenge of evolving content for a multiplayer game that contains elements of stochasticity and hidden information. Our algorithm leverages different metrics based on the type of content it is trying to generate. However, both metrics were selected following very strong indications that they were clearly linked to those game pieces and that optimizing them can have significant impact on quality of gameplay. We showcase content that our algorithm generated and compare and analyze it with respect to the results extracted from released versions of the game.

2 RELATED WORK

Procedural content generation (PCG) is by now commonplace in video games, and there is a whole research field devoted to finding better (by various criteria) content generation methods [15]. Content has been generated for a large variety of content types and game genres, including platform game levels [10], strategy game maps [16], weapons in space shooters [9] and even flowers [14]. A popular approach is search-based PCG, where an evolutionary algorithm or other stochastic search algorithm is used to find good game content [17].

Within board games and card games, there is less work on content generation, despite the great potential for generation in such games. A prominent exception is the work of Cameron Browne on generating completely new board games [1]; in a similar vein, there has been work on generating rules for card games [7]. PCG can also be used to balance card games, for example by generating card decks that balance players in Dominion [13].

All search-based PCG methods rely on evaluation functions that quantify some aspect of the quality of the content artifact. Many such evaluation functions are *simulation-based*, where an agent

FDG18, August 7–10, 2018, Malmö, Sweden

ACM ISBN 978-1-4503-6571-0/18/08.
https://doi.org/10.1145/3235765.3235813

plays through the content. Measuring content quality is in general a very hard problem, and successful solutions generally rely on reductive or approximate solutions. Browne et. al developed a criticism method for measuring quality in combinatorial games [1], having hidden information and non-deterministic elements. However, our approach of utilizing the game arc for evaluating the most fit content does share connections with some of the aspects of their evolution of design. The game arc, which we describe in the sections bellow, has ties to game depth [11] and to the skill ladder [3].

3 TICKET TO RIDE

Ticket to Ride is a 2 to 5 player board game released in 2004 [2]. In it, players collect resources and spend them to claim different train routes on a map to gain points. The goal of the game is to acquire more points than the other players. The game has been very successful commercially, selling over 3 million copies as of 2014. This success paved the way for over 15 expansions and new titles released for the series, spawning new boards, variants, card games and etc.

A traditional game of Ticket to Ride is composed of a board, typically based on a real-world location, representing the map composed of cities and train routes connecting them, a deck of train cards, the collectible resource that players need to claim train routes, a deck of destination cards, representing objectives that players complete in secret through the game to obtain additional points, and an individual pool of train tokens for each player that is used to mark claimed routes. The game ends once a player has exhausted their pool of train tokens.

On the board, multiple cities are represented and connected between different train routes. The train routes are the primary resource of the game. Players claim routes by spending the appropriate train cards and placing their train tokens to signify that a route has been claimed. The core of the gameplay comes from when a player chooses to claim a route. Once a route has been claimed by a player, no one else can use it. Routes have two attributes: color, which defines the type of train card needed to claim it, and size, which define how many of the same resource is needed to claim it.

In addition to planning around routes, players must also plan for their destination cards. Destination cards act as additional objectives for the players that hold them. They display two cities that the player must connect by claiming routes in between them. At the end of the game, players are rewarded or penalized based on whether or not they were able to complete these objectives. These cities are usually not adjacent, requiring multiple routes to be connected. Cards with cities that are farther apart are typically worth more points, but also present greater risk. Each card that is not completed by the end of the game penalizes the player by subtracting the same amount of points that would be won for completing it.

A match is played over several rounds, with players alternating turns. Every turn a player takes one of three possible actions: drawing new train cards, drawing new destination cards or claiming a route on the board. The end of the game is reached once any player ends their turn with 2 or less train tokens. At this point, each player gets one more turn. The game is then over and points are tallied.

4 SIMULATING THE GAME

We chose to evaluate the generated content with a large number of simulations. These simulations require access to a game engine, as well as a set of agents.

The game engine was intended to replicate the standard version of the game. The engine was optimized to run simulations quickly to enable high-volume simulations.

