trAIns: An Artificial Inteligence for OpenTTD

Luis Henrique Oliveira Rios, Luiz Chaimowicz

Departamento de Ciência da Computação
Universidade Federal de Minas Gerais

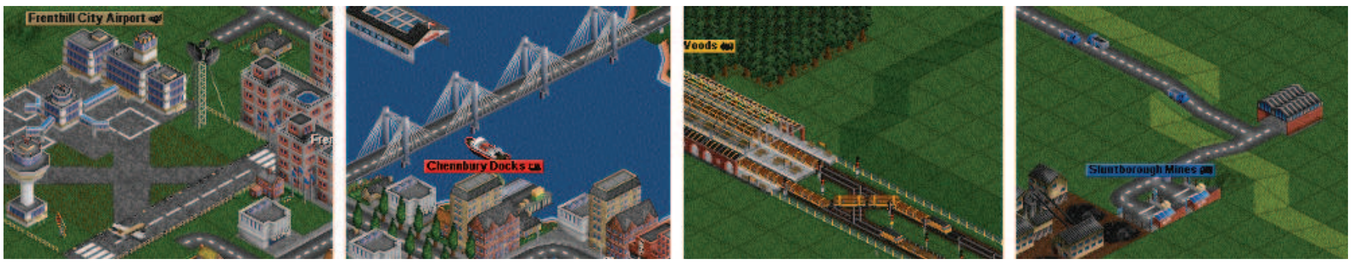


Figure 1: *The four transport types available in OpenTTD: aerial, maritime, railroad and road.*

Abstract

Simulation games present several challenges for computer controlled players. As a result of this, most of the artificial intelligence algorithms developed so far, especially for construction and management simulation games, do not give satisfactory results when compared to human performance. In this paper we develop an AI to control an agent of OpenTTD, a open-source clone of Transport Tycoon Deluxe, one of the premier construction and management simulation games. To do this, we create and adapt artificial intelligence techniques to allow their use in a dynamic, multi-agent strategic environment. Named *trAIns*, the developed AI constructs and manages railroad routes, the most challenging transport type in the game. Several qualitative and quantitative experiments comparing *trAIns* with another AI are performed, bringing very good results.

Keywords: Artificial Intelligence, Construction and Management Simulation Games, Path Planning, OpenTTD.

Author's Contact: {lhrios, chaimo}@dcc.ufmg.br

1 Introduction

The advances in hardware and software have allowed game developers to improve the quality of digital games, augmenting their degree of immersion and realism. The increase in game complexity has demanded the development and adaptation of techniques to deal with a great number of variables and details still respecting time constraints. In this context, the research and development of artificial intelligence algorithms have gained importance. The quality of other game elements such as graphics and gameplay has increased players' expectations regarding artificial intelligence. Thus, it is becoming one of the main components of digital games and should have a level of sophistication similar to other game elements.

In digital games, artificial intelligence algorithms are responsible for making decisions and determining actions of game agents - we will call agent a character or institution in the game that is controlled by a computer. A general metric is that these algorithms must generate actions and decisions resulting in behaviors similar to the ones caused by human players [Byl 2004]. A common problem faced by these algorithms is the time constraints. Despite the increase of computational resources available, the response time is critical because most of the games happen in a dynamic, fast paced environment. These requirements demand an adaptation of classical approaches and the development of new AI algorithms.

In this paper, we are interested on artificial intelligence algorithms for simulation games. Among the several existing modalities, we focus on a style called simulation of construction and management (as stated by [Rollings and Adams 2003] taxonomy). Some classical examples are: *Capitalism*, *Caesar*, *SimCity* and *Transport Tycoon*. In this game style, the player's main goal is to construct, manage and expand communities, institutions, companies or empires using limited resources. Because of the complexity involved in constructing and managing resources as well as the time restrictions, artificial intelligence algorithms suitable for these games are not sophisticated - to our knowledge, there are few artificial intelligence techniques developed for simulation games that are able to deal well with these issues. Therefore, when compared to human players, the generated behavior is poor.

Thus, the main objective of this paper is to adapt, implement and evaluate artificial intelligence algorithms to control agents in construction and management simulation games. In particular, this work considers AI algorithms that will be used to control an agent in a game called OpenTTD [OpenTTD 2009].

OpenTTD is an open-source clone of Transport Tycoon Deluxe, a game released in 1994. The main objective is to construct and manage routes to become the transport tycoon. To achieve this, players must build lucrative transport routes connecting industries and cites. There are four kinds of transport types (figure 1): railroad, road, aerial and maritime. Normally, the most used is the railroad since it is capable of carrying much cargo for great distances in a fast way. It is also the most challenging one as will be discussed.

In this paper, we will use the acronym AI to denote a set of algorithms that control an agent in the game. These algorithms are implemented using a specific API and can play against human players. Currently, there are about 13 AIs available for the game, but only four use railroads. These AIs have several problems in common - part of them caused by the naive approaches adopted. For example, they can not build complex railroads, are not able to plan large routes, can not change the railroad track type and use poor algorithms to choose the locomotive engines. Furthermore, the design of tracks constructed by the AI is very different from the ones built by human players. These problems affect the performance of the company controlled by the computer and influence the gameplay.

Thus, the main contribution of this work is the development of *trAIns*, an AI specifically developed for the construction and management of railroads in OpenTTD. Adapting traditional AI algorithms such as A* and proposing new techniques for a dynamic, multi-agent environment, *trAIns* solves most of current OpenTTD AIs' problems and introduces many other improvements. *trAIns* is evaluated qualitatively, analyzing its construction decisions as well as quantitatively, comparing its performance against Admiral AI, the most complete AI current available in the game.

This paper is organized as follows: next section presents OpenTTD, describing its main features. Section 3 discusses several aspects related to the development of intelligent agents for playing OpenTTD. In section 4, we introduce *trAIns* and in section 5 we present the experimental results. Finally, Section 6 brings the conclusions and possibilities for future work.

2 OpenTTD

As mentioned, OpenTTD is an open-source clone of Transport Tycoon Deluxe, a game released in 1994. It was developed using reverse engineering, but now has an expressive number of new functionalities that improves the gameplay.

Players in OpenTTD own a transport company and are responsible for managing it. These players can be humans or computer controlled agents. The primary goal is to become the transport tycoon, that is, to make the controlled company becomes the best one. The criteria used to evaluate a company contemplates aspects like: the total number of vehicles and stations, the number of cargo types transported, the total number of stations and the amount of money the company has. A more general criterion can be the company net worth: the sum of available cash and company's properties value minus the total debts.

So, to be better evaluated on these criteria the company owner needs to build lucrative transport routes. The routes must connect cities and industries. There are some types of cargo (coal and wood, for example) that must be transported among industries. Others, like passengers and mail, must be carried from one city to another. Finally, some cargo must be transported from one industry to a city (examples are: goods and food).

Four different transport types can be used and combined to create routes: aerial, maritime, railroad and road (as shown in figure 1). Each one has pros and cons that vary according to different factors such as the amount of cargo that must be carried, the landscape around the industry/city, the distance between source and destination, the difficulty to construct the ways and stations, and the amount of money available.

