Symfonia Events

Poradnik dla chcących pisać scenariusze do symulatora MaSzyna

Autor: Transkei

Sierpień 2017 r.

Spis treści

1. Od autora

Do symulatora MaSzyna powstało wiele scenerii i wciąż powstają kolejne. Sama jednak sceneria, choć nie wiadomo jak perfekcyjna i dopracowana, pozostanie nieciekawa dopóki nie będzie dostępnych dla niej ciekawych scenariuszy. Tematyka pisania scenariuszy w symulatorze MaSzyna jak dotąd nie doczekała się pełnego opracowania. Materiały pomocnicze umożliwiające jakiekolwiek prace w tym zakresie były rozproszone, niektóre zaszyte głęboko w wątkach forum, inne na prywatnych stronach, a jeszcze inne ukryte w strukturze katalogowej symulatora.

Pierwszym najpoważniejszym dla scenarzystów materiałem był słynny scenery.doc, manual w bardzo dokładny sposób opisujący sposób powstawania scenerii. Pomimo upływu lat i prób stworzenia coraz lepszych i doskonalszych poradników, plik ten nieprzerwanie cieszy się sławą najlepszego poradnika, i to nie tylko dla scenarzystów, ale nawet dla trasopisarzy.

Kolejną próbą stworzenia dokumentacji i jednocześnie edytora służącego do pisania scenariuszy był Generator Eventów SKP, nadal dostępny w strukturze katalogowej symulatora, w podkatalogu *Programy_na_potrzeby_symulatora*. Choć sam w sobie nie zawierał rewolucyjnych rozwiązań a załączony plik pomocy był stosunkowo skromny, to program sam w sobie stanowił (i dalej stanowi) znakomite źródło wiedzy o wszelkich mechanizmach sterujących scenariuszami – dość wspomnieć, że autor do dnia dzisiejszego korzysta z tego generatora, choć głównie w sytuacji, gdy nie może sobie przypomnieć szczegółów składni eventlauncherów.

Ogromnym źródłem wiedzy jest forum symulatora, w którym można znaleźć praktycznie wszystko, i które ma bardzo poważną wadę – trzeba wiedzieć, jakich słów kluczowych użyć podczas wyszukiwania.

Ostatnią szczególnie istotną pozycją w pozyskiwaniu wiedzy na temat funkcjonowania scenariuszy jest strona prowadzona przez Ra zawierająca informacje o wszelkich modyfikacjach w exe. Niektóre zawarte tam dane są niezwykle wyczerpujące, problemem jest mało sensowne pogrupowanie informacji – aby doszukać się jakieś konkretnej informacji, należy przejrzeć kilkadziesiąt podstron, zawierających dane na temat modyfikacji wprowadzanych w kolejnych exe.

Autora niniejszego opracowania niegdyś smucił fakt, że na jego ulubionej scenerii Całkowo jest bardzo mało misji. Pojawiła się wprawdzie zapowiedź odświeżenia scenerii i dodania ogromnej ilości scenariuszy (obecna sceneria Całkowo v.2, zapowiadana w 2012 r., upubliczniona 5 lat później) ale czas biegł i zupełnie nic się nie działo. Z tego oczekiwania wzięły się pomysły, potem pierwsze bardzo proste próby, aż w końcu nastąpił 24.08.2014 r., kiedy to autor spróbował zaimplementować jeden ze swoich pomysłów, najpierw w sposób bardzo prosty – poprzez kolejne naciskanie klawiszy Shift+cyfra, potem zagłębiał się coraz bardziej w tajniki eventów, memcellów, itd... Wystarczył zaledwie jeden dzień, aby powstał scenariusz, który do złudzenia przypominał dobrze znany dzisiaj c*alkowo_osobowy*. Tak się wszystko zaczęło... W każdym bądź razie autor uznał, że pisanie scenariuszy jest rzeczą bardzo prostą i stosunkowo mało czasochłonną.

Ponieważ niewiele osób podziela zdanie autora odnośnie stopnia trudności pisania scenariuszy, autor postanowił przygotować niniejsze opracowanie, aby podzielić się swoją wiedzą z pozostałymi, jak również przygotować w miarę bezbolesną ścieżkę dla nowego pokolenia scenarzystów, którzy zapewnią dalszy rozwój symulatora MaSzyna.

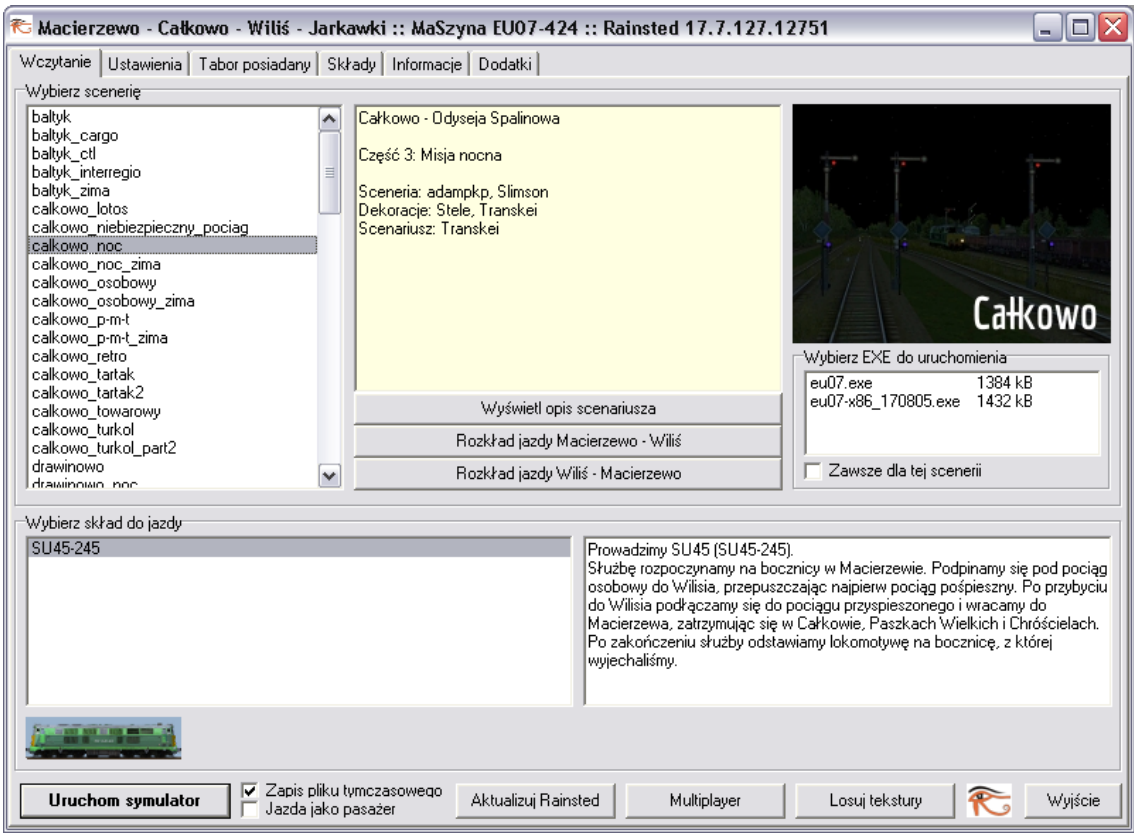
2. Przygotowanie do pracy nad scenariuszem

W tym rozdziale poznamy absolutne podstawy: gdzie szukać plików wykonawczych scenariuszy, które pliki co zawierają, jakie narzędzia sobie przygotować przed przystąpieniem do pracy, bez czego nie należy rozpoczynać pracy.

Gratuluję czytelniku! Zakładam, że masz pomysł na znakomity scenariusz! Jeżeli nie, to proponuję zakończyć lekturę, gdyż bez pomysłu nie ma co się w ogóle brać za pisanie scenariuszy. Załóżmy zatem, że pomysł jest następujący: stwórzmy misję na scenerii Całkowo, gdzie chcielibyśmy prowadzić pociąg towarowy z Macierzewa do Jarkawek (czyli z jednego krańca scenerii na drugi kraniec). Przyjęcie od razu jakiegoś przykładu pozwoli na pokazywanie konkretów, a nie tylko na teoretyczne gdybanie.

2.1. Stworzenie własnego pliku .scn

Na początek otwórzmy Rainsted. Naszym oczom ukaże się lista dostępnych scenariuszy. Wybierzmy jeden z nich, np.: *calkowo_noc*.



Teraz otwórzmy w eksploratorze plików katalog zawierający symulator. Wejdźmy do podkatalogu *scenery*. Teraz odszukajmy plik *calkowo_noc.scn*. W podstawowej wersji, Rainsted jest tak skonfigurowany, że pokazuje wszystkie pliki z rozszerzeniem .scn znajdujące się w katalogu *scenery*. Otwórzmy znaleziony plik za pomocą notatnika – naszego najważniejszego narzędzia do pisania scenariuszy. Naszym oczom ukaże się coś takiego:

```
//$n Macierzewo - Całkowo - Wiliś - Jarkawki
//$d Całkowo - Odyseja Spalinowa
//$d
//$d Część 3: Misja nocna
//$d
//$d Sceneria: adampkp, Slimson
//$d Dekoracje: Stele, Transkei
//$d Scenariusz: Transkei
//$i calkowo_noc.jpg
//$g EU07-424 459 503
```

```
//$f pl inne/calkowo_odyseja/rps420.PDF Rozkład jazdy Wiliś - Macierzewo
//$f pl inne/calkowo_odyseja/ros419.PDF Rozkład jazdy Macierzewo - Wiliś
//$f pl inne/calkowo_odyseja/calkowo_noc_opis_misji.PDF Wyświetl opis scenariusza

time 21:30 05:56 19:50 endtime

atmo 0.423 0.702 1.0 1700 2000 0.70 0.80 0.9 0 endatmo

config movelight 240 endconfig

include slimson/calkowo_tory.scm end
include slimson/teren.scm end
include slimson/pozostale_os.scm end
include slimson/tri_os.scm end
include slimson/zielen_os.scm end
include slimson/pola_c1.scm end
include slimson/drogi3_os.scm end
include slimson/slupy_os.scm end
include slimson/druty_os.scm end
include slimson/elementy_lampy.scm end
include slimson/hekt.scm end
include slimson/calkowo_posers.scm end
include slimson/calkowo_lawki.scm end
include slimson/calkowo_animacje.ctr end
include slimson/calkowo_wskazniki.scm end
include slimson/calkowo_wskazniki_w31.scm end
include calkowo/events_noc.ctr end

//Predefiniowane pozycje kamer dla testerów scenerii
//camera 20672.460 4.678 18081.390 1.343 -100.115 0.000 1 endcamera
//camera 11468.250 15.174 15728.200 -2.668 -101.833 0.000 2 endcamera
//camera 5458.672 10.380 9837.975 0.770 70.054 0.000 3 endcamera
//camera 4474.592 10.380 9487.434 -3.814 -108.709 0.000 4 endcamera
//camera 28.075 5.722 -163.736 -4.960 166.311 0.000 5 endcamera
//camera -2530.627 8.542 -6444.643 -4.387 19.347 0.000 6 endcamera
//camera -5063.041 16.868 -9464.609 -6.106 17.199 0.000 7 endcamera
//camera -8220.647 10.305 -15112.580 0.770 2.158 0.000 8 endcamera
//camera -8226.743 10.305 -16160.290 0.197 -177.894 0.000 9 endcamera

FirstInit

trainset rozklad none3972 10 0
//$o Służbę rozpoczynamy na bocznicy w Macierzewie. Podpinamy się pod pociąg osobowy do
          Wilisia, przepuszczając najpierw pociąg pośpieszny. Po przybyciu do Wilisia
          podłączamy się do pociągu przyspieszonego i wracamy do Macierzewa, zatrzymując się
          w Całkowie, Paszkach Wielkich i Chróścielach. Po zakończeniu służby odstawiamy
          lokomotywę na bocznicę, z której wyjechaliśmy.
node -1 0 SU45-245 dynamic PKP\SU45_V2 301D-245 301D  0 reardriver 3 0 enddynamic
endtrainset
```

Co zrobimy w pierwszej kolejności? Wybierzemy z menu Zapisz jako... i zapiszemy ten plik pod nazwą scenariusza, który chcemy stworzyć, np.: *calkowo_cargo.scn*.

