TECHNICAL UNIVERSITY OF DENMARK
DEPARTMENT OF INFORMATICS AND MATHEMATICAL MODELING

MASTER THESIS

Artificial Intelligence for the OpenTTD Game

Author:
Maciej WISNIEWSKI

Supervisor:
Dr. Carsten WITT

Kongens Lyngby 2011
IMM-M.Sc.-2011-56

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

IMM-M.Sc.: ISSN 0909-3192

Abstract

This master thesis project report is a result of an analysis of artificial intelligence applications in the field of transport management, focusing on optimal economic strategy and based on the example of OpenTTD, a simulation game. During this analysis a custom artificial intelligent agent has been designed and implemented for this game.

OpenTTD is a simulation game available for free on its website, as an open-source project. The objective of the game is to create and manage your own transport company and potentially achieve the best performance company ratings.

The project presented had several specified aims. In the beginning the author gives a brief game description and describes related problems which are solvable through the usage of knowledge from the field of artificial intelligence. In the next chapters an analysis of human behaviors, playing strategies and a typical human approach to the game is presented.

The game API and artificial intelligence implementation language Squirrel are investigated and learned. Based on the gained knowledge existing artificial intelligence implementations and their designs are analyzed and described. Comparison classes and properties for categorization of artificial players are determined.

Using these results and the gathered knowledge an artificial intelligence called SPRING is implemented and presented. Afterwards, an evaluation of the results based on selected test cases is made.

In the end of the report a discussion (with examples) of real-life applications is presented. Additionally, a short discussion of possible game improvements in the area of artificial intelligence is presented.

Acknowledgements

I would like to thank my supervisor for a lot of patience and good advice. Likewise, I would like to thank to my family and friends for the support they have provided.

Maciej Wisniewski

Contents

Chapter 1

Introduction

Artificial Intelligence is an exciting field of Computer Science, a substantial portion of which is directed towards studying, design and creation of intelligent agents. These agents are able to perceive their environment and respond in a way that could bring them closer to achieving a success. In a lot of cases, artificial intelligence patterns are inspired by human behaviors and implementations of ”intelligent acting” are intended to be simulations of human mind processes.

Intelligent agent studies focus on problems related to reasoning, learning, inference, communication, collecting and processing knowledge, perception, planning, abilities to move and operate on objects. Each of these problems can represent a separate set of challenges in the artificial intelligence world. In this report the acronym AI will be used to denote a set of algorithms representing an artificial intelligence in the context of an intelligent agent.

The idea of creating an AI is to improve human reasoning skills and automate the process of inference and decision making. Currently a dominating application of AI is the development of solutions which support humans by side giving support based on measurable parameters. But the main goal is to build independent systems being able to support themselves in their specific environments.

The number of problems that could be solved with the help of an AI is practically uncountable as it is comparable to the number of problems facing the human race. Artificial intelligence algorithms are widely used in economics, expert systems, decision support, prediction, pattern recognition, uncertainty reasoning, as well as in search and optimization problems.

In daily life an ordinary person can encounter AI in computer games, on the Internet as automated on-line assistants or automatic ”bots” built for data-processing. In many cases the quality of today’s computer games and on-line assistants is highly dependent on the controlling AI’s behavior and skills. Artificial intelligence agents are even present on stock exchanges and currency markets where special algorithms are used for the detection of negative trends and automatic protection against investor losses. Another example relevant here is the presence of transportation management systems (TMS), decision support systems (DSS) and expert systems that are used by the industry to improve the process flow and to support human experts in gathering, analyzing and reasoning about data.

1.1 Game Description

Computer games are usually strongly AI-dependent. This means that the most of the fun which a player draws from the play is based on competition or some sort of ally with an AI player. Even ff the game is a multiplayer one, this does not mean there is no AI, because they may appear in the form of computer-controlled characters or some (or even all) elements of the game environment.

In this work focus is placed on OpenTTD game AI development. Due to the complexity and special properties, OpenTTD presents a unique challenge in the AI context.

OpenTTD is an open source simulation game based on Transport Tycoon Deluxe, the latter originally developed by MicroProse Software, Inc. in 1994 and upgraded to "deluxe" in 1995.

The main goal of the game is to build and develop your own transport company in such a way that it gets the highest possible revenue, thanks to an optimal transport organization, transporting as many cargo as possible in the most efficient way. There are several modes of transportation available: road-based (buses, trucks, trams), aerial (airplanes, helicopters), rail (trains) and nautical (ships). There are multiple types of cargo to be transported, their names and properties (source, destinations) may depend on the landscape style chosen, but in the default ('temperate') mode, these are: passengers, mail, coal, iron ore, steel, wood, oil, livestock, grain and goods.

Passenger and mail sources/destinations are towns, goods are produced by industries and should be delivered to towns. Other cargo should be transported between appropriate industries.

1.2 Aims and Basic Rules

The competition in the game is focused on building the right connections and managing infrastructure. Additionally there can be multiple companies present at the same map. There can be multiple human players as well as multiple AIs competing against each other. The game world reacts actively in response to player behavior. Industries without transport infrastructure go bankrupt, good connections, in turn, increase production. Towns grow with time and passenger exchange. An important part of the game are local authorities which monitor the natural surroundings and noise levels and can decline permission for the construction of new infrastructure.

Economics play an important role in the game. The game begins with a bank loan which should be well and wisely invested. The player has to pay not only for the new infrastructure, but also loan interests and maintenance for existing vehicles and routes.

Players have additional town actions like advertisement campaigns, bribing local authorities (very risky) or other actions improving company ratings and reputation at their disposal.

Games contain random events like economic flow changes, accidents, prototypes, areal cargo acceptance changes and subsidies for particular connections. Vehicles are getting older while time passes and they require replacement. New vehicles are introduced depending on the game date, they have different parameters and can improve the company's transport capabilities.

Interactions between companies is extended by the ability to buy shares in companies or staging an outright takeover. Multilayer LAN or Internet sessions can support up to 255 players

in a single game. A single company can be controlled by multiple players at the same time. Game server can also be used for AI competitions. Single player game against artificial competitors is also available.

Due to OpenTTD's complexity, wide options and extensive economy model, it is emerging as a transport company simulator, more then just a game.

All game sources are open source and can be modified up to anyone's needs. Elements such as graphics, themes, vehicle sets are highly customizable. AI players can be created by volunteers and are stored in the so-called "BaNaNas" repository, which is available to download by all players in the main game screen through menu options.

1.3 Goals and Strategies for an Artificial Player

Based on the description given above, what we will be dealing with is adaptive AI techniques used in a dynamic, multi-agent strategic environment. There are several decision and planning problems we have to face initially:

- planning towns/industry connections,
- action planning,
- transport type choice,
- building infrastructure,
- management, support and maintenance,
- general company strategy, priorities, tactics.

Connection planning is referring to the connection network which will be built using the game's available infrastructure. Different types of transport have different needs and vary in optimization of routes.

Actions planning refers to the order of performed operations - what should be done first, and what can wait. It consists of dependencies between actions (airplanes can not be bought before building an airport).

Transport choice is a decision problem related to the chosen strategy of player preferences. Building infrastructure refers to the right location no only of the roads or railways but also of the right placement of stations and depots.

A correct and optimal placement can have a crucial impact on the a route's financial performance.

General company strategy, priorities and tactics should gather all other problems in a common goal in a way that single solutions are coordinated and prevented from disturbing each other.

The chosen transport type has an important meaning for every aspect of the rest of the problems which an AI has to solve. Aircraft does not have path limits, because airplanes and helicopters are traveling "above" the map. But airports require a lot of available ground space, which might require landscaping. Building costs of airports, terrain changes and especially hight price

of airplanes makes it a relatively high-risk, but also very fast and possibly lucrative transport solution. Ship transport have similar properties except the limitation for only water and a very limited speed of vehicles. In opposite for aircraft the cargo capacity is high.

Railways have high capacitance and high speed, but they are most complicated in construction and maintenance. Keeping more then one train on the route is not a trivial task. Road transport is the simplest, building roads is relatively cheap and they are reusable by all players. Road vehicles are not that much expensive and once built right they can bring in cash almost immediately.

Depending on the used transport type the AI has to adopt different strategies. The best way to understand how big an influence it might have on the strategy is to observe how human players behave in the game and what is their way of reasoning. Chapter 2 will describe this in detail.

Different transport modes and their vehicles have various properties. The most important are capacity, speed and reliability, which defines the frequency of failures. A vehicle failure stops the vehicle, which delays delivery of the cargo, but may also block other vehicles. It may also lead to a crash causing a loss of the participating vehicle(s).

Each transport type has to have an individually designed connection network with most optimal distances, path in between connection points and the right infrastructure for used vehicles.

The whole success depends on the economic performance of a single connection as well as on the coordination of the whole transport company. Well-designed transport routes can bring outstanding profits, badly designed ones will cause financial losses and can bury your company.

The game world from an artificial intelligence aspect is a full observability multi-agent environment where agents compete against each other. Agents are interacting AIs in the game world. Full observability stands for the agent ability to see the whole world with no hidden informations and obstacles. Multi-agent means that there are other AIs which are interacting with the game world.

The challenges for an AI capable of supporting itself in the OpenTTD world are the main point of discussion in the report. The conclusions will be a base for own AI design and implementation presented in Chapter 5.

Chapter 2

Human Behaviors

Humans playing OpenTTD use many different strategies and it is hard to generalize them to one specific playing style. Players may have transport type preferences, which have the biggest influence on the playing strategy. Also players pay attention to different things, for example some of them are fully income-oriented, some try to compete with others since start and some try to play in isolation so the most things depend only on them.

The thing which is common for all the players is a special attention for the construction and building infrastructure. The human brain is capable of planning and decision making with no special effort and good results. Apparently a set of simple rules can evolve into well- and nicely-built transport network. Humans playing OpenTTD are often choosing spontaneous and unique solutions, sometimes they obey a predefined strategy and they accordingly hold it against obstacles.

2.1 Gameplay

During design and construction the player is capable of freely creating and modifying all elements of the infrastructure and even the terrain properties, including the reduction of slopes and leveling the hills.

The reader is highly encouraged to try a short game session just to have a feeling of the gameplay. Let us assume that we start the game in the most common settings, without modifications, just right after installing it. For the purpose of this report the OpenTTD 1.1.0 version will be used. It has been downloaded from the game website (`http://openttd.org`). Just after starting the game we can chose the ”New game” option. Then we will see the map generator window, the best choice for now will be not to change anything and, after clicking on ”Generate”, going directly to the game.

In the top part of the screen we will a menu bar with several options. The best way of discovering the possibilities is just to try each one. Once we achieve some basic knowledge about the menu and the options we can start playing. Usually the best way start is to analyze the map. Maps are generated randomly, but with some algorithms behind which keeps them reasonable. To see how the map is we can use very helpful ”Display map” option (fifth button

Figure 2.1: The map view for planning the future network.

from the left in menu bar). By using mouse scroll we can zoom things in and out. In the lower part of the Map window we can notice some options modifying the view. We can show/hide city names, show roads, industries and more. It is good idea to get rid of the names at the beginning and look at the map with towns and industries. The view like presented in Figure 2.1 will help us to decide what to do next.

That is the starting point for any more experienced players to evaluate the map, find best places, estimate the future infrastructure design and choose the transportation methods.

Depending on the map conditions like terrain flatness and water coverage we can primarily say what and where could be easily built. Especially at the beginning it is very important to maximize the potential income of everything we build and to create routes which will bring stable and constant profit to our new company, so it could handle rapid development, maintenance costs and loan interest.

If we chose to focus on passengers and mail the starting point should be the biggest towns. If we focus on cargo transport, let us say coal, we should check the coal mine details regarding possible monthly coal production capabilities and check the closest delivery place (for coal it is a power station).

The basic rule every player abides by is to keep in mind the dependencies between cargo source and delivery place and minimizing the distance (transportation cost, delivery time). The example how the income may change is presented in Figure 2.2. We can observe the slope presenting the relation between estimate income and the train route length. But the design and creation of the route is not the only concern. Once in a while all routes should be checked for eventual problems and expected profit. According to these rules under default game settings it should be possible to establish a company which creates profit.

If a player would like to achieve an outstanding company financial performance there are also

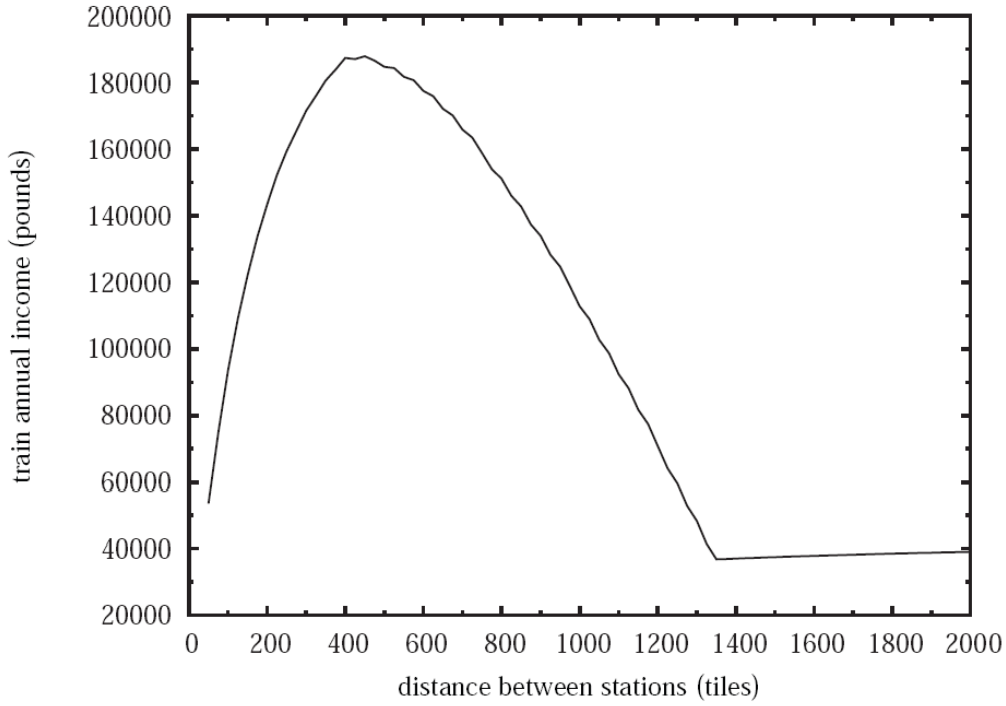

Figure 2.2: The curve presenting how annual train income varies from the distance. The plot from [23]).

additional game elements worth keeping in mind. For many players also the visual side of the game is important, they set a nice looking graphics set so it is also crucial to keep the design of routes according to an original idea. But in our case the visual style is something immeasurable and could be very subjective. We focus only on the elements that improve profit.

2.2 Survey Results

During preparation of the report a survey on the OpenTTD's Internet discussion board has been made. Experienced players have been asked what is their view on the behavior of an AI player. How they recognize that it is not a human player and what kind of improvements they would suggest.

One of the most noticeable property of the AI player is ability to effectively control and maintain a large fleet of vehicles. But it works only on simple routes and in the case of repeatable schemata. AI players currently available are not building massive connection hubs, e.g. combined from multiple train stations as it is shown in Figure 2.3.

Another limitation of AIs are repeated schemata for building infrastructure. They are not creating the junctions and lines in a creative way, i.e. in a way that is adjusted individually in each situation. AIs have ready-to-use schemata and they are just choosing one possibly best solution from many preprogrammed options. It is done to simplify the process of decision taking but the result of such construction in most cases could be easily improved by a human player.

The limitation in constructions patterns creates another problem: not taking into account

the acceleration model. The acceleration model is a physical model in the game simulating the changes of speed of the vehicle caused by slopes of the terrain and meanders of e.g. railways. The best infrastructure should take the acceleration model into account and has minimized the angles of turns and number of terrain slopes (terrain level differences).

The problem with the right terrain slopes is also an element of more complex handicap with general terrain leveling and smoothly flowing tracks and roads. A person is able to instantly optimize the route with respect to elevation changes; to apply viaducts or bridges when necessary and change the terrain levels while keeping in mind the costs (minimalization of the leveled surface area) and best performance of vehicles on the route.

A problem partially covered by existing AIs is also upgrading existing services according to the increasing demand. In the following chapter a few existing AIs will be described and this problem will be extended in the discussion of our own AI implementation.

The problem with changing the train length and adaptation of the vehicles was also mentioned in the survey. The problem is related to vehicle properties considered for the vehicle switching. Depending on the circumstances the priorities of vehicle properties while selection may wary. Most often the important parameter during vehicle selection is capacity. There is also reliability, speed, cost, maintenance cost per year. If we try to maximize the profit, directly dependent on the volume of cargo transported the capacity property seems to be most important. That might not be always true. The cargo generation algorithms which increases the load of stations are dependent on the frequency and stability of the cargo out-flow.

We could imagine a situation when for a long air-route the cargo is not that frequently present.

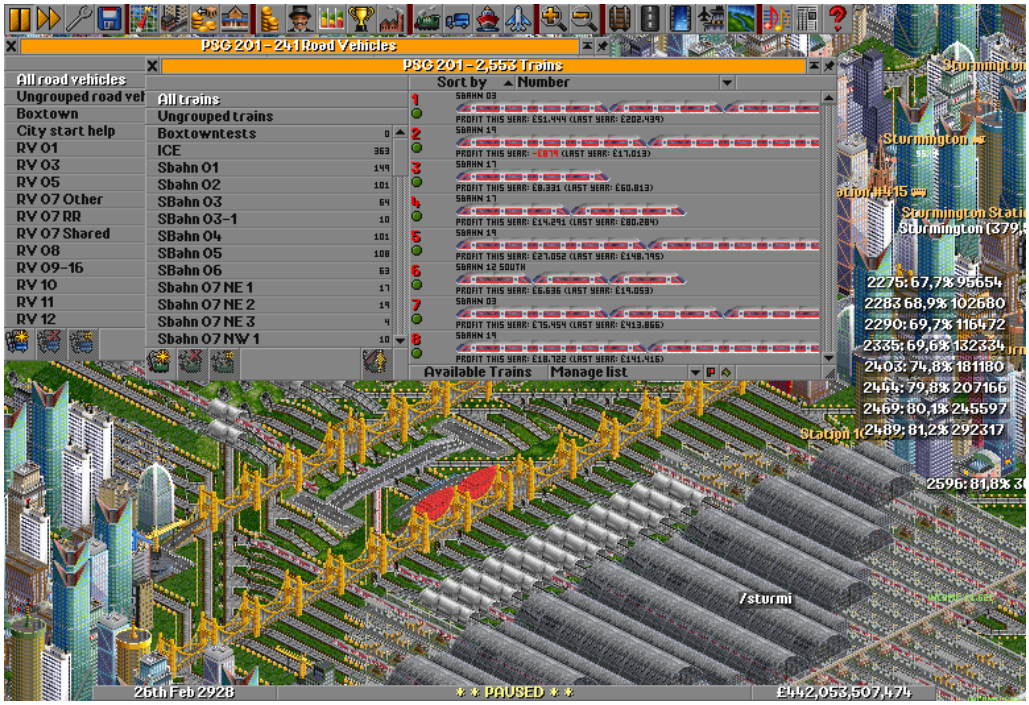

Figure 2.3: Large train station combined of multiple routes into a hub, long lists of trains presented. The image from [3].

If we place there a big plane which will just stay on the airport waiting for full-load this could be a big waste. Cargo is not frequently transported so it is not increased. We loose money on the maintenance cost of a big airplane and additionally the speed of its flight might be slow. In such case it could be better to have a small, fast and cheap airplane which would transport the cargo frequently increasing its number much faster. After a while when the cargo number would be appropriate the airplane could be switched to a bigger one which would fit the economical conditions.

The last issue mentioned by players is upgrading existing infrastructure up to the currently available technology. During the game every mode of transport evolves. There appear new more sophisticated vehicles which have better parameters, but also new infrastructure becomes available. There are new bigger airports and better planes. Railway also evolves, first electrical rail becomes available and later maglev tracks shows up. On the road new buses and trucks are present. It is not a trivial task for an AI to adjust to these changes and modify its strategy. It is especially hard to upgrade existing routes to new standards.

All of these described problems regarding the AI behavior can be summarized as follows:

- massed approach in management and concentration (hubs),
- variations of lines, junctions, stations,
- take the acceleration model into account,
- taking an advantage of increasing supply,
- leveling small areas of land in order to get a smoothly flowing track,
- upgrade existing services as demand increases,
- lengthen the trains,
- improved a junction and station for capacity,
- upgrade it to high speed rail or maglev, upgrade of airports or substitution of road vehicles.