The aerial transport, for example, requires a large area of flat ground for the construction of an airport capable of operating with large airplanes. If compared with the capacity of a train, each airplane can carry only a small amount of cargo and is also more expensive. However, there is no need to create paths connecting the airports. An already existing airport can easily receive new airplanes coming from anywhere, which is not always true for the other transport types. Also, as a general rule, airplanes are the fastest vehicles available in the game.

In contrast with the aerial transport, road transport has the cheapest vehicles of the game. They are also slower and able to carry only small amount of cargo. To reach some destination, vehicles must travel using the available roads. Therefore, the company owner that wants to use this kind of transport needs to build the roads connecting the route's source and destination. Road vehicles can automatically share the roads as they are two-way.

On the other hand, railroads are not able to share their tracks automatically. To create railroads capable of supporting more than one train running at the same time, the player needs to use a resource called *signal*. There are 6 different types of signals. The two most important are the block signal (allows only one train to be in the same block at the same time) and the path signal (allows more than one train to enter in a block if their paths do not intercept each other). These resources must be carefully used because misplaced signals and the use of a wrong signal type can cause deadlocks and accidents. That is why railroads are the most flexible transport type.

A carefully planned railroad can operate with dozens of trains and transport a large amount of cargo connecting distant points. The train cost is not too high if compared to other transport types because a single locomotive can drag several inexpensive wagons.

As in aerial transport, the construction of railroad stations demands a large area of flat land. It has also similarities with road transport: both must connect endpoints and offer tunnels and bridges as way to transpose obstacles. Thus, the construction of railroads incorporate almost all the challenges present in the construction of other transport types routes. That is why it has the most complex construction process.

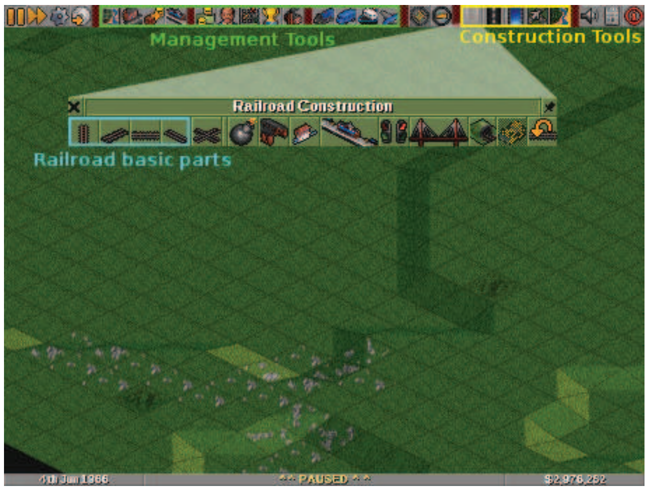


Figure 2: *An OpenTTD screenshot: there two rectangles highlighting some tools available on game menu. The yellow tools predominantly have construction functions. By contrast, the green tools are essentially used for management tasks. It also shows the railway construction tools (the window positioned near the middle). The basic parts provided by the game for railroad track construction have been highlighted using the cyan color.*

2.1 Gameplay

OpenTTD is a construction and management simulator of transport routes. Thus, it has tools used to build the routes and other tools that enable the player to manage the created routes. Figure 2 shows an OpenTTD screenshot. The game menu is depicted at the top of the figure. Some tools have been highlighted. The yellow tools are used in route construction while the green tools are used mainly for management.

The main point of OpenTTD gameplay is that the player must build all elements related to the route. So, suppose that one player chooses to create a route using trains. The first step will be the selection of the industries and/or cites that will be connected by the route. There are some important criteria: production rate and distance between source and destination. After that, he needs to construct the stations and the tracks. Finally, he will buy the proper locomotive and wagons, and program them to execute the route. Other transport types have similar steps but the most powerful and complex is the railroad type.

Management tools are applied to manage the created routes throughout the game. During the game, new vehicle models will be released. All vehicles have some attributes: running cost, maximum reliability (determines the chances of a failure), maximum speed and capacity. Generally, new vehicles are better, since they are faster and are able to carry much more cargo. Therefore, vehicles need to be replaced during the game because they will become outdated. For railroads, it is also possible to change the type of the rail. There are four types available in the game: common, electrified, monorail and maglev. As these new rail types become available, it will be possible to use new kinds of vehicles. Thus, sometimes, changing the railway rail type can be lucrative.

The industry production rates tend to increase during the game. If this industry is used in a route, the number of vehicles must be revised to better transport the production. The production rate can also decrease, requiring a reduction in number of vehicles. It is also possible that another transport company decides to carry cargo from an already explored industry. The production will now be split between the companies forcing the old company to adjust the number of vehicles in its route.

To create the routes, it is important to understand the transported cargo payment mechanism. The payment received for delivering an amount of cargo to some industry depends on some factors. They are: the distance between source and destination, the type of cargo, the number of days that the cargo traveled and the amount of cargo

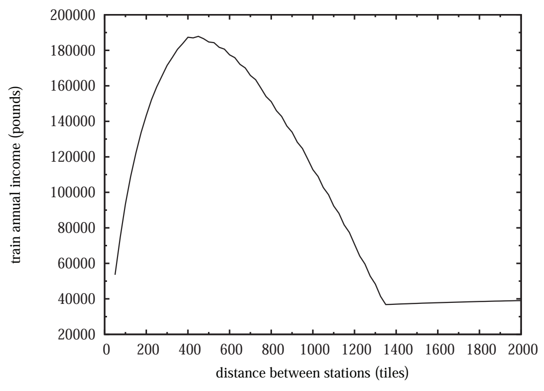


Figure 3: *This curve shows how the annual train income varies according to the distance not considering the inflation and supposing the train travels at constant speed. The used train has maximum speed of 112 km/h and the transported cargo was gold.*

delivered. So, the final payment equation is:

$$payment = dist \times time\_factor \times num\_pieces \times cargo\_base$$

Where $dist$ is the distance between stations, num_pieces is related with the amount of cargo delivered and $cargo\_base$ is the base price of a cargo unit. $Time\_factor$ represents the delay for transporting the cargo. It is inversely proportional to the number of days the cargo traveled. The rate of variation depends on cargo type.

Observing the equation, a trade off can be perceived: long distances will contribute for a bigger delivery payment. However, the delay will increase penalizing the time factor. Thus, there must be an optimal distance.

It is possible to estimate the annual train income for a specific cargo and distance. First, supposing the train travels at constant speed it is possible to estimate the travel time. The train has also a load and an unload time. With this information, one can compute the number of times the train travels the route in one year. Using this approximation we have computed the curve in the Figure 3 that shows approximately what the optimal distance is.

Observing the curve, it is possible to see that the optimal distance is close to 450 tiles (a tile is a cell in the discrete game map). A good AI should be able to build routes connecting industries separated by distances of this magnitude or even bigger distances, considering that trains will become faster during the game.

3 AI for OpenTTD

This section presents several aspects related to the development of an AI for OpenTTD. It first classifies the game environment using [Russell and Norvig 2003] taxonomy. This classification is important to better understand the challenges involved in programming OpenTTD agents. Then, NoAI - the OpenTTD API that enables the development of AI algorithms - is described. Finally, the already existent AIs are discussed.

3.1 Game Environment

Considering a company owner as an agent we can classify the environment where this agent will act. Using [Russell and Norvig 2003] taxonomy, it can be classified as fully observable, strategic, sequential, dynamic, discrete and competitive multiagent.