Na samej górze mamy – jak łatwo się domyśleć – nagłówek scenariusza wyświetlany przez Rainsted. Odpowiednie rozpoczęcie linii pozwala nam na umieszczenie tekstu, obrazka, bądź linka do pliku z opisem scenariusza lub rozkładem jazdy. Ta rzecz zasadniczo interesuje nas w tym momencie najmniej, omówimy to dopiero, gdy będziemy wykańczać scenariusz. Możemy te linie zostawić takie jakie są, albo wstawić tytuł naszego scenariusza i ewentualnie jakiś opis.

Kolejna linia zawiera komendę czasu. Jest to linia, której nie wolno kasować. Musimy zdecydować się, o której godzinie rozpocznie się nasz scenariusz. Tak naprawdę istotny jest pierwszy parametr po komendzie *time*. Dalsze 2 parametry są nieaktywne, zastąpiły je znacznie lepsze mechanizmy, możemy ich nie ruszać. Przypuśćmy, że chcemy rozpocząć misję w samo południe, toteż wpisujemy:

```
time 12:00 05:56 19:50 endtime
```

Kolejny parametr opisuje parametry atmosferyczne. Proponuję również ich nie ruszać, zostawić sobie na później. Dalej mamy komendę *movelight*. Kiedyś była opcjonalna, w obecnym wydaniu MaSzyny jest niemal obowiązkowa, za jej pomocą możemy ustalić godzinę wschodu i zachodu Słońca. Tajemnicza liczba widniejąca za

wpisem *movelight* to nic innego jak numer dnia w roku. Jeżeli chcemy mieć wschód i zachód o takiej godzinie jak w okolicach 30 czerwca, to musimy wpisać 180, jeżeli chcemy mieć scenariusz zimowy i jechać w okolicy Bożego Narodzenia, to możemy wpisać 360. My chcemy pojechać 1 października (274 dzień roku), toteż wpisujemy:

```
config movelight 274 endconfig
```

A teraz dochodzimy do tajników scenerii. Widzimy kilkanaście linii zaczynających się od *include*, i zawierających ścieżkę dostępu do jakiegoś pliku. Oto właśnie pliki zawierające dane na temat scenerii. Dla każdej scenerii różnie się nazywają, jednak zasada jest taka, że nazwa powinna w sposób jednoznaczny identyfikować zawartość takiego pliku. Ważne jest również rozszerzenie pliku – scm oznacza, że w danym pliku znajdują się definicje trójkątów terenu, są powstawiane rozmaite inne obiekty, albo drogi i tory. W pliku z rozszerzeniem .ctr są wypisywane tzw. eventy sterujące przebiegiem scenariusza.

Nas interesują dwa pliki: ten zawierający tory, oraz zawierający tzw. eventy. Te pliki będziemy musieli mieć odrębne, toteż zmodyfikujemy linie w następujący sposób:

```
include slimson/calkowo_tory_nasze.scm end
include calkowo/events_cargo.ctr end
```

A teraz poprzez eksploatora wchodzimy do katalogu *scenery/slimson*, kopiujemy plik *calkowo_tory.scm* i wklejamy go z nową nazwą *calkowo_tory_nasze.scm*. Potem wchodzimy do katalogu *scenery/calkowo* i zwyczajnie tworzymy nowy pusty plik tekstowy, nazywając go *events_cargo.ctr*. Do przygotowanych plików wrócimy w kolejnym podrozdziale.

Kolejne linie zaczynają się od znaków //. Oznacza to, że treść danej linii jest pomijana przez parser MaSzyny. Możemy po tych znakach umieścić komentarze, albo za ich pomocą unieważnić jakąś komendę. W tym przykładzie widzimy gotowe komendy predefiniujące pozycje kamer w scenerii. Możemy je odkomentować (poprzez usunięcie z początku linii znaków //), po uruchomieniu scenariusza naciskając klawisze 1,2, 3...9 będziemy przenoszeni na kolejne stacje całkowskiej scenerii.

Porada:

Możemy zakomentować na czas pisania scenariusza wszystkie include nie zawierające torów, eventów sterujących (czyli plików z rozszerzeniem .ctr), oraz semaforów czy innych wskaźników. Efekt będzie taki, że podczas testów scenariusza będziemy jeździć po torach zawieszonych w powietrzu, ale w zamian scenariusz będzie się uruchamiał tak szybko, jak sceneria TD. Wszystko zależy od pisarza: czy chce oglądać docelowe widoki, czy zależy mu na szybkim wczytywaniu scenerii. ***Uwaga!*** *Zakomentowanie niektórych include może doprowadzić do błędnego działania scenariusza, dlatego na tym etapie takie działanie nie jest wskazane.*

Linia *FirstInit* oznacza nam granicę, od której rozpoczyna się definiowanie pociągów. Nie ruszamy jej.
Dalej mamy fragment oznaczający pociąg. Słowa kluczowe następujące po *trainset* są następujące:

- *rozklad* – możemy tu podać link do pliku z rozkładem jazdy, który się pojawi gdy uruchomimy symulator. Można też wpisać *none*, wówczas w ogóle nie będzie rozkładu. Słowo kluczowe *rozklad* jest przydatne w trybie autopilota. Na początek wpiszmy wartość *none*.
- *none3972* – nazwa toru, na którym stoi lokomotywa po uruchomieniu symulacji. Wrócimy do tego w następnym podrozdziale.
- *10* – odległość początku pociągu od krańca toru. Na razie zostawmy ten parametr taki jaki jest.
- *0* – prędkość, z jaką ma się poruszać pociąg zaraz po uruchomieniu symulacji. Często daje się ten parametr jako 0.1, co ma znaczenie dla pociągów prowadzonych przez autopilota (AI). Zostawmy parametr 0.
- *//$o* – po tej linii dajemy opis naszej misji, np.: *//$o Prowadzimy pociąg towarowy Macierzewo – Jarkawki.*
- Kolejne *node* oznaczają kolejne lokomotywy i wagony w składzie. Można je edytować ręcznie ale istnieje znacznie łatwiejszy sposób wstawiania wagonów, opisany w kolejnym podrozdziale.
- *endtrainset* – komenda kończąca definicję pociągu.

Pozostałe pociągi możemy usunąć z naszego pliku, tak aby na początek po scenerii jeździł tylko jeden pociąg – ten, który chcemy prowadzić.

2.2. Wstawianie pociągów do scenariusza

Pierwszym krokiem będzie zdefiniowanie toru, na którym chcemy, aby nasz pociąg stał. Od momentu ukazania się MaSzyny 17.07 proces ten jest bardzo prosty. Wystarczy uruchomić jakikolwiek scenariusz na żądanej scenerii (w poniższym oknie uruchomiony został *calkowo_turkol*, można było uruchomić równie dobrze *calkowo_noc*, ale po prostu będzie lepiej widać). Po uruchomieniu wychodzimy z lokomotywy, ustawiamy się przy żądanym torze, naciskamy klawisz Alt, i naprowadzamy kursor na żądany tor. Przy kursorze pojawi się nam nazwa toru, który wskazaliśmy.

Chcemy rozpocząć scenariusz w Macierzewie, na torze głównym dodatkowym. Nazwa toru brzmi: *macie4*. Możemy wrócić do naszego pliku *calkowo_cargo.scn* i poprawić wpis lokomotywy:

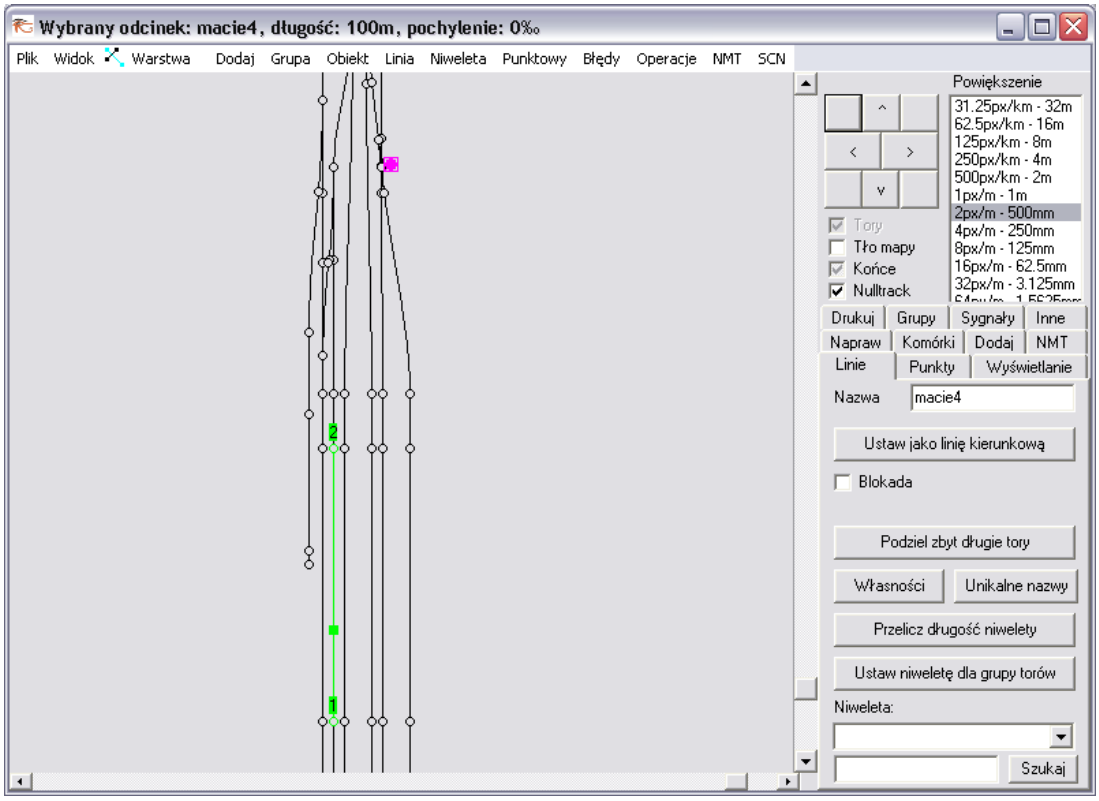
```
trainset rozklad macie4 10 0
```

To najprostszy sposób, najszybszy, ale jednocześnie mocno ograniczony. Podczas dalszego pisania scenariusza potrzebna będzie nam większa znajomość układu torowego. Możemy skorzystać z narzędzi, które oferuje nam Rainsted.

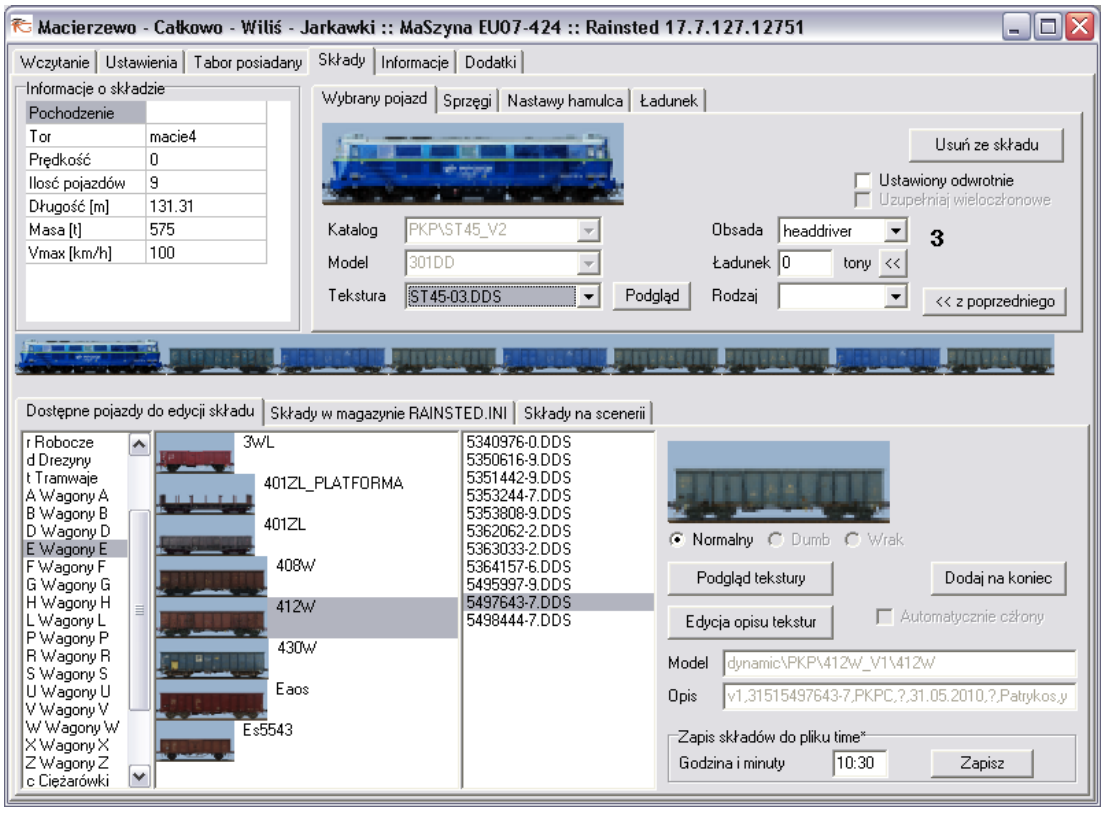
W tym celu należy:

- W zakładce *Ustawienia* wybrać tryb pracy *specjalny – dla trasopisarzy*.
- Wracamy do zakładki *Wczytanie*. Pod listą scenariuszy widzimy 4 przyciski, klikamy *.scm*.
- Lista w okienku powyżej bardzo mocno się rozszerzyła, wyszukujemy nasz plik z torami, czyli *slimson/calkowo_tory_nasze.scm* i zaznaczamy ten plik.
- Przechodzimy do zakładki *Debugger*, klikamy przycisk *Eksport scenerii do RSF*, a następnie klikamy *Edytor scenerii RSF*. Pojawi się okno z widokiem torów.
- Na tej mapie zaznaczamy interesujący nas tor, w nagłówku okna pokaże się jego nazwa, oraz informacja o jego długości. Dodatkowo program pokaże kierunek toru, czyli gdzie jest jego początek (zielona cyfra 1) i koniec (zielona cyfra 2).