For the gameplaying agents we implemented 4 handcrafted heuristic-based agents, designed specifically to play Ticket to Ride. Each agent utilizes a different strategy, with varying levels of success. Our agents are based on those used by Silva et al. to evaluate components of Ticket to Ride [4, 5]. The decision to use handcrafted agents is due in large part to the nature of the game. The space of available moves in a turn can be quite large, sometimes reaching above 100 different possibilities, with the average above 60. In addition, stochasticity and hidden information play a significant role in gameplay. Destination cards and train cards are drawn from a shuffled deck and players keep their both their train and destination cards hidden from the other players until the game is over. For these reasons the game can be very challenging for search-driven approaches.

5 BOARD GRAPH AND BRANCHING FACTOR

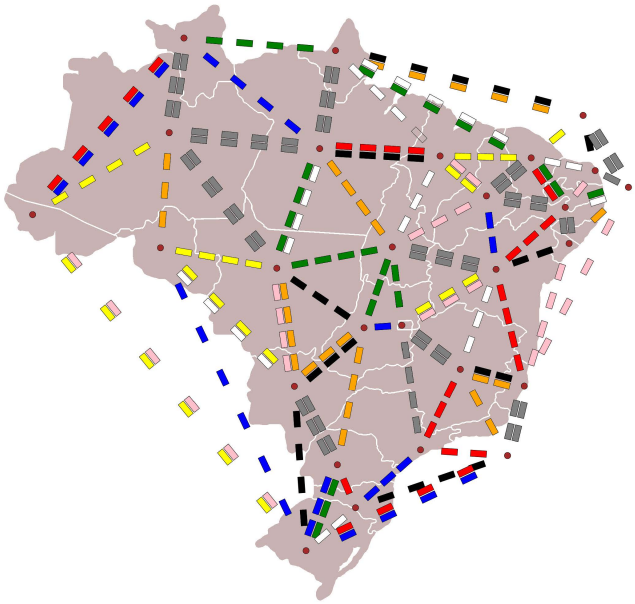


Figure 1: A completely new board generated by our algorithm. The nodes given as the initial inputs were the capitals of the 26 states of Brazil. In the image, nodes/cities are represented by the circles and routes are the dashed lines connecting any 2 nodes. The size of the route is equal to the number of dashes that compose it. The color of the dashes matches the color of the route.

Arguably the main component of Ticket to Ride, board design is largely responsible for steering gameplay. It is the most differentiable element between versions of the game and the main source

of interaction between players. Small changes to the features of the map represented in the board can have large effects on the flow of the game. For these reasons, generating a new board for Ticket to Ride is the most delicate and complex component we are generating in this work. Figure 1 shows a board generated by our algorithm.

The board can be represented as a graph: the cities are nodes and routes are edges. Cities are relevant in terms of their positions in relation to each other on the board and can be distinguished by their names. Routes have 2 major features in terms of gameplay: size (to which we can refer to as edge weight) and color. In existing versions of the game, route size is mainly kept between 1 and 6, although a small handful of titles feature larger routes. For the purpose of this work, we will not attribute sizes larger than 6 to any routes we generate. In terms of route color there are 8 possible regular colors in Ticket to Ride, which correspond to the colors train cards can take. To claim a colored route, a number of cards of the appropriate color must be discarded. There is also 1 extra route color, gray, which can be claimed with train cards of any color, so long as all cards are of the same color.

The problem of generating a board for Ticket to Ride is then a problem of populating a graph with nodes and connecting these nodes with edges that have color and weight. There is also a set of features that we must guarantee that our graph possesses in order to ensure that it doesn't degenerate the gameplay. These features are discussed on the subsection below.

5.1 Board Graph Evolution

To maintain the theming surrounding the game we want to generate boards that resemble real world locations. Most game maps represent real cities, although on different scales: Cities from the United States and Canada make up the map that comes with the standard version of the game, while Ticket to Ride Europe uses cities from the entire European continent. The relative position of the cities tend to represent their real world counterparts. Therefore, as a first step in the generation of a game board, we selected the nodes of the graph. This was done by selecting a real place to represent with the graph. One could, as we have done for the map on Figure 1, use the capital cities of each state in a given country as the nodes. That gives us the nodes for the graph as well as the relative position of the nodes in relation to one another. The nodes themselves are unchanged throughout the evolution and are rather given as input. This not only facilitates the process, but also provides a semantic meaning to the graph.