The game environment is fully observable because an agent using its sensors has access to the complete state of the environment at each point in time. It is capable of seeing all vehicles in the game (including other companies' vehicles), all routes and all existing industries. Moreover, an agent can see the plans being executed by other companies' vehicles. However, some of these characteristics are not implemented in NoAI API yet.

Considering only players' actions the environment is strategic, *i.e.*, the next game state is completely determined by the agent actions and the other players' actions (that can not be predicted). Although, the environment has also some stochastic characteristics mainly related with the economy. New industries can arise randomly. Their production rates changes are also determined randomly. The economy phase, contraction for example, is not predictable. The game disasters (UFO landing, for example), commonly disabled, are an environment stochastic element as well.

The OpenTTD game environment is sequential since current actions will affect future ones. For example, the construction of a railroad route influences future managing decisions regarding industry selection. It also generates new management tasks like: controlling the number of vehicles in the route and choosing the moment to change route vehicles models.

While a player is acting or deciding what he has to do, the game environment is changing: other players can act at the same time, changes such as the construction of a new industry can happen, etc. If a player is constructing a railroad to connect a specific industry, others can do the same and occasionally finish first. The environment, therefore, can be classified as dynamic. This dynamism will always generate some uncertainty in the planning and in its execution.

The game map is divided into small cells called tiles. These tiles are the smallest map units, *i.e.*, the construction of an element always demands at least one map tile. The time is also discrete. Internally it is represented using fractions of days (ticks).

As mentioned, there are various companies competing against each other. Hence, the environment is competitive multiagent. A company can collaborate with another by sending to it some money, but they still compete against each other for resources: industries, cities and map tiles.

Some of the environment characteristics increase the challenges involving the construction of an AI for the OpenTTD. The dynamic environment together with the presence of other agents demand from the AI the ability of fast planning. If planning takes too long, when it finishes, the current game state can be very different from the state initially considered in the planning. Thereby, it will not be valid anymore. Small differences can be resolved using a replanning that, again, needs to be fast.

The route planner needs to be fast while considering a lot of details available in the game such as the construction of bridges and tunnels; the use of terraforming (a tool that enables company owners to modify the land form); the route configuration (winding routes reduce the maximum allowed speed) and the large number of possible paths to connect two points.

Besides, the presence of other agents also demands some flexibility from the algorithms responsible for managing the routes. For example, consider that one company transports some cargo from one industry to a city. If another company decides to transport cargo from this same industry, the number of vehicles in the route should be decreased. Furthermore, variations on industry production rates are not deterministic. A route manager, thus, needs to be able to increase or decrease the number of vehicles in a route according to the changes in the production rates.

These are the main challenges that must be surpassed by the AI that will control a company in OpenTTD.

3.2 NoAI API

An agent has mechanism to perceive the environment (called sensors) and some tools that enable it to change this environment (called actuators). On OpenTTD, sensors and actuators are implemented as an API called NoAI Framework.

The NoAI API allows users to create AIs for the game, programming them in a script language called Squirrel [Demichelis 2009].

This is one of the many improvements introduced by OpenTTD community. Squirrel is an imperative object oriented script language strongly based on Lua language [Ierusalimschy et al. 1996]. This section will discuss some important characteristics of this API and how it works.

The primary principle used in the construction of NoAI Framework is fairness. That is, the actuators must correspond to the tools available for human players and the sensors must generate environment perceptions similar to human players'. As a consequence of this principle, the resources available in both interfaces (for the human players and for the computer controlled players) are equivalent.

To better understand the API internal functioning, one can consider each AI running in the game as a process in an operating system. Each AI will be scheduled using a round-robin scheduler and will be preempted after executing a certain number of instructions. Then, the game will execute its proper functions and call the AIs again, restarting the loop.

The API is composed by various classes that aggregate game functionalities related with some game resource. The class named *AIController* must be extend, so when the AI company is launched in the game, OpenTTD will call the method *Start*. This method must never return, if so the AI process will die.

The classes *AIAirport*, *AIRail*, *AIRoad* and *AIMarine* aggregate the basic functionality related with the four kinds of transport available in the game. For example, the *AIRail* class has a function named *BuildRailTrack(TileIndex tile , RailTrack rail_track)* used to construct a rail on a given tile. Another important function, used to create railroad station, is *BuildRailStation*.

To perceive the game environment one important class is the *AITile*. It has functions like *IsWaterTile*, *GetHeight* and *GetSlope* that can be used to get information about the map. It is important to observe here that when one of these functions is called, it will return a value based on the state of the map when it was executed. Therefore, if the game state changes after this, the AI will need to evoke the function again to note the difference.

The NoAI Framework also works with some events that are equivalent to the news published for human players during the game. When a new company is launched, when a new train becomes available or when a new industry is created the AI event stack will receive the proper information. These events are very important and can be used to help AI decisions process.

All the actuators implemented on the API inform if the corresponding action could be executed successfully. Thus, the AI programmer can always know when some action fails. One can execute an action in the test mode to check if it can be currently executed. Again, the returned value is associated with a specific game state.

3.3 Existing AIs

Currently, there are about 13 published AIs [OpenTTDForum 2009] available for OpenTTD. Most of them are very simple and just create straightforward routes using the aerial and/or the road transport type. Others, more sophisticated, are able to plan complex routes and even combine some transport types. However, only four of them work with trains, the most complex game transport type.

Generally speaking, all these four AIs have some problems. One of these problems, related with the pathfinding, is the limitation in the size of the planned routes. As shown, long routes can be very lucrative especially for trains. Some AIs can not deal with failures during the construction of a route. That is, if the game state changes during the planning the AI will fail in the construction of the route.

Other problem is the absence of rail type changes. This change is very important because it enables the use of new locomotives that are faster and more trustworthy. Differently from human players, none of these AIs is able to create double railways (some AIs create two independent railroad tracks that together compose the route). They neither are capable of making good decisions. Generally, they use poor algorithms to choose the locomotive engines and industries that must be connect by a route. In our tests, *trAIns* AI won

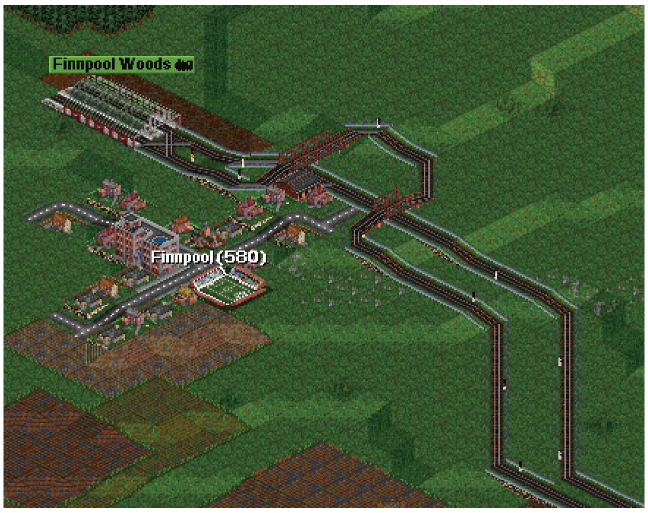


Figure 4: *The figure shows a route created by Admiral AI that operates with trains. To compose the two-way railway it uses two independent one-way railroad tracks.*

easily from some of these AIs because they are very naive and lack powerful management resources. On some tests, they bankrupted on the first years of the game.