Istnieje jeszcze jedna możliwość podglądu nazw torów, czyli uruchomienie programu STV (Szopa Track Viewer). Znajduje się on w katalogu *programy_na_potrzeby_symulatora/podglad_scn*. Nie będzie tutaj omawiany, gdyż jego uruchomienie może być problematyczne – chętnych odsyłam do rozdziału 8.2.2.



Skoro mamy już zdefiniowany tor, to możemy sobie zdefiniować skład, który na nim będzie stał. Tutaj najłatwiej użyć samego Rainsted. Zaznaczamy nasz scenariusz *Calkowo_cargo*, klikamy kartę *Składy* i wybieramy lokomotywę i wagony dla naszego pociągu, chociażby takie:



Wracamy do karty *Wczytanie*, zaznaczamy nasz pociąg, klikamy *Uruchom Symulator*, a następnie klikamy przycisk *Przerwij* (uruchomić symulator też możemy, ale teraz nic to nam nie da). Przechodzimy do eksploatora plików i wyszukujemy w katalogu *scenery* pliku o nazwie *$.scn*. Otwieramy go w notatniku, po czym na jego samym dole powinna się znajdować definicja przygotowanego przez nas składu:

```
trainset rozklad macie4 10 0
//$o Prowadzimy pociąg towarowy Macierzewo - Jarkawki.
node -1 0 ST45-03 dynamic PKP\ST45_V2 ST45-03 301DD  0 headdriver 3 0 enddynamic
node -1 0 a36752 dynamic PKP\408W_V1 5316830-7 408W  0 nobody 3 40 Mial1 enddynamic
node -1 0 a33109 dynamic PKP\412W_V1 5340976-0 412WC  0 nobody 3 40 Mial2 enddynamic
node -1 0 a15651 dynamic PKP\412W_V1 5350616-9 412W  0 nobody 3 40 Mial5 enddynamic
node -1 0 a43976 dynamic PKP\412W_V1 5351442-9 412W  0 nobody 3 40 Wegiel4 enddynamic
node -1 0 a48601 dynamic PKP\412W_V1 5353244-7 412W  0 nobody 3 40 Wegiel3 enddynamic
node -1 0 a51498 dynamic PKP\412W_V1 5362062-2 412W  0 nobody 3 40 Mial5 enddynamic
node -1 0 a7018 dynamic PKP\412W_V1 5495997-9 412WB  0 nobody 3 40 Mial1 enddynamic
node -1 0 a8632 dynamic PKP\412W_V1 5497643-7 412W  0 nobody 0 40 Wegiel3 enddynamic
endtrainset
```

Cały ten wpis kopiujemy do naszego pliku *calkowo_cargo.scn*. Pociąg jest już ustawiony, pora przygotować tory i infrastrukturę scenariusza.

2.3. Przygotowanie torów

Jeżeli uruchomimy nasz scenariusz, to znajdziemy się w wybranej przez nas lokomotywie, będziemy mogli ruszyć, ale niestety nie będzie żadnej interakcji ze strony symulatora. W dalszym ciągu tego podrozdziału otworzymy plik zawierający tory i będziemy zgłębiać jego tajniki.

Otwórzmy zatem w notatniku plik *slimson/calkowo_tory_nasze.scm*. Zobaczymy bardzo długi kod, który może nic nam nie mówić. Aby ułatwić sobie zadanie, naciśnijmy Ctrl+F i wpiszmy do odnalezienia ciąg *macie4*. Notatnik odnajdzie nam wpis toru, na którym ustawiliśmy pociąg:

```
node 1000 0 macie4 track normal 100.0 1.435 0.25 25.0 20 0 flat vis
 rail_screw_used1 4 tpbps-new2 0.2 0.5 1.1
-8217.74 6.2 -15515.6 0.0
0.0 0.0 33.333
0.0 0.0 -33.333
-8217.74 6.2 -15415.6 0.0
0
event2 Macierzewo#9_stopinfo
endtrack
```

Dokładny opis wszelkich parametrów odnajdziemy w dokumentacji symulatora *scenery.doc*. My skupimy się tylko na tych elementach, które mają realny wpływ na przebieg scenariusza. W przedostatniej linii widzimy wpis zaczynający się od *event2*. Oznacza to, że w momencie, gdy pociąg wjedzie na tor w kierunku od punktu 1 do punktu 2 (kolejność tych punktów możemy zobaczyć w edytorze Rainsted, jak to zostało pokazane na jednym z poprzednich obrazków), to zostanie uruchomione **zdarzenie** o nazwie *Macierzewo#9_stopinfo*. Tutaj doszliśmy do sedna funkcjonowania scenariuszy: wszystko co się w nich dzieje, to kolejno uruchamiane zdarzenia (eventy).

Ważna informacja:

Zdecydowana większość scenariuszy w MaSzynie posiada własny układ torowy, gdzie większość torów posiada odpowiednio przypisane zdarzenia. Aktualnie developerzy MaSzyny nie zalecają tego sposobu pisania scenariuszy – zalecane jest wykorzystywanie tzw. odcinków izolowanych, ale w praktyce nikt za bardzo nie wie jak właściwie z nich korzystać (a niektórzy nawet nie wiedzą właściwie dlaczego te odcinki izolowane są zalecane).
Na początku szkolenia stworzymy scenariusz z własnym układem torowym, dzięki temu łatwiej się wdrożymy w pisanie scenariuszy. Jak zdobędziemy trochę doświadczenia, będzie można się zająć pisaniem bardziej skomplikowanych scenariuszy. Cierpliwość jest cnotą!

Na początek kilka informacji teoretycznych: nazwa zdarzenia (eventu) może być poprzedzona słowem kluczowym:

- *event0* – zdarzenie uruchamiane przez stojący na tym torze obsadzony wagon (czyli najczęściej lokomotywę z wpisem *headdriver* lub *reardriver*, wagon z wpisem innym niż *nobody*),
- *eventall0* – zdarzenie uruchamiane przez stojący na tym torze dowolny wagon i lokomotywę (nawet z obsadą *nobody*),
- *event1* – zdarzenie uruchamiane przez wjeżdżający na tor z punktu 2 w kierunku punktu 1 obsadzony wagon,

- *eventall1* – zdarzenie uruchamiane przez wjeżdżający na tor z punktu 2 w kierunku punktu 1 dowolny wagon,
- *event2* – zdarzenie uruchamiane przez wjeżdżający na tor z punktu 1 w kierunku punktu 2 obsadzony wagon,
- *eventall2* – zdarzenie uruchamiane przez wjeżdżający na tor z punktu 1 w kierunku punktu 2 dowolny wagon.

W praktyce stosuje się najczęściej *event1* i *event2*. Zaleca się, aby korzystanie z *event0* ograniczyć do minimum.

Teraz poszukajmy innych wpisów. Np.: tor o nazwie *none3847* ma następujące wpisy:

```
event1 prz_paszm_otwieraj
event2 prz_paszm_zamykaj
```

Domyślamy się co te wpisy mogą oznaczać? Jak pociąg wjeżdża z jednej strony, to zamyka przejazd kolejowy w Paszkach Małych, a jak wjeżdża na tor od drugiej strony, to przejazd jest otwierany.

Przygotowanie pliku z torami polega na usunięciu wpisów zdarzeń, tak abyśmy mogli spokojnie poumieszczać tam własne zdarzenia. I tu niestety tkwi pewna trudność: możemy oczywiście pousuwać wszystkie wpisy jak leci, ale w ten sposób usuniemy całą przydatną nam automatykę scenariusza. Podczas usuwania zostawiamy wpisy, gdzie nazwy zdarzeń mają w nazwie: *sem_info*, *lineinfo*, *stopinfo*, *Warning*, *zamykaj*, *otwieraj*, *shp*, *sbl*, *s1*, *sr1*. Czyli jeżeli odnajdziemy wpis taki:

```
event2 Milewice_jezioro2_Warning
```

lub

```
event1 Milewice#1_stopinfo
```

to taki event zostawiamy, gdyż pełni uniwersalną rolę (np.: jeśli usuniemy ten drugi to stracimy możliwość przewijania rozkładu jazdy, ponieważ event informuje pociąg o mijaniu przystanku o nazwie Milewice). Jeśli jednak znajdziemy wpis taki:

```
event2 czg_sem_d
```

to możemy bez obaw taki event usunąć z wpisu toru.

UWAGA!

Każdy tor może mieć tylko jeden event danego typu, czyli jak wpiszemy do toru jeden po drugim event2, to drugi event nie zostanie wykonany, i zostanie zgłoszony błąd!

Ważna informacja:

W pokazywanym jako przykład pliku torów występuje dość nietypowe zamykanie semaforów. Na wszystkich innych sceneriach zamykanie odbywa się poprzez event z nazwą S1 *lub* Sr1. *Tutaj event zamykający semafor ma nazwie* wylacz *– nie usuwajmy również tych eventów. Jeżeli powyższe kroki sprawiają nam trudności, to możemy usunąć z pliku calkowo_tory_nasze.scm absolutnie wszystkie wpisy event. Nie spowoduje to błędnego działania naszego pierwszego scenariusza.*

Porada:

Możemy do usuwania eventów wykorzystać program EventoUsuwacz_v1.0 – więcej informacji w rozdziale 8.2. Jeżeli jednak korzystamy z Notepad++, to możemy wpisać wyszukanie: **event. \w*$** *i zaznaczyć tryb szukania -wyrażenia regularne. Jeżeli klikniemy „Zamień wszystkie”, to od razu usunie nam wszystkie eventy. Jeśli chcemy zostawić przypisanie semaforów, to musimy przelecieć po kolei cały plik – ale nie zajmie to nam więcej niż 10 minut.*

Skoro już usunęliśmy wszystkie niepotrzebne eventy (albo nawet i wszystkie), to oznacza, że mamy w pełni gotową infrastrukturę, i możemy przystąpić do właściwej pracy.