This could directly improve the profit of an AI player and could be a result of a good planning. Such features would bring the AI player closer to a human in performance.

2.3 Non-optimal Goals

In the previous section we listed features which are specific for the human behavior in achieving the higher profits. There could be mentioned other AI behaviors which can not be considered as improvements of economic performance, but would make it more human-like:

- shape of infrastructure inspired by a real patterns, realistic junctions,
- using a few different sorts of vehicles,
- so called ”cultivation of towns” in a chosen way,

- adding decorative non-track tiles to stations,
- dynamic style of building infrastructure,
- aesthetic optimization,
- unlimited re-planning capabilities, damage detection before losses.

The term ”cultivation of towns” refers to the game concept of playing in a way that would develop town in a certain way. The most popular tactic related to this concept is the increase of town citizens by frequent passenger transportation. There is also possibility to form or in other words point the development direction of a town according to one’s wishes.

This list may not necessary improve the profit but these are features which are definitely specific for a human player. This as especially hard to program or simulate. It is difficult for an AI to make a decisions without clear and measurable evaluation criteria for visual side of the infrastructure. Repeating the patterns inspired by real-life is also hard for an AI due to limited observation ability and the necessity of creating a backend base of schemata in advance.

2.4 Other Artificial Intelligence Aspects

OpenTTD, apart from the artificial intelligence substituting a player also contains other AI mechanisms. The most noticeable is the vehicle control on which the player has almost no influence. While playing we specify the route destinations but we are unable to control e.g. how a bus is driven. Also there appear mentioned map events which are controlled by internal game mechanics but have built-in AI algorithms.

Sometimes these ”hidden” AI elements have an impact on our game. For example, and notoriously, the algorithms used for the vehicle movement and path finding on roads/railways can make our fleet looping around in a strange way. The game built in mechanism for detecting such situations exists. In the case of a human player a person would be informed about this situation by a screen message. In the case of AI there exists *AIEventVehicleLost* static class with a method *AIEventVehicleLost(VehicleID vehicle_id)*. The fact of receiving notification does not solve the problem. Once the vehicle is localized it should be sent to a closest known infrastructure. If the infrastructure has been modified then the surrounding area of the map needs to be rescanned and new paths, destinations and the route should be renewed. If the infrastructure is damaged it may require new constructions. For an AI player a right reaction might be hard in practice.

Chapter 3

Programming environment

OpenTTD, starting from release 0.7.0, allows a user to create her own AI. The developer team provided a clean and simple game API and example scripts. As a bridge between the game back-end and API the Squirrel script language is used. Game sources have an built-in interpreter, so AI developers do not need to install or use any external tools, only the game itself and a simple text editor. Additional tools are theoretically available, but they do not offer much help. For simple development the setup mentioned is enough to create an AI.

3.1 Squirrel

As already mentioned, the language used for AI implementation is Squirrel. Squirrel is a simple scripting language with a C++-style syntax. There are mainly two reasons why it has been chosen by developers. First, it is similar to C++ which is the major programming language used for the entire game source code. Second, it is a high level imperative, object-oriented programming language designed with an effort to provide the following: light-weight scripting, low memory usage and full-fill real-time requirements of applications like games. Squirrel is shared on an Open Source MIT license.

The Squirrel syntax has been inspired by C/C++/Java and its performance and adopted concepts have been inspired by languages like Python, JavaScript and Lua. Lua has been a big inspiration for the API design and it is a very good example of a great success of an intermediate language. Lua has been used in Apple iOS SDK, Adobe Photoshop Lightroom, Apache HTTP Server, Cisco, Lego Mindstorms NXT and NXT 2.0, LuaTeX, MySQL Workbench, Vim, VLC media player, Wireshark and many, many more. The game applications can be found in e.g. Far Cry and Civilization V.

It can be noticed that the concept of a specialized API scripting language is popular and there is a strong demand for such solutions. Squirrel does not have such considerable achievements as Lua, but a few applications in more popular software can be found, mainly in games, e.g. OpenTTD, Portal 2, Grand Theft Auto IV Multiplayer Mod and Final Fantasy Crystal Chronicles: My Life as a King.

The major features of Squirrel can be listed as:

- dynamic typing,
- delegation,
- classes & inheritance,
- high-order functions,
- lexical scoping,
- generators,
- cooperative threads (coroutines),
- tail recursion,
- exception handling,
- automatic memory management,
- small and efficient implementation size.

Due to above features and its construction as a short simple implementation in C/C++, Squirrel is considered to be a very flexible tool, perfect fit for a game API. The advantages of Squirrel are identical to the reasons why OpenTTD developers decided to drop C++ AI support and stay with Squirrel. These can be listed:

- built-in multi-platform support,
- Squirrel is a more abstract approach to AI programming,
- testing the AI in C++ required recompiling the entire OpenTTD code, even with modules cross-platform compatibility could be questioned,
- Squirrel is mature enough to take over,
- mistakes in C++ could crash the game, with Squirrel any problem stays in the Virtual Machine which can, at worst, kill the AI.

The clear disadvantages of Squirrel are the speed, which is inferior to C++, but for this particular application the difference is not that big and noticeable. C++ has its pointers, which can make the code much more efficient.

By removing C++ as a possible language to implement AI the OpenTTD develpers gained simplicity in the framework, security, and most of all: portability. That is why the currently supported NoAI framework is Squirrel-based.

The current development state of Squirrel is version 3.1 beta, however in the OpenTTD sources version 2.2.4 from the previous 2.x series is used.

It is necessary to mention that Squirrel 2.2.4 is not applied in its clean version. The series of modifications has been applied to version 2.2.4 which has been integrated with OpenTTD. The changed version serves basic functions, although they may also differ then original 2.2.4 version.

The standard libraries, freely available to standard Squirrel releases are not enabled. Anything, not mentioned in the documentation to the AI framework, may not work in under Squirrel Virtual Machine (VM). It it possible to extend the functionality, but it may require modifying the game source-code written in C++.

In theory there are available programming tools for Squirrel:

- SQDEV is Eclipse plug-in with syntax coloring and debug features,
- SQDBG is a remote debugger for SQDEV plug-in,
- Squirrel_JIT is an improved optimized compiler,
- SquirrelStudioIntegrated is a Squirrel support package for Microsoft Visual Studio.

There are community patches, build systems and other related tools available. Practically due to mentioned modifications in the Squirrel source-code for the OpenTTD purposes most of the mentioned tools are limited in usage.

The author of this report tried to use them, but they seemed to be in a very early development stages. The SQDEV Eclipse plug-in did not work as expected and it has not been possible to use it. For SQDBG there appeared problems with compilation and execution. Some files necessary to make it work seemed to be missing even after a successful compilation, but further investigation has been dropped. SquirrelStudioIntegrated as a support package for Microsoft Visual Studio also has been very unstable, basically it was installed but it did not work.

During the entire development and implementation of custom AI the only tools used had to be OpenTTD for testing and debuging and Notepad++ with Java syntax coloring for source code edition. Lack of any programming tools had an impact on the source code development speed.

3.2 NoAI Framework

All currently supported OpenTTD AIs are based on the NoAI framework. It is recommended to use as a development and testing platform for the latest stable release of the game. An alternative is the latest source code from OpenTTD SVN compiled by oneself. Because in this report the focus is on AI, for the reader's convenience it is suggested to use the current stable release. All implementations have been tested with OpenTTD version 1.0.5 released on 2010-11-20 20:25 UTC.

There are two major paths for the OpenTTD AI developer. One is to start from scratch and the second is to download one of a few already existing AIs. By looking at other AIs developer can take an advantage by borrowing portions of code or simply modifying the AI up to one's wishes. An important remark should be stated: taking portions of code or modifying others source code is highly license-dependent. Almost all of the available AIs are open-source and released with the General Public Licence (GPL) version 2.0 or 3.0. However, there is a couple of modules licensed on non-standard conditions, so it is important to be aware.

In the next few sections a mixed approach will be presented. The first implementation example is done from scratch and later on the development will follow its needs. It may be that

implementation of some algorithms are used from existing sources, but in such case it is clearly stated what is used and based on which source code.

Another important aspect of the NoAI framework is accessibility of ready, general-use libraries. They contain a road pathfinder and a rail pathfinder, that help at the early stages to build an AI and make it work much faster. It is also allowed to implement pathfinders on your own.

3.3 AI Skeleton

Let us start with the simplest possible AI. To build the simplest AI it is required to create a new folder according to AI name under the game path in the directory */ai*. The expected directory tree should look as shown:

OpenTTD/

```
--- ai/
------ MyNewAI/
--------- info.nut
--------- main.nut
```

There are two main files for each AI. The definition of an AI is placed in *info.nut* where we can specify the main AI class and a fixed set of information about it: author's name, AI name, description showed in the game, current version and release date. The class has four major methods returning values important for the in game AI control system. There is also a special method returning the settings list and its specification. The AI can have series of settings available for the user in the game GUI. These settings can be the AI playing speed, limitations on vehicle type usage or others. To register in the OpenTTD core as an AI the function *RegisterAI()* is used. The example content of the information file can be found in [7].

Once the AI information is specified, the behavior part has to be set. The second file, *main.nut*, should contain the whole AI implementation, its main loop and references to other classes or source files. The class should be build according to the interface specified, which means it has to hold implementations of these methods:

- *Start()* specifies actions to run,
- *Stop()* should contain actions performed when the AI is halted,
- *Save()* support of the AI state for saving purposes,
- *Load()* loading routines for actions before restarting the game,
- *SetCompanyName()* name the company dynamically in the game because might happen one instance already exists.

The example content of the file is as shown in the Listings B.1.

3.4 Debugging

The AI can be tested within the game. There are two features available for it. One is the game console available under the key ~ (on US standard keyboard). Using the game console developer

is able to start and stop an AI. For starting it the command *startai <AIName>* should be typed, e.g. *startai MyNewAI*. Respectively for stopping the command *stopai <AINumber>* is present, e.g. *stopai 2*. AI numbering usually starts with two because first slot is reserved for the current player which runs the game. It may happen that there are more AIs running so to control them right and avoid mistakes in their indexes it is good to also have the AI Debug Window open.

The AI Debug Window can be found after the game board generation in the the last main menu bar option under big blue question mark icon; the option is labeled "AI debug". The AI Debug Window allows to observe the current AIs running and their execution. The developer has access to *AILog* output, including info and error and crash messages, but unfortunately the output can not be redirected to e.g. an external file and can be read only in the AI Debug Window, which has a very limited capacity.

3.5 Game API

As we mentioned before that game has a comprehensive API available for the AI developer. Documentation is available in [6]. It allows all the normal game operations accessible by the user while playing. Everything is grouped into static classes with names related to the purpose and can be applied with a good practice of using design patterns.

The developer is also allowed to create dynamic objects on his own using a provided static class as a factory. There is access to additional controls e.g. *AIAccounting* for better money management, *AIBase* for randomness features, *AIController* for the overall game and AI special setting access, *AIError* for detecting and retrieving errors, various types of *AIEvent* for handling special circumstances like a vehicle crash, resource reduction, bankruptcy, etc.

A very important class for general usage is *AIMap* that allows to scan the terrain divided into tiles, which have their counterpart in the API as *AITile* class objects. Another important feature of *AIMap* are the following methods: *DistanceManhattan (TileIndex tile_from, TileIndex tile_to)* for calculating the Manhattan distance, *DistanceMax (TileIndex tile_from, TileIndex tile_to)* for calculating the distance via 1D calculation, *DistanceSquare (TileIndex tile_from, TileIndex tile_to)* for ordinary squared distance, *DistanceFromEdge (TileIndex tile)* for shortest distance to the edge.

Especially *DistanceManhattan ()* method is important because it allows to estimate the shortest travel distance for ground vehicles and planned routes. There is also an *AILog* class available for printing out AI debug messages. There can be printed info or error messages, with a difference in the AI debug window that error messages are marked red.

The API has multiple classes to control vehicle features, e.g. *AIVehicles*, *AIVehiclesList*, *AIVehicleList_Group*, and its behavior, e.g. *AIOrder*.

There are some small differences between the API documentation and actual behavior. A simple example is related to the precondition list for a *AIAirport .BuildAirport()* method. One of the preconditions mentioned in documentation is *AirportAvailable()* which should be executed before *BuildAirport()*. However, the actual precondition is *AIAirport.IsValidAirportType()*. These differences creates obstacles for AI developer who should be aware of them. The only method is to always test in practice single portions of code to see if the program behavior is as

expected. If an error appears then the developer can only look at other up-to-date AI examples and search for the same method used in the similar context.

3.6 Implementations and Design Problems

The implementation has been influenced by several language features which made it complicated for a beginner Squirrel programmer. Let us take a look on the list of the main obstacles and complications met during the implementation:

- Method preconditions
  Most of the methods require several preconditions before they can be successfully executed. In the documentation the preconditions are mentioned, but it is not mentioned how they should be fulfilled in a practical way.

- Not enough explanation regarding data initialization;
  Assuming a static class which returns an custom object (not an NoAI framework object). In most cases it is not documented what kind of object it is.

- Unknown types, references, conversion of variable's types and unknown or not documented functions/methods parameters;
  Lack of clear comments regarding the passing argument type. E.g. assume we have a piece of code containing of a class's method and we would like to perform a slight modification. It is hard to estimate what are the argument types and how to make a conversion. There is no tool to automatically detect where the value comes from, so the whole source has to be analyzed in order to find it.

- Advanced syntax used without docs;
  E.g. `::<variable>`, which is defined inline. After further investigation it appears to be `::<variable><-this` and it means static field, newly assigned, containing the current object.

- Unexpected behavior of arithmetic operations;
  E.g. 3/2 returns integer by default, which requires to be converted to float by using *.tofloat()* method. Squirrel documentation lacks informations about this kind of behavior.

The example AIs have non-unified design structure, except the overall basis of *main.nut* and *info.nut* files. For this reason navigation of the unknown source code was not an easy task. Some of the existing AIs have better organization and internal comments (like PathZilla or AdmiralAI), but most of them lack basic concept explanations. None have model or class diagram-based descriptions typically used in software engineering.

There were also practical obstacles at the beginning, namely a lack of Squirrel tutorials or beginner examples. There appeared that programming tools like IDE tool, editor syntax checker, on-line debugger or others were lacking or defective. Such tools are very helpful to speed-up the process of software development.

The development process had to based on a necessity of ”trial and error”. Reading and analyzing the existing implementations was helpful, but the best examples have comprehensive and complicated structure and it was not an easy task for a beginner.

All mentioned causes had an influence on the work on the analysis described in the next chapter and the implementation described with details in Chapter 5.

Chapter 4

Existing AI Analysis

In the following chapter a summary of existing and available AIs is presented. The comparison classes have been selected and a table combining all gathered informations has been presented. Selected solutions have been described in details.

4.1 Overview

In the current version of OpenTTD (1.1.0) all available AIs are placed in a repository calls BaNaNaS available on the game website [4]. They are also available from the in-game interface, which provides automatic download and install options.

First, descriptions of AIs have been prepared. For every AI a brief description has been made and written in the form of a side-card. All side-cards are available in Appendix A. Based on the side-cards a summary has been created as a one easy-to-read table, Table 4.1. In the table AI names are listed in rows and their specialization is marked. Special symbols are used: $\times$ means that the AI is capable of using particular type of transport, $\otimes$ means that AI is somehow specialized in using certain type of transport.

The most interesting AIs for this project have been denoted with a $\bullet$ and the reader can find their more detailed descriptions later on in this chapter. In the ”Remarks” column short notes referring to special properties have been placed. More details about other AIs can be found in Appendix A.

	Road			**Railway**	**Water**	**Aircraft**						
	Buses	Trucks	Trams	Trains	Ships	Helicopters	Planes	Passengers	Mail	Cargo	Extended description	Remarks
Convoy	⊗							⊗				-
PathZilla	⊗		×					⊗			•	adv. graph theory used
Chopper						×		⊗	×			-
Denver & Rio Grande	×			⊗				×		⊗		three sub-pathfinders
PAXLink	⊗						×	⊗				"towns cultivation concept"
Trans	⊗	×	×	×	×	×	×	⊗		×		"cargo concept"
trAIns				⊗				⊗	×	⊗	•	scientific approach, human behaviors
AdmiralAI	×	×	×	⊗	×	×	×	×	×	×	•	comprehensive usage of the game API
ChooChoo				⊗				⊗	×		•	network
SimpleAI	×			⊗			×	⊗	×			-
MogulAI		⊗								⊗		-
CluelessPlus	⊗							⊗				-
AIAI	⊗	×		×	×		×	⊗		×		-
NoCAB	×	×		×	×		×	×		⊗		extended planning, sub-sum algorithm
RoadRunner	⊗	×	×					⊗	×	×		-
OtviAI	×	×	×	×				⊗	×	×		-
Rondje											•	winner of TJIP 2008, unique approach
MailAI		⊗		×					⊗			-

Table 4.1: Summary of the available AIs from BaNaNaS (state for 2011-03-01). Informations about AIs from [3].

4.2 Properties of AIs

There are various approaches present to the AI implementation in OpenTTD game. Some are more sophisticated and comprehensive like: PathZilla, Denver & Rio Grande, Trans, trAIns, AdmiralAI, ChooChoo or Rondje, but there are some other simple AIs which were created with limited capabilities. Besides major AIs which have the main aim to play, there exist AIs with special purposes. Their major goals are: building roads or creating artificial traffic on the roads, so the game play is more similar to the real life.

As the summary Table 4.1 shows, not in all cases every type of transport is supported by an AI. Even if the type of transport is supported, an AI might be specialized in one type of vehicles, e.g. Chopper uses only helicopters, MogulAI uses only trucks and Convoy uses only buses.

AIs are using different strategies and algorithms to generate profit during the game play. Some of them use libraries built by the NoAI framework developers, where basic AI algorithms can be found. More extended description of the NoAI framework can be found in Chapter 3, Section 3.2.

In the next few sections several AIs will be described in a more detailed way due to their special properties. The selection of these has been made based on the gathered information, side-cards and summary table. The focus has been made on the most interesting algorithms used, some non-standard unique approaches, reasoning methods and strategy with a reference to the project aims.

4.3 PathZilla AI

The PathZilla AI has been selected as an interesting example of using advanced graph theory in the context of connection network planning for transport paths. PathZilla AI has been written by George Weller (alias 'Zutty') and the history of development begins in 2008. The author takes an advantage of the strategy and planning based on graphs (for the original description please look in the details in Appendix A). The idea is to start with a Delaunay Triangulation for a set of points representing cities in the game.

4.3.1 Base Algorithms

By "triangulation" we mean a method for an efficient way of defining triangles between the triple of points. The example of such connections is presented in Figure 4.1. Each of those points represents a town and vertices in the graph are connected by edges to their closest neighbours; in a way that all edges are parts of triangles without edges intersections on a plane. Also there is no triangles overlap and the surface between vertices is covered with a layer of various triangular tiles.

The Delaunay Triangulation has been invented by Russian mathematician Boris Nikolaevich Delaunay in 1934. We can find various applications of this method in science and computer graphics. It is very common method to be used in the graphic representation of geometrically irregularly distributed data, such as weather maps or altitude maps.