With the nodes set, the next step, in board creation is edge generation. To generate the edges we use a genetic algorithm. To start, the initial population is generated using the nodes defined earlier. By expressing them as points to be connected, we create the edges that compose a delaunay triangulation [12] for those nodes. Delaunay triangulation allows us to guarantee that no point is inside the circumference of any generated triangle.

From this initial graph we generate the initial population. Each individual is formed by choosing randomly from 3 possible actions to apply to this starting list of edges. The possible operations are: remove an edge chosen at random, choose an edge at random to duplicate and choose a random edge to flip. Duplicating edges is the only way of increasing the amount of edges from the initial list, and is an important operation since it is a common feature of the maps in Ticket to Ride. Essentially, it is possible to have two nodes be connected by two equally weighted edges simultaneously. Flipping an edge is an operation analogous to that performed by the delaunay triangulation algorithm: take the common edge between 2 adjacent triangles and flip it (for example, the triangles ABC and ACD that share the edge AC will become ABD and BCD, sharing the edge BD). The step of choosing 1 random operation from these 3 listed is repeated between 1 to 5 times, so each individual has between 1 and 5 operations applied to them.

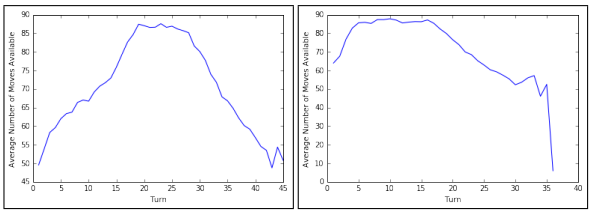
With the initial population set, we then run a genetic algorithm. Mating happens as a two-point crossover operation between the list of edges of two individuals. Mutation is done by selecting at random 1 of the 3 operations listed above (removal, duplication, and flipping) and applying it to a random edge. Tournament selection is performed to choose individuals for crossover.

Individual candidate evaluation makes use of a custom fitness function. It starts by penalizing undesirable graphs. To guarantee that the graph is not degenerate in terms of gameplay we require that it contains certain properties. First, the graph must be planar. That is, no two routes in any Ticket to Ride map cross each other. This way the maps are clean and more readable for human players. The graph must also be a single strongly connected component. Having a single component ensures that, before taking any edges, players can form a route between any 2 cities of the board. Additionally, the graph should have no nodes with a degree above 10. Having a node that is too connected makes it disproportionately desirable while detracting value from other nodes. The number 10 was inferred from the most connected node in the Ticket to Ride games already released. We want the number of edges to be within an interval. Having too few edges both disconnects the graph early in a game and creates a possible scenario in which there simply are not enough edges to exhaust the train pool and end the game. Additionally, too few edges restricts the space of possible moves the players have. We reject graphs that have fewer than 95 edges and more than 110, once again based off the average amount of nodes of already released game boards. A graph that does not meet any of these constraint is set to a negative fitness value. For each constraint it fails to meet we add -100 to its fitness value.

After filtering out degenerate graphs, we evaluate candidates based on the variable upon which we want to optimize. Optimally we would like to have a function that can measure the quality of gameplay for players, but without an objective way to calculate quality of gameplay, we have chosen to evaluate the fitness of a board in terms of the game arc derived from it.

5.2 Game Arc and Branching Factor

We make use of Elias et al. definition of game arc [6]. The game arc is defined by the size of the space of available moves to a player over the course of the game. As Elias, G.S. et al point out, the game arc is analogous to a game arc for the game. Many popular games share a common structure for their game arc: The number of possible moves gradually increases, starting from few available decisions, until mid-game where the size of this space peaks, followed by a decline where the consequences of the decisions begin to show and the game proceeds to its final stages.



(a) Europe board **(b) Asia board**

Figure 2: Comparison of game arcs between two Ticket to Ride maps: Europe, on the left, and Asia, on the right. The x-axis represents the turn and the y-axis represents the average amount of moves. This indicates that the board affects the shape of the game arc.