Among the existent AIs capable of playing with trains, the only one that could generate results similar to *trAIns* was Admiral AI (Admiral AI is able to play with all kinds of transport available in the game except the maritime transport type). It is the most powerful AI available for the game so far. It operates with trains creating routes which distance between source and destination is about 75 tiles. It is able to reuse already built stations, *i.e.*, Admiral AI can connect different source industries to the same destination industry. To be able to construct a two-way railway, it creates two independent railroads tracks that together compose the route (figure 4). Each independent railway is one-way. Thus, Admiral AI railway routes can operate with multiple trains.

Admiral AI is also able to manage the created routes. During the game it adjusts the number of vehicles in a route according to the production of the industry that is connected by the route. Throughout the game, Admiral AI replaces the locomotive types considering the new introduced types. Thus, we used Admiral AI to play against *trAIns* AI in the experiments presented in Section 5.

4 *trAIns* AI

Using the NoAI Framework and the Squirrel language we have developed a new AI for OpenTTD: *trAIns*. It is named *trAIns* because it only plays with trains, *i.e.*, it basically creates and manages train routes that connect industries. It works as follows: if there is some money available, it will decide if it should build a new route or spend the money improving already existent ones.

Route improvement has higher priority and includes increasing the number of trains in a route, changing the locomotive type or changing the rail type. If no route needs to be updated, *trAIns* will create a new route between two industries. Firstly, the source industry and destination industries are selected. If possible it tries to use an already existent destination industry. This is done sharing the same railway to multiple routes using a mechanism called junction. Then, the railways to connect the industries are created. To do this, it executes A* algorithm to find a path between the stations. We use an abstraction called double parts that enables the construction of double railways.

This section will describe in details each part of this process.

4.1 Railway Construction

One important component of *trAIns* AI is the module responsible for the construction of railways. *trAIns* builds only double railroad

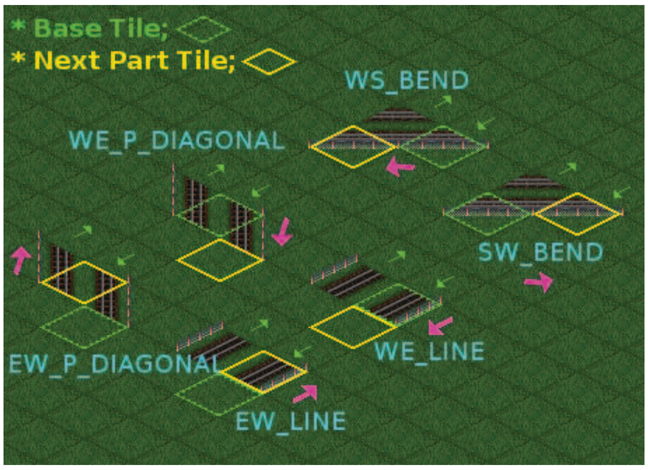


Figure 5: *This figure shows 6 double parts used to construct double railways. Each double part is created by the union of the basic parts provided by the game (figure 2). There are three basic kinds of double parts: bends, diagonals and lines. They have a direction and a orientation as shown in the figure. It also shows the base and next points. The base point is placed over the next point of a part to connect them.*

tracks, which enables many trains to circulate on the same route. As a result of the restriction that on each track of the railroad the train can travel only in one direction, the traffic on double railways is two-way. This approach minimizes traffic jams while allowing a large number of trains on the same railroad. To guarantee that trains will only be able to travel in one direction, each track is signaled using one-way block signals. Next, the process of constructing these railways will be detailed.

4.1.1 Double Railway Parts

OpenTTD provides four basic parts (figure 2) for the construction of railways. We have combined these basic parts to create an abstraction called double railway parts (figure 5). From now on, the term part will refer to a double part created by the combination of the basic parts provided by the game, unless the opposite is mentioned. These parts can be connected to create a double railway. There are 22 parts (bridges and tunnels are also considered parts) that have an orientation and a direction. The connection between parts has a restriction: some parts can not connect with others (*i.e.*, each part has a set of successors). Each part has two important points called *base* and *next*. The base point of a child part (successor part) must be placed over the next point of the parent part (predecessor part) to connect the parts.

4.1.2 Railway Planning

As mentioned, *trAIns* uses A* algorithm [Hart et al. 1968] to plan a path between source and destination points. A* is a classical path planning algorithm that is widely used in games to find shortest paths in a graph or grid. Basically, at each step, A* expands the node x that has the smaller cost $f(x)$ according to the equation $f(x) = g(x) + h(x)$, where $g(x)$ is the actual cost of moving from the source node to x and $h(x)$ is an heuristic function that estimates the cost between x and the destination node. If this heuristic is *admissible*, *i.e.*, if it never overestimates the actual cost, A* can be proved optimal. Details of A* can be found in [Russell and Norvig 2003] and [Bourg and Seemann 2004] among others.

To be able of performing an efficient planning of long railways, the original A* implementation provided as library in OpenTTD was modified. To save time and resources, we removed from the algorithm the code that updates the cost (g) of the nodes in the *open list*. That is, we assume that when a node is inserted in the *open list* it has already the minimum cost, *i.e.*, repeated nodes are visited in an increasing order of cost. Although we have not formally proved it, we believe that A* optimality is preserved, since, in our case, all edges have the same cost. So the first edges to be inserted will have the smallest values of g.

This modified implementation of A* tries to check, as early as possible, if a node has already been visited, that is, if it is already in the *closed list*. The original implementation postpones this verification, first generating the node and then checking if it is on the *closed list*. Moreover, as a consequence of the first change in the code, nodes are closed (marked as visited) in the moment they are inserted in the *open list*.

To create the double railways it considers double parts instead of the basic parts provided by the game during path computation (a node is considered a pair $< tile\ ,\ part >$). Hence, with this approach, *trAIns* needs to execute the search algorithm only one time to build a double railway while other AIs need two executions, each one creating an independent single railway in which trains move just in one direction.

As mentioned, one of the goals of this project is to develop an AI capable of building railways similar to the ones constructed by human players. Part of this goal is achieved by the use of double railroad tracks. However, the railway shape is also an important aspect that can determine the similarities of a human player's railway and a railway created by an AI. The railway tracing is important because it influences the acceleration of locomotives. Depending on the path configuration (number of curves), the train will be forced to decrease its speed. So, is important to try to minimize the number of curves. One possible approach to solve this problem is to punish each direction change using A* function cost (g) as suggested in [Rabin 2000]. But this can increase the time of execution since it augments the number of expanded nodes. The solution proposed here tries to avoid direction changes during tie-break procedures, as explained below.

To allow A* to efficiently plan long railways, it is necessary to carefully choose the heuristics since the number of expanded nodes in A* can be exponential in the length of the solution. The use of appropriate heuristics can attenuate this problem. The h function used by our algorithm is the diagonal distance. The traditional Manhattan Distance Heuristic, generally used in grid environments, is not admissible in our context since using double parts, we may have rails oriented diagonally. When some nodes have the same f value, we have to employ some tie-break procedures to chose which one will be expanded. In case of ties, our algorithm firstly chooses the node x that has the smallest $h(x)$, that is, the node that is supposedly closest to the goal. If all nodes have the same $h(x)$, the node that will minimize direction changes is selected. With this approach, the number of expanded nodes is reduced without sacrificing the solution quality.