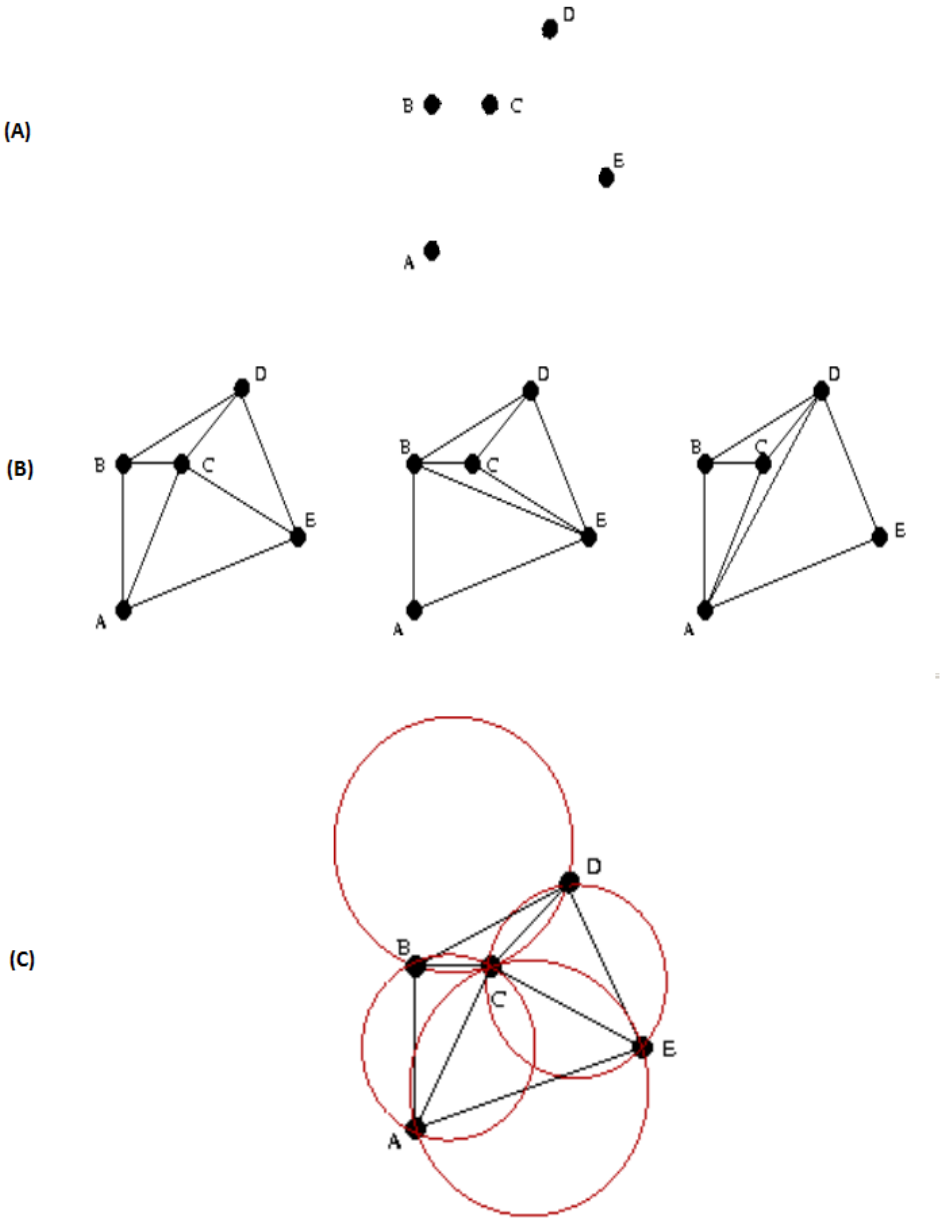

Figure 4.1: The example for the visualization of a triangulation process.
(A) Initial graph state; (B) Possible triangulations; (C) Delaunay Triangulation.

The 3D-variant is the most common method for representing virtual worlds in video games. A closely related popular solutions are Voronoi diagrams, alternatively called "Dirichlet tessellation" (more details can be found in [11]). An example of triangulation process is presented in Figure 4.1.

In the next part of this section several definitions are presented. They are based on the theory from [16], [10], [11] and [24]. Definitions of the basic terms as: graph, subgraph, edge, vertex and edge weight and other can be found there.

We define Delaunay Triangulation for a set P of points in a two-dimensional plane, as a triangulation $DT(P)$ such that no point in P is inside the circumcircle of any triangle in $DT(P)$.

Once triangulation is done and paths between points are stored the next step of the algorithm amounts to determining the minimum spanning tree and the shortest path tree.

A spanning tree of a connected, undirected graph is a subgraph that is a tree and connects all the vertices together.

The Euclidean minimum spanning tree is a minimum spanning tree of a set of n points in the plane, where the weight of the edge between each pair of points is the distance between those two points. In other words a minimum spanning tree connects pairs of points in a way that the total length of all connections is minimum and any point can be reached from any other by going along the tree's vertices.

The set of points in the Euclidean minimum spanning tree is a subset of the Delaunay Triangulation of the same points. This fact has been used here for more efficient computation.

As a shortest path tree we define a subgraph of a given weighted graph constructed so that the distance between a selected root node and all other nodes is minimal.

In the case of the distances between cities the weights are always positive so the fast and common way to determine shortest path tree is to use Dijkstra's algorithm (more about the algorithm can be found in [10]).

The next step done by this AI is to combine edges from the minimum spanning tree and the shortest path tree into one graph based on the same set of vertices. And then, based on a such a network, build the roads.

The visualization of this algorithm is presented in Figure 4.2, which is based on the author's original animation posted on OpenTTD Forum [3].

4.3.2 Interesting Features

The construction of the network is partial (not all at once). To make it so the AI has a queue of planned actions.

Another feature of this AI is ability to build tram lines alongside roads and handle multiple stations per town which increases its growth. It supports all primary industries and uses a two step pathfinder to improve the lines' reuse.

As the advantages of such approach we can list:

- nice looking and functional connection network,
- trams alongside roads so trams do not inhibit automobile traffic,

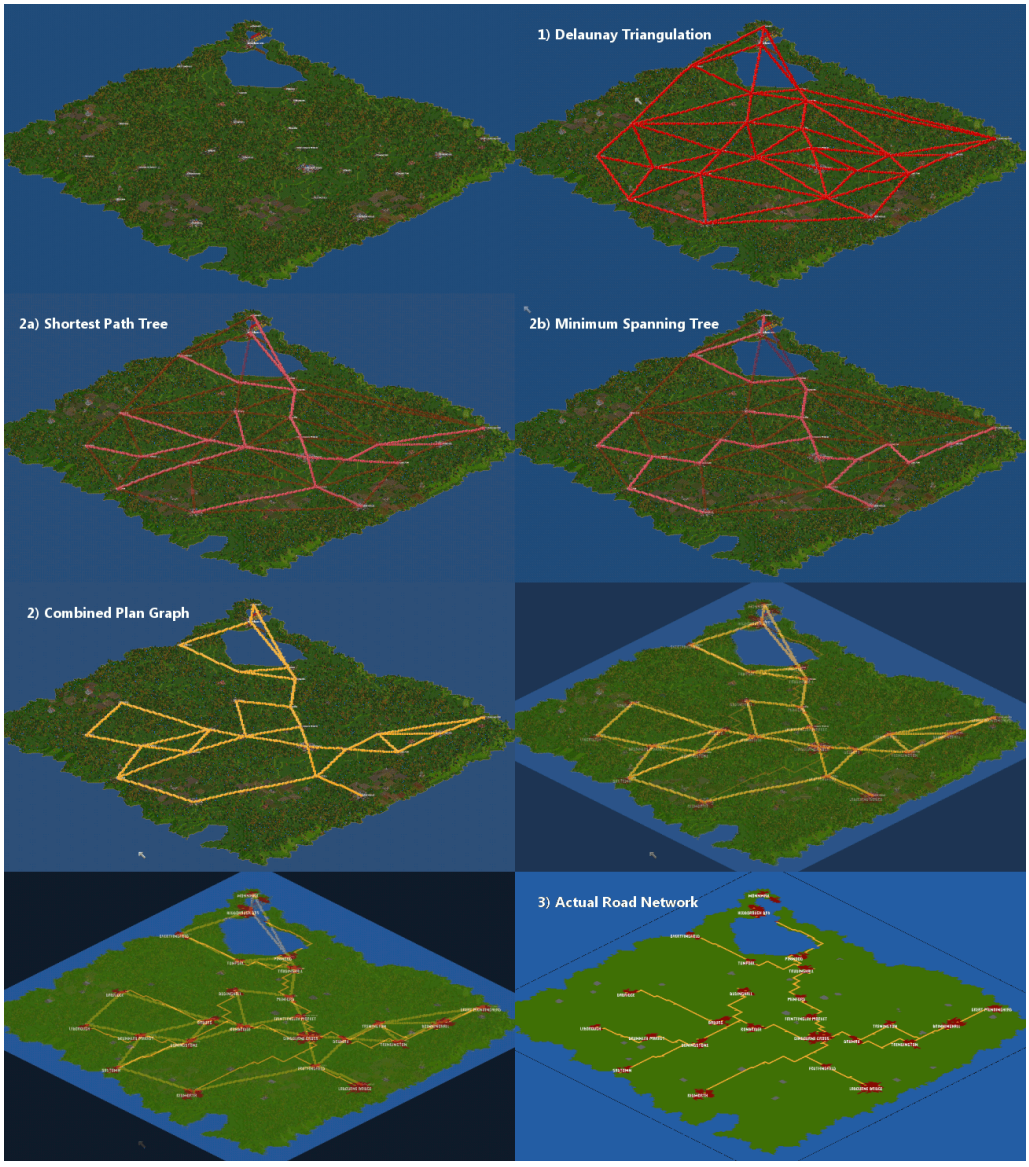

Figure 4.2: PathZilla network planning algorithm
(based on animation from [3]).

- reuse of routes,
- support for local authorities appeal.

Very nice strategic property of this AI is to keep high appeal rating of the local authorities by keeping and maintaining ”green belts” around the cities. Appeal rating is a value describing the local authority approval for the company actions in a particular area. If the rating is low then our building actions may be blocked with a message ”Local authority refuses to allow this”. The local authority rating can be improved by a number of actions:

- plant trees,
- advertising campaigns,
- build statue of company owner,
- bribe the local authority,

How the game mechanics works in the matter of the local authority it is presented with details in [5].

PathZilla AI improves its ratings by planting trees around towns.

4.3.3 Possible Improvements and Conclusions

PathZilla works fine and generates profit. In its selected specialty it has good results. Although, there are several possible improvements:

- re-planning abilities,
- expand and upgrade busy stations,
- upgrade or replace old vehicles,
- other types of transport support,
- on-the-fly landscaping (lands' slopes changes),
- building two-way highways.

This AI has been selected as an interesting approach for connection network planning. If we try to look for a more aggressive AI with road vehicles than there are certainly competitors, but the PathZilla AI uses very professional solutions.

4.4 trAIns AI

The ”trAIns” AI is the most scientific approach for a OpenTTD game. The author is Luis Henrique Oliveira Rios which started his project as a graduation project [22] in 2008 and later it has evolved into a research project [23].

The main aim of the creator was to make an AI which would be a competitive and intelligent, and that can handle railroads. To quote the author: ”A general metric of intelligence for AIs is that its actions and decisions must result in behaviors similar to the ones caused by human players. OpenTTD has a lot of AIs but few can use trains and generate good results according to this metric.” [22].

Several features have been implemented in this context, the main one is the focus placed on railroad development, but with double railways. This imitates construction style similar to a human players. There have been taken care of junctions and complex railway networks which allow railroad parts sharing for cargo transposition to the same destiny.

The so-called ”concentration of production” concept (more details in [7]) has been applied which is a very common human player strategy. It means that the processing industries like e.g. a Power Plant, Factory or Refinery are supplied from multiple sources, not only one. In the case of several types of industries, they create Goods from supplies. Goods can be be transported to towns which accept them.

In this meaning the ”concentration of production” concept has double advantage. First, it allows to centralize the production place, so all resources accepted can be delivered to one place. Therefore the production is centralized, so large number of Goods is generated and concentrated in one place and can be easily transported further. The ”trAIns” AI covers the concentration of supplies.

For improving the production speed game mechanic properties have been used. The AI tries to always keep a reserve train in all stations. Such behavior guarantees that the entire generated cargo is always loaded as fast as possible. When the waiting time is minimal, the cargo production increases.

The ”trAIns” AI has ability to replace old locomotive engines and built the best available. Selection is based on their parameters. Improved and self-implemented A* algorithm allows to plan and build considerably long railroads up to 250 tiles in length. The AI tries to keep straight-lined railways to save train travel time and keep its maximum speed at the highest possible level.

An interesting feature is the way how it creates the routes. It connects industries with themselves and railways that connects towns in pairs. In the case of town connections only passengers are transported (no Mail). Unprofitable routes are detected and demolished. It selects industries based on the current transported cargo rate to chose only these with the biggest potential generation of income.

The author admits on his web-page that his AI has some elements which could be improved:

- there is no upgrade of routes for bridges,
- the junction placement is sometimes surprising,
- old vehicles are not sold,
- trAIns constructs two parallel bridges instead of one,
- there is no support for more then one cargo industry
- trAIns cannot connect industries with cities.

We could add few additional improvements to this list:

- support for acceleration model,
- extension stations development within time,
- more concentrated cargo model, so the production results of one industry are passed to the other,
- combining existing stations in one location as one.

With no doubt the ”trAIns” AI is the most interesting AI developed so far the from scientific point of view. This is mainly determined by its aim to be a graduation project. All its concepts are worth consideration for further use. Unfortunately the author does not provide access to the full content of his project report, the available sources are the personal web page, source code available in BaNaNas and the article [23].

4.5 Admiral AI

The Admiral AI by Thijs Marinussen (alias ’Yexo’) has been developed since June 2008. It is an attempt to implement as many features from the OpenTTD game API as possible. It supports most of the types of transport: trains, road vehicles (including trams) and planes. One of the main goals is to make an AI that is fun to play against, mainly by making it play using several types of transport.

This AI is worth a mention because of the wide usage of the game API. It contains a lot of source code examples used in the right context. As is the case with other AIs we cannot count on the documentation, but it contains some comments in the source code. Unfortunately the comments, when present, define only the methods and functions. We can read what the method does briefly, but there is not much explained inside. If the method/function has 300-500 lines of code it is impossible not to get lost in the context.

We can say that the structure of the AI is well done. All features responsible for particular actions or types of transport are logically divided into classes. Classes and functions have their place in separated files and everything looks very organized.

This AI is a good source of inspiration and examples, but it is necessary to be familiar with the NoAI framework before being able to gain advantage from it. As far as major improvements go more comments could be placed the code.

It is the oldest and most comprehensive AI available and supported by the one of the OpenTTD developers himself. It is well-tested and it is distinguished by proven quality.

4.6 ChooChoo AI

The ”ChooChoo” AI differs from others with the presence of special properties. It has been developed since July 2009 by Michiel Konstapel (alias ’Michiel’). The focus is directed towards network development. Primarily this AI transports only passengers and mail, but the author

Figure 4.3: ChooChoo network. The image from [3].

has placed some updates for transporting cargo instead of passengers and basic bus services to improve ratings.

The main idea of the transport network is to start with a four-way railway crossing at a random location and then extend the network from this point to towns in four directions. Each time the town is not directly on the way a new crossing is built and that is how it is connected with the station. This process continues until the network cannot be extended. If the network cannot be extended any more the new random crossing is built and the process starts over. The visualization of the network which is build by this AI is presented on Figure 4.3.

Another very nice feature of this AI is an implemented ”to do list”. Each task has its own object representation holding a state, which marks if the task has been completed or if it can not then what is the cause. If the cause is temporary (e.g. vehicle in the way) then the action can be resume.

ChooChoo AI is an example of, relatively to others, simple and short implementation of a train-managing AI.

4.7 Rondje AI

The ”Rondje om de kerk” (’Rondje’ in short) AI is a very unique approach. The authors are Marnix Bakker (alias ’Marnix’), Willem de Neve (alias ’Willem’), Michiel Konstapel (alias ’Michiel’) and Otto Visser (alias ’Maninthebox’). Development started in August/September 2008. The initial inspiration was the TJIP Challenge 2008, a competition sponsored by a Dutch company with the following assignment: program an AI to play OpenTTD, using the NoAI

framework. The challenge was to achieve the highest company value within 10 years of gameplay. The competition's finals were on September 20, 2008 in Delft and the first prize was a hardware package of the winner's choice. worth €2,500.

The main aim at the early stages was to abuse or to use the routes built by others. The best description of its basic strategy is summarized by a quote from the original description:

"We first tried this concept manually, with three normal players and one of us leaching the routes the others built." This turned out to be both very profitable for the AI and frustrating for the victims.

Authors described further: "This formed the basis of our AI: we scan the map for serviced industries and towns (by checking for stations in the vicinity), then we follow the roads to determine which ones are connected. All the connected routes then go into a list which is sorted by estimated route profit. From this list we pick the most profitable ones to build our own stations on." [3].

The idea evolved while development. New interesting features has been noticed:

- Passengers are more profitable then cargo, because the transport of them generates profit both ways, while cargo trucks are going one way empty. This led to an idea of selling trucks at the end of the route. There was a small loss, but lower then the loss of the empty truck going whole way back and the vehicle devaluations (around 16% a year);
- The town centers are getting crowded with in time. Authors decided to build drop-off stations (for cargo) in the town outskirts and use the center only for passengers.
- They noticed that the profit was counted as the unloading starts, so they did not waste time for it to finish. Once the profit for deliver has been counted the vehicle has been sent to the depot and sold with whole cargo inside. This idea reduced the delay time spent on unloading significantly, so the stations could handle more vehicles in a time period;
- They have made a research what is generating most of the company value (which was a main competition challenge). After investigating the game source-code they noticed that the stations are worth 10 times more the building cost, after 10 years. In the game there are maintenance cost, so they could not buy properties during the whole game play, because that would slow the company finance a lot. They manage to optimize the exact moment, perfect to sell everything (all vehicles) and invest everything into building the bus/truck stations. Under normal circumstances such strategy in the next year of the game would kill a company, but since the challenge is for 10 years they did not have to care;
- They also noticed that the constructor of the class is not limited to 10000 squirrel VM commands as the rest of the AI implementation, so they placed the most significant computations there, to save as much processing power and computations as possible for other operations;
- Taking the advantage of the unlimited processing power, they preprocess each and every tile on the map determining the best places for the stations at the endgame. Due to the computational expensive checking they limited themselves to the 10 biggest towns for profitable connections.

Authors noticed also that every operation in the game has a cost of Squirrel VM ticks. By checking the game source code they estimated each operation tick weight and minimized the number of ticks by switching more costly operations for less costly. For example instead of building a new vehicle and give it orders it's better to clone an existing one.

Scanning the map for profitable routes of other players is a very costly operation in the context of processing power. The AI has been adjusted not to wait too long and start building vehicles and connections based on a sufficient number of routes found.

Rondje has a specially designed pathfinder specialized in checking connectivity of roads. It was based on the OtviAI pathfinder, additionally optimized in the scope of speed and used VM ticks.

They wanted to hinder the appearance of traffic jams or broken routes to appear, and that's why they came with an idea to verify the success of a route based on the age of the vehicle. By knowing the path, they are able to estimate the travel time. Once the vehicles are "one-use" only, it is easy to just compare the real vehicle age with expected time travel. If the vehicle is much older then it is most likely that there is something wrong with the route. Until it is fixed no new vehicles are added.

The strategy of Rondje is very unique and the authors managed to creatively "utilize" the game's mechanics. Most of the exploits they used were fixed immediately after, but still their approach contains a lot of great ideas.

We were intrigued about this strategy due to its approach to follow other players and compete on the most profitable routes. It is a direct imitation of human player behavior.

Chapter 5

Design and Implementation of SPRING

The majority of existing AIs are specialized in one area or composed of several modules. Each separate module could be separated and handles a piece of a job.

For custom AI development it has been decided to develop a coordinated AI that could base on passenger transport between cities and simple cargo routes between industries and their destinations.

In this chapter the details of custom AI design and implementation will be presented. Not all the designed parts are reflected in the implementation, but this will be discussed later on.

5.1 Separation of Problems

The initial idea was to separate the problem of constructing an AI into three main sub-domains:

- overall coordination of gameplay,
- management over particular transport types,
- financial issues handling (loan control, pay-off control),
- maintenance and vehicle replacement.

Each of these problems' sub-domains consist of another set of problems, each requiring a solution, so the AI could behave in the desired complex manner.