As, once again, stated by Elias, G.S. et all, most boardgames reflect the game arc described above. Ticket to Ride is no exception, but experimentation has demonstrated that Ticket to Ride's game arc can be affected by the game board. Figure 2 shows the comparison of the average number of moves per turn for different, commercially released, Ticket to Ride maps, over 200 simulations. As can be seen, the Europe map follows this common game arc, while Asia displays different behavior. It is not under the scope of this work to try to explore a relationship between the shape of the game arc and direct quality of gameplay, but rather, demonstrate the impact the graph has on game arc. With this in mind, we chose to guide our generation towards a graph that can better exhibit the game arc described by Elias, G.S. et al.

We therefore tailored our fitness function to favor graphs that resemble said game arc. If the list of edges of an individual passes all the constraints previously outlined in section 5.1, we proceed to evaluating it in terms of its game arc. To do so, we need a graph that can be playable, and thus we must attribute sizes and colors to all edges.

We first define the sizes. Since we used real geographic data to decide the position of the nodes, we can also use it to decide the distance between any 2 nodes. For every edge, we calculate the distance it represents. Then, we extract the maximum, minimum and average distance among all edges. With these metrics, we build 3 buckets of equal size between the minimum and average value, and another 3 buckets of equal size between the average and maximum value. We classify each edge using these 6 buckets, based on the edge's represented distance value. We then order the buckets in ascending order, from 1 to 6, from the bucket closest to the minimum value to the bucket closest to the maximum, we attribute all edges a weight based on which bucket it falls into.

After all edges have weights, we proceed to coloring each edge. By analyzing the existing Ticket to Ride maps, we encountered a trend in edge coloring. If we sum the weights present in the graph, each basic color represents a total of 9% to 10% of the total weight as edges of that color. The rest are gray colored edges. To follow a similar trend, we calculate the sum of the weight of all edges in our graph. We set the minimum that any color can have as being 9% of that weight, and the maximum as being 10%. We proceed to count how many edges of each weight there are. We then calculate the number of edges of each size that each color will have. To know such, we try to evenly distribute the weights among the colors. Any remaining edges are then colored gray.

Once each edge was assigned a weight and color, we proceed to generate the destination deck. Our goal is to generate both a board graph and a deck of destination cards automatically, but a destination deck cannot be generated without a board. In order to have both components, we generate the board first and, after fixing and validating it, generate the deck of destination cards. Since we need a deck of destinations to play the game and therefore evaluate a board, we create a deck with all possible destination cards, that are comprised of the combination of any 2 different cities on the board (without repetitions). The value of the card, based on analysis of the released games, is attributed to be the sum of the weights of the shortest weighted path between the 2 cities.

Once these elements are all in place, we can simulate the game. A 4-player game is played between all four agents 200 times. We then calculate the aggregate average amount of moves of all agents on every turn of the game. Our fitness function then tries to maximize for the area of the triangle formed by the point on the first turn, the point on the last turn, and the "peak" of the graph. This can then be considered as an approximation of a graph with the general shape of the game arc that we are looking for.

6 DESTINATION DECK AND CITY DESIRABILITY

When generating a destination deck, it is imperative that a graph has been defined in advance. To create, or rather choose, the cards to be placed in the deck, we first need to know the cities and the routes in play on the board. For that reason we chose to tackle the problem of creating a destination deck using a separate set of evolutions.

The destination cards play an important part in the game as they give, for most players, a set of objectives around which a strategy is built. As such, the cards drawn often guide the density of interaction between players in different parts of the board. Mid-game questions such as: "Why are players concentrating their efforts on connecting the east coast?" or "why is someone else so upset that you claimed the only route connecting New York and Toronto?" can usually be tracked back to which destination cards the players are holding.

An "unbalanced" destination deck can have a severe impact on gameplay. This work focuses on the destination deck's influence on city desirability. An undesirable city would be a node on the map whose edges are largely unclaimed by players. The most common features that directly influence undesirability are: low connectivity of the node (usually nodes with only 2 or 3 routes connecting it) and being surrounded by more "valuable" nodes, those that are very often connected. The destination exerts a great deal of influence over the desirability of a city. There are noticeable differences in the desirability of specific cities depending on the deck, as shown by our previous work [4, 5].