Differently from common implementations, our A* implementation does not keep a field to indicate the parent node. Thus, when the execution finishes, if a path is found, the algorithm starts from the goal and chooses the successors with the lowest cost until it reaches the start node. If successors have the same cost it selects the successor that will not cause a direction change. When the start is found, there's a path where is possible (there is no guarantee since the game environment is dynamic) to construct a railway.

If during the railway construction - during the part construction at the position calculate by A* - a change is detected, *i.e.*, it is not possible to create a part at that point, it is necessary to replan the path. The last built parts are destroyed and A* is executed again changing only the start point, since the construction is done in direction to the goal.

The structure generated to represent the path is stored to be used in the future. It is important for changing the rail type and computing junction points.

4.1.3 Bridges

Sometimes, it is necessary to transpose an obstacle (a river, a road, another railway) during the construction of a railway. This can be done using bridges. In general, it is better to avoid bridges by bordering obstacles when possible, since bridges are more expensive (monetarily speaking). But this may significantly increase the num-

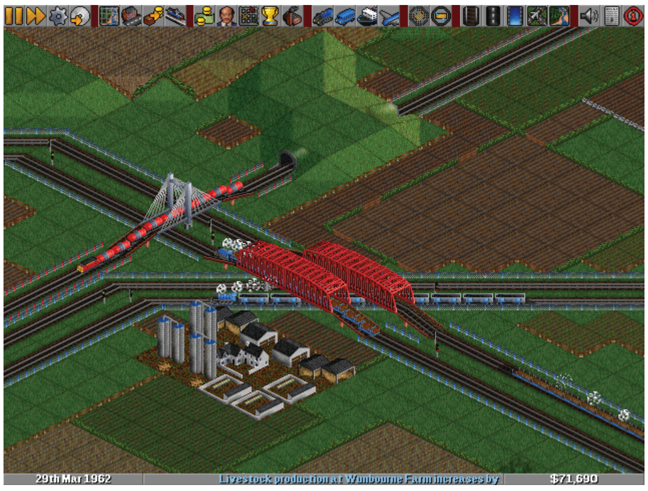


Figure 6: *The double railways were constructed by the trAIns AI. The bridge part was used to transpose other railway. As shown, the bridges that form a bridge part do not need to be aligned. The picture also shows a tunnel that can transpose obstacles too. trAIns AI does not work with tunnels.*

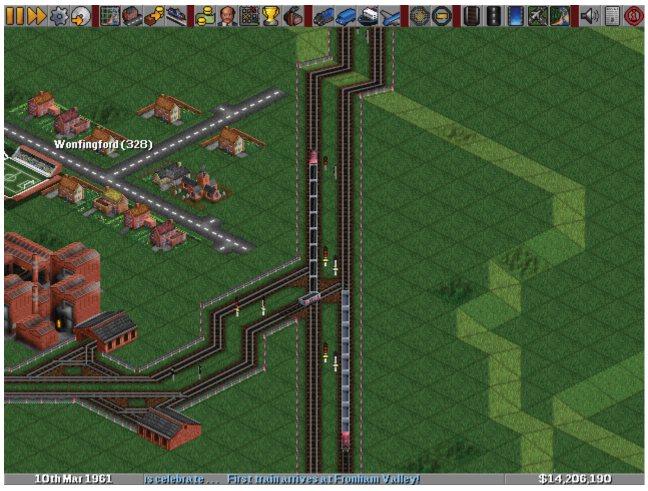


Figure 7: *This figure shows a junction used to create a branch on a railway connecting a Coal Mine to a Power Station. After the construction of the junction, there are two Coal Mines connected to one Power Station. It also shows that two trains can pass through the junction at the same time if their paths are independent,* i.e., *they do not cross.*

ber of expanded nodes. To avoid this growth, our algorithm always considers bridge construction during path planning instead of trying to avoid the obstacles.

As the railway created by *trAIns* AI are double, bridges need to be double too. Another important restriction is related with the execution time. During the planning of a railway it is not possible to check for every node if a bridge should be build. Therefore, the approach adopted here limits the number of bridge construction tries. It only tries to create a bridge if it detects that there is an obstacle avoiding the construction of the next part. If so, it will try to find an ending point for the bridge that transpose the obstacle and permits the construction of a part after the bridge.

Bridges are modeled as parts. One bridge part can be composed of multiple bridges. Each track in the double part has independent size bridges, that is, the bridges do not need to be aligned in a part neither start and end on neighbor tiles. There is also the possibility of mixing bridges with common rails as shows figure 6. Besides saving processing time, this technique produces good results because the bridges built are very similar to the bridges created by human players.

4.1.4 Junctions

Junctions are important tools that allow the creation of complex railway networks, since they permit the creation of branches. Let's say, for example, that there is a route connecting two industries. Near the source industry there is another industry that produces the same type of cargo (that is, its production can be transported to the destination industry already connected by the route). Without the use of junctions, it would be necessary to create a new railway connecting the new industry to the destination industry. However, it is possible to use the already existing railway by connecting the new industry to it. This connection will be made using a junction as shown in figure 7.

The adoption of junctions on railways permits the concentration of production. The production of multiple source industries can be carried to a single industry. Some industries can only produce cargos by processing other industries production (a Sawmill, for example, uses wood to produce goods). If the production of various industries can be routed to a single processing industry its production rate will be very high. Thus, this industry will be an excellent candidate for a route.

Junctions are also modeled as double parts. There are 12 different junctions and each one can be placed over some specific parts. Thus, to create a branch at specific point of the route, one must choose the proper junction part that fits on the part at that point.

At the junctions, the tracks of the railway cross with each other. So, signals must be placed to avoid accidents. The signals used here are from the path signal type, which allows more than one train to enter in a block if their paths do not intercept themselves. Hence, it permits that two trains use the junction at the same time, in the best case. This approach attenuates traffic jams caused by the necessity of mutual exclusion on junctions.

4.2 Railroad Station Construction

Cargos are loaded and unloaded at stations, which should also be constructed by the company owner. These stations should be compatible with the double railways adopted. We selected a format in which there is a single entry point and a single exit point in each station. Each point connects with one of the tracks that compose the double railway. A station can operate with multiples trains at the same time as the number of platforms can be configured. We use two platforms for unloading station and one for the loading station. The number of unloading platforms is bigger to avoid traffic jams since multiple source industries can share the same destination industry.

To create a station, it is necessary to find a large area of flat land close to the desired industry (so far, *trAIns* only creates routes connecting industries). Unfortunately, there is rarely an available area satisfying these constraints so it is necessary to use the terraforming tool to flat the land. Since the cost of this operation can be very high, the approach adopted here is to test a number of different possible lands to then choose the least expensive. The number of tests is not high since there is a distance restriction between the station and the industry (if this distance is too large the station will not be associated with the industry).

4.3 Management

There are two primarily management tasks that should be treated by the AI: route management and the investment of the company's available money. The first task has higher priority: the AI will only invest the money in new routes if none of the current existing routes needs it. In this case, the last task is treated using a very simple approach: always invest (although some restrictions are adopted to avoid spending money on industries with very low production rates). Another important restriction is that *trAIns* AI must have a minimum amount of money before starting to construct any new route.