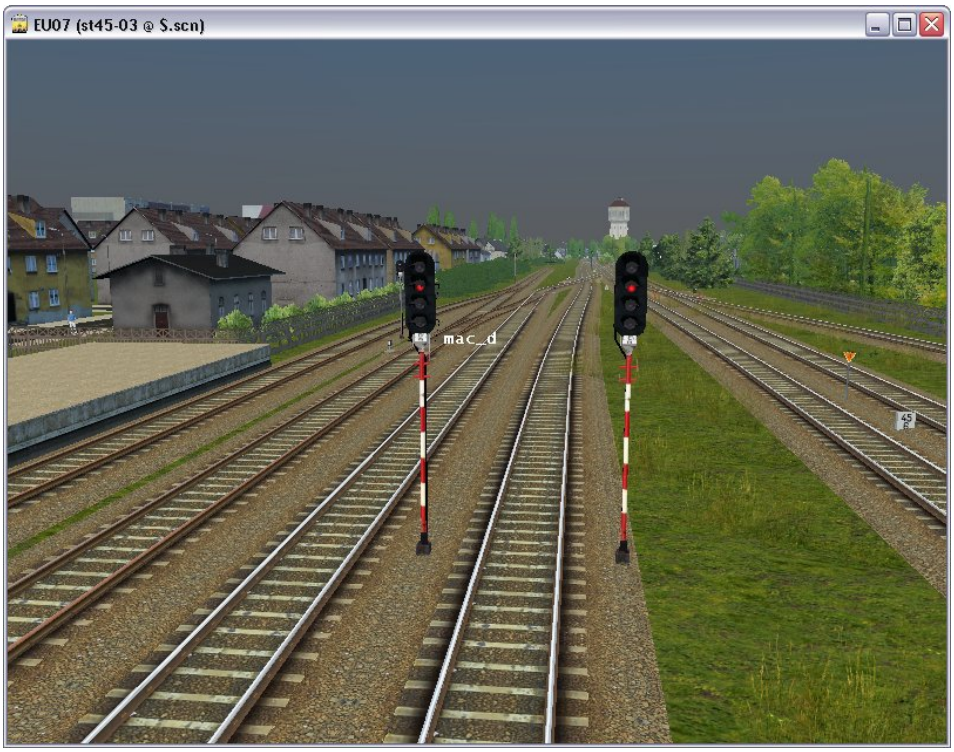
3. Tworzenie najprostszych scenariuszy

W tym rozdziale przekonamy się, że jest możliwe napisanie scenariusza w dosłownie 10 minut.

Otwieramy za pomocą notatnika z katalogu *scenery/calkowo* wcześniej utworzony plik *events_cargo.ctr*. W tym pliku będziemy wpisywali wszelakie procedury sterujące tym scenariuszem.

3.1. Sterowanie zwrotnicami i semaforami

Ponieważ mechanizm scenariusza w mniej lub bardziej inteligentny sposób układa drogę dla pociągów – czyli przekłada zwrotnice i zapala semafory – to można powiedzieć, że jak poznamy mechanizm sterujący zwrotnic i semaforów, to będziemy potrafili wszędzie wjechać. Na początek możemy uruchomić nasz scenariusz *calkowo_cargo*. Po uruchomieniu nic wprawdzie nie będzie się działo, ale będziemy mogli wyjść z lokomotywy, i po naciśnięciu klawisza Alt, oraz po wskazaniu semafora kursorem myszy zobaczymy nazwę semafora wyjazdowego.



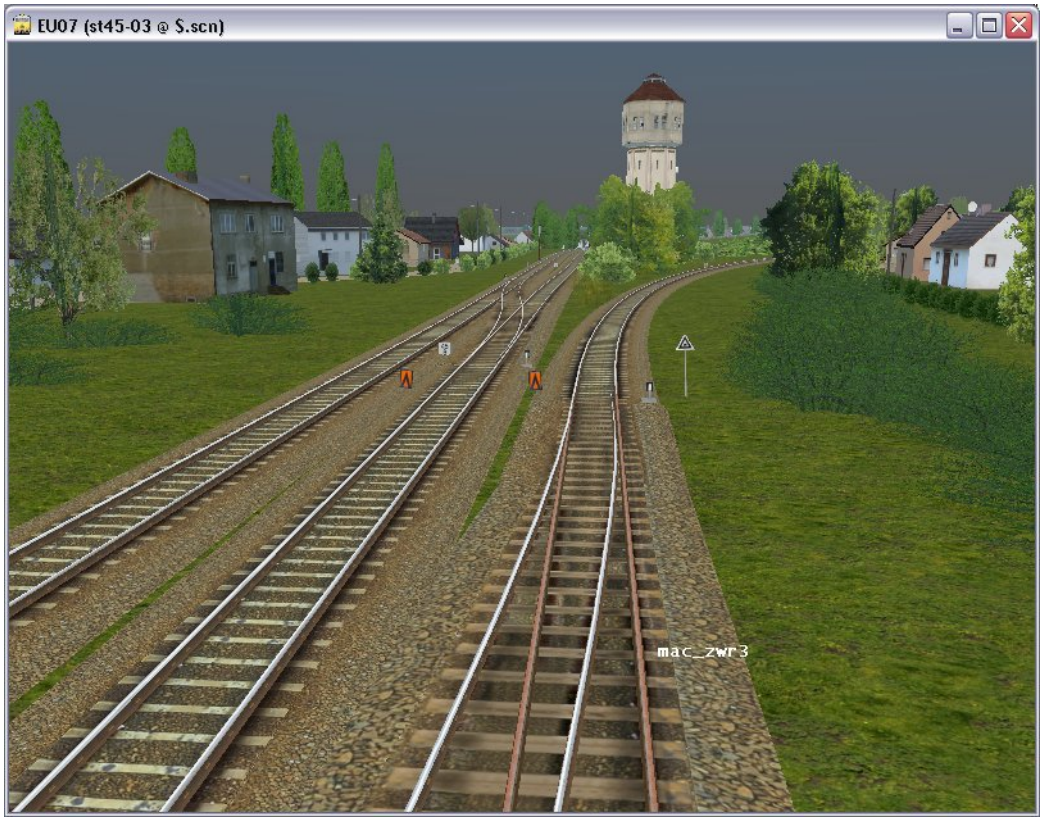
Nasz semafor wyjazdowy nazywa się *mac_d*. Teraz możemy sprawdzić, jakie zwrotnice należy poprzestawiać, aby móc wyjechać do Jarkawek: najeżdżając kursorem na odpowiednią zwrotnicę, odczytamy jej nazwę. Zwrotnice, które musimy ustawić to kolejno: *mac_zwr3*, *mac_zwr4*, *mac_zwr5*, *mac_zwr6*, *mac_zwr9*.

Do odczytania nazw zwrotnic możemy również skorzystać z edytora Rainsted.

Teraz wpiszmy w pliku *events_cargo.ctr* następujący tekst:

```
event keyctrl01 multiple 0 none mac_zwr3- mac_zwr4- mac_zwr5- mac_zwr6ac mac_zwr9+ mac_d_S10
          endevent
```

Możemy uruchomić ponownie symulator, nacisnąć kombinację klawiszy Shift+1. Otrzymamy na semaforze sygnał S10, a zwrotnice ułożą się tak, że będziemy mogli wyjechać z Macierzewa w kierunku Jarkawek.



Na podstawie powyższego przykładu widzimy jak sterować zwrotnicami i semaforami:

- Zwrotnicami steruje się za pomocą znaków – i +. Jeżeli chcemy ustawić zwrotnicę do jazdy na wprost, to musimy wpisać jej nazwę ze znakiem + na końcu. Jeżeli zwrotnica ma być ustawiona do jazdy na bok, to musimy wpisać jej nazwę ze znakiem – na końcu.
- Zwrotnica numer 6 jest zwrotnicą angielską, sterowanie taką zwrotnicą odbywa się poprzez wpisanie na końcu kombinacji liter:
  - *ac*, *bd* – jazda na wprost,
  - *bc*, *ad* – jazda na bok.
- Semafory świetlne ustawia się poprzez wpisanie po jego nazwie podkreślnika i nazwy sygnału, który ma być wyświetlony, np.: *mac_d_S2* – światło zielone, *mac_d_S10* – światło zielone i żółte (v=40km/h).

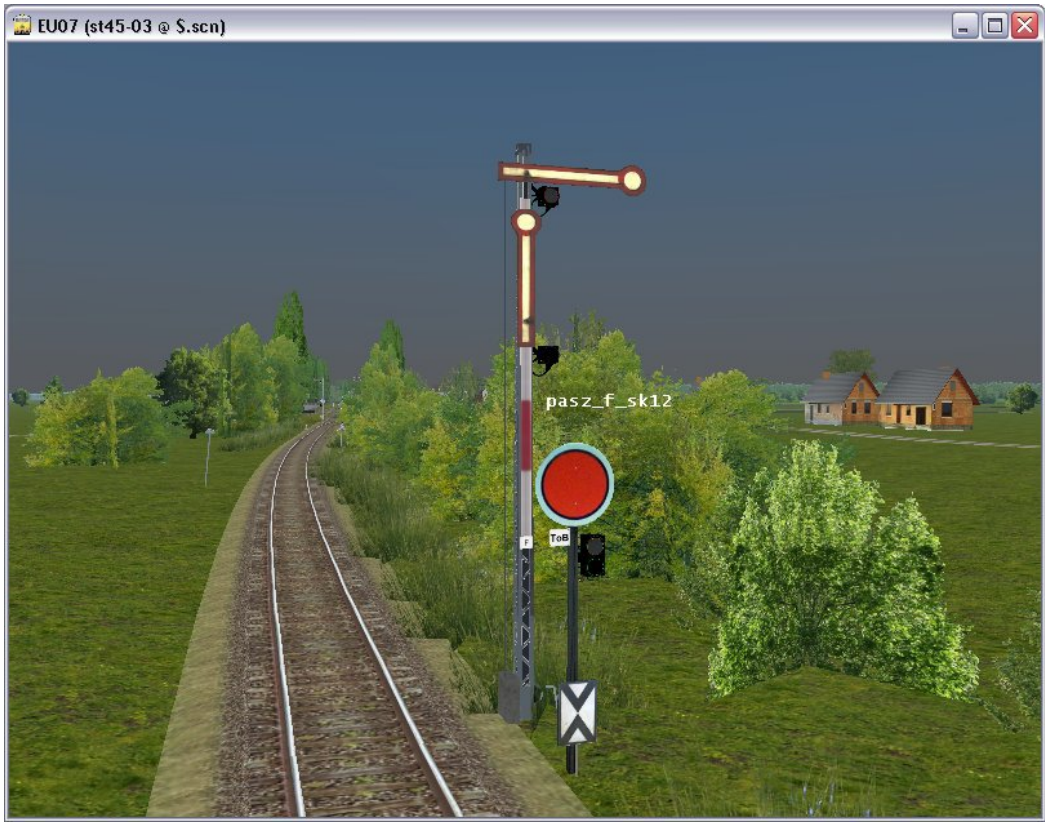
3.2. Sterowanie za pomocą klawiszy Shift+Num

Wiemy już jak sterować zwrotnicami i semaforami, toteż możemy pisać nasze eventy. Przeanalizujmy linię, którą wpisaliśmy do naszego pliku *events_cargo.ctr*:

- *event* – informacja dla parsera, że rozpoczynamy pisać zdarzenie,
- *keyctrl01* – nazwa dla tego zdarzenia. Może być dowolna, ale możemy również skorzystać z nazwy zastrzeżonej, tak jak w tym przypadku. Nazwa *keyctrl01* oznacza, że event będzie uruchomiony po naciśnięciu kombinacji klawiszy Shift+1. Dla kombinacji Shift+2 nazwa eventu będzie keyctrl02, itd.
- *multiple* – informacja, że event polega na wywoływaniu innych eventów (tak, przełożenie zwrotnicy i zmiana światła na semaforze to nic innego, jak event o wpisanej nazwie).
- *0* – opóźnienie w sekundach między wywołaniem, a rozpoczęciem uruchamiania eventu. Jeżeli wpiszemy tutaj wartość 10, to po naciśnięciu będziemy czekać 10 sekund, aż się zapali S10 na semaforze.
- *none* – na razie po prostu wpisujemy *none*, w rozdziale 4.3. wyjaśnimy do czego to służy.
- Nazwy innych wywoływanych eventów, których może być maksymalnie 8.
- *endevent* – tym kończymy definicję zdarzenia.

No dobrze, udało się nam wyjechać z Macierzewa. Dalsza jazda nie wymaga jakiegoś specjalnego sterowania, aż do momentu, kiedy dojedziemy do stacji Paszki Wielkie. Jednakże wiemy już jak uaktywniać semafory i przekładać zwrotnice, użyjemy kombinacji Shift+2, aby przejechać przez Paszki.