6.1 Destination Deck Evolution

Evolution of the destination deck is done as follows. First, a list is created out of the possible pairs of cities. We create a list of all combinations of the n nodes of the graph taken 2 at a time. This acts as a baseline for the individuals we generate and evaluate. An

individual is represented as a list of k booleans, where k is the number of elements on our combinations list. So, if a boolean on index x of the individual is **True**, the card represented by the 2 cities on the index x of the list of combinations is part of the deck. Outside of the 2 cities, the destination card needs to have a point value. By analyzing the released decks of the game there is a simple rule that holds for the vast majority of the destination cards: the point value of the card is equal to the sum of weights of the edges that make the path of minimum total weight between the two cities. That is therefore the method we use to define the point value for a card.

During evolution, a selection tournament is done to choose individuals for mating with two-point crossover. The mutation operation involves randomly selecting one of the booleans on the individuals list and flipping its value.

The fitness function tries to minimize the overall undesirability of cities. To achieve this, we simulate 200 games for each individual. Then, we create a map to represent how undesirable the cities are in the simulation. The keys are the cities in the map and the values are the number of games in the simulation in which no player claimed any routes connecting that city. As a last step, we calculate the variance of the values in the map, and the most fit individual is the one with the lowest variance.

7 RESULTS AND DISCUSSION

Our method performs two separate generative processes. It can generate a deck of destination cards given a graph, as well as generate a graph independently of the destination card deck. As such, we will discuss the results for the board generation step and deck generation step separately.

7.1 Board Results

With our algorithm we are able to generate novel maps. We showed the results of a completely new map in Figure 1. However, for the purpose of analysis, we decided to generate a map using the same nodes as an existing, commercially released, board. Figure 3 shows the original USA board on the left and the board we evolved on the right.

There is an immediately recognizable regularity in the shape of the original map as compared to the generated map, especially around the more external edges. The original map's outline resembles that of the geographic United States , due primarily to the fact that external nodes connect to neighboring external nodes. It would be simple to codify that into a constraint and enforce it with the ones we already established at the first step of our fitness function, but we decided against enforcing this rule.

The coloring of routes on our board appears subpar in comparison to the original. It is undesirable to have adjacent routes that are of the same simple color (not gray) as this could require a player to collect many cards of a specific color. That said, since there is an element of randomness to our coloring, one could either run the evolution several times and pick one's favorite coloring or impose a constraint preventing adjacent edges from being painted the same color.

In terms of routes, our graph has 6 fewer edges then the original. Out of all edges, there are 32 which connect the same two cities on both graphs, 17 of which have the same weight and 5 have the same color (all being gray). The average node degree of both graphs is close to 5.5 and the variance of node degree of both is close to 3.1.

Deck	Variance	# Cards	# Cities	Avg pts
USA	29888	30	30	11.4
1910	11410	35	33	11.09
Evolved	3128	38	31	9.37

Table 1: Table comparing 3 different decks being played on the same map. Variance is the variance of undesirability between cities. # Cards is the size of the deck. # Cities is the number of different cities present in the cards. Avg pts is the average amount of points in the cards of the deck.

In terms of game arc, we simulated 1000 matches between the 4 agents on both maps. In both maps we used the same deck of destination cards: the deck that has every single possible destination card. Additionally, we took the game arc of a randomly generated board to act as a baseline for comparison. Figure 4 shows the resulting game arc all three both maps.

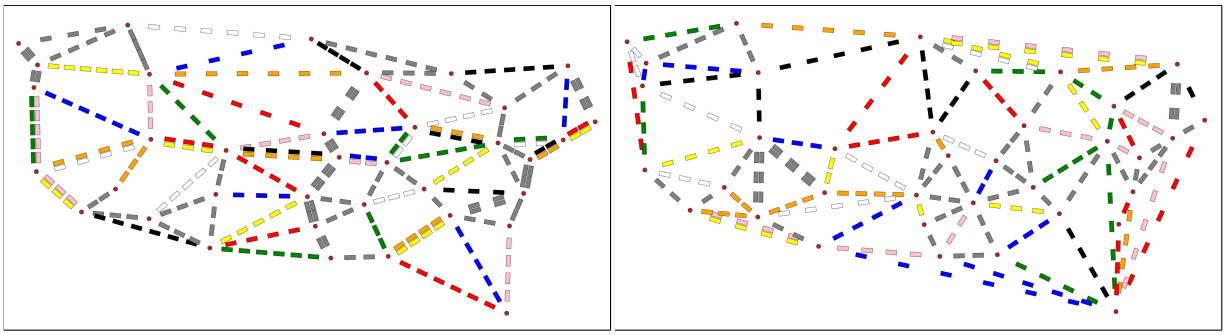
As can be seen, the fully evolved map's game arc closely matches that of the original, which is significantly different from the baseline. This means that our algorithm was successful in evolving a map from its degenerate baseline to one that can create the same game arc as the original Ticket to Ride. Not only that, this also gives an indication that the original game map may possibly be close to some optimal game arc according to our evaluation function.