Overall coordination should guarantee that the right connection network is selected and individual transport type managers will not conflict with each other. Specifically, it is important the context of building the infrastructure from the point of view of the expenses. Expenses control the field of gameplay coordination, and should include the most important rule: once the investment in a new route has been started, it should always be completed. If it could not be completed,

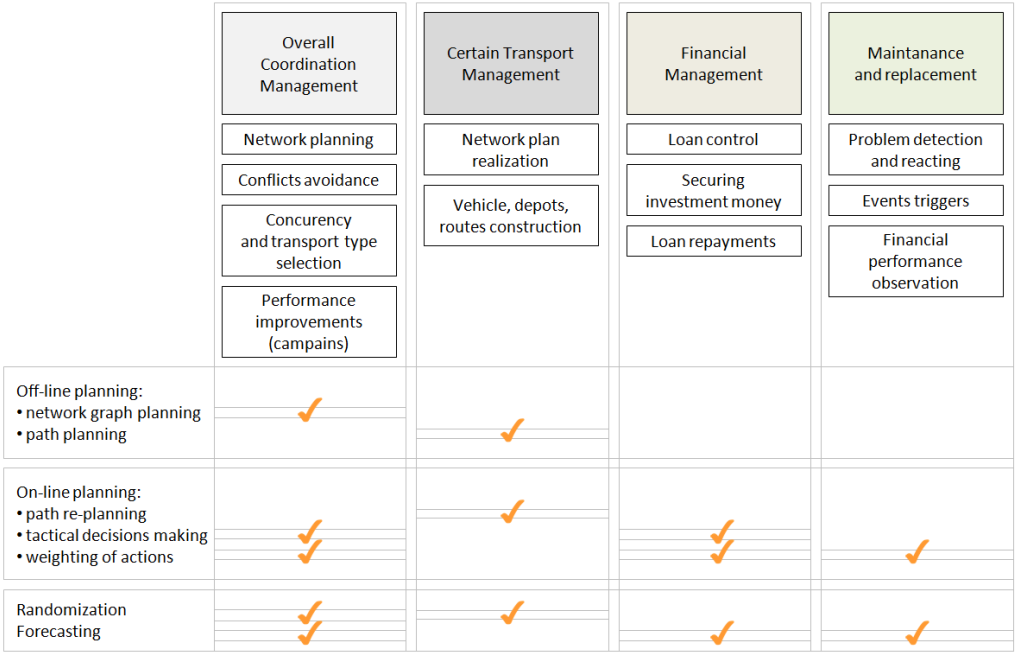

Figure 5.1: Domains of problems and appropriate techniques.

then it should not block or interrupt the whole AI, but there should a mechanism which skips that and returns to the problem later (or tries again with some changes).

The specific transport type management should contain AI algorithms that are able to estimate the route cost, infrastructure shape and location, final pathfinder, vehicle selection, construction and the number of vehicles per the route.

In order to achieve these aims the following techniques from the field of artificial intelligence are considered:

- off-line planning,
- re-planning,
- on-line planning,
- statistic and forecasting,
- path-finding,
- weighting of actions or possible choices,
- randomization.

Detailed descriptions of every aspect are presented in the following sections, with references to actual implementation details. In Figure 5.1 the reader can see a graphical representation of AI design when taking into account the aforementioned problems and techniques.

5.2 Software Design

“Divide and conquer“ has been an inspiration for the design. In the design itself the problems’ sub-domains are also clearly visible.

Let us take a look at Figure 5.2. The design has been directed towards optimizing module separation and independence.

In the following section the role and construction of each element class will be described. In Figure 5.2 only the main design is presented, during implementation many additional classes have been developed. The design fits in the general programming environment structure and the NoAI framework frames.

5.3 Overall Management and Coordination

The overall network is planned in accordance to various transportation types. Road and air transportation is passenger-based. Conversely, trains will be used for transporting cargo only.

Each network consists of three subnetworks, each computed independently for every type of transport. Road vehicles and aircraft are focused on passengers only, so based on the idea from the PathZilla AI the connections are based on a graph in which towns are represented by nodes and the routes are established in between. With the aircraft the control mechanism is based on extending the road routes efficiency or connecting towns with the highest growth potential.

5.3.1 Graph Plan for the Network

Similarly to the PathZilla AI a Delaunay triangulation is computed, but, instead of considering all existing towns, only bigger ones are chosen. A town is considered ”big“ if it contains more then a certain number of citizens. This bound is dynamic based on the towns list ordered by population and its median.

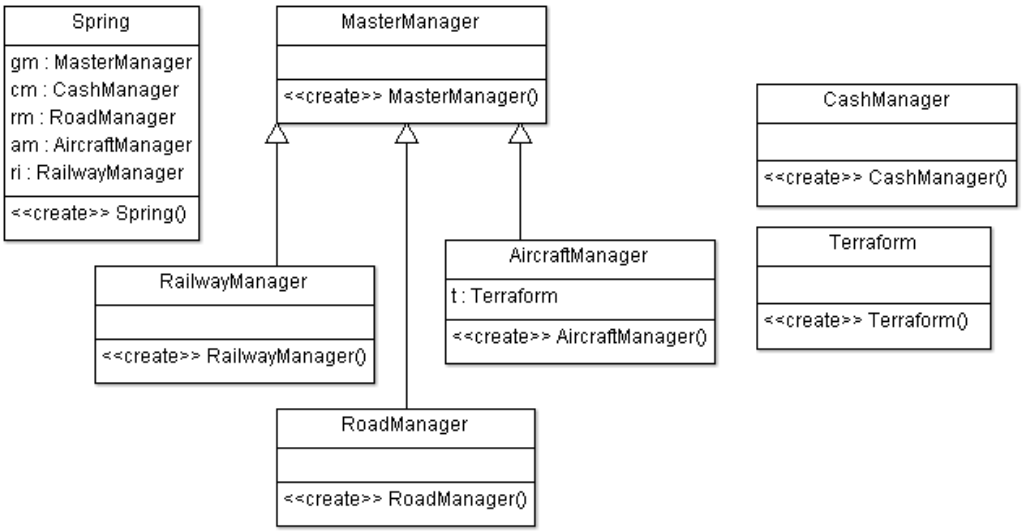

Figure 5.2: The initial class diagram of the AI design model.

On very small maps this approach might not be best, since the number of cities could be very limited. For such cases, as an exception, all towns are considered. For medium and large maps a limit is good practice because it allows us to reduce the computation time and, and the AI can start building much faster.

After making the triangulation, similarly the shortest path tree (SPT definition in [16]) is found with Dijkstra's algorithm. Next, a minimum spanning tree (MST) is computed with Prim's algorithm. Algorithm details can be found in [10]. The union of SPT and MST is a network plan for the RoadManager class. The towns set differs, but the process is inspired by PathZilla AI. A visualization of it is presented in Figure 4.2.

In the case of the aircraft network it was considered to be built upon two schemata:

- Most efficient towns connections: once we a have road network at least partially operational we can notice the number of passengers waiting for pick-up at the bus stops. If the number of waiting passengers is high enough we establish an air route between two qualifying towns.

- Most promising town connections: After the road connection network is computed it contains the minimum spanning tree graph. The idea is to pick up non-leaf vertices and build the aircraft routes upon them. A diagram presenting this approach is in Figure 5.3. Towns selected by this method are the most likely to grow due to at least two connections with others and possibly high passengers traffic.

The first method's advantage is the certainty about the success of the route. Aircrafts are very expensive, so it is wise to build it in a places where the invested funds won't be wasted. The disadvantage is the time necessary for the road network to develop and cultivate towns, additionally it may appear that when we will be ready to build airports there will not be enough space for them.

The second method is time-independent - the AI does not need to wait until the road network is built. If it has enough money it can take a big advantage at the start. The airport locations reserved can be a strategic advantage in the future. This is also much more risky because it may happen that our predicted town growth is not that good as expected or before the traffic became profitable the maintenance costs generate big loses.

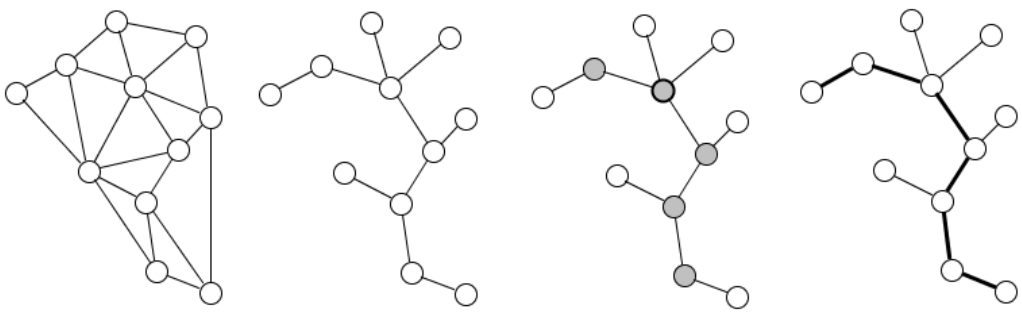

Figure 5.3: Variant of aircraft network planning working under the assumption the selected towns achieve maximum growth.

At the end, the first solution has been chosen to be implemented, because it guaranties profit from the start.

The train network for cargo routes is based on the idea similar to AdmiralAI. The main plan is to connect matching industries by single-train lines and according to the production rates and available engines increase the train's size.

For graph handling PathZilla's existing data structures were used, due to their clear design and well documented sources. Implementation of graph based algorithms have also been based on the existing source code.

5.4 Final Program Structure

The final project structure is shown in Figures 5.4, 5.5, 5.6 and 5.7 in UML notation.

There is no Squirrel tool to support the class diagram generation. Considering the size of the final source code the way to automatically generate the diagram had to invented. The author came with an idea to convert the classes description in Squirrel to Java code. Eventual mistakes were manually corrected.

It is immediately apparent that all attributes and methods are public – this is the only possibility in the OpenTTD dialect of Squirrel. Moreover, in this dialect, although static calls and references are feasible, there exists no way to qualify class members as static; it is decided depending on the context.

The figures contain one more important difference from the actual Squirrel code: native API collection, specific data structures and classes are not present, due to practical reasons. Collections have been substituted by e.g. *int[]* (array), because in practice they are Squirrel arrays of indexes. The reader should be aware of this convention while analyzing the class diagram.

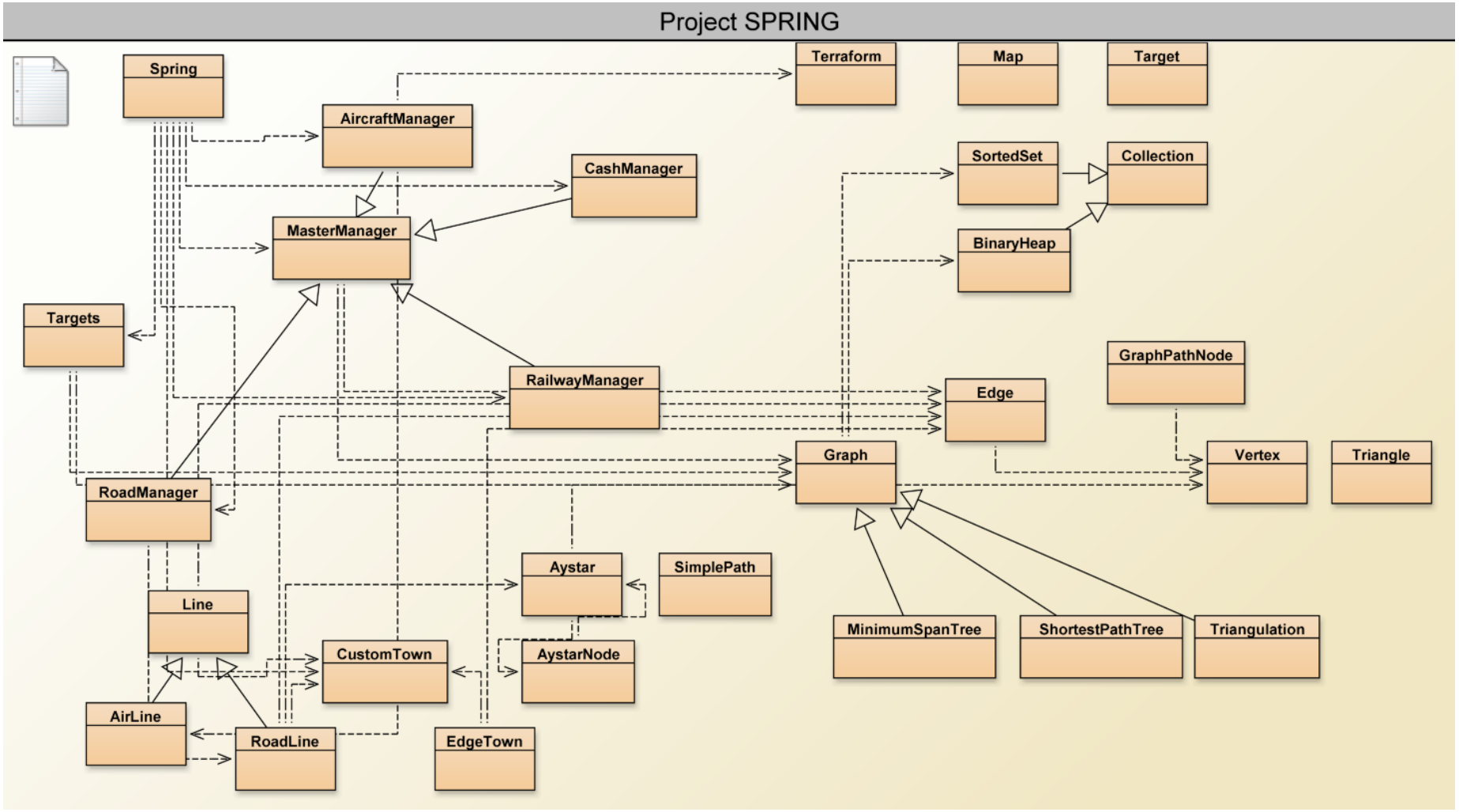

Figure 5.4: Final program structure – class diagram overview with class relations.

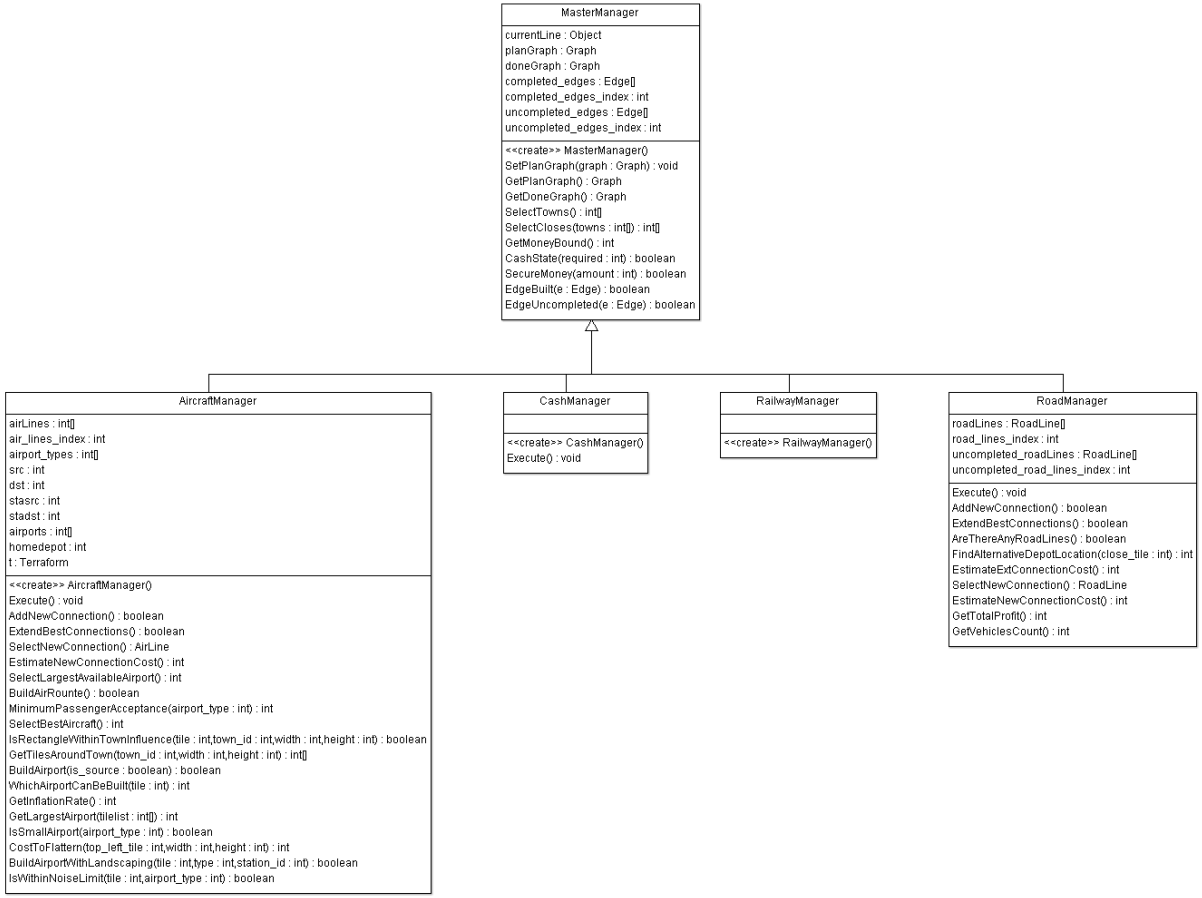

Figure 5.5: Final program structure – class diagram part 1/3:
Main classes, containing operational elements of the AI.

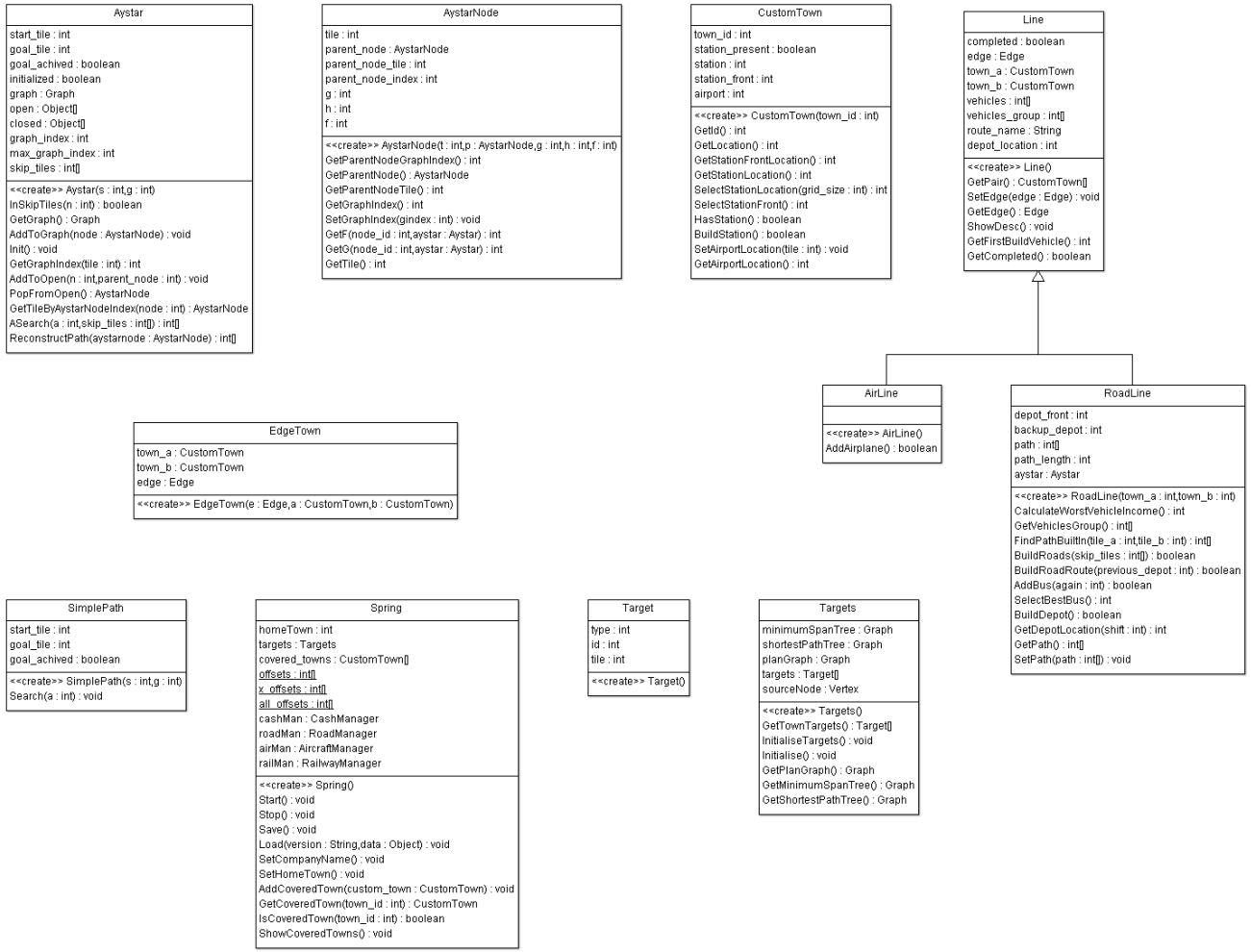

Figure 5.6: Final program structure – class diagram part 2/3:
SPRING main class, A* algorithm implementation and several additional elements.