Na scenerii całkowskiej mamy dużo semaforów kształtowych. Odczytajmy w symulatorze ich nazwy:



W przypadku semaforów kształtowych symulator pokazuje nam również nazwę podtypu semafora. W rzeczywistości nazwa semafora widzianego na powyższym screenie to *pasz_f*. Jego sterowanie odbywa się poprzez podanie sygnałów Sr1, Sr2 lub Sr3. Po sprawdzeniu nazw zwrotnic oraz kolejnego semafora, możemy wpisać następny event:

```
event keyctrl02 multiple 0 none pasz_f_sr2 pasz_c_sr3 pasz_zwr3+ pasz_zwr2- pasz_zwr1-
          endevent
```

W powyższym przykładzie nie ma wpisu do tarczy ostrzegawczej, ani powtarzacza znajdującego się przed semaforem F w Paszkach Wielkich. O ile tarcze ostrzegawcze świetlne są w pełni zautomatyzowane i ustawiają się same w zależności od przypisanego dla nich semafora, to tarcze kształtowe musimy uruchamiać ręcznie. Zrobimy to później, najpierw dojedźmy do Jarkawek.

Kolejne kombinacje klawiszy używamy aby przejechać przez Całkowo, Wiliś Wschód, Wiliś, Pomianki, i wjechać do Jarkawek:

```
event keyctrl03 multiple 0 none cal_a_sr2 cal_f_sr2 cal_zwr1+ cal_zwr2+ cal_zwr4+ cal_zwr6+
          endevent
event keyctrl04 multiple 0 none lacz_a_S2 lacz_h_S2 laczni_zwr1+ laczni_zwr2+ laczni_zwr3+
          endevent
event kecytrl05 multiple 0 none wil_n_Sr2 wil_e_Sr2 wilis_zwr16+ wilis_zwr12+ wilis_zwr9+
          endevent
event keyctrl06 multiple 0 none pom_a_Sr2 pom_d_Sr2 pom_zwr1+ pom_zwr2+ pom_zwr3+ endevent
//W Jarkawkach możemy wjechać na bok:
event keyctrl07 multiple 0 none jar_a_Sr3 jar_zwr3- jar_zwr2+ endevent
```

W ten sposób napisaliśmy nasz scenariusz. Jest przejezdny od Macierzewa do Jarkawek, pod warunkiem, że będziemy przed każdą kolejną stacją wciskać kombinację klawiszy. Bardzo prymitywny scenariusz? Ale możliwy do napisania w 10 minut.

3.3. Jak rozwinąć najprostszy scenariusz

Najprostsze rozwiązanie nas jednak nie zadowala, więc chcemy trochę rozwinąć nasz przykład.

Po pierwsze: trochę nienaturalnie wygląda to, że semafor się zapala zanim jeszcze zwrotnice się przestawią do właściwej pozycji. Możemy zastąpić pierwszy event taką kombinacją:

```
event keyctrl01 multiple 0 none macierzewo_ustaw_zwr macierzewo_ustaw_sem endevent
event macierzewo_ustaw_zwr multiple 0 none mac_zwr9+ mac_zwr6ac mac_zwr5- mac_zwr4- mac_zwr3-
        endevent
event macierzewo_ustaw_sem multiple 5 none mac_d_S10 endevent
```

Zdefiniowaliśmy sobie dodatkowe dwa zdarzenia, pierwsze ustawi nam zwrotnice w Macierzewie, drugie ustawi semafor – przy czym zostanie wywołane z 5-sekundowym opóźnieniem (liczba 5 za *multiple*), a poprzez kombinację klawiszy Shift+1 wywołujemy oba zdarzenia. Efekt wizualny będzie taki, że najpierw zaczną się przestawiać zwrotnice, a po 5 sekundach zaświeci się semafor.

Po drugie: tarcze ostrzegawcze kształtowe niestety są nieruchome. Ale można je uaktywnić. Tarcze na scenerii całkowskiej są dwustawne, czyli mogą pokazywać sygnały Od1 i Od2. Zmodyfikujmy zatem ustawianie przebiegu w Paszkach:

```
event keyctrl02 multiple 0 none paszki_ustaw_zwr paszki_ustaw_sem endevent
event paszki_ustaw_zwr multiple 0 none pasz_zwr3+ pasz_zwr2- pasz_zwr1- endevent
event paszki_ustaw_sem multiple 5 none pasz_f_sr2 pasz_tof_od2 pasz_f1_sp2 pasz_c_sr3
        pasz_tob_od2 endevent
```

Po trzecie: przejazd w Paszkach Wielkich jest otwarty, podczas przejazdu wypadałoby aby został zamknięty. Nazwę przejazdu odczytamy w sposób identyczny jak to robiliśmy z semaforami. Poniżej może nie będzie screena, ale przejazd nazywa się *paszprz01*. Event zamykający ten przejazd nazywa się *paszprz01_zamykaj*. Przejazd zamyka się ok. 10 sekund, tak więc wypada, aby semafor się otworzył dopiero jak przejazd się zamknie. Modyfikacja eventów dla Paszek będzie wyglądała tak:

```
event keyctrl02 multiple 0 none paszki_ustaw_zwr paszki_ustaw_sem paszprz01_zamykaj endevent
event paszki_ustaw_zwr multiple 0 none pasz_zwr3+ pasz_zwr2- pasz_zwr1- endevent
event paszki_ustaw_sem multiple 11 none pasz_f_sr2 pasz_tof_od2 pasz_f1_sp2 pasz_c_sr3
        pasz_tob_od2 endevent
```

czyli semafory się ustawią 11 sekund po rozpoczęciu zamykania przejazdu. Po tym czasie rogatki będą już opuszczone.

Porada:

Jako narzędzie do wpisywania kodu został wspomniany program Notatnik. Jest to faktycznie narzędzie umożliwiające edycję plików .scn, .scm i .ctr, jednakże jest również bardzo ułomne. Zaleca się korzystanie z jakiegoś bardziej zaawansowanego edytora, np.: Notepad++. Oprócz ogromnej ilości niezwykle przydatnych funkcji ma możliwość podświetlenia składni używanej w języku eventów. Do tego celu należy ściągnąć jeden ze skryptów udostępnionych na forum symulatora. Więcej informacji w rozdziale 8.2.3.

3.4. Eventy warunkowe i losowanie

W poprzednim podrozdziale byliśmy w Paszkach, i może naszą uwagę zwrócił specyficzny układ torowy tej stacji. Być może zadaliśmy sobie pytanie: „A gdyby tak zrobić przelot drugim torem?” Wówczas może powstało pytanie: „A gdyby tak symulator losował, którym torem będzie przelot?” Takie rzeczy umożliwiają nam eventy warunkowe.

Na początek przyjrzyjmy się takiemu eventowi:

```
event keyctrl02 multiple 0 none zdarzenie1 condition propability 0.5 endevent
```

Przed *endevent* pojawiło się coś nowego: słowo *condition* to informacja dla parsera, że będziemy podawali warunek wykonania tego zdarzenia. Za słowem kluczowym dajemy kolejne słowo kluczowe, które określa typ warunku: *propability* oznacza, że będziemy podawać prawdopodobieństwo z jakim event się wykona. Następnie podajemy liczbę z przedziału <0;1>, oznaczającą prawdopodobieństwo wykonania zdarzenia. W tym przykładzie podana jest liczba *0.5*, co oznacza, że zdarzenie zostanie wykonane z prawdopodobieństwem 50%.

No dobra, ale co się stanie, jeżeli wynik losowania będzie negatywny? *zdarzenie1* się po prostu nie wykona. Z pomocą przychodzi nam jeszcze jedno słowo kluczowe, znane chyba wszystkim programistom:

```
event keyctrl02 multiple 0 none zdarzenie1 else zdarzenie2 condition propability 0.5 endevent
```

Przy tej składni, jeżeli wynik losowania był negatywny i nie wykona się *zdarzenie1*, to wykona się *zdarzenie2*. W samej składni zdarzenia losowego można umieścić więcej eventów, oczywiście pamiętając aby ich ilość nie była większa niż 8. Słowo kluczowe *else* oddziela dwa zbiory eventów przewidziane dla danego wyniku losowania.

Skoro umiemy już pisać zdarzenia losowe, to możemy ponownie zmodyfikować nasz przelot przez Paszki:

```
event keyctrl02 multiple 0 none paszki_ustaw_zwr1 paszki_ustaw_sem1 else paszki_ustaw_zwr2
          paszki_ustaw_sem2 condition propability 0.5 endevent
event paszki_ustaw_zwr1 multiple 0 none pasz_zwr3- pasz_zwr1+ paszprz01_zamykaj endevent
event paszki_ustaw_sem1 multiple 11 none pasz_f_sr3 pasz_tof_od2 pasz_f1_sp4 pasz_b_sr2
          pasz_tob_od2 endevent
event paszki_ustaw_zwr2 multiple 0 none pasz_zwr3+ pasz_zwr2- pasz_zwr1- paszprz01_zamykaj
          endevent
event paszki_ustaw_sem2 multiple 11 none pasz_f_sr2 pasz_tof_od2 pasz_f1_sp2 pasz_c_sr3
          pasz_tob_od2 endevent
```

W ten sposób urozmaiciliśmy sobie przejazd przez Paszki Wielkie – nie wiadomo, przez który tor dyżurny da nam przelot.

Istnieje jeszcze jeden typ zdarzenia losowego – opóźnienie o losowy interwał czasowy. Możemy zasymulować, że w Całkowie pracuje jakiś przymulony dyżurny, który da nam przelot z nie wiadomo jakim opóźnieniem od zgłoszenia. Najpierw przeróbmy eventy tak, aby w Całkowie działały tarcze ostrzegawcze i zamykał się przejazd:

```
event keyctrl03 multiple 0 none calkowo_ustaw_zwr calkowo_ustaw_sem cal_prz1_zamykaj endevent
event calkowo_ustaw_zwr multiple 0 none cal_zwr1+ cal_zwr2+ cal_zwr4+ cal_zwr6+ endevent
event calkowo_ustaw_sem multiple 11 none cal_a_sr2 cal_toa_od2 cal_f_sr2 cal_tof_od2 endevent
```

A teraz dopiszemy komendę, która opóźni nam uruchomienie wszystkich eventów o losową wartość czasu:

```
event keyctrl03 multiple 0 none calkowo_ustaw_zwr calkowo_ustaw_sem cal_prz1_zamykaj
          randomdelay 30 endevent
event calkowo_ustaw_zwr multiple 0 none cal_zwr1+ cal_zwr2+ cal_zwr4+ cal_zwr6+ endevent
event calkowo_ustaw_sem multiple 11 none cal_a_sr2 cal_toa_od2 cal_f_sr2 cal_tof_od2 endevent
```

Pogrubiono komendę *randomdelay*. Jej użycie opóźni wykonanie eventu o wartość czasu losowaną z przedziału <0;30> sekund. Wartość podawana za *randomdelay* jest górną granicą przedziału (podawaną w sekundach), z którego losowana jest wartość opóźnienia. Uwaga! Do losowanej wartości jest także dodawana wartość opóźnienia podana za *multiple*, w tym przypadku 0. Gdybyśmy zdarzenie zapisali w taki sposób:

```
event keyctrl03 multiple 10 none calkowo_ustaw_zwr calkowo_ustaw_sem cal_prz1_zamykaj
          randomdelay 20 endevent
```

wówczas zdarzenie wykona się z losowym opóźnieniem mieszczącym się w przedziale <10;30> sekund.