7.2 Destination Deck Results

In order to compare destination decks, we decided to generate a deck for the original USA map. This way, we can compare it to both the original deck of destination cards, as well as the one released later, USA 1910. Table 1 shows the comparison between decks.

Through the table we can notice that through evolution we manage to reduce the variance significantly compared to the other decks. It is also worth noting that the lower average in points indicates that, in general, cities that share a card on our generated deck are closer than on the others. The fact that reducing the variance in city desirability required many generations and ultimately a substantially different destination deck indicates that the distribution of desirability is a function of the entire deck, rather than a few component cards within it.

We can also evaluate distance between decks. By creating a vector from each deck, where each dimension corresponds to a city and the value for that dimension is the total amount of cards that have that city in the deck. Then, we can check how similar the decks are using euclidean distance. Turns out, the two most similar decks are USA and USA 1910, with a distance of 7.2, while our generated deck has a distance of 8.0 to USA 1910 and a distance of 10.6 to USA. Since the difference in variance is smaller between USA 1910 and the evolved deck, this shows that a specific city being present in the deck does not translate directly into variance.



(a) Original USA map

(b) USA map generated by our algorithm

Figure 3: Comparison between the original USA map that comes with the regular version of the game and an USA map generated by our algorithm after being initialized with the same nodes as the original. The maps are shown with the same rendering style to facilitate comparison.

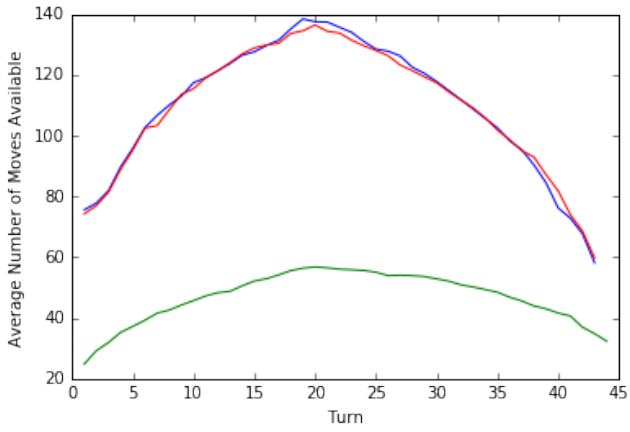


Figure 4: Comparing the game arc for the original USA map, in blue, our evolved USA map, in red, and a map with randomly generated edges, in green. 1000 simulations between the 4 agents were run for each map, and for both maps a "full" deck of destination cards was used.

8 CONCLUSION

The techniques we have shown in this work are capable of generating the two main components of the game Ticket to Ride: the board, which is the center of action in the game, and the deck of destination cards, which is commonly responsible for guiding player strategy during the game.

We detailed our evolutionary approaches to these problems and talked about our choices for fitness functions. While trying to achieve a specific game arc in the game, we look to guarantee a game arc that is common to most board games. Our results also confirm that the board used for the game has great impact on the shape of the game arc. With the destination deck we desired to tackle a different detail on the design. By reducing the variance in undesirability of cities, we are trying to ensure that interaction will be spread through the board, and that most nodes are perceived to have more similar value in terms of strategy.

Our results are compared with components released for the game and achieve very close results, for the maps, or better results, for the destination deck, in terms of the features we try to fit our evolution for.