The process of creating a new route demands some decisions. One of them is the choice of which pair of industries will be connected by the route. To solve this problem, *trAIns* AI considers some industry characteristics such as the number of stations around the industry, the cargo type and the amount of cargo already transported. Thus, the algorithm computes a ratio for each industry in the game and then chooses the industry with the highest ratio. This ratio is given by the production that is not already transport divided by the number of stations around the industry (at least one) and by the price paid for one unit of that cargo transported across 20 tiles with just 10 days of delay. That is, the *trAIns* AI tries to select the industry with the largest potential of money generation.

After choosing the source industry, it is necessary to find an industry to where the production must be transported. The optimal distance between two industries follows a curve similar to the one shown in figure 3, but can not be too long as time to plan the railway will increase too much. Firstly, it tries to use an industry that already has a route and railways. The condition to do this is that the distance between industries and the distance between the source industry and the closest part of the railway that forms the already existent route are limited for a given range. If so, a junction will be created. Otherwise, a new railway will be constructed.

Finally, the number of trains and the locomotive type must be chosen. As the number of trains in the route depends on the locomotive type, it is selected first. The selection process is also based on the computation of a ratio. For locomotives, this ratio varies from 0 to 1 and considers aspects such as the price, the maximum reliability, the maximum speed, its weight and power. All these variables are normalized according to the largest value available. Basically, the *trAIns* AI computes the financial cost of the locomotive benefits. All benefits have the same importance.

The created routes must be managed during the game. One of the managing tasks is the decision of the number of trains in a route. The process used to solve this problem estimates the load time and the travel time using the distance between stations, the production rate and the locomotive maximum speed. Thus, the number of trains is given by the load time plus the total travel time divided by the load time. This formula tries to guarantee that there will always be a train waiting to be load.

Another important management problem is the decision of when the locomotive type must be changed. This problem is also related with the rail type change. To solve these problems, *trAIns* calculates the same ratio used to decide with locomotive is the best. If the current locomotive is not the best it will be replaced. If the new locomotive can not operate on the current rails it must be changed as well. To avoid constant locomotive changes the AI only replaces locomotives in intervals of five years.

5 Experiments and Results

We performed some experiments to evaluate the proposed AI. These experiments compared *trAIns* AI with the Admiral AI playing with only trains. Fourteen scenarios have been used: half with flat terrain and the other half with mountainous terrains. All maps have 512x512 tiles, very small amount of seas (they are predomenantly are formed by lands) and the landscape style used was the *temperate* (the game has 4 different landscape styles). Games begin at the first day of 1960 and go for about 15 years. The locomotive failures were disabled and the other attributes were configured using the medium difficulty level. The games were executed in OpenTTD version 16724, available at the game SVN server.

Tables 1 and 2 present a summary of the results. Two metrics were used to compare the performance of the AIs: the company value and the detailed performance rating. The last considers different aspects such as the total number of vehicles, the total number of stations, the minimum and maximum incomes, the minimum profit, the number of cargo types transported and the total amount of money. The evaluation is based on some thresholds that must be reached by the company. Thus, if the company reaches all thresholds it will be evaluated with the maximum value: 1000 (figure 10 shows more details about this evaluation process).

The tables also present the number of routes created (a route is considered a pair of connected industries). They also exhibit the route

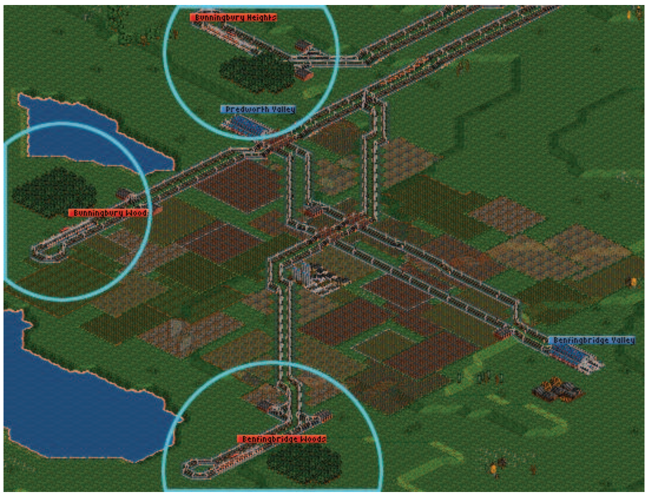


Figure 8: *All highlighted industries have their production transported to the same destination industry. That is, they share the destination station and part of the railway.*

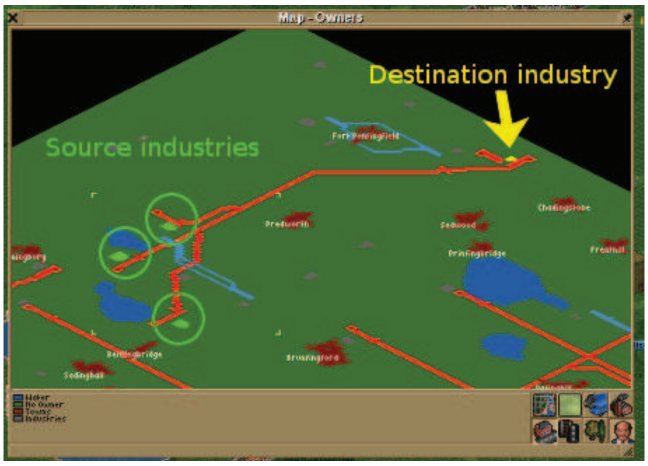


Figure 9: *The figure shows an overview of the routes presented in figure 8. The source industries have been circulated and the destination industry is pointed by an arrow. It is also possible to see that the created railways have a small number of direction changes.*