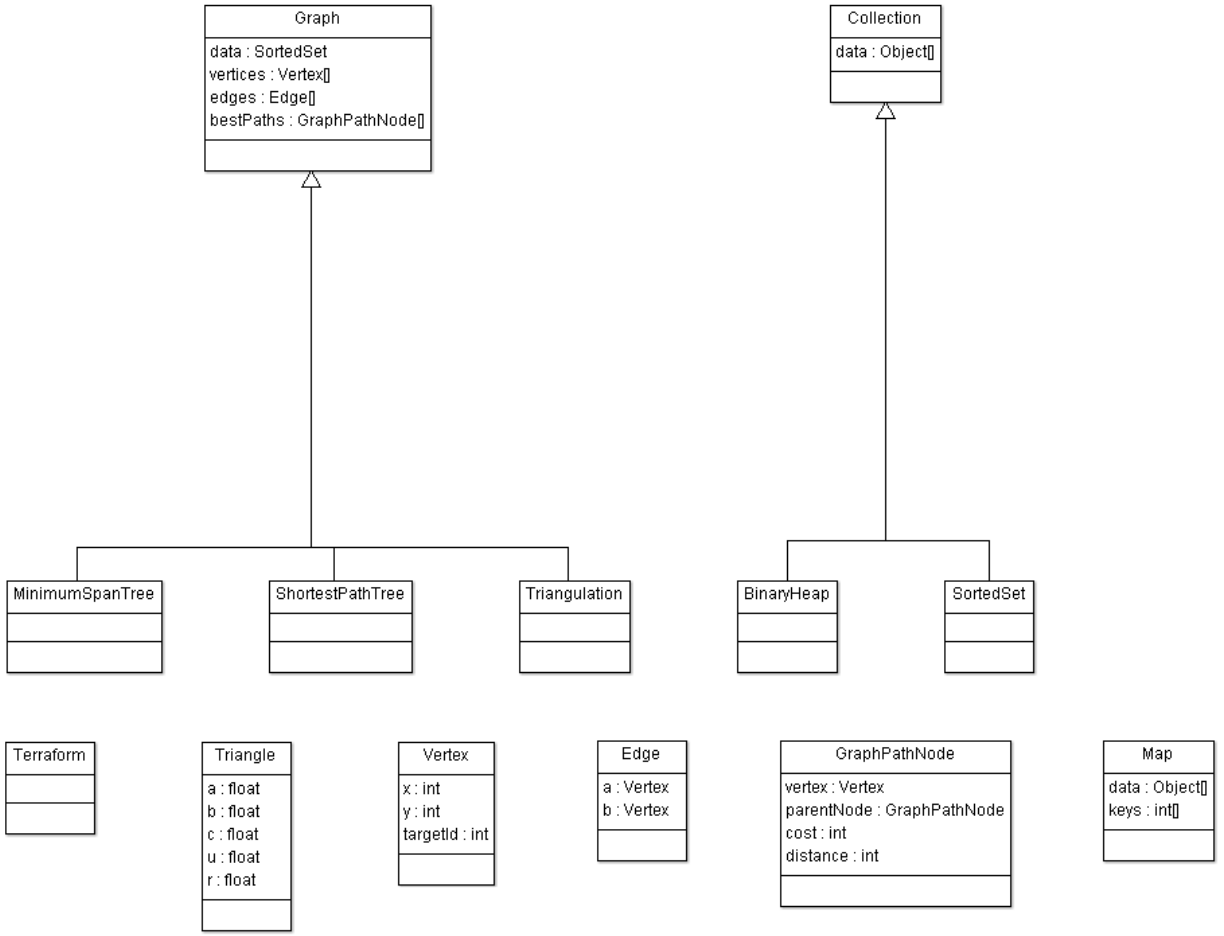

Figure 5.7: Final program structure – class diagram part 3/3:
classes present in SPRING, reused from PathZilla and Terraform class from NoCAB.

5.5 Aircraft Manager

The first challenge for our OpenTTD AI is the management of aircraft and airports. In real life the set of problems related to this area includes a series of managing services, the expertise of which have an impact on safety and costs. The topic of management touches problems like:

- fleet discounts on fuel, insurance and crew services,
- reporting,
- budgeting and regulatory compliance,
- interior and exterior planning, including scheduling and administration,
- bases, hangar spaces, storehouse management,
- crew selection, training and oversight,
- coordinated maintenance.

Aircraft management is a well-developed segment of the commercial market which goes in million of dollars. In OpenTTD aircraft management is presented in a simplified form. The main aim is focused the same way as the game: on cargo delivery, route planning, vehicle acquisition and maintenance and efficient investments in infrastructure. The set of problems is strongly reduced. Because it is the initial chapter which presents the developed solution used in the AI for the game, the description will go in details from the very beginning.

5.5.1 Basic Aircraft Planning

Let us start with the simplest and basic problem of cargo delivering. A similar problem can be found, for example, in [24]. Consider a set of airports, cargoes and planes. Cargoes and planes are located at the airports. Cargoes have destination airport where they should be transported by planes which could be loaded and unloaded. To describe the problem in [24] the PDDL notation has been used.

PDDL, the Planning Domain Definition Language, allows us to express the state of the world by a collection of variables in a comprehensive representation. PDDL is an attempt to standardize planning domain and problem description languages. It has been developed mainly due to the Internation Planning Competitions. It is a successor of the original STRIPS planning language developed by Richard Fikes and Nils Nilsson in 1971. PDDL allows to unify state description and state generation.

With it, it is possible to express predicates and the currently valid logical statements in standard propositional logic form. Also, it allows us to specify possible actions with their preconditions, and results of the actions in the value changes.

Another feature in PDDL is specifying the object interacting in the world, e.g. we write $Plane(P_1)$, $Cargo(C_1)$ and $Airport(JFK)$ having in mind informally that "P_1 is a plane" and "C_1 is a cargo". Similar methods apply to state description sentences, e.g. by writing $At(C_1, JFK)$ we mean that the statement "cargo C_1 is at the JFK airport" is true.

$$
\begin{aligned}
&Init(At(C_1, SFO) \land At(C_2, JFK) \land At(P_1, SFO) \land At(P_2, JFK) \\
&\qquad \land Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2) \\
&\qquad \land Airport(JFK) \land Airport(SFO) \\
&Goal(At(C_1, JFK) \land At(C_2, SFO)) \\
&Action(Load(c, p, a), \\
&\qquad Precond : At(c, a) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a) \\
&\qquad Effect : \neg At(c, a) \land In(c, p)) \\
&Action(Unload(c, p, a), \\
&\qquad Precond : In(c, p) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a) \\
&\qquad Effect : At(c, a) \land \neg In(c, p)) \\
&Action(Fly(p, from, to), \\
&\qquad Precond : At(p, from) \land Plane(p) \land Airport(from) \land Airport(to) \\
&\qquad Effect : \neg At(p, from) \land At(p, to))
\end{aligned}
$$

Figure 5.8: Planning definition example from [24]

Let us take a look on our problem definition in PDDL presented in [24], demonstrated with 5.8.

The initial state definition contains: planes and cargoes; which have their object types specified. The goal has been defined as a state in which the C_1 is present at JFK airport and C_2 at SFO airport. We have three actions at our disposal, which could lead to the goal. These are: load, unload, fly. They have a list of preconditions (states or partial state descriptions) which specify the states where the action could be applied.

The presented definition is an example of a classical planning problem in which the set of objects is known, states are fully-deterministic and do not change until the agent acts. We can simplify this definition and reduce the size of the state tree description by disregarding the negative state values and keeping only positive ones.

This problem description is related to the OpenTTD game if we assume that player is in a full control of planes or cargoes and the players compete separately. Also we should assume that no random events occur in relation to described world.

In other words, it is applicable to our scenario in some special conditions, but let us see if maybe we can make it fit better. If we look closer we notice that in the game airports serve multiple units of cargoes or passengers. What's more the cargoes do not have specific destinations, so basically we don't have to worry where the taken cargo is unloaded if the destination airport is able to accept certain type of cargo. Another important feature of the game is that we do not have to perform a single action of loading and unloading. We are able to set a route between two or more airports and specify what actions are going to be performed at each of them. The game features cover the problem partially, because we can avoid single actions and focus on the right airports connections.

5.5.2 Aircraft Problem Definition

Let us try to modify the problem defined above with the game's features taken into account. First of all, the actions will change, instead of single load, unload and fly, we just set a route by action named 'MakeRoute'. So far we assumed that the airports exist, which is not the case in

the game. In the game we invest and build airports near the cities we like. It would be nice if the planner could make so if the route is under construction between cities with no airports, so the action named 'BuildAirport' is required.

The previous definition also assumes that we have a few planes available at airports. In the game it might happen that we have some spare plane around but first we have to build them so the planner should have an action 'BuildPlane'. For now the problems like selecting the best plane or selecting the set of destination cities is skipped, because these actions do not require the planner to be executed, so they are irrelevant to the problem definition for our planner.

Three actions have been determined: MakeRoute, BuildAirport and BuildPlane. To form a complete definition we need action preconditions, effects and the general goal. The goal would be to build a network of plane connections between selected cities according to some sort of overall plan.

The plan of selecting the cities and generating the path and how they should be connected, i.e. what type of transport should be used, is a matter of the previously discussed Overall Management and Coordination Section. At this point we assume that our planner will get the set of cities and requested routes ready for aircraft transport development. The order of connection setup, the number of planes and other properties will be a result of the heuristic and its economic performance.

We can specify the goal of the planner to achieve the requested network between given cities where the airplanes are transporting passengers. Let us form the goal as a set of statements $RouteExists(from, to)$. We could finish here, but let us assume a situation when we build the airport everywhere possible or we are unable to complete all desired routes set due to e.g. lack of local authority approval. We have money at our disposal, so the natural way of proceeding further would be to set up extra routes for the most efficient connections. The decision of choosing such action brings complications, but this the planner's job to make the right choice based on the provided definition. To create an extra route the action named 'MakeExtraRoute' could be used.

The preconditions should be defined according to the example, so the parameters can be verified and requirements regarding existing infrastructure can be checked. It is hard to cover with the definition a dynamic number of objects (planes), so let it be the case related to the previous one where we would have initially two cities only (the set of cities is constant). The complete definition is formed as presented in Figure 5.9.

Based on the above definition the planner performs a search for the goal. There exists a number of various techniques to estimate the goal and to customize the heuristic to find the most optimal solution. But let us observe how such a definition would work and what state sequence could appear – and whether we are truly able to get from the initial state to the goal.

For the reader the initial state description might be confusing due to its laconic form. The initial state is defined as $City(C_1) \wedge City(C_2)$, but we could write it in a full form as $City(C_1) \wedge City(C_2) \wedge \neg AirportAt(C_1) \wedge \neg AirportAt(C_2) \wedge ...$ and continue with an uncounted number of negative statements. For simplicity we assume that if other statements are negative they can be omitted.

To solve this problem we could ask how the human would behave in such situation? First we would build an airport in one of two cities, so the action 'BuildAirport' would be called.

In the end we get the series of actions and states presented on Figure 5.10.

The example shows that achieving a goal through a series of actions defined in our world is possible. By means of an example presented we see that planning formally works. The next step would be to generate this series of actions based on the given data and the planning definition.

5.5.3 Aircraft Planner Properties

At this stage we have a complete definition and the achievable goal on the horizon. The next step is to make a fast algorithm which could generate a plan (series of actions) optimal for the current situation, considering additional properties of transitions between states. In the game actions like establishing an airport or buying an aircraft cost money.

Cities have various distances, planes use fuel and have a certain ranges. So we have to keep an eye on the budget and evaluate the 'cost' of performing an action. Also an action can be blocked or become inefficient which crossed the border of classical planning and leads to on-line planning (and some sort of interaction with the environment).

In this particular case the planner has a relatively easy task, because the subject of the decision is to basically select the route to be created from the set of planned ones, and execute all required actions leading to it. There is also one more property which has to be considered. We would like to keep the starting actions as cheap as possible, because at the game start our budget is limited.

5.5.4 Aircraft Planner Prototype and Conclusions

Due to implementation limitations in Squirrel related to dynamic code usage a PHP planner prototype has been implemented in a basic version. The source code is available at the report website. The main purpose of the prototype was to check how the planner performs in practical manner and that there is a justified reason to apply such an approach. The goal searching algorithm have not been chosen for their sophistication: an ordinary depth-first search (details in [10]) has been used.

The chosen searching algorithm would not give satisfactory results even with search space limiting. After several runs a few observations has been made:

- even in such a simple example a large number of states is present;
- there are repeating patterns of the same actions order (permutations of actions with the same effect).

This leads to the conclusion that the classic planning approach is not the method which fits our domain. Indeed, if we imagine a problem description covering more then the case presented above the number of possible actions and the search tree grow enormously.

The direct solution is to gather the obviously ordered sets of actions as shown in Figure 5.10. This ordered set of actions can be treated as a subgoal state. This state means that the route between C_1 and C_2 has been successfully established.

If we think about the subgoals required by the aircraft manager in the matter of actions planning, they all consist of establishing routes between pairs of points.

$Init(City(C_1) \wedge City(C_2))$
$Goal(RouteExists(C_1, C_2))$
$Action(BuildAirport(c),$
$\quad Precond : \neg AirportAt(c)$
$\quad Effect : AirportAt(c)$
$Action(BuildPlane(c),$
$\quad Precond : AirportAt(c)$
$\quad Effect : Plane(P_x) \wedge PlaneAt(P_x, c) \wedge \neg PlaneOnRoute(P_x)$
$Action(MakeRoute(p, from, to),$
$\quad Precond : AirportAt(from) \wedge AirportAt(to)$
$\quad\quad \wedge \neg RouteExists(from, to)$
$\quad\quad \wedge (PlaneAt(p, to) \vee PlaneAt(p, from))$
$\quad\quad \wedge \neg PlaneOnRoute(p)$
$\quad\quad \wedge City(from) \wedge City(to) \wedge Plane(p)$
$\quad Effect : RouteExists(from, to) \wedge PlaneOnRoute(p)$
$\quad\quad \wedge \neg PlaneAt(p, from) \wedge \neg PlaneAt(p, to)$
$Action(MakeExtraRoute(p, from, to),$
$\quad Precond : AirportAt(from) \wedge AirportAt(to)$
$\quad\quad \wedge RouteExists(from, to)$
$\quad\quad \wedge (PlaneAt(p, to) \vee PlaneAt(p, from))$
$\quad\quad \wedge \neg PlaneOnRoute(p)$
$\quad\quad \wedge City(from) \wedge City(to) \wedge Plane(p)$
$\quad Effect : RouteExists(from, to) \wedge PlaneOnRoute(p)$
$\quad\quad \wedge \neg PlaneAt(p, from) \wedge \neg PlaneAt(p, to)$

Figure 5.9: Aircraft management planning definition

Action	**State**
Init	$City(C_1) \wedge City(C_2)$
$BuildAirport(C_1)$	$City(C_2) \wedge City(C_2) \wedge AirportAt(C_1)$
$BuildAirport(C_2)$	$City(C_2) \wedge City(C_2) \wedge AirportAt(C_1)$ $\wedge AirportAt(C_2)$
$BuildPlane(C_1)$	$City(C_2) \wedge City(C_2) \wedge AirportAt(C_1)$ $\wedge AirportAt(C_2) \wedge Plane(P_1)$ $\wedge PlaneAt(P_1, C_1)$
$MakeRoute(P_1, C_1, C_2)$	$City(C_2) \wedge City(C_2) \wedge AirportAt(C_1)$ $\wedge AirportAt(C_2) \wedge Plane(P_1)$ $\wedge PlaneOnRoute(P_1)$ $\wedge$ **RouteExists(C_1, C_2)**

Figure 5.10: Series of actions and states

Then the conclusion is that the planner should not waste computation time and memory to plan every single action, but focus on the subgoals of choosing the best route to build.

As the situation on the map is changing rapidly, and while the AI is running in the loop its decisions should be based on the current situation, not on the initial game state.

This clearly shows that the classical off-line planing is not the best solution and the subgoal should be chosen and achieved based on the on-line evaluation. Planning-in-advance capabilities should be limited to finding construction locations or road path finding algorithms as it is done in the case of Road Manager described in the next section.

5.5.5 Aircraft Management Implementation

The fact that the classical planning approach has not been found suitable does not mean that the module is not functional. Instead of implementing a complete action planner the solution with preordered actions has been chosen.

A procedure for selecting pairs of towns for airport construction has been implemented. The pair is selected according to population size. The best airplane to operate on the route is chosen by calculating a special rate from available plane properties:

- speed,
- reliability,
- running cost,
- capacity,
- max. age – maximum vehicle life-time,

The selection rate is estimated by:

$$\text{score} = (\text{capacity} * \text{speed} * \text{max. age})/(\text{running cost} * \text{relability} * 100).$$

The highest-scoring vehicle is accepted to be the best one. It is not a perfect selection method, because we do not explicitly consider the number of passengers on the route. But this data can be acquired only after some vehicle completes the route several times. The differences in running cost are not that high, so the really important properties are capacity and the speed.

Some parts of SimpleAI and PAXLink have been reused in our code. A possible improvement for the Aircraft Manager could be an algorithm switching the airplane for a more suitable one after a certain time.

5.6 Road Manager

The second big challenge for our OpenTTD AI is the management of roads and road vehicles, in our case buses. The related real-life problems and challenges are perceptible everyday when waiting at a typical bus stop. Besides concerns about the same elements as in the aircraft case: fleet maintenance, human resources, fuel, insurances, etc., we need to face additional problems:

- route planning,
- choosing bus stop locations,
- pathfinding for vehicles,
- frequency of picking up passengers, waiting time and general schedule,
- vehicle utilization,
- shorter vehicle lifetime,
- on-road failures,
- road traffic,
- high market volatility.

Road transport is a very common mode of transportation in the world. Its properties are high flexibility, relatively low cost, but also low capacity.

Rad transport is probably, next to railways, the most challenging transport type in OpenTTD. In the following subsection we will try to provide solutions to selected problems and describe their implementations.

5.6.1 Station and Depot Localization

The initial problem is choosing which route should be built first. Because we would like the route to bring profit with high probability, pairs of largest towns should be selected first as initial routes.

The next problem is bus station localization in towns. In the game there are several bus stop/station types. It has been decided to use only one type which is a single tile bus station, oriented towards the closest road. It is not built on the road, so the problems related to road ownership are omitted.

To find a location for a station, we select the closest possible location to the town center tile, remembering to keep the *CC_PASSENGERS* cargo type acceptance positive.

We select a close surrounding in some certain radius and trying to find the best suitable and available place. If this procedure fails the radius is extended and a second search is performed. If this fails again then the route is put on the uncompleted routes list and the next one is chosen.

Once we have locations of stations chosen we can try to construct a connection between them by finding an existing road or building a new one.

5.6.2 Road Pathfinder

A very important element of OpenTTD is the problem of path-finding. This refers to the road and railway building process. We can immediately determine the goals of the route, i.e. its destinations, but the latter need to be connected correctly in the most efficient way. Most efficient

means the cheapest to lay down, but also short enough so the vehicles do not waste additional time for travel.

In general the shortest path between two points is a very important element in almost all computer games. The easiest solution would be to let the vehicle drive in a straight line between the points, but this is not always possible.

First of all, the game's world is based on square tiles. Every element is placed on the tile so the paths also can go only between tiles if they are direct four-consistent neighbors. The four-member neighborhood also influences the basic distance measurement in the game, which is the Manhattan metric (more details can be found in [15]).

Additionally, the pathfinding problem is not trivial due to the obstacles. In the case of OpenTTD these obstacles can be divided into two groups: "natural" - caused by the map generation and usually modifiable (surmountable); or "unnatural" - unmodifiable (unsurmountable).

The surmountable obstacles are:

- Terrain slopes, which often make the connection of two pieces of a road or a railway impossible to make. The example visualization of this problem is presented in Figure 5.11. This could be neutralized by terrain alignment, but it has to be determined how it should be changed and what to do with the inevitable additional costs.
- Water tiles, which makes it impossible to build a road or a railway in a direct path. Can be modified by raising the terrain above the water level, which is very expensive or by building bridges. Building bridges is quite complicated due to special properties of the terrain on the ends (coast lines should be of equal elevation) and sometimes unexpected distances achieved from the second side.
- Town buildings. They cab be demolished if the local authority approves, but it is not recommended because it decreases the the authority ratings for the company and additionally lowers the town's population.

The unsurmountable obstacles are:

- Industries, which can not be removed, they can be only bypassed.
- Other player's infrastructure. In the case of roads they can be reused, so our vehicles will just use any available roads. Trains can not do that, so we need to build bridges above or tunnels underneath.