Znając już losowe wykonywanie eventów spróbujmy napisać takie zdarzenie, po którym z prawdopodobieństwem 20% przy wyjeździe z Macierzewa dostaniemy sygnał zastępczy zamiast S10:

```
event macierzewo_ustaw_sem multiple 5 none mac_d_Sz1 else mac_d_S10 condition propability 0.2
          endevent
```

Podsumujmy zatem nasz scenariusz. Dopisane zostały komentarze umożliwiające nawigację po kodzie źródłowym, oraz procedura zamykania przejazdu na wyjeździe z Macierzewa. Dla pierwszych trzech stacji będzie wyglądał następująco:

```
//Wyjazd z Macierzewa
event keyctrl01 multiple 0 none macierzewo_ustaw_zwr macierzewo_ustaw_sem macprz2_zamykaj
          endevent
event macierzewo_ustaw_zwr multiple 0 none mac_zwr9+ mac_zwr6ac mac_zwr5- mac_zwr4- mac_zwr3-
          endevent
event macierzewo_ustaw_sem multiple 11 none mac_d_Sz1 else mac_d_S10 condition propability 0.2
          endevent
//Przelot przez Paszki Wielkie
```

```
event keyctrl02 multiple 0 none paszki_ustaw_zwr1 paszki_ustaw_sem1 else paszki_ustaw_zwr2
          paszki_ustaw_sem2 condition propability 0.5 endevent
event paszki_ustaw_zwr1 multiple 0 none pasz_zwr3- pasz_zwr1+ paszprz01_zamykaj endevent
event paszki_ustaw_sem1 multiple 11 none pasz_f_sr3 pasz_tof_od2 pasz_f1_sp4 pasz_b_sr2
          pasz_tob_od2 endevent
event paszki_ustaw_zwr2 multiple 0 none pasz_zwr3+ pasz_zwr2- pasz_zwr1- paszprz01_zamykaj
          endevent
event paszki_ustaw_sem2 multiple 11 none pasz_f_sr2 pasz_tof_od2 pasz_f1_sp2 pasz_c_sr3
          pasz_tob_od2 endevent
//Przelot przez Całkowo
event keyctrl03 multiple 0 none calkowo_ustaw_zwr calkowo_ustaw_sem cal_prz1_zamykaj
          randomdelay 30 endevent
event calkowo_ustaw_zwr multiple 0 none cal_zwr1+ cal_zwr2+ cal_zwr4+ cal_zwr6+ endevent
event calkowo_ustaw_sem multiple 11 none cal_a_sr2 cal_toa_od2 cal_f_sr2 cal_tof_od2 endevent
```

Uzupełnienie pozostałych stacji o uruchamianie tarcz ostrzegawczych, zamykanie przejazdów, ewentualne zdarzenia losowe, pozostawiam użytkownikowi.

4. Pierwsze bardziej zaawansowane kroki

W tym rozdziale poznajemy jak przypisywać eventy z poziomu układu torów, wykonujemy pierwsze doświadczenia z udźwiękowieniem scenariuszy, oraz zgłębiamy tajniki zdarzeń warunkowych – poznajemy struktury danych, czyli komórki pamięci. Cały czas udoskonalamy nasz próbny scenariusz calkowo_cargo.

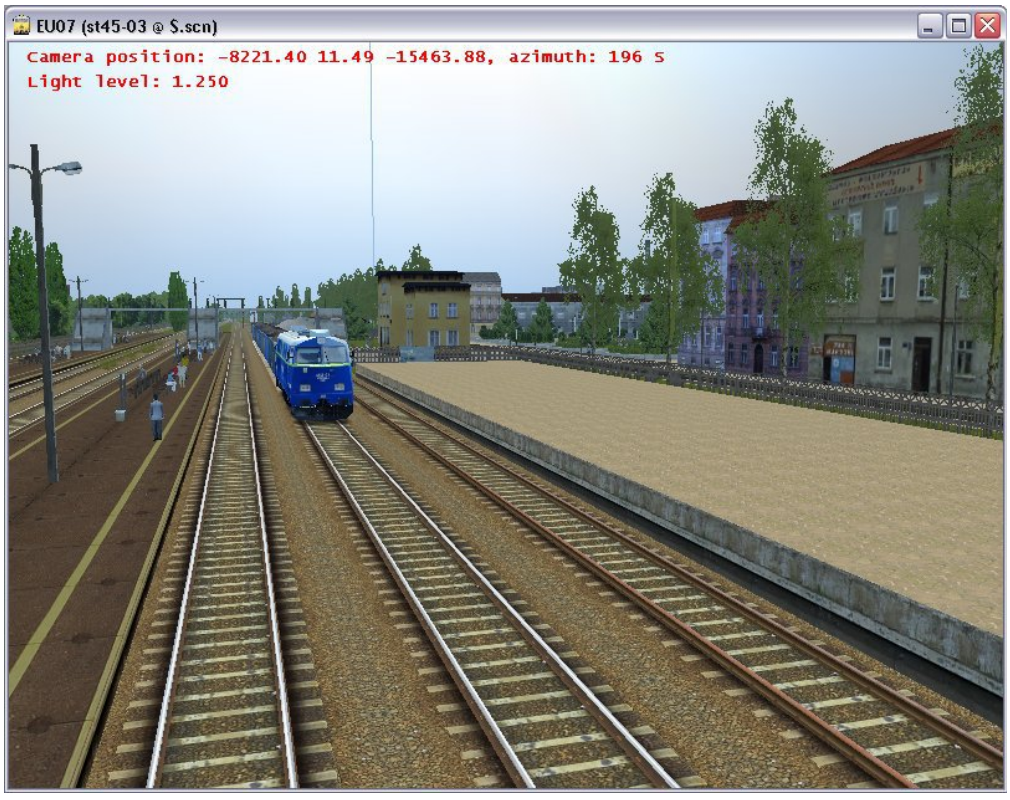
4.1. Udźwiękowienie scenariusza

Wyjazd z Macierzewa możemy otrzymać na sygnał zastępczy. Można do tego dorobić komunikat dyżurnego przez radiotelefon. W katalogu *sounds* odnajdziemy plik *wyjazd_sz.wav*, gdy go odtworzymy, usłyszymy komunikat nadawany przez radio: „Wyjazd będzie na sygnał zastępczy". W tym celu wpisujemy do naszego pliku z eventami (*events_cargo.ctr*) następującą linię:

```
node -1 0 macierzewo_radio1 sound -8221.4 11.49 -15463.88 wyjazd_sz.wav endsound
```

Zdefiniowaliśmy sobie w ten sposób obiekt dźwiękowy. Składnia jest następująca:

- *node* – definicja obiektu,
- *-1* – maksymalna odległość w metrach, z której obiekt będzie widoczny (w naszym przypadku słyszalny). Wartość -1 oznacza nieskończoność – taką wartość wpisujemy, jeśli chcemy mieć rozmowę przez radio.
- *0* – minimalna odległość w metra, przy której obiekt przestaje być widoczny (w naszym przypadku słyszalny). Zawsze wpisujemy tutaj 0.
- *macierzewo_radio1* – nazwa obiektu,
- *sound* – typ obiektu, w naszym przypadku jest to dźwięk,
- *X Y Z* – współrzędne, w których dźwięk ma być umieszczony. Współrzędne te spisujemy korzystając z funkcji symulatora: wychodzimy z kabiny w żądanym miejscu, ustawiamy się, następnie naciskamy klawisz F3 – na górze ekranu pojawią się współrzędne naszego aktualnego położenia. Ponieważ chcemy aby dźwięk był słyszalny w Macierzewie, wybieramy jakieś współrzędne znajdujące się w Macierzewie.
- *wyjazd_sz.wav* – podajemy nazwę pliku dźwiękowego, który będziemy chcieli odtwarzać.
- *endsound* – koniec definicji obiektu.



Sam obiekt to nie wszystko, musimy jeszcze napisać event, który wywoła ten obiekt:

```
event macierzewo_radio1 sound 0 macierzewo_radio1 1 endevent
```

Tutaj mamy pierwszy przykład zdarzenia innego niż *multiple* – dotychczas w naszym scenariuszu stosowaliśmy wyłącznie zdarzenia, które wywoływały inne zdarzenia, tutaj mamy natomiast zdarzenie, które wywołuje dźwięk. Jego składnia jest następująca:

- *event* – informacja dla parsera, że rozpoczęliśmy pisać zdarzenie,
- *macierzewo_radio1* – nazwa tego zdarzenia, może być inna niż nazwa obiektu dźwiękowego,
- *sound* – informacja, że zdarzenie wywołuje dźwięk,
- *0* – opóźnienie, z jakim zostanie odtworzony dźwięk,
- *macierzewo_radio1* – nazwa obiektu, który ma być odtworzony,
- *1* – zawsze wpisujemy 1,
- *endevent* – koniec składni zdarzenia.

Na sam koniec nie pozostaje nam nic innego, tylko dopisać to zdarzenie do innych wywoływanych w sytuacji, gdy wyjazd z Macierzewa jest na sygnał zastępczy:

```
event macierzewo_ustaw_sem multiple 11 none mac_d_Sz1 macierzewo_radio1 else mac_d_S10
          condition propability 0.2 endevent
```

Całościowo, kod dla Macierzewa wygląda następująco:

```
//Wyjazd z Macierzewa
event keyctrl01 multiple 0 none macierzewo_ustaw_zwr macierzewo_ustaw_sem macprz2_zamykaj
          endevent
event macierzewo_ustaw_zwr multiple 0 none mac_zwr9+ mac_zwr6ac mac_zwr5- mac_zwr4- mac_zwr3-
          endevent
event macierzewo_ustaw_sem multiple 11 none mac_d_Sz1 macierzewo_radio1 else mac_d_S10
          condition propability 0.2 endevent
node -1 0 macierzewo_radio1 sound -8221.4 11.49 -15463.88 wyjazd_sz.wav endsound
event macierzewo_radio1 sound 0 macierzewo_radio1 1 endevent
```

Ważna informacja:

Gdy przy wpisie node podajemy wartość -1, to wbrew pozorom wcale nie oznacza, że dźwięk będzie słyszalny z drugiego końca scenerii. W rzeczywistości jeżeli współrzędne dźwięku znajdują się w Macierzewie, to w Paszkach Wielkich dźwięk ten będzie już bardzo cichy. Optymalna odległość, z której dźwięk jest słyszalny to ok. 5 kilometrów, dlatego zawsze wybieramy współrzędne dla dźwięku blisko miejsca, z którego ma być słyszalny.

I jeszcze jedna uwaga na temat odtwarzanych plików *.wav*. Wymaga się, aby pliki te, były 1-kanałowe (mono), jakość 22050 próbek / sekundę, rozdzielczość 8 bit. Jeżeli korzystamy z gotowych nagrań w katalogu *sounds*, to nie musimy na to zwracać uwagi. Jeżeli nagrywamy nowe dźwięki np.: do radiotelefonu, to przestrzegajmy powyższego zalecenia. W przypadku dźwięków pozostałych np.: odgłosy natury i cywilizacji, można zaryzykować – wbrew zaleceniom – lepszą jakość nagrania. Symulator nie wysypie się od tego.

4.2. Przypisywanie eventów do torów

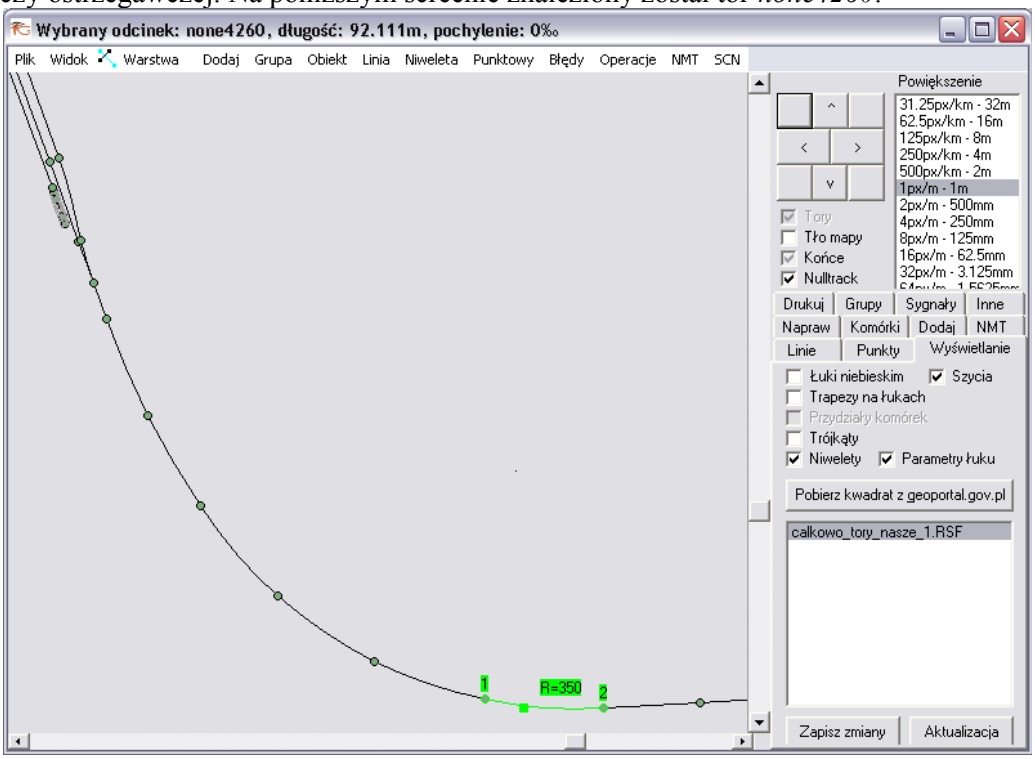
4.2.1. Metoda bezpośrednia

Najwyższy czas wyeliminować główną niedogodność napisanego przez nas scenariusza, czyli konieczność ciągłego naciskania kombinacji klawiszy Shift+2, Shift+3... W pierwszej kolejności pozbędziemy się konieczności ręcznego uruchamiania przebiegu przed stacjami Paszki, Całkowo, i dalej do Jarkawek. Zostawimy kombinację Shift+1 w Macierzewie, mechanizmy pozwalające na likwidację tego ograniczenia poznamy w kolejnych rozdziałach.