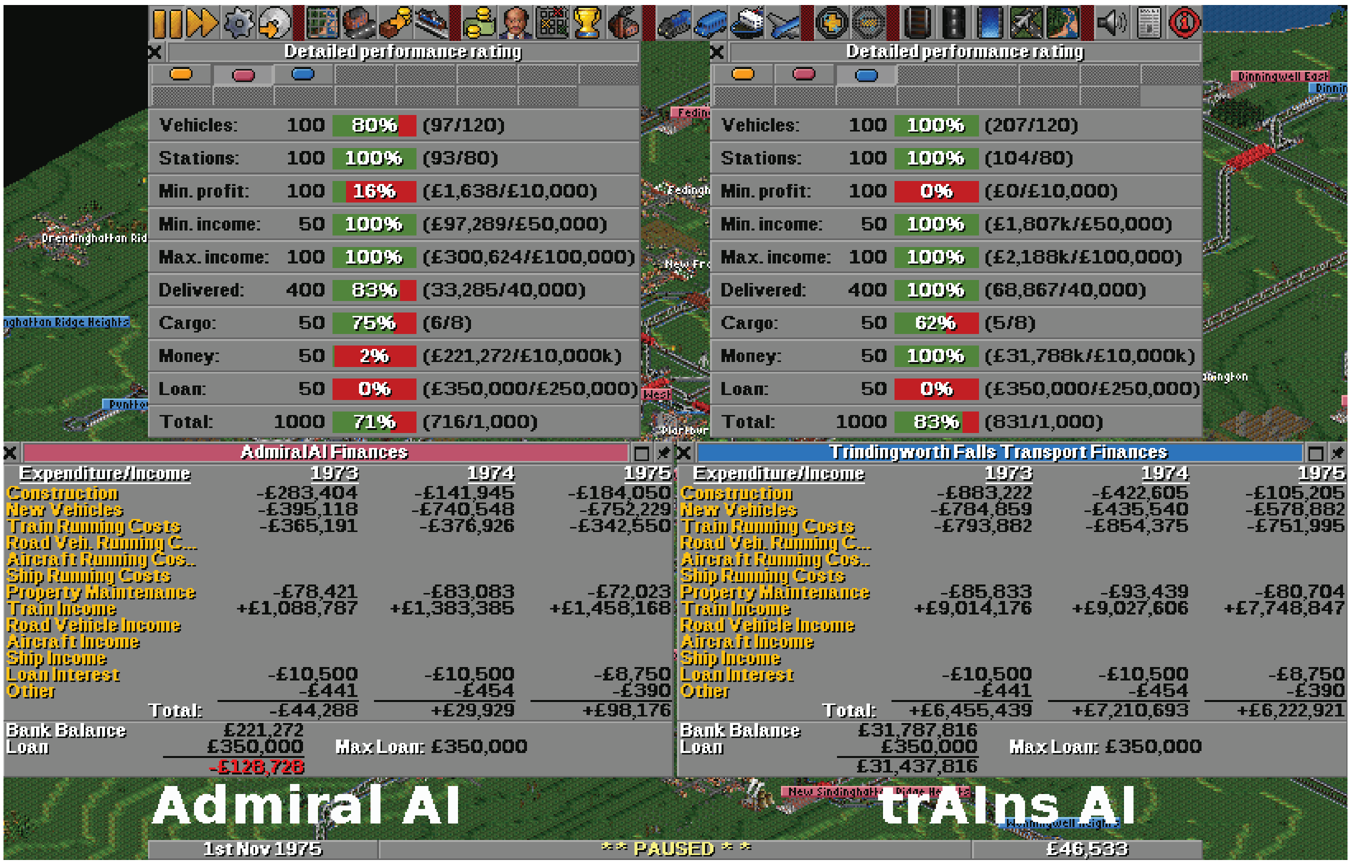
average size that is computed using the Manhattan distance between the stations. Other selected fields are the total number of trains and the total number of stations.

Considering the company value as metric, *trAIns* AI defeated Admiral AI in all scenarios. On average, it reached a company value about eight times bigger than Admiral AI company value.

One of the causes for this success is the size of constructed routes. The routes created by our AI are longer than Admiral's as shown in the last column. They are also more lucrative. Table 3 shows the ratios: company value per trains and per routes. The *trAIns* AI routes are about 3 times more lucrative than Admiral AI routes. This reiterates the results shown in figure 3. Moreover, long railways demand a large number of trains per route because of the increase in travel time. That is, if the distance between stations increase, to keep the monthly transported cargo rate, it is necessary to operate with more trains on the route. On average, *trAIns* AI had about 2.66 trains per route against about 2 trains used by Admiral AI (table 3).

The use of junctions allowed the sharing of a large number of stations among routes. *trAIns* AI uses on average 1.44 stations per route against 1.86 used by Admiral AI. It also provided a mechanism to create railroad networks as shown in figures 8 and 9. During the experiments we observed that, in some situations, five different source industries shared the same destination industry.

When the terrain type is mountainous, it is more difficult to plan railways. On hilly terrains, the number of restrictions for path plan-



Detailed performance rating

Vehicles:	100	80%	(97/120)
Stations:	100	100%	(93/80)
Min. profit:	100	16%	(£1,638/£10,000)
Min. income:	50	100%	(£97,289/£50,000)
Max. income:	100	100%	(£300,624/£100,000)
Delivered:	400	83%	(33,285/40,000)
Cargo:	50	75%	(6/8)
Money:	50	2%	(£221,272/£10,000k)
Loan:	50	0%	(£350,000/£250,000)
Total:	1000	71%	(716/1,000)

Detailed performance rating

Vehicles:	100	100%	(207/120)
Stations:	100	100%	(104/80)
Min. profit:	100	0%	(£0/£10,000)
Min. income:	50	100%	(£1,807k/£50,000)
Max. income:	100	100%	(£2,188k/£100,000)
Delivered:	400	100%	(68,867/40,000)
Cargo:	50	62%	(5/8)
Money:	50	100%	(£31,788k/£10,000k)
Loan:	50	0%	(£350,000/£250,000)
Total:	1000	83%	(831/1,000)

AdmiralAI Finances

Expenditure/Income	1973	1974	1975
Construction	-£283,404	-£141,945	-£184,050
New Vehicles	-£395,118	-£740,548	-£752,229
Train Running Costs	-£365,191	-£376,926	-£342,550
Road Veh. Running C...			
Aircraft Running Cos..			
Ship Running Costs			
Property Maintenance	-£78,421	-£83,083	-£72,023
Train Income	+£1,088,787	+£1,383,385	+£1,458,168
Road Vehicle Income			
Aircraft Income			
Ship Income			
Loan Interest	-£10,500	-£10,500	-£8,750
Other	-£441	-£454	-£390
Total:	-£44,288	+£29,929	+£98,176

Bank Balance £221,272
Loan £350,000 Max Loan: £350,000
-£128,728

Trindingworth Falls Transport Finances

Expenditure/Income	1973	1974	1975
Construction	-£883,222	-£422,605	-£105,205
New Vehicles	-£784,859	-£435,540	-£578,882
Train Running Costs	-£793,882	-£854,375	-£751,995
Road Veh. Running C...			
Aircraft Running Cos..			
Ship Running Costs			
Property Maintenance	-£85,833	-£93,439	-£80,704
Train Income	+£9,014,176	+£9,027,606	+£7,748,847
Road Vehicle Income			
Aircraft Income			
Ship Income			
Loan Interest	-£10,500	-£10,500	-£8,750
Other	-£441	-£454	-£390
Total:	+£6,455,439	+£7,210,693	+£6,222,921

Bank Balance £31,787,816
Loan £350,000 Max Loan: £350,000
£31,437,816

1st Nov 1975 ** PAUSED ** £46,533

Figure 10: *This figure shows the companies detailed performance rating in match number 6 with flat terrain type. As mentioned, it is computed based on different criteria. It also shows the companies finances.*

AI	Performance rating	Company value (pounds)	# trains	# routes	# stations	Route average size (tiles)
Match 1, finished date: *8 - Jun - 1975*.						
Admiral	841	6,090,193	121	67	110	63.79
trAIns	837	46,124,685	221	81	112	193.04
Match 2, finished date: *1 - Feb - 1975*.						
Admiral	780	4,634,504	98	52	90	59.46
trAIns	786	18,437,790	103	43	65	192.74
Match 3, finished date: *11 - Jan - 1975* .						
Admiral	775	8,272,364	175	84	129	67.01
trAIns	831	43,477,220	202	83	114	195.19
Match 4, finished date: *4 - Jan - 1975*.						
Admiral	598	3,729,935	79	44	79	65.43
trAIns	812	41,622,238	174	67	96	188.28
Match 5, finished date: *21 - Jan - 1975*.						
Admiral	595	4,073,063	92	51	91	62
trAIns	831	35,198,641	153	61	94	185.34
Match 6, finished date: *1 - Nov - 1975*.						
Admiral	716	4,955,126	97	49	93	61.25
trAIns	831	39,619,536	207	77	104	192.20
Match 7, finished date: *2 - Jan - 1975*.						
Admiral	801	6,591,956	120	62	107	66.50
trAIns	791	19,586,151	143	43	68	195.20