To simplify let us assume that natural obstacles appear on the map, but in limited amount. Such setting can be created by selecting, in the map option window, "Flat" or "Very Flat" for the "Terrain type" parameter and "Very Smooth" for the "Smoothness" parameter. To be sure that the water will appear only in a form of lakes the parameter "Sea level" should be set to "Very low" or "Custom" with value not bigger the 2%. Our map should not contain islands which could prevent the described version of the pathfinder from working.

Regarding the unsurmountable obstacles, as a simplification, we will treat them as things which should be somehow bypassed.

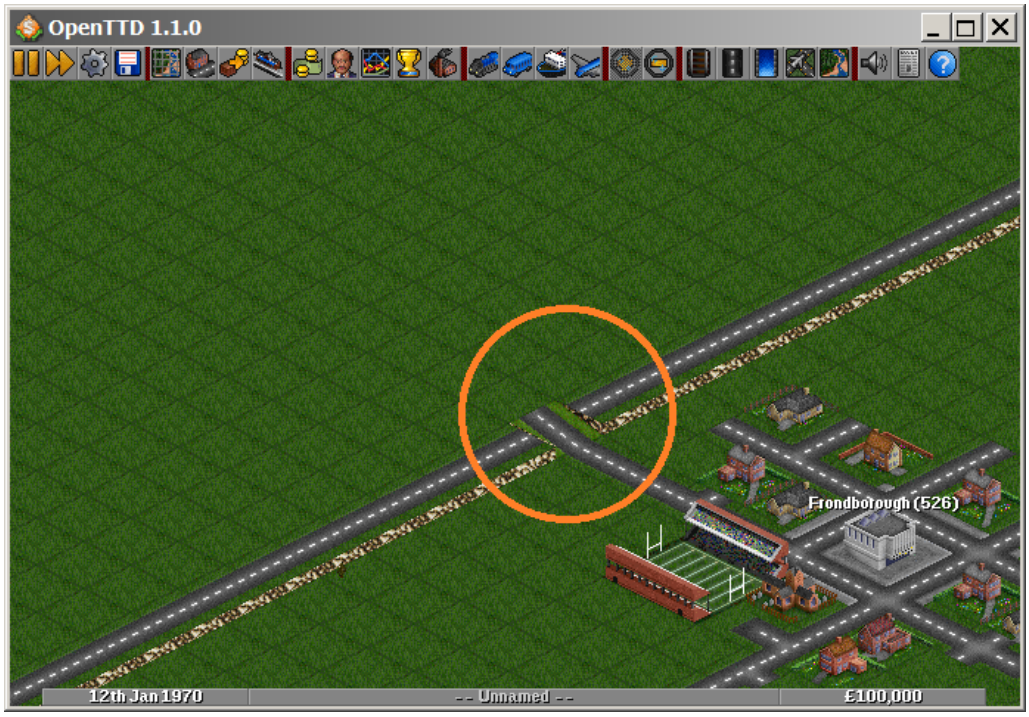

Figure 5.11: A terrain slope obstacle example. A road has been build considering the neighbored and other roads existence, but the road connectivity flaw appear caused by the terrain slope.

5.6.3 A* Algorithm

The A* search algorithm is an algorithm widely used in pathfinding. A pathfinding problem is the problem of detecting the best possible route between two points in a graph. It was described first by Peter Hard, Nils Nilsson and Bertram Raphael in 1968 [20, 24]. It is an extension of Dijkstra's algorithm from 1959. A* has an advantage in better time performance and in using plug-in heuristics. If we use monotone heuristics the algorithm is equivalent to Dijkstra's.

First, we assume that nodes in our initial graph represents the map tiles. Relations between nodes are graph's edges. If the tile is an obstacle the transition between tiles cannot be performed. The relations between tiles are in four-neighborhood. The construction of roads in the game requires the property of direct neighborhood between tiles. By direct neighborhood we understand the adjacency of the tiles' edges.

The explanation of the A* algorithm based on [20] will be presented below.

The algorithm accepts as input a set consisting of nodes with relations between them (representing the entire map, i.e. our search space), the starting node n_0 and the expected goal node n_g.

1. We start with a graph G, consisting of a start node, n_0. Additionally we need two lists, first called OPEN and the second called CLOSED. Initially we put n_0 in OPEN, leaving CLOSED empty.

2. If OPEN is empty the algorithm should exit returning false. It means that the path can not

be found.

3. If it is not empty we pick up the first node in OPEN, remove it from OPEN and put it in CLOSED. Let us called this node as n. During the first iteration n is the same as n_0, i.e. our starting point.

4. If n is a goal node, $n = n_g$, the algorithm can end with success and return the solution by tracking a path along the pointers from n_g to n_0 in G. The pointers define a search tree and are assigned during the neighbors' scanning phase (point 6.).

5. We expand node n, which gives us a set M, of its successors that are not already ancestors of n in G. We add these M members as successors of n in G.

6. We select those members of M which are not already in G (i.e., not already in either OPEN or CLOSED). We add them to OPEN and for every one we create a pointer from each member of M pointing to n.

7. For each member m, of M which has been on OPEN on CLOSED already, we redirect the pointer to n if the best path found was through n.

8. For every member of M present in CLOSED, redirect the pointers of each of its descendants in G so they point backward, and that creates the best paths along found so far to these descendants.

9. Sort the list OPEN in order of increasing f values.

10. Repeat again from point 3.

The aforementioned value f is a special one, weighting the nodes according to their usefulness or shorter distance. In our case these value will not only represent distances, but the cost of the path up to these node as well.

Let us first take a look on how f is constructed in the classic A* definition:

$$f'(n) = g(n) + h'(n),$$

where:

- $g(n)$ is the total distance it takes to get from n_0 to n.
- $h'(n)$ is the estimated distance from the current position to the goal destination. A heuristic function is used to create this estimate, from n to the goal n_g.

$f'(n)$ is the sum of $g(n)$ and $h'(n)$ and $f'(n)$ represents the estimated shortest path for current node n, until the real $f(n)$ which is the true shortest path is determined by the A* algorithm.

An important element of the algorithm's implementation is the rejection of inappropriate tiles at the early stage of expanding the OPEN set for the current tile's neighbors. This subprocedure not only encompasses checking OPEN and CLOSED sets, but also all neighborhood tiles for the fulfillment of the following conditions:

- have to be valid map tiles,
- are not obstacles (can be buildable or are connectable roads already),
- they are not water or coast tiles,
- they are not part of a bridge; if they are a part of a bridge with a road transport type, a special routine is executed which makes the algorithm ”jump” to the other side of the bridge,
- the terrain slopes have to match when the road is built (tested with *AITestMode*),

It is possible to verify more conditions, but the aforementioned ones have been determined as sufficient. The conditions cannot be too strict, because they may unnecessarily complicate road shape, which should be kept as straight as possible in order not to slow down the vehicle.

5.6.4 Selection of the Heuristic Function

The most difficult element of implementing A* is choosing a good heuristic function $h'(n)$. Heuristic functions vary from implementation to implementation, because it is an estimate and is not usually provably correct. An algorithm forms a set of steps which can be verified and proven for a particularly given set of input.

In A*, the heuristic function plays an important role, the better it is the faster the algorithm runs and the better the solution itself is.

In the case of road construction the heuristic $h'(n)$ could just be the Manhattan distance between the node n and the goal node n_g. That would work with the assumption that no obstacles appear and the cost of each tile on the path is equal.

Unfortunately obstacles exists and the cost can be different for each tile, especially if we reuse existing roads. The usage of existing roads does not require to pay for the construction, so the real cost of reuse is equal to zero. If we accept that it is really a zero this can lead to a situation present in Figure 5.12. A conclusion is that accepting the zero price for reused road is not a best solution, because that will increase the time of traveling of a road vehicle. Which decreases the profit afterwards.

5.6.5 Data Structures and Performance Improvements

For handling graphs and A* sets various data structures have been devised. Simple lists are based on a class presents in the NoAI framework, *AIList*. It has built-in evaluator and sorting functionalities. Planning graph data structures have been based on the same principle as the general planning of the network. A* has been implemented by the author. The implementation is not perfect among those in the filed of algorithmic optimization, however. Some possible improvements suggested by the author would be:

- limit the search space of the map by cutting a portion of it between the starting and goal points.

Figure 5.12: Examples of an A* algorithm result with heuristics differently prioritizing existing roads reuse. (A) The route is directed towards a town. (B) The variant is based on reuse of existing roads and minimalization of the expenses. (C) The variant is taking into account the eventual vehicle travel distance. The reader can notice that the actual distance for a traveling vehicle is longer in the case (B) then in (C).

- OPEN lists based on a binary heap [16] to speed up the sorting and first element peek based on the $f(n)$ value.
- Clustering the tiles of the map. Instead of searching each tile cut off bigger areas where the path is obvious.
- Specialize into two or more pathfinding systems depending on the path length or map situation;
- Consider preprocessing of the map and the calculation of universal paths, which could be later reused;
- Marking checked tiles which cannot lead anywhere as ”dead-ends”.
- Better memory management: cleaning up the used objects from the memory after each path found could be improved by keeping some selected information regarding tiles for later runs.

Considering the modular structure of the source code, and provided a better understanding and gained experience with Squirrel programming and OpenTTD API, the list of these improvements could be applied if some additional time was available.

5.6.6 Expectations and Results

The road manager module as implemented and ran on its own meets AI requirements for the default game settings with a flat map. It is able to create road connections network between cities, successfully establish bus routes and transport passengers. It is able to create a profit for the company and extend existing and most profitable routes by adding new vehicles.

In very special and rare cases the pathfinder experienced some connectivity flaws on terrain slopes. To make our AI fully reliable the implementation has been extended. In the final version it contains a second variant of A*. The alternatives are switchable in the "AI Settings" window. One is the author's A* implementation described previously, the second being a library based on the road pathfinder available in OpenTTD's API extensions.

5.7 Railway Manager

The railway management list of tasks is similar to the ones for aircraft and road transport. The biggest difference is the infrastructure complexity involved and vehicle control – this applies both to simulation and "real life".

Creating a railway requires designing a network of connections with reliable train control signals. Deficiencies in signal and/or train crossing designs may result in train damage, contrary to aircraft or road traffic (in the game), where vehicles are not that much dependent on traffic control mechanisms.

5.7.1 Design Concept

The railway manager been only partially implemented. The inspiration behind it is AdmiralAI's simple strategy to connect industries directly with each other using a single train line.

The selection of industries should be based on distance first (according to 2.2), and on production rates of the supply industry second.

With the current software design presented in Section 5.4, extending SPRING AI with railway management could be quite simple. The modular structure allows us to combine other train-specialized AIs with SPRING, as a possible future development.

5.8 Randomization

For SPRING, randomization has been considered in the context of:

- route build order,
- route planning,
- choice of actions and vehicle selection,
- defining infrastructure shape before its built,
- other applications.

Randomization of the build order has a practical meaning when we run several instances of the same AI. By choosing routes to build randomly, we change the order so that the companies are not duplicating themselves.

For route planning, it is applied in situations when two equivalent solutions are present.

Randomization of action choice appears partially possible, but such randomization cannot conflict with decisions based on economic factors.

Infrastructure shape determination has been randomized by choosing bus station orientation randomly if there is no road nearby. In such a case, there is no optimal way of deciding in which direction the bus station should be pointing to, so randomization of this decision introduces some variety (not all the stations point in the same direction). The decision may increase or decrease a vehicle's travel path, but has only marginal impact on the overall results.

Randomization has an application in all cases when actions we reason about seem to have equivalent results. Moreover, it can be used for estimating costs of planned routes. Based on terrain properties, we can randomize the spread in certain probability bounds.

Unfortunately the number of situations in the game where the decision process can be randomized is limited, for practical reasons.

5.9 Project Website

The source code of the AI is freely available on the Internet. For the purpose of the project a website has been created with the following URL:

`http://www.student.dtu.dk/~s090898/spring/`

Its contents can be summarized as follows:

- source code of SPRING AI and the prototyped planner,
- the Java source code used for class diagram generation,
- a short tutorial entitled: "How to run an AI in OpenTTD?",
- hyperlinked bibliography.

By the time of publication of this report SPRING should also be available through the OpenTTD game interface, from the BaNaNas repository. An application for inclusion has been sent and the relevant procedure has started. If it is successful, a notification regarding this fact will appear on SPRING's website.

Chapter 6

Experiments and Results

In this chapter the chosen test case is described. The implemented AI has been run several times and the results are presented. A comparison with other selected AIs is presented. Finally, possible improvements are discussed.

6.1 Preface

The implementation of the author's AI is working. It generates profit, but it does not cover all possible situations. It is important to underline that the best and most popular AIs available for the game have almost two years of development behind them. Sometimes the main source code has not changed substantially during this period, but improvements added to fix the behavior in certain situations grant them an advantage.

Some situations are very hard to predict and detect in advance. That is why having an extensive testing phase is so important in AI development. Available AIs are tested by users in the long run. Any flaws detected can be posted on a discussion board or in a bug report system. By investigating historic data we can observe substantial activity in this context. Most of the problems are related to:

- network design and planning,
- pathfinder effectiveness,
- roads, rail and train route construction and obstacle avoidance,
- location-choosing algorithms,
- transport amount balance,
- vehicle number and road traffic overload management,
- economic balance,
- reuse of existing infrastructure.

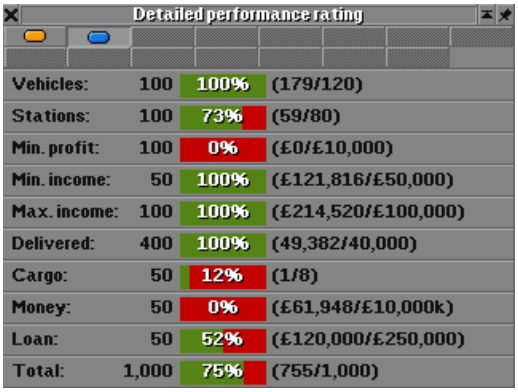

Figure 6.1: Results of test session for SPRING.

More specific examples could be given. The importance is to note, that even after a long testing period, flaws in even the oldest and most comprehensive AIs are still detected and reported. This may lead to a conclusion that no matter how much testing is done, in any complex artificial world, where several AIs and players are interacting with each other, it is very hard to achieve perfect results in all possible cases.

6.2 Test Case

In the previous section it has been stressed that perfect results in every situation are hard to achieve. The testing time is, by necessity, limited and not all possible situations in the game are covered. An AI implementation is a complex task and thesis research time is limited.

The test case has been chosen to limit the possibility of AI failure. Our test aim is to see the economic growth of AI, not to check it for conformance to all API features. The difficulty level ”Medium” has been chosen. Before map generation the following properties have been modified:

- Difficulty Level: Medium;
- Landscape: temperate;
- Size: 512x512;
- Date: 1st Jan 1960;
- No. of towns: High;
- No. of industries: Low;
- Terrain type: Very Flat;
- Smoothness: Very Smooth;
- Sea level: Custom (2%);
- Disasters: Off.

Changing the properties forces the difficulty level to ”Custom”, but we still keep the other values unchanged relative to the ”Medium” difficulty level.

Year:	**1965**
Company age:	5 years
Company value	£133,098
Road Vehicle Income (last year):	+£109,138
Vehicles:	33
Stations:	15
Performance rating:	188
Year:	**1970**
Company age:	10 years
Company value:	£1,078,128
Road Vehicle Income (last year):	+£485,658
Vehicles:	125
Stations:	35
Performance rating:	536
Year:	**1975**
Company age:	15 years
Company value:	£2,552,978
Road Vehicle Income (last year):	+£1,096,449
Vehicles:	179
Stations:	59
Performance rating:	755

Table 6.1: Summary of an example testing session with SPRING.

6.3 Testing

The example results achieved by the SPRING running in an isolated fashion during a test game session are presented in Figure 6.1. During 15 game years of the company's existence, the controlling AI was able to obtain a substantial financial reserve and create a complex road network. The observed results can be found in Table 6.1.

It is readily apparent that the AI is able to generate profit and ensure self-sustenance. In the next section we will try to see how it behaves as a competitor.

6.4 Competitor Selection and Experiment Results

AIAI has been chosen as the competitor, as it has similar priorities (buses and airplanes) and it is not as much competitive as the other road-transport-based AIs. Also, its development time is not that extensive as in other cases – it has started in January 2010. Three game matches have been run with the same test case settings. The results can be observed in Table 6.2

	Match #1		Match #2		Match #3	
Parameters	**AIAI**	**SPRING**	**AIAI**	**SPRING**	**AIAI**	**SPRING**
	Year: 1965					
	Company age: 5 years					
Company value	£518,363	£178,084	£269,830	£154,327	£556,453	£409,030
Road Vehicle Income:	+£153,108	+£87,464	+£143,831	+£111,293	+£257,291	+£242,167
Average income per vehicle:	£2,469	**£3,123**	£2,765	**£3,837**	£3,298	**£4,248**
Vehicles:	62	28	52	29	78	57
Stations:	7	13	6	11	11	19
Performance rating:	200	168	178	**188**	268	**331**
	Year: 1970					
	Company age: 10 years					
Company value:	£1,513,544	£876,000	£899,276	£779,771	£1,572,000	£1,379,939
Road Vehicle Income:	+£618,056	+£427,745	+£355,309	**+£367,612**	+£605,757	+£588,844
Average income per vehicle:	£4,148	**£5,780**	£3,116	**£4,324**	£2,804	**£4,206**
Vehicles:	149	74	114	85	216	140
Stations:	26	31	17	31	28	43
Performance rating:	463	**472**	344	**452**	474	**648**
	Year: 1975					
	Company age: 15 years					
Company value:	£5,645,707	£2,594,946	£3,096,809	£2,826,035	£4,546,405	£3,502,605
Road Vehicle Income:	+£1,808,148	+£1,015,030	+£1,163,310	+£1,189,007	+£1,486,883	+£883,182
Average income per vehicle:	£3,531	**£4,413**	£3.102	**£5,608**	£3,304	**£5,589**
Vehicles:	512	230	375	212	450	158
Stations:	54	66	50	62	66	45
Performance rating:	690	**758**	530	**786**	715	**774**
Loan:	-£250,000	**-£130,000**	-£230,000	**£0**	-£200,000	**£0**
Bank balance:	+£1,486	+**£56,782**	+£23,543	+**£1,350,234**	+£51,512	+**£2,513,294**

Table 6.2: Summary of the competition between SPRING and AIAI.

Parameters	AIAI	SPRING
	Year: 1980	
	Company age: 20 years	
Company value	£6,551,249	£6,031,256
Road Vehicle Income:	+£918,517	+£685,542
Average income per vehicle:	£1,882	**£3,188**
Vehicles:	488	215
Stations:	60	61
Performance rating:	739	**805**
Loan:	-£80,000	**£0**
Bank balance:	+£25,270	+**£4,677,672**

Table 6.3: Results of the match #2 continued till 1980.

If take into closer look on all of the parameters and take into consideration the game performance rating we see that SPRING achieved better results. AIAI managed to achieve higher company value, but other parameters were worse. AIAI has two important features:

- 'drive-through' bus stations,
- multiple bus stations per city.

This approach improves the traffic flow and helps to avoid traffic jams in front of the stations. AIAI created a lot more vehicles which increased the company value significantly, but their utilization was much worse then in SPRING case.

SPRING always shows much better average vehicle income and ability to collect big amounts of money. How big is the difference we can observe in Table 6.3.

The collected money could be spent on infrastructure or new vehicles. Although, the ability to save money like that could have a practical meaning in real-life applications, when we are oriented more towards getting a revenue.

6.5 Comparison with Highly-Competitive AIs

To show the distance between ordinary and highly-competitive AIs optimized for score-based games, a match between 'the best' has been arranged. The results are presented in Table 6.4. The value of few years development and testing experience can be immediately noticed. Moreover, the developers' OpenTTD playing experience has not been without significance.

Parameters	**AdmiralAI**	**PathZilla**	**ChooChoo**	**PAXLink**	**CluelessPlus**	**NoCAB**	**Convoy**	**RoadRunner**	**trAIns**
	Year: 1965								
	Company age: 5 years								
Company value	£191,531	£64,214	£1,403,750	*bankrupt*	£5,880,279	**£12,583,384**	£849,885	£179,566	£388,535
Vehicles (all types):	35	25	46	-	284	109	127	35	7
Stations:	6	13	42	-	44	54	10	5	1
Performance rating:	128	100	419	-	617	**744**	370	110	144
	Year: 1970								
	Company age: 10 years								
Company value	£658,113	£103,495	£3,078,629	-	(!) £13,015,974	**£18,242,386**	£3,294,057	£594,876	£3,459,404
Vehicles (all types):	75	34	85	-	326	438	313	87	20
Stations:	17	19	95	-	57	158	28	8	9
Performance rating:	256	143	666	-	825	**866**	612	206	426
	Year: 1975								
	Company age: 15 years								
Company value	£1,770,544	£185,978	£6,919,613	-	-	**£67,287,808**	£6,527,013	£1,933,311	£8,746,365
Vehicles (all types):	172	55	130	-	-	620	308	210	28
Stations:	24	22	136	-	-	209	26	24	10
Performance rating:	363	187	825	-	-	**860**	611	452	383

Table 6.4: Summary of the match between highly-competitive AIs.
(”!” indicates that AI stopped, because of an error)

6.6 Summary

In this chapter we showed that the created AI works and creates revenue. It is balanced more to creat the profit then increase the company value.