Na początek udrożnijmy Paszki Wielkie. Zmieńmy nazwę eventu uruchamiającego z *keyctrl02* na *przelot_paszki*:

```
event przelot_paszki multiple 0 none paszki_ustaw_zwr1 paszki_ustaw_sem1 else
          paszki_ustaw_zwr2 paszki_ustaw_sem2 condition propability 0.5 endevent
```

Teraz musimy przypisać go do odpowiedniego toru, tak aby pociąg przyjeżdżający z Macierzewa uruchomił ten event. Najpierw otwórzmy edytor Rainsted, do którego zaimportowaliśmy układ torowy naszego scenariusza. Zaznaczmy tor znajdujący się w pewnej odległości od semafora wjazdowego – najlepiej gdzieś na wysokości tarczy ostrzegawczej. Na poniższym screenie znaleziony został tor *none4260*:



Zgodnie z informacjami z rozdziału 2.3. aby zdarzenie *przelot_paszki* zostało uaktywnione przez pociąg nadjeżdżający od strony Macierzewa, musi być do definicji toru dodany wpis:

```
event1 przelot_paszki
```

ponieważ od strony Macierzewa znajduje się punkt 2 interesującego nas toru. Otwórzmy zatem z katalogu *slimson* plik *calkowo_tory_nasze.scm*, i poszukajmy toru *none4260*. Najpierw musimy sprawdzić, czy nie ma tam już przypisanego jakiegoś eventu – jeżeli tak, to musimy sobie znaleźć inny tor. Tak jednak nie jest, bo wprawdzie był przypisany event *pzg_sem_f*, ale go usunęliśmy podczas przygotowania pliku z torami (rozdział 2.3.). Tak więc możemy przypisać event do toru, za współrzędnymi, ale przed informacją o prędkości szlakowej (velocity):

```
node 1000 0 none4260 track normal 91.6294 1.435 0.25 25.0 20 0 flat vis
 rail_screw_used2 4 tpd-old1 0.2 0.5 1.1
-2975.83 4.2 -7177.44 -1.65
-29.8052 0.0 -6.22705
31.2026 0.0 -1.81543
-3067.69 4.2 -7184.23 -1.65
350.0
event1 przelot_paszki
velocity 70.0
endtrack
```

Możemy teraz sprawdzić doświadczalnie: kiedy pojedziemy z Macierzewa, zaraz za tarczą ostrzegawczą przed Paszkami Wielkimi, po najechaniu na tor *none4260* powinien zostać wywołany event *przelot_paszki*. Nie ma już zatem konieczności naciskania kombinacji klawiszy Shift+2.

Następnie możemy zrobić to samo dla Całkowa:

- Nazwę eventu *keyctrl03* zmieniamy na *przelot_calkowo* (nazwa oczywiście może być dowolna inna, ale należy także pamiętać, że każdy event musi mieć unikalną nazwę),
- Wyszukujemy odpowiedni tor do przypisania eventu – może to być *none3777*,

- Sprawdzamy w edytorze kierunek ułożenia toru, ponieważ pociąg wjedzie najpierw na punkt 1, nazwa eventu musi być poprzedzona *event2*,
- Odnajdujemy wpis toru, i dopisujemy wywołanie:

```
node 1000 0 none3777 track normal 75.0 1.435 0.25 25.0 20 0 flat vis
 rail_screw_used1 4 tpbps-new2 0.2 0.5 1.1
-168.522 0.200002 -762.243 0.0
17.6777 0.0 17.6777
-17.6777 0.0 -17.6777
-115.489 0.200002 -709.21 1.28
0
event2 przelot_calkowo
velocity 70.0
endtrack
```

4.2.2. Metoda zdalna

Opisany w poprzednim podrozdziale sposób przypisywania eventów do torów ma głęboką tradycję, jest również najbardziej stabilnym sposobem, jednakże ma znaczącą wadę: wymaga osobnego układu torów dla każdego scenariusza. Taki sposób pisania scenariuszy już się zemścił na developerach MaSzyny, kiedy to w 2014 r. była poprawiana sieć trakcyjna. Ponadto wprowadzanie wszelkich poprawek do scenerii jest utrudnione, gdyż trzeba ją wprowadzać we wszystkich „klonach". Pół biedy, jeżeli wszystkie te klony mają identycznie nazwane tory, prawdziwy koszmar scenerii zaczyna się, gdy scenarzyści dowolnie zmieniają sobie nazwy torów (obecnie L053, niegdyś L61). Aby zapobiec podobnym problemom w przyszłości można zastosować 2 rozwiązania.

Pierwsze rozwiązanie: pisać scenariusze w taki sposób, aby do każdego wykorzystywać jeden układ torowy. Zadanie ekstremalnie trudne, dlatego nie będzie teraz omawiane. W rozdziale poznamy algorytm umożliwiający pisanie wielu scenariuszy na wspólnym układzie torowym. Niemniej jednak sposób pisania scenariuszy na jednym układzie torów jest wciąż zagwozdką wielu developerów. Przykładem takich scenerii / scenariuszy są:

- Całkowo Odyseja Spalinowa – wprawdzie są w użyciu 3 klony torów, ale głównie z uwagi na wersję letnią / zimową / letnią zarośniętą. Każdy scenariusz może być uruchomiony w wykorzystaniem dowolnego innego klona torów,
- Drawinowo (choć głównie dlatego, że dla tej scenerii dostępny jest jeden scenariusz),
- Krzyżowa (choć głównie dlatego, że dla tej scenerii dostępny jest jeden scenariusz),
- Quarkmce2007,
- Całkowo v2.

Pełen uniwersalizm – czyli można bez problemu w scenariuszach dopisać kolejne pociągi, albo napisać nowy scenariusz bez ingerencji w układ torów – osiągnięto tylko w Odysei Spalinowej i w Quarkmce2007.

Drugie rozwiązanie: przypisywanie eventu do toru z poziomu pliku .ctr. Od 2014 roku w MaSzynie jest taka możliwość. Możemy nasze eventy *przelot_paszki* i *przelot_calkowo* nazwać w inny sposób, albo dla większej przejrzystości dodać jeszcze po jednym evencie:

```
event none4260:event1 multiple 0 none przelot_paszki endevent
event none3777:event2 multiple 0 none przelot_calkowo endevent
```

I na samym początku pliku *event_cargo.ctr* dopisujemy informację dla parsera, że korzystamy ze zdalnego przypisywania eventów do torów:

```
config hiddenevents 1 endconfig
```

Wówczas nie musimy wpisywać eventów bezpośrednio do definicji toru, w ogóle możemy nie ruszać pliku z torami. Ale... niestety jest kilka ale:

- Musimy na samym początku pliku wpisać tekst informujący o użyciu zdalnego przypisania eventów, jeśli go nie będzie – nic nam nie zadziała!
- Jeżeli będziemy chcieli przypisać w ten sposób jakiś event do toru, w którego definicji jest już jakiś event wpisany, to event się nie wykona, a symulator nawet nie zgłosi błędu... (patrz uwaga w rozdz. 2.3.)
- Tor do którego przypisujemy event musi mieć nazwę. Może w Całkowie to nie jest problem, ale na L053 już tak.

Niemniej te wszystkie „ale" nie powodują, że taki sposób przypisywania eventów do torów jest

niewłaściwy. Taki sposób z powodzeniem został wykorzystany w Odysei Spalinowej (w ograniczonym stopniu) i w jednym ze scenariuszy na L053.

4.2.3. Wybór metody przypisywania eventów

Metoda bezpośrednia jest najlepsza, jeżeli chodzi o eventy sterujące automatyką scenerii – wygaszanie semaforów, przypisanie semaforów, przypisanie W4, W5 i W6, itd. Takie eventy są ewidentnie związane z infrastrukturą scenerii, toteż powinny być przypisane na stałe do konkretnych torów.

Dla początkujących scenarzystów proponuję metodę zdalną. Oczywiście, możemy napisać na potrzeby edukacyjne scenariusz z odrębnym układem torów, ale pamiętajmy, że musimy również pilnować porządku w plikach MaSzyny, zwłaszcza, że niekoniecznie nam przyjdzie kiedyś wprowadzać poprawki.

Pozostajemy zatem przy metodzie zdalnej. Kod dla pierwszych 3 stacji wygląda następująco:

```
config hiddenevents 1 endconfig
event none4260:event1 multiple 0 none przelot_paszki endevent
event none3777:event2 multiple 0 none przelot_calkowo endevent
//Wyjazd z Macierzewa
event keyctrl01 multiple 0 none macierzewo_ustaw_zwr macierzewo_ustaw_sem macprz2_zamykaj
        endevent
event macierzewo_ustaw_zwr multiple 0 none mac_zwr9+ mac_zwr6ac mac_zwr5- mac_zwr4- mac_zwr3-
        endevent
event macierzewo_ustaw_sem multiple 11 none mac_d_Sz1 macierzewo_radio1 else mac_d_S10
        condition propability 0.2 endevent
node -1 0 macierzewo_radio1 sound -8221.4 11.49 -15463.88 wyjazd_sz.wav endsound
event macierzewo_radio1 sound 0 macierzewo_radio1 1 endevent
//Przelot przez Paszki Wielkie
event przelot_paszki multiple 0 none paszki_ustaw_zwr1 paszki_ustaw_sem1 else
        paszki_ustaw_zwr2 paszki_ustaw_sem2 condition propability 0.5 endevent
event paszki_ustaw_zwr1 multiple 0 none pasz_zwr3- pasz_zwr1+ paszprz01_zamykaj endevent
event paszki_ustaw_sem1 multiple 11 none pasz_f_sr3 pasz_tof_od2 pasz_f1_sp4 pasz_b_sr2
        pasz_tob_od2 endevent
event paszki_ustaw_zwr2 multiple 0 none pasz_zwr3+ pasz_zwr2- pasz_zwr1- paszprz01_zamykaj
        endevent
event paszki_ustaw_sem2 multiple 11 none pasz_f_sr2 pasz_tof_od2 pasz_f1_sp2 pasz_c_sr3
        pasz_tob_od2 endevent
//Przelot przez Całkowo
event przelot_calkowo multiple 0 none calkowo_ustaw_zwr calkowo_ustaw_sem cal_prz1_zamykaj
        randomdelay 30 endevent
event calkowo_ustaw_zwr multiple 0 none cal_zwr1+ cal_zwr2+ cal_zwr4+ cal_zwr6+ endevent
event calkowo_ustaw_sem multiple 11 none cal_a_sr2 cal_toa_od2 cal_f_sr2 cal_tof_od2 endevent
```

4.3. Struktury danych – komórki pamięci

Pora na wprowadzenie do jednego z najbardziej użytecznych w scenariuszach mechanizmu: możliwości definiowania własnych zmiennych.

Komórka pamięci to struktura (używając języka programistów) zawierająca 3 zmienne: ciąg tekstowy, oraz 2 zmienne liczbowe (stało lub zmiennoprzecinkowe). Na początek zdefiniujmy sobie taką jedną komórkę:

```
node -1 0 komorka1 memcell 1.0 1.0 1.0 tekst 1 2 none endmemcell
```

Składnia jest następująca:

- *node* – informacja, że definiujemy obiekt,
- *-1 0* – odległości, z której obiekt będzie widzialny, dla komórek pamięci zawsze wpisujemy -1 0,
- *komorka1* – nazwa naszej komórki,
- *memcell* – informacja dla parsera, że obiekt jest komórką pamięci,
- *1.0 1.0 1.0* – współrzędne położenia, na ogół można wpisać dowolne wartości, tylko w niektórych przypadkach ma to znaczenie,
- *tekst 1 2* – wartości trzech zmiennych wewnątrz komórki, pierwsza wartość to ciąg tekstowy, można wpisywać dowolne rzeczy, dwie kolejne wartości są wartościami liczbowymi,
- *none* – możliwość przypisania komórki pamięci do innego obiektu, np.: toru. Na ogół jednak się z tego nie korzysta, wówczas wpisuje się none,
- *endmemcell* – informacja, że zakończyliśmy definicję obiektu.