Table 1: *Admiral AI versus trAIns AI: played in scenarios with flat terrain type.*

AI	**Performance rating**	**Company value (pounds)**	**# trains**	**# routes**	**# stations**	**Route median size (tiles)**
Match 1, finished date: *1 - Jan - 1975*.						
Admiral	703	4,581,543	103	47	88	67.74
trAIns	796	23,004,009	123	50	72	179.74
Match 2, finished date: *23 - Set - 1975*.						
Admiral	133	29,700	9	4	10	69
trAIns	823	22,927,440	137	53	74	195.32
Match 3, finished date: *26 - Jan - 1975*.						
Admiral	716	5,408,988	120	65	110	62.35
trAIns	831	21,110,332	200	73	102	190.79
Match 4, finished date: *5 - Aug - 1975*.						
Admiral	439	2,280,652	48	23	46	62.82
trAIns	831	40,170,005	192	70	98	194.10
Match 5, finished date: *7 - May - 1975*.						
Admiral	285	3,030,691	24	14	28	72.28
trAIns	837	34,544,942	180	64	91	207.57
Match 6, finished date: *26 - Jul - 1975*.						
Admiral	332	1,283,846	29	15	27	60
trAIns	831	33,816,215	173	63	91	198.28
Match 7, finished date: *5 - Jan - 1975*.						
Admiral	468	1,462,148	34	16	34	61
trAIns	816	17,162,685	121	49	73	201.28

Table 2: *Admiral AI versus trAIns AI: played in scenarios with mountainous terrain type.*

AI	$\frac{\#\text{stations}}{\#\text{routes}}$	$\frac{\#\text{trains}}{\#\text{routes}}$	$\frac{\text{company value}}{\#\text{routes}}$	$\frac{\text{company value}}{\#\text{trains}}$	Company Value (pounds)	Relative company value
Scenarios with flat terrain type.						
Admiral	1.73	1.9	92,941	48,914	5,478,163	1
trAIns	1.45	2.67	527,191	200,083	34,866,609	6.36
Scenarios with mountainous terrain type.						
Admiral	2	2.02	97,247	50,560	2,582,510	1
trAIns	1.43	2.65	399,024	149,679	24,585,847	9.52
All scenarios.						
Admiral	1.86	1.96	95,094	49,737	4,030,336	1
trAIns	1.44	2.66	463,107	174,881	29,726,228	7.94

Table 3: *This table summarizes some statistics related with the matches presented in tables 1 and 2. The ratios that used the company value are in pounds.*

ning increases. The terrain landscape also causes a decrease in average locomotive acceleration. So, the total company value tends to reduce. As table 3 shows that Admiral AI was more affected by land format.

If the criterion used to compare the AIs is the performance rating, *trAIns* AI still defeats Admiral AI. The aspects evaluated by this criterion cover various game details. It is based on some thresholds and if the company reached the minimum values it will be ranked using the high value: 1000.

The railways created by *trAIns* AI do not have too many direction changes, in spite of the absence of punishment for direction changes. This is a very important result because enables the creation of large railways without degenerating their quality. Figure 9 shows a railway created by *trAIns* AI.

6 Conclusion

Artificial intelligence is one of the main components of a game and largely influences its quality. In OpenTTD, artificial intelligence algorithms are mainly responsible for controlling game agents, specifically, the companies controlled by the computer. These algorithms must generate actions and decisions so that agents behave similarly to the human players.

In this work, we presented *trAIns*, an AI for OpenTTD. The main motivation for its creation was the lack of good AIs capable of playing using trains. The existent AIs have some common problems: they can not deal with complex railroads, are not able to plan large railroads, can not change railroad track type, use poor algorithms to choose the locomotive engines and also construct very differently from human players. These problems affect the performance of the company controlled by the computer and also degrade game's quality.

trAIns AI presented some approaches to better deal with these problems. A careful implementation of A* search algorithm increased the performance without lowering solution quality. Despite the use of simple function costs, the generated railways do not have too many direction changes and are similar to human players' railways. Double railways enabled the use of various trains on the same route and with the adoption of the junctions the sharing of stations were also possible. Finally, the decision processes implemented in *trAIns* were able to satisfactorily manage the transport company during the entire game.

In the future, we intend to improve the AI decision and planning processes. We believe they can be refined with the adoption of optimization techniques. However, these techniques must be adapted to generate fast responses. Another way to upgrade these processes is the adoption of some techniques, like GOAP [Orkin 2004], commonly used in digital games.

The pathfinding algorithm can also be improved. There are some approaches capable of performing a replanning without the necessity of recomputing the whole solution. These approaches, for example [Koenig et al. 2004; Koenig and Likhachev 2002], can be adapted and used for programming an OpenTTD AI. Another possibility is the use of real time algorithms such as [Koenig and Likhachev 2006] to generate fast solution for the game.

Finally, there are some other game resources that still can be implemented in the AI. The use of tunnels is one of them. Like bridges, they can transpose some kinds of obstacles and are important in the game. Another important resource is the adoption of terraforming to decrease the number of curves and altitude changes.

Acknowledgments

The authors would like to thank the financial support provided by CNPq and Fapemig in the development of this work.

References

BOURG, D., AND SEEMANN, G. 2004. *AI for Game Developers*. O'Reilly Media, Inc., July.

BYL, P. B.-D. 2004. *Programming Believable Characters for Computer Games (Game Development Series)*. Charles River Media, Inc., Rockland, MA, USA.

DEMICHELIS, A., 2009. Squirrel. http://squirrel-lang.org/default.aspx, June.

HART, P. E., NILSSON, N. J., AND RAPHAEL, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on 4*, 2, 100–107.

IERUSALIMSCHY, R., DE FIGUEIREDO, L. H., HENRIQUE, L., WALDEMAR, F., AND FILHO, W. C., 1996. Lua - an extensible extension language.

KOENIG, S., AND LIKHACHEV, M. 2002. D*lite. In *Eighteenth national conference on Artificial intelligence*, American Association for Artificial Intelligence, Menlo Park, CA, USA, 476–483.

KOENIG, S., AND LIKHACHEV, M. 2006. Real-time adaptive a*. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, ACM, New York, NY, USA, 281–288.

KOENIG, S., LIKHACHEV, M., AND FURCY, D. 2004. Lifelong planning a*. *Artif. Intell. 155*, 1-2, 93–146.

OPENTTD, 2009. Openttd. http://www.openttd.org/en/, June.

OPENTTDFORUM, 2009. Transport tycoon forums - noai discussion. http://www.tt-forums.net/viewforum.php?f=65, June.

ORKIN, J. 2004. Applying goal-oriented action planning to games. *Game Programming Gems*, 217–228.

RABIN, S. 2000. A* aesthetic optimizations. *Game Programming Gems*, 264–271.

ROLLINGS, A., AND ADAMS, E. 2003. *Andrew Rollings and Ernest Adams on Game Design*. New Riders Publishing, May.

RUSSELL, S. J., AND NORVIG, P. 2003. *Artificial Intelligence: A Modern Approach*. Pearson Education.