9 FUTURE WORK

Although the results show great potential, since there is no direct way to measure for quality of gameplay or factor of fun, the only way to test the quality of the design behind our boards and decks is to have them played by humans, rather than AI agents. As future work we intend to conduct a study to gather feedback from players as to how they would rate the content created so we can further validate the ideas presented in this paper. An option is to have groups of people playing games on the maps and decks we evolved and on components of low fitness and ask them for a ranking, in order to check if our metrics are working in the direction of quality and fun.

ACKNOWLEDGMENTS

Authors thank the support of CAPES, Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brazil.

REFERENCES

[1] Cameron Browne and Frederic Maire. 2010. Evolutionary game design. *IEEE Transactions on Computational Intelligence and AI in Games* 2, 1 (2010), 1–16.
[2] Days of Wonder. 2004. *Ticket to Ride.* https://en.wikipedia.org/wiki/Ticket_to_Ride_(board_game) Accessed: 2016-05-15.
[3] Fernando de Mesentier Silva, Aaron Isaksen, Julian Togelius, and Andy Nealen. 2016. Generating Heuristics for Novice Players. *2016 IEEE Conference on Computational Intelligence and Games* (2016).
[4] Fernando de Mesentier Silva, Scott Lee, Julian Togelius, and Andy Nealen. 2017. AI as Evaluator: Search Driven Playtesting of Modern Board Games. (2017).
[5] Fernando de Mesentier Silva, Scott Lee, Julian Togelius, and Andy Nealen. 2017. AI-based Playtesting of Contemporary Board Games. (2017).
[6] George Skaff Elias, Richard Garfield, and K Robert Gutschera. 2012. *Characteristics of games.*
[7] José María Font Fernández, Daniel Manrique Gamo, Tobias Mahlmann, and Julian Togelius. 2013. Towards the automatic generation of card games through grammar-guided genetic programming. (2013).
[8] Hasbro. 2007. *Monopoly: Electronic Banking.* https://boardgamegeek.com/boardgame/32032/monopoly-electronic-banking Accessed: 2017-05-15.

[9] Erin J Hastings, Ratan K Guha, and Kenneth O Stanley. 2009. Evolving content in the galactic arms race video game. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*. IEEE, 241–248.
[10] Britton Horn, Steve Dahlskog, Noor Shaker, Gillian Smith, and Julian Togelius. 2014. A comparative evaluation of procedural level generators in the mario ai framework. (2014).
[11] Frank Lantz, Aaron Isaksen, Alexander Jaffe, Andy Nealen, and Julian Togelius. 2017. Depth in Strategic Games. *AAAI 2017 Workshop: What's Next for AI in Games* (2017).
[12] Der-Tsai Lee and Bruce J Schachter. 1980. Two algorithms for constructing a Delaunay triangulation. *International Journal of Computer & Information Sciences* 9, 3 (1980), 219–242.
[13] Tobias Mahlmann, Julian Togelius, and Georgios N Yannakakis. 2012. Evolving card sets towards balancing dominion. In *Evolutionary Computation (CEC), 2012 IEEE Congress on*. IEEE, 1–8.
[14] Sebastian Risi, Joel Lehman, David B D'Ambrosio, Ryan Hall, and Kenneth O Stanley. 2016. Petalz: Search-based procedural content generation for the casual gamer. *IEEE Transactions on Computational Intelligence and AI in Games* 8, 3 (2016), 244–255.
[15] Noor Shaker, Julian Togelius, and M Nelson. 2016. *Procedural Content Generation In Games*. Springer.
[16] Julian Togelius, Mike Preuss, Nicola Beume, Simon Wessing, Johan Hagelbäck, Georgios N Yannakakis, and Corrado Grappiolo. 2013. Controllable procedural map generation via multiobjective evolution. *Genetic Programming and Evolvable Machines* 14, 2 (2013), 245–277.
[17] Julian Togelius, Georgios N Yannakakis, Kenneth O Stanley, and Cameron Browne. 2011. Search-based procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games* 3, 3 (2011), 172–186.
[18] Z-Man Games. 2016. *Pandemic: Reign of Cthulhu*. https://boardgamegeek.com/boardgame/192153/pandemic-reign-cthulhu Accessed: 2017-05-15.