Zamiast podawania wszystkich wartości, możemy wykorzystać znak *. Np.:

```
node -1 0 komorka1 memcell 1.0 1.0 1.0 * 0 0 none endmemcell
```

Wówczas komórka po zdefiniowaniu będzie miała dwie zmienne liczbowe zdefiniowane jako 0, natomiast zmienna tekstowa może być dowolna. Znak * oznacza wartość dowolną.

Do dyspozycji mamy 2 operacje na komórkach: zmiana wartości, oraz dodawanie wartości. Do tego celu musimy posłużyć się specjalnymi eventami:

```
event komorka1_up updatevalues 0 komorka1 * 2 3 endevent
```

Taki event ma następującą składnię:

- *komorka1_up* – nazwa eventu zmieniającego wartość komórki,
- *updatevalues* – informacja dla parsera, że event zmienia wartość jakieś komórki danych,
- *0* – opóźnienie w sekundach, po którym nastąpi wykonanie eventu (zmiana wartości w komórce),
- *komorka1* – nazwa komórki pamięci, w której będą zmieniane dane,
- ** 2 3* – nowy zestaw danych. Użycie symbolu * powoduje, że dana zmienna nie będzie zmieniana. W tym przypadku zmienne liczbowe zostaną ustawione na 2 i 3, natomiast zmienna tekstowa pozostanie taka sama, jak przed uruchomieniem eventu,
- *endevent* – zakończenie eventu zmiany wartości.

W ten sposób wygląda sztywna zmiana wartości w komórce pamięci. Istnieje również możliwość dodania wartości – składnia eventu jest identyczna jak dla powyższego przykładu, jednakże event zamiast *updatevalues* musi nazywać się *addvalues*.

Jeżeli *komórka1* początkowo miała wartości: *tekst 2 3*, to po wykonaniu eventu:

```
event komorka1_add addvalues 0 komorka1 * 1 1 endevent
```

będzie miała wartości *tekst 3 4*.

Zrozumiałe? W kolejnym podrozdziale odkryjemy po co są struktury danych, i jak gigantyczne możliwości otworzą się przed nami, gdy zaczniemy z nich korzystać.

4.4. Eventy warunkowe – wykorzystanie komórek

W poprzednim rozdziale poznaliśmy najprostsze eventy warunkowe – takie, których wykonanie jest uzależnione wyłącznie od wyniku losowania. Mamy jednak do wyboru mechanizm, który pozwoli nam na warunkowe wykonywanie eventów, w zależności od stanu zmiennych w komórce pamięci. Np.:

```
event zdarzenie1 multiple 0 komorka1 zdarzenie2 condition memcompare * 0 0 endevent
```

Powyższy event warunkowy ma następującą składnię:

- *zdarzenie1* – nazwa zdarzenia warunkowego,
- *multiple* – jest to event uruchamiający inne eventy,
- *0* – event uruchomiony zostanie z opóźnieniem 0,
- *komorka1* – tutaj zmiana w porównaniu do dotychczas pisanych eventów, wcześniej było none, teraz musimy podać nazwę komórki, której wartości będą sprawdzane w warunku,
- *zdarzenie2* – zdarzenie które zostanie wykonane, jeśli test warunku wypadnie pozytywnie,
- *condition* – słowo kluczowe informujące, że wykonanie eventu jest warunkowe,
- *memcompare* – słowo kluczowe informujące, że warunek zostanie spełniony, jeżeli podana komórka pamięci (tutaj *komorka1*) będzie miała takie same wartości jak:
- ** 0 0* – tutaj podajemy wartości, które musi mieć komórka pamięci, aby event został wykonany. Symbol * oznacza, że dana zmienna nie będzie brana pod uwagę – czyli event zostanie wykonany, jeżeli *komorka1* będzie miała dowolną wartość tekstową, oraz obie wartości liczbowe będą równe 0.

  Możemy również wykorzystać wcześniej poznane słowo kluczowe *else*:

```
event zdarzenie1 multiple 0 komorka1 zdarzenie2 else zdarzenie3 condition memcompare * 0 0
          endevent
```

czyli jeśli test wypadnie negatywnie (*komorka1* będzie miała inne wartości liczbowe niż 0 i 0), to wykonany zostanie event o nazwie *zdarzenie3*.

Do czego można wykorzystać komórki pamięci? W zasadzie do wszystkiego, na co pozwolą nam pomysły na scenariusze. Przypuśćmy, że chcemy, aby event *przelot_calkowo* został wykonany tylko jeden raz, tak aby kolejny pociąg po podjechaniu do semafora wjazdowego w Całkowie nie otrzymał wjazdu. Stwórzmy sobie zatem pomocniczą komórkę pamięci:

```
node -1 0 calkowo_kom memcell 1.0 1.0 1.0 * 0 0 none endmemcell
```

oraz event, który zmieni nam wartość tej komórki:

```
event calkowo_kom_up updatevalues 0 calkowo_kom * 1 1 endevent
```

Na koniec zmieniamy składnię eventu *przelot_calkowo*:

```
event przelot_calkowo multiple 0 calkowo_kom calkowo_ustaw_zwr calkowo_ustaw_sem
          cal_prz1_zamykaj calkowo_kom_up condition memcompare * 0 0 endevent
```

Zrozumieliście? Przy pierwszym wywołaniu komórka pamięci ma wartości * 0 0, toteż event się wykona. Ale event wraz z wykonaniem uruchomi event (*calkowo_kom_up*), który zmieni wartości komórki pamięci, toteż przy ponownym wywołaniu eventu *przelot_calkowo* już nie nastąpi jego uruchomienie.

Porada:

W tym momencie poznałeś/łaś już praktycznie wszystkie mechanizmy pozwalające na pisanie zaawansowanych scenariuszy. Wszystko co dalej następuje, to już tylko drobne sztuczki pozwalające na precyzyjniejsze działanie scenariuszy.

A zatem miłego i owocnego tworzenia scenariuszy! ;-)

No dobra, jeszcze zostańcie, można się nauczyć wielu ciekawych rzeczy.

Nie da się opisać i zaprezentować wszystkich możliwych sposobów użycia komórek pamięci. Polecam poszperać w katalogu scenery, wyszukać wszelkie pliki .ctr, i pooglądać, jak twórcy scenariuszy wykorzystują komórki pamięci do sterowania scenariuszami. Bardzo często są wykorzystywane przy obsłudze bardziej skomplikowanych zdarzeń losowych.

4.5. Eventy rekurencyjne

A co to takiego? A może przyszło do głowy wam pytanie, czy event potrafi wywołać samego siebie? Taka konstrukcja jest możliwa, np:

```
event zdarzenie1 multiple 5.5 none zdarzenie2 zdarzenie1 endevent
```

Czy jest jakieś „ale”? Niestety jest – jeżeli czas opóźnienia wykonania eventu jest krótszy niż 5 sekund, wówczas zdarzenie nie wywoła samo siebie. Przypuśćmy, że jednak zachodzi konieczność stworzenia eventu rekurencyjnego, który będzie siebie wywoływał co 2 sekundy. Da się to zrobić taką kombinacją:

```
event zdarzenie1 multiple 2 none zdarzenie2 zdarzenie1a endevent
event zdarzenie1a multiple 2 none zdarzenie2 zdarzenie1 endevent
```

Taki mechanizm jest bardzo często spotykany w Odysei Spalinowej.

Do czego można wykorzystywać eventy rekurencyjne? O tym w następnym podrozdziale.

4.6. Jeszcze o eventach warunkowych

Do tej pory poznaliśmy następujące możliwości tworzenia eventów warunkowych – czyli następujące słowa kluczowe, które można wpisać po *condition*:

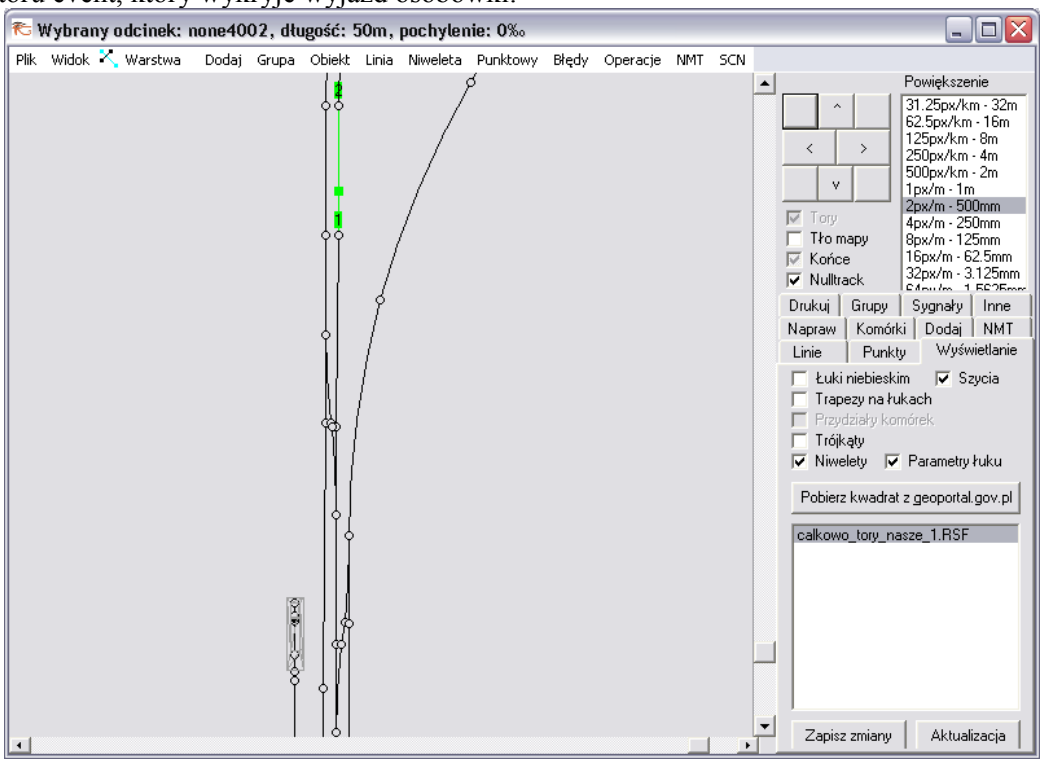
- *propability* – warunkiem jest pozytywny wynik losowania, podaje się prawdopodobieństwo uzyskania pozytywnego wyniku,

- *memcompare* – porównanie, czy podana komórka pamięci posiada identyczne wartości jak w teście.

Oprócz tego są jeszcze dwa inne typy eventów warunkowych – czyli następne słowa kluczowe:

- *trackfree* – event zostanie wykonany, jeżeli tor o podanej nazwie jest pusty,
- *trackoccupied* – event zostanie wykonany, jeżeli tor o podanej nazwie jest zajęty.

Ich działanie najlepiej pokazać na jakimś przykładzie. W tym celu rozwińmy nieco początek misji *calkowo_cargo*. Przypuśćmy, że wyjazd dostajemy dopiero, gdy ze stacji wyjedzie pociąg osobowy. Musimy przypisać do toru event, który wykryje wyjazd osobówki:



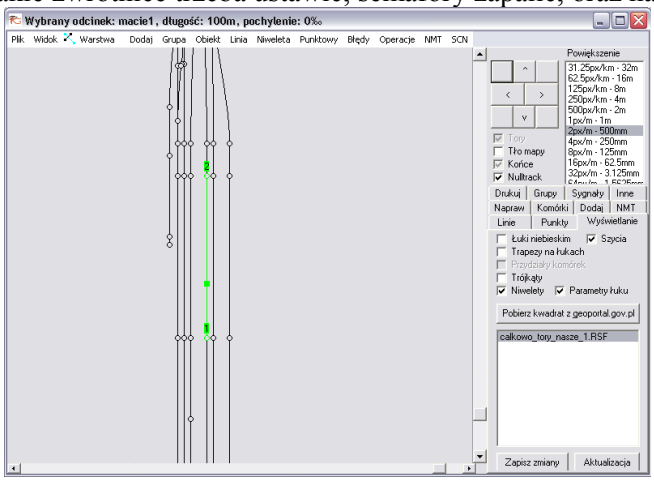