The task of developing a reliable AI for a computer game is not a trivial one. Many circumstances have to be taken into consideration and as much as it is feasible all solutions should be always checked in practice. Extensive testing, continual improvements and a long period of development are essential for an AI to be working well and for it to provide a challenge to the player.

Chapter 7

Real-life Applications

In this chapter real-life applications for artificial intelligence approaches are discussed, both in a general manner, as well as in the specific case of an AI developed for the game's purpose. A comparison between the OpenTTD simulation and the "real world" is presented. Selected aspects of real-world decision problems in the transport are commented on in the context of possible simulation-based solutions. Afterwards, a few additional applications of OpenTTD are presented, some more practical, some merely interesting and probably unexpected for the reader.

7.1 Intelligent Systems

Upon hearing "AI", the majority of people tend to think along the lines of HAL from "2001: A Space Odyssey". It is an artistic vision of what might come to in the future, but today's artificial intelligence is struggling with tasks like parsing and deriving the meaning of human-language data. We find applied artificial intelligence in cognition and game theory, but also some practical applications of complex decision systems.

We can distinguish three types of intelligent systems, based on functioning industrial applications used for solving real-world problems:

- Decision Support Systems – an attempt to assist in finding a "semi-structured" problem solution by using specified decision rules, often combined with human intuition. The DSS main purpose is to give an advice.
- Expert Systems – a system containing codified expert knowledge with a set of relevant circumstances. The structured decision rules are collected from a human expert and then applied to real-world data.
- Machine Learning – concerned with the design and development of algorithms that automatically learn from their empirical experiences.

While working on this report a large set of decision problems in the matter of a transport company has been found. All kinds of industrial solutions mentioned could address them. In the next section we will try to look at the differences between the simulated world of OpenTTD and the real world.

7.2 Real and Simulated

To find possible applications of artificial intelligence algorithms discussed in this report we need to ask a question about the similarities between the simulation world in OpenTTD and our real world of transport.

The questions we face are: which solutions could be reused and what objects could be identified with each other? Is there a method to bridge reality and the simulation environment? What are the largest differences and can they be somehow relaxed or alleviated?

The game is based on the real world, but only in a partial view. Even if we select only some elements and limit our expectation to transport management problems selected and discussed in each transport type sections in Chapter 5, we can easily see that it is not perfect, but in many cases very close.

Let us draw your attention to the major similarities:

- the revenue source, by transporting cargopassengers;
- established contiguous routes between two (or more) points;
- road vehicle parameters, general movement and reliability;
- traffic load, traffic obstacles, jams;
- train control, signals, acceleration model;
- various terrain slopes and shapes;
- the economic model in the context of the relation between the distance, transport volume and time;
- demand/supply relation kept;
- competition between companies.

The major differences are:

- waiting for cargo data;
- a discreet, tile-based division of the map;
- limited pre-made construction elements;
- the time concept adjusted to the game's purpose only, several related problems could be separated;
  - general time discretization to one day in the economic model;
  - operations in "no-time", e.g. infrastructure or vehicles built and bought;
  - distance and travel time relation (several days of game time often equals to several minutes of equivalent real time);
- auto-navigation of the vehicles once the orders are given;

- oversimplified cargo requirements (destination, travel conditions);
- limited scheduling.

There are more differences as well as similarities, but the ones mentioned above play the most substantial role in an analysis of an attempt to combine reality with the simulated environment.

The discreet nature of the simulation is scalable. The tiles used for the map are far from perfect, their 4-connected neighborhood is a big simplification. Its main cause is the graphical interface limitations still based on the old version of the game.

The biggest problem to make it more realistic is the availability of the waiting cargo data and the timing in the game. Cargo (as well as passengers) are appearing in the game in very specific way according to the game mechanics [5]. In real world application it should be based on the statistical data or real-time observations.

The same lack of real data appears in the case of information being available through the simulator. It uses data generation which makes it good for pretending environment for the purpose of a game, but unrealistic as a simulation. This applies to random events (e.g. failures) in the game as well. The time model is adjusted for fluid game play and mechanics, but not to show how the vehicles actually behave. Their properties and the overall model is promising, but the time model is very much abstracted from reality.

Finally the company control itself is very well reflected in the simulator. The decision level can be moved on the very abstract level of choosing best routes, vehicles and being profit-focused, gained by solving decision problems. Still all of them requires very detailed programming of every action, but this can be grouped in larger and more complex methods representing actions fulfilling subgoals. Costs are taken into account as well, but could be improved with more realistic data acquired in the real world.

Let us sum up this discussion. There are serious differences, but also promising similarities. The scale of the simulation, timing model and visual representation requires improvements. More realistic data are necessary to bring the simulation closer to the level of fidelity necessary for real life applications. What is important – the idea of controlling the company and the problems solved by AIs at the abstract level are true real-life problems.

7.3 Real World Maps

In the previous section several differences and similarities between the real world and the game world have been mentioned. Map topology is one of the big departures from reality, due to the quite large tiles, which makes the simulation not that precise and discreet as it could be.

But before we describe how a game AI can be reapplied from the simulated environment into real-life applications, allow us to focus on OpenTTD map features. We will try to investigate how map models can be generated for real world locations (the idea of it has been found in [3]).

OpenTTD allows to load so-called ”heightmaps”. A heightmap is a specially prepared greyscale image stored in the PNG format. Such image can be loaded into the game scenario editor and used as a pattern for the terrain.

The scenario editor can be run by adding the parameter `-e` when launching the game. But before we can do this, we need several tools to prepare special a PNG file with the terrain template. To prepare the image representation of an area the Geographic Information system (GIS) will be required. In general the PNG file can be prepared using any GIS-software able to generate high-resolution image files of the chosen terrain area, based on elevation data. In our example the following tools were selected for use:

- Google Earth ([1]),
- MicroDem ([2]),
- SRTM Overlay for Google Earth ([14]),
- Microsoft Windows 7 Paint.

First, all programs and extensions should be installed. Then, Google Earth should be started. The program allows to select a certain area. By turning on the SRTM module we can create a dataset for a particular region. A result of following these instructions is presented in Figure 7.1. The terrain map can be downloaded from the SRTM database, and saved to the hard drive as a file in the "DEM" format.

By using MicroDem we can read the DEM file, specify the selected region more precisely (with a "crop" tool). A very important function can be found in the options, by selecting Raster GIS-¿Thin DEM. It allows us to significantly reduce the amount of data, which speeds up the computation. In our case, the parameter was equal to 3.

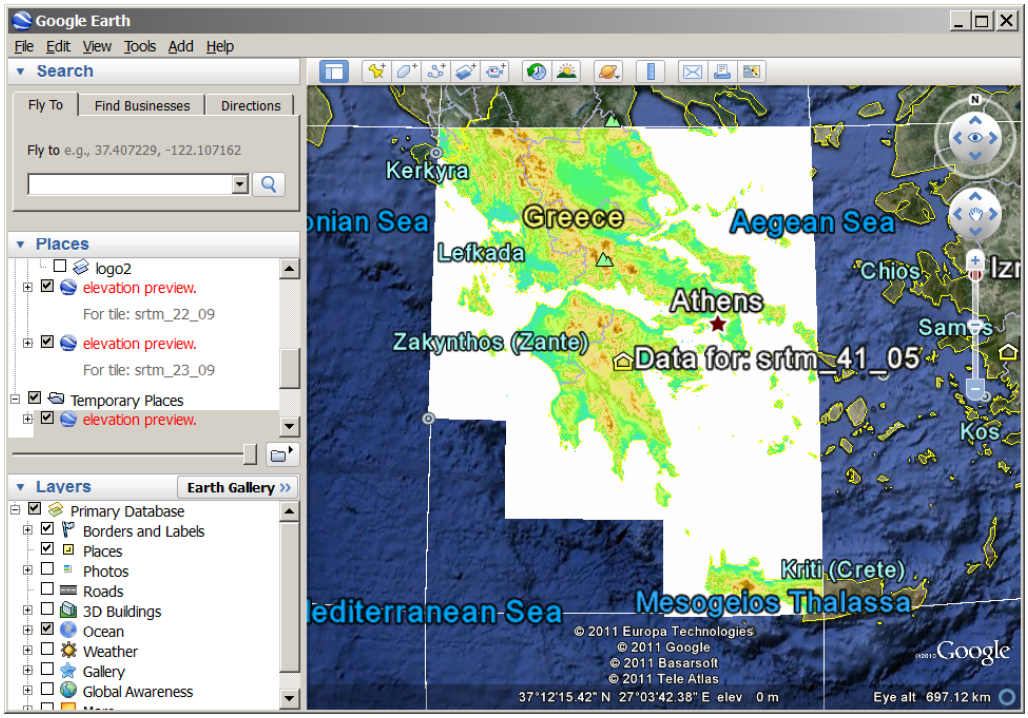

Figure 7.1: Google Earth preview with selected region and loaded elevation data from an SRTM Overlay.

A possibility to adjust the elevation parameters and choose the minimum and maximum values of data considered is also present. That is very helpful when adjusting the sea level for maps with a lot of small islands. Once the elevation parameter is converted to gray-scale, it is possible to save the results as an ASCII Arc Grid. After choosing 1:1 zoom we can export the full view into a GeoTIFF (TIF) file. The result can be observed in Figure 7.2.

When the TIF image is ready we can us an image editor to rescale it, change its size and, most importantly, convert into a PNG file, which should be saved in the game documents' folder, */Documents/OpenTTD/scenario/heightmap*. If everything has worked successfully the user should be able to obtain results similar to Figure 7.3.

7.4 Logistic Management with AI

In the previous section we showed how real-world maps can be loaded into the simulator. However, they are not perfect due to the geometric properties of the game's tiles. We will try to find a way to use the AI outside its original environment.

It is possible to separate the AI from the simulator environment and combine it with a more realistic one. The sole requirement is the availability of a dataset similar to the one found in the game. The real world data should be reflected in the form of NoAI framework objects containing

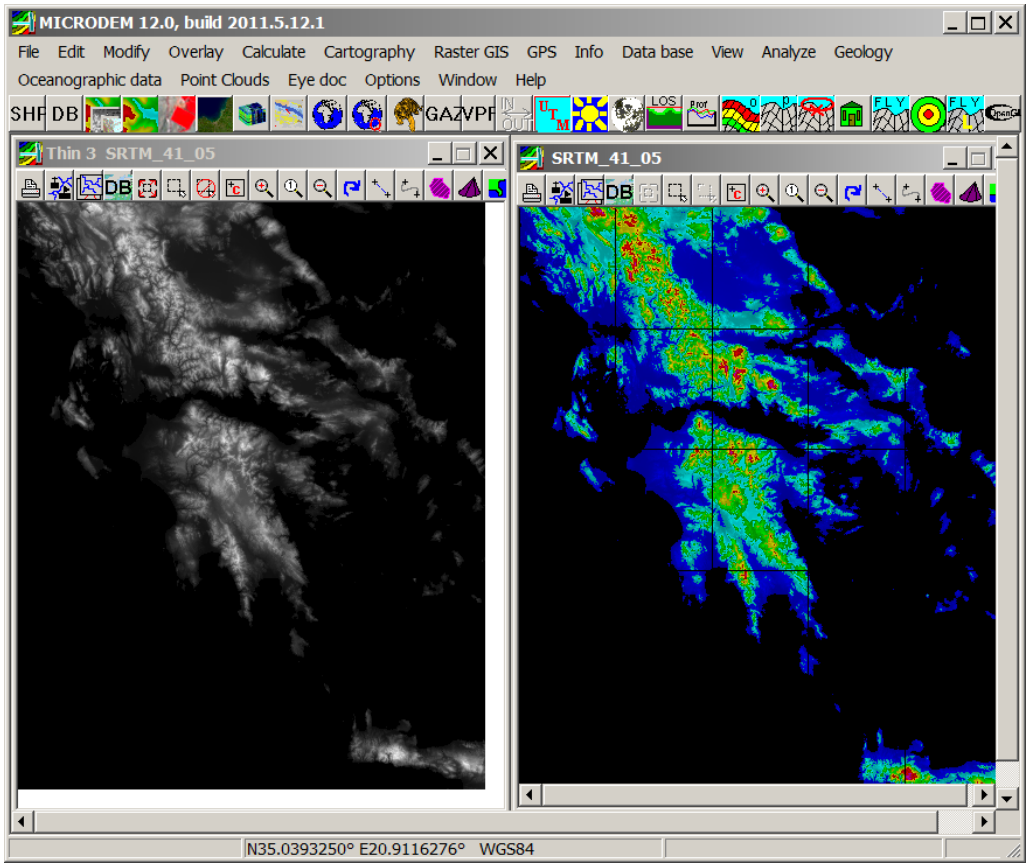

Figure 7.2: MicroDem with previews of raw loaded (righ) and processed (left) data.

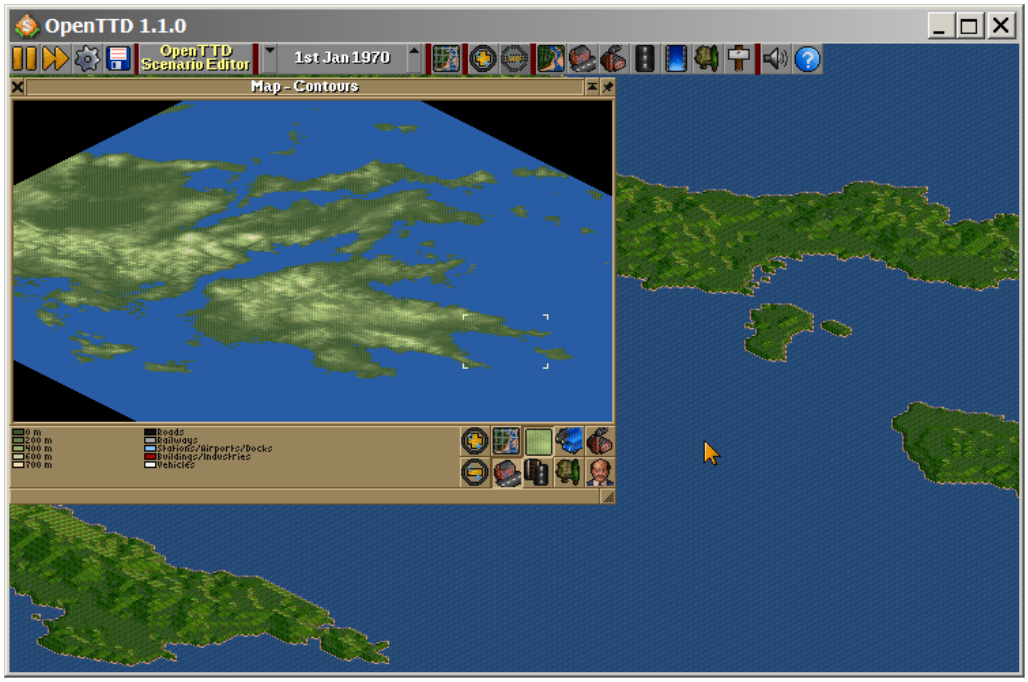

Figure 7.3: The real terrain map successfully loaded.

real parameters instead of some generated.

Squirrel's advantage is that the language itself can be adjusted to any purpose as an API. If we would like to keep the whole implementation as it is, the idea would be to modify the API in a way that would make it connected to database containing real-world data.

Fictional vehicles can be substituted by real-world ones with authentic properties. Additionally, the database of towns and infrastructure can be imported directly. The important differences are related to the distance representation. In OpenTTD we have tiles. However, in the case of a realistic version we can forget about tiles, due to two facts:

- new infrastructure will not be built, only existing one will be used;
- the metric changes, it will not be Manhattan distance.

The AI abilities will be limited by the lack of the ability to create new infrastructure. In the real world creating infrastructure is a very complicated process and cannot be automated. Regardless, if an interface for acting in real-world scenarios would be created, the AI could help solve the following major challenges:

- establishment of new routes;
- determining the most profitable routes
- vehicles selection and extension of the most profitable routes;
- vehicle replacement/service;
- elimination of unprofitable vehicles;

- as a result of all the above, generation of revenue.

At this point, assuming that the object property adjustment has been done right, the presented AI is able to advise in managing a transport company in a real-world scenario.

7.5 Fully-formed Application

By using extra data we could also extend the model with the addition of traffic-control elements – in the context of density control and reaction to real unexpected events (e.g. traffic jams, blockades). Additionally, by improving the time control model we should be able to extend the scheduling and see how the traffic schedule influences transportation efficiency (i.e. the amount transported during a certain period of time).

If we drop the existing implementation in its current form and create a new one in some other technology e.g. as a web application, the control model could be extended even further. By tracking down the vehicle positions and using GPS combined with compatible geographical data, the process could be visualized on real maps (e.g. Google Maps). The web application could gather data on road traffic conditions as well as the cargo delivery process and, based on this, by using the same algorithms, maintain a sustainable economical performance. A simple drawing showing the architectural differences is presented in Figure 7.4.

7.6 Expert Systems and Decision Support Systems

In the previous section an attempt to separate the solution from the game environment into real world application. Here we will take a short overview on the commercial or research products covering similar topics.

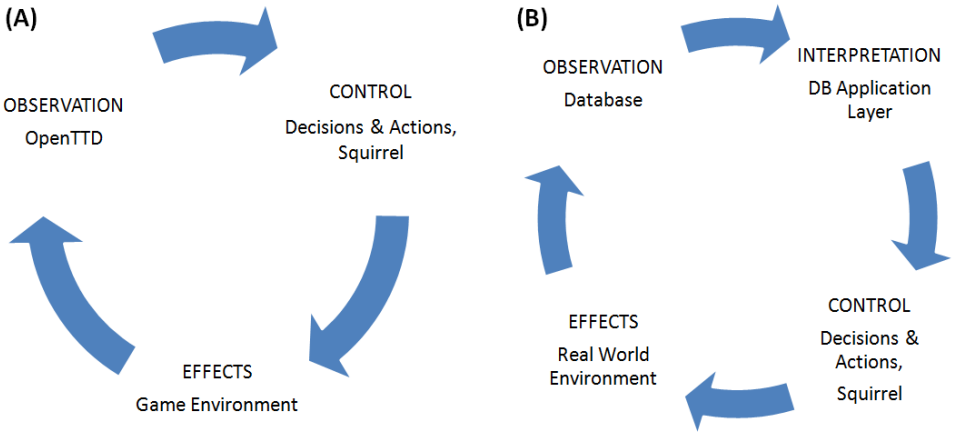

Figure 7.4: (A) OpenTTD system architecture. (B) A possible solution for real life applications.

7.6.1 EXPEDITE

A lot of effort in the research is directed to forecasting and measuring the traffic load. An example of the system designed for this purpose is EXPEDITE, EXpert-system based PrEdictions of Demand for Internal Transport in Europe (more in [13]) prepared for the European Commission Directorate-General for Energy & Transport. The main objectives were to:

- generate forecasts for passenger and freight transport in specified future scenarios;
- evaluate the efficiency of various policies that focus on transferring from automobile, lorry and air transportation to other modes;
- identify market segments that are sensitive in terms of modal usage modifications.

As we can read in the report [13] the expert system is directed more towards measuring, forecasting and discovering sustainable mobility and inter-modality, than transport network management. On the other hand, the forecast output can be strategic for the management and infrastructure development.

7.6.2 Minimal Transport Standard

The next example is more related to transport network management. In the publication [9] we found an approach to the problem of strategic spatial development through transport network re-organization. We find a description of an expert system for strategic transport planning and an introduction to the concept of the Minimal Transport Standard (MTS), which has been applied there.

The authors notice that transportation may play a crucial role in area development. The primary task of the expert system was to devise a tool to increase the quality of life and business activity. The reflection of this concept is the idea of “town cultivation” present in OpenTTD, as mentioned in Chapter 5.

7.6.3 Sustainable Transport Concept

A topic which often appears in the area of transportation management is the ”sustainable transport” concept. The practical application of such concepts are mostly attractive for cities, where the transport improvements can quickly and directly improve peoples’ lives and bring economical benefits. The project found in [17] responds to these needs by introducing UTOPIA (Urban Transport Operations and Planning using Intelligent Analysis), which is a an expert system covering the analysis and the decision making process in cities.

It helps in investigating the development of a new public transport system. The paper outlines the features required of an expert system, the systematic method of deciding which public transport technology is the most appropriate. It also discusses knowledge acquisition techniques based on fourteen British and thirty-two non-British systems.

The analysis of the information allows to gain a general knowledge about the nature of cities, the systems, and the particular rules of the decision process.

7.6.4 Transport Security

We additionally found transport expert systems with aims on traffic measuring, forecasting, sustainability and planning to increase the quality of life. On top of that, there exist solutions focused on security. One of such solutions is presented in [18] as an example of integration of an expert systems into a railway electric transport safety control. The authors' objective was to examine the basics of expert systems and artificial immune systems and design a scheme of the transport control system combining their advantages. The authors review the features and evaluate them for possible integration.

7.6.5 Examples Summary

There exist much more examples, as the field is in rapid development, both in pure research and in industrial applications. The current trends in the field of transportation design research are moving towards CO_2 reduction, which is a highly motivating challenge and a desirable direction for development.

Research on quality, improvements and economical benefits of efficient transportation inside and outside urban areas will have a major impact on future city shape and development. The underlying design problems, present due to the large amounts of data and knowledge requirements, are creating an opportunity for the implementation of IT solutions from the field of artificial intelligence and decision support systems.

Chapter 8

Conclusions

In this report an analysis of applying artificial-intelligence-based solutions to transport management and economic strategy evaluation was presented. The example of "OpenTTD", a simulation game, was used for a custom implementation and additional experiments.

In the first part of the work the existing game AIs were analyzed. They were presented in a single table, on the basis of which the reader could find out about their properties and comparison classes. From that summary several examples were selected and described in more detail. The reasoning and the strategies behind them have been presented.

The typical human approach for playing the game was investigated and commented on. The most common human behaviors in the game were identified and the playing concepts were specified.

The game API and the AI programming environment (based on the Squirrel programming language) were described. Advantages and disadvantages of Squirrel were presented. The reader could find the detailed description of the implementation, architecture and the problems met.

A custom AI design has been proposed and described. Not all designed features were implemented, but the version attached to this report has successfully met experimental requirements.

Several experiments evaluating the relative performance of the AIs were performed and results were provided.

Finally, real-life applications for similar artificial intelligence approaches have been described. The author tried to find a potential bridge between the algorithms, AI implementation in Squirrel and some possible real-world applications. The potential ability to model the real world and the limitations of OpenTTD in this context have been discussed. Several examples of expert systems or decision support systems have been presented, as an overview of existing real-life solutions. By being presented with these example the reader was provided with information about the extent of the subject and possible simulation improvements.

8.1 Future Work

During the project development appeared several ideas of further development. To begin with, SPRING AI can be extended for new features.

Furthermore the algorithms used could be improved or substituted with more advanced. A large part of presented implementation could be improved with the gained experience. The author noticed that the software design is not minimized in the memory usage. Thus, the memory usage can be possibly better designed.

Squirrel port in OpenTTD could be updated to the last Squirrel version and common libraries could be available to use

In the matter of real life applications simulator requires improvements. The time control model does not reflect real world and the objects lacks the real data. Based on the gathered informations, the postulate of a possibility to make OpenTTD a real life transport simulator in many aspects is believed.

Hence there is also a possibility of further work in the real data accusation and connecting the database to the API interface.

Moreover while development and experiments a new idea came. The game uses very conservative approach for vehicles control. Once the route is established vehicle keeps traveling between certain points defined by a human or non-human player.

The idea is to modify slightly the game and remove a necessity of defining the route points to a vehicle. The vehicle could have dynamically computed range, with-in the most profitable distances. Then, a vehicle itself could choose what to do according to the current situation with-in its range and its specialty (different if it is a bus, a coal truck, a wood truck, etc.).

A multi-agent approach might maximize the profit according to the route distances and increase the transported cargo amounts. The economical model would change, instead of buying vehicles for a certain route, a player would buy vehicles for a certain area to cover the present demand.

Bibliography

[1] Google Earth Project. Website. http://earth.google.com, access: June 2011.

[2] MicroDem Project. Website. http://www.usna.edu/Users/oceano/pguth/website/microdem/microdem.htm, access: June 2011.

[3] OpenTTD Forum. Website. http://www.tt-forums.net/viewforum.php?f=55, access: March 2011.

[4] OpenTTD Game. Website. http://wiki.openttd.org/Main_Page/, access: March 2011.

[5] OpenTTD Game Mechanics. Website. http://wiki.openttd.org/Game_mechanics, access: March 2011.

[6] OpenTTD NoAI API Documentation. Website. http://noai.openttd.org/api/, access: March 2011.

[7] OpenTTD Wiki. Website. http://openttd.org, access: March 2011.

[8] Bobby Anguelov. Optimizing the A* Algorithm. Website. http://takinginitiative.net/2011/05/02/optimizing-the-a-algorithm/, access: April 2011.

[9] Vladimir Bougromenko. An expert system for sustainable urban and regional transport development. *Conference Proceeding Paper. Proceedings of ICTTS 2002, Guilin, China*, 2:1369–1376, July 2002.

[10] Thomas H. Cormen. *Introduction to Algorithms*. MIT Press, 2001.

[11] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry (2nd revised edition ed.)*, page 147163. Springer-Verlag, 2000.

[12] Alberto Demichelis. Squirrel: the programming language. Website. http://www.squirrel-lang.org, access: March 2011.

[13] RAND Europe. Expert-system based predictions of demand for internal transport in europe. December 2002.

[14] T. G. Farr. The shuttle radar topography mission. *Rev. Geophys. American Geophysical Union*, 45, 2007.

[15] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques*, pages 65–79. Elsevier, 2011.

[16] Jon Kleinberg and Eva Tardos. *Algorithm Design*. Pearson/Addison-Wesley, 2006.

[17] Roger Mackett and Marion Edwards. An expert system to advise on urban public transport technologies. *Computers, Environment and Urban Systems. Elsevier Ltd. Volume 20.*, pages 261–273, July-September 1996.

[18] Andrew Mor-Yaroslavtsev and Anatoly Levchenkov. Modeling the integration of expert systems into railway electric transport safety control. *10th International Symposium, Topical Problems in the Field of Electrical and Power Engineering, Parnu, Estonia*, pages 10–15, 2011.

[19] Andrew Nash. Web 2.0 Applications for Improving Public Participation in Transport Planning. *Transportation Research Board Annual Meeting 2010, Washington D.C.*, November 2010.

[20] Nils J. Nilsson. *Artificial Intelligence: A New Synthesis*. Morgan Kaufmann, 1998.

[21] Amit Pate. A* Algorithm: Implementation Notes. Website. `http://theory.stanford.edu/~amitp/GameProgramming/ImplementationNotes.html`.

[22] Luis Henrique Oliveira Rios. trAIns Project. Website. `http://homepages.dcc.ufmg.br/~lhrios/trains/`, access: March 2011.

[23] Luis Henrique Oliveira Rios and Luiz Chaimowicz. trAIns: An Artificial Intelligence for OpenTTD. *VII Brazillian Symposium on Games and Digital Entertainment*, December 2009.

[24] Stuart Jonathan Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2010.

[25] Carsten Schnober. Projects on the move, June 2009. `http://www.linux-magazine.com/w3/issue/103/Free_Software_Projects.pdf`, access: March 2011.

Appendix A

Side-cards of AIs

Convoy	
Transport:	road
Vehicles:	buses
Author:	GeekToo
Licence:	GPL v2
URL:	http://www.tt-forums.net/viewtopic.php?f=65&t=37946

Features

- Uses only buses
- Modified path-finding so the roads are nonlinear

Designer aims

- to create an AI that human players like to play against
- not the best financial performance
- improve the author's programming skills

Algorithms, behavior and strategy

- A* based pathfinder (not fully implemented)
- only one station per city is built
- only unused towns are connected
- no bus line merging

PathZilla	
Transport:	road
Vehicles:	buses
Author:	Zutty
Licence:	GPL v2
URL:	http://www.tt-forums.net/viewtopic.php?f=65&t=38645

Features

- network planning using graph theory
- two steps of path-finding to improve line reuse
- aesthetic path-finding builds tram lines alongside roads and honors town grid layouts where applicable
- support for articulated vehicles and trams
- ”green belts” for improving local authority rating
- multiple road stations per town and fleet maintenance
- supports all primary industries
- currently no support for Pathzilla

Designer aims

- high level planning
- neat, realistic construction

Algorithms, behavior and strategy

- Graph theory: Delaunay triangulation, Shortest path tree, Minimum spanning tree, Combined planar graph

Full AI description can be found in the report, Section 4.3.

Chopper	
Transport:	air
Vehicles:	helicopters
Author:	Marbs
Licence:	GPL v2
URL:	http://www.tt-forums.net/viewtopic.php?f=65&t=45070

Features

- mail and passenger transport
- attempts giving subsidies
- oil rig support
- auto-replaces vehicles
- sleeps until heliport and heli depot are available (usually 1976)
- a lot of parameters in AI settings, several intended for making it more challenging to humans (limits, speed, subsidy coverage, etc.)

Designer aims

- AI which uses only helicopters

Algorithms, behavior and strategy

- Chopper can choose station locations randomly or cleverly

Denver & Rio Grande	
Transport:	railway
Vehicles:	trains
Author:	Dustin
Licence:	BSD
URL:	http://www.tt-forums.net/viewtopic.php?f=65&t=44287

Features

- prefers the NARS (North-America Railway Set) train set, but supports default
- two way tracks
- train management
- cleans up after closed industry
- custom pathfinder that is faster than the default rail pathfinder
- pathfinder with three sub-pathfinders
- non-network style connections
- poor to decent code
- reuse of consumer stations for mini networks
- lots of settings
- some limited bus capability might be forthcoming

Designer aims

- a train-focused AI that's a reasonable replacement for the "oldAI"
- completely custom path-finding
- profitable, robust and interesting in the default game and with popular NewGRFs

Algorithms, behavior and strategy

- customized pathfinder with three sub-pathfinders

PAXLink	
Transport:	road, air
Vehicles:	buses, aircraft
Author:	Zuu
Licence:	GPL v2
URL:	http://www.tt-forums.net/viewtopic.php?f=65&t=41518

Features

- a passenger only AI
- inter-city transport

Designer aims

- to build bus services which helps towns grow; when they do grow, it creates airports nearby

Algorithms, behavior and strategy

- does not store any meta state information using Squirrel, but it reads all state data from the map and reasons based on this
- controls the number of buses in the town
- controls the number of airplanes on an air connection

Trans (old name FanAI)	
Transport:	road, water, air, railway
Vehicles:	buses, trucks, trams, aircraft, ships
Author:	fanioz
Licence:	GPL v2
URL:	`http://www.tt-forums.net/viewtopic.php?f=65&t=42272`
URL:	`http://noai.openttd.org/projects/show/ai-trans`

Features

- buggy

Designer aims

- try to transport cargo from/to all possible towns and industries using Bus, Truck, Tram, Aircraft or Ship
- in the future all available vehicles will be used

Algorithms, behavior and strategy

- uses only one type of industry (if possible) as the destination of transporting cargo, this strategy is based on so called "cargo concept" (more can be found in [7])

trAIns	
Transport:	road, water, air, railway
Vehicles:	buses, trucks, trams, aircraft, ships
Author:	lhrios (Luis Henrique Oliveira Rios)
Licence:	GPL v2
URL:	http://homepages.dcc.ufmg.br/~lhrios/trains/
URL:	http://www.tt-forums.net/viewtopic.php?f=65&t=41175

Features

- constructs and manages railroad routes
- upgrades infrastructure and vehicles

Designer aims

- started as a graduation project, later evolving into a research project
- created to provide a competitive and intelligent AI that can play with railroads
- ”A general metric of intelligence for AIs is that its actions and decisions must result in behaviors similar to the ones caused by human players. OpenTTD has a lot of AIs but few can use trains and generate good results according to this metric.”

trAIns *(continuation)*

Algorithms, behavior and strategy

- double railways (human-like behavior):
  - imitates construction style of human players
  - many trains can circulate on the same route
- several industries can have their production transported to one destination industry sharing parts of the railroad (human-like behavior)
- concentration of production (human-like behavior)
- dynamic number of trains computation so there is always one kept waiting at the loading station
- infrastructure and vehicle upgrades
- a carefully implemented version of the A* algorithm
- fast planning (planner with good results)
- connects industries with each other, or two towns with each other (passengers only)
- detection of unprofitable routes and consequent demolishing
- select the industries with the largest potential of money generation

Full AI description can be found in the report, Section 4.4.

AdmiralAI	
Transport:	railway, road, air
Vehicles:	trains, road vehicles (including trams), aircraft
Author:	Yexo
Licence:	GPL v2
URL:	`http://www.tt-forums.net/viewtopic.php?f=65&t=38057`

Features

- OpenTTD developer project

Designer aims

- implement as many features from the game API as possible

Full AI description can be found in the report, Section 4.5.

ChooChoo	
Transport:	railway
Vehicles:	trains
Author:	Michiel
Licence:	GPL v2
URL:	http://www.tt-forums.net/viewtopic.php?f=65&t=44225

Features

- OpenTTD developer project

Designer aims

- ”interesting”, nice looking networks

Algorithms, behavior and strategy

- randomized starting spots
- main processing loop as a ”to do list” of tasks
- no train replacement except an auto-renew function
- ”It works by placing a four way crossing at a random location and extending its rail network to towns in 4 directions. When a destination town is ’off to the side’, it’ll build a new crossing, connect the crossings and the station, and extend the newly placed crossing. This continues until the network can’t be extended further. Then, it places a new seed crossing to build a new network.”

Full AI description can be found in the report, Section 4.6.

SimpleAI	
Transport:	road, railway, air
Vehicles:	road vehicles, trains, aircraft
Author:	Brumi
Licence:	GPL v2
URL:	http://www.tt-forums.net/viewtopic.php?f=65&t=44809

Designer aims

- remake of oldAI in the NoAI framework, with some improvements

Algorithms, behavior and strategy

- gives subsidies
- if not giving subsidies it sets up cargo transportation in industries/towns randomly
- uses the library's pathfinder
- corrects route if interrupted
- contains code from PAXLink and NoCAB

MogulAI	
Transport:	road
Vehicles:	trucks
Author:	Dezmond_snz
Licence:	Custom
URL:	http://www.tt-forums.net/viewtopic.php?f=65&t=44760

Features

- industry-industry truck routes
- vehicle upgrading
- route-, station- and vehicle-count management
- possibility of reaching RV's limit

Designer aims

- only trucks

Algorithms, behavior and strategy

- only trucks, tries to grow as fast as possible
- bank balance monitoring

CluelessPlus	
Transport:	road
Vehicles:	buses
Author:	Zuu
Licence:	GPL v2
URL:	http://www.tt-forums.net/viewtopic.php?f=65&t=41462

Features

- bus transport between towns

Designer aims

- remake of the old version (Clueless) for the new NoAI framework
- make it up-to-date
- improved pathfinder (used from libs)

Algorithms, behavior and strategy

- library-based pathfinder

AIAI	
Transport:	air, road, ships, railway
Vehicles:	airplanes, trucks, busses, ships, trains
Author:	Kogut
Licence:	GPL v2
URL:	http://www.tt-forums.net/viewtopic.php?f=65&t=47298

Features

- support for all unified graphic sets
- support all possible settings
- full road vehicle (RV) support
- freight train support
- plane support
- uses all cargoes (except mail, tourists)

Designer aims

- intended to be both challenging and nice looking

NoCAB	
Transport:	air, road, ships, railway
Vehicles:	airplanes, trucks, buses, ships, trains
Author:	Morloth
Licence:	GPL v2
URL:	http://www.tt-forums.net/viewtopic.php?f=65&t=40203

Features

- very fast transport network development
- outstanding speed of the company value grow

Designer aims

- originally released to compete in the ”Tjip competition” in 2008 with only lorry support
- extensively developed since then
- not competitive against experienced human players, but fully competitive against other AIs

Algorithms, behavior and strategy

- custom A* implementation designed for speed
- during play vehicles load replacements (for routes with more volume) and upgrades are performed
- plans out different routes, uses sub-sum algorithms to decide which one to build
- while it has no money to build new connections it uses time to plan ahead

Roadrunner	
Transport:	road
Vehicles:	buses, trucks
Author:	Steffl
Licence:	GPL v2
URL:	http://www.tt-forums.net/viewtopic.php?f=65&t=48281

Features

- transports all kinds of cargo
- calculation of optimal fleet size
- protection against going bankrupt

Designer aims

- improvement of the original PathZilla algorithm

Algorithms, behavior and strategy

- based on the PathZilla AI

OtviAI	
Transport:	road, railway
Vehicles:	busses, trucks, tram, trains
Author:	Maninthebox
Licence:	GPL v2
URL:	http://www.tt-forums.net/viewtopic.php?f=65&t=39707

Features

- transports all kinds of cargo
- calculation of the fleet size
- protection against going bankrupt

Designer aims

- part of a collection of test AIs written for training and improving Rondje AI

Algorithms, behavior and strategy

- based on PathZilla AI

Rondje om de kerk (Rondje)	
Transport:	road, railway
Vehicles:	busses, trucks, tram, trains
Author:	Willem, Marnix, Michiel, and Otto (Maninthebox)
Licence:	GPL v2
URL:	`http://www.tt-forums.net/viewtopic.php?f=65&t=39756`

Features

- winner of the TJIP challenge 2008

Designer aims

Algorithms, behavior and strategy

- partially inspired by PathZilla AI

Full AI description can be found in the report, Section 4.7.

MailAI	
Transport:	road, railway
Vehicles:	trucks, trains
Author:	Hephi
Licence:	GPL v3
URL:	http://www.tt-forums.net/viewtopic.php?f=65&t=52478
URL:	http://dev.openttdcoop.org/projects/ai-mailai

Features

- the goal is to have a fun competitor to play against
- only mail with trucks and trains

Designer aims

- transport mail with trucks and trains

Appendix B

Initial AI Classes

Listing B.1: MyNewAI/main.nut

```
class MyNewAI extends AIController
{
  function Start();
}

function MyNewAI::Start()
{
  AILog.Info("MyNewAI started.");
  SetCompanyName();
  local types = AIRoadTypeList(); // set a legal roadtype
  AIRoad.SetCurrentRoadType(types.Begin());
  // main program loop
  while (true) {
    AILog.Info(".. time is passing, tick " + this.GetTick());
    this.Sleep(50);
  }
}

function MyNewAI::Stop() { AILog.Info("AI stopped"); }

function MyNewAI::Save()
{
  local table = {}; // Add save data to the table.
  return table;
}

function MyNewAI::Load(version, data)
{
  AILog.Info("Loaded"); // loading routines
}

function MyNewAI::SetCompanyName()
{
  AICompany.SetName("MyNewAI"); // our company name
  //if this company name already exists then add a number
  if (!AICompany.SetName("MyNewAI")) {
    local i = 2;
    while (!AICompany.SetName("MyNewAI #" + i)) {
      i = i + 1;
    }
  }
  AICompany.SetPresidentName("M. New");
}
```

Listing B.2: MyNewAI/info.nut

```
class MyNewAI extends AIInfo
{
  function GetAuthor()      { return "Maciej W."; }
  function GetName()        { return "MyNewAI"; }
  function GetDescription() { return "An example AI"; }
  function GetVersion()     { return 1; }
  function GetDate()        { return "2011-03-24"; }
  function CreateInstance() { return "MyNewAI"; }
  function GetShortName()   { return "MNAI"; }
  function GetAPIVersion()  { return "1.0"; }

  function GetSettings()
  {
    AddSetting({name = "bool_setting",
      description = "a bool setting, default off",
      easy_value = 0,
      medium_value = 0,
      hard_value = 0,
      custom_value = 0,
      flags = AICONFIG_BOOLEAN});

    AddSetting({name = "int_setting",
      description = "an int setting",
      easy_value = 30,
      medium_value = 20,
      hard_value = 10,
      custom_value = 20,
      flags = 0,
      min_value = 1,
      max_value = 100});
  }
}

/* Tell the core we are an AI */
RegisterAI(MyNewAI());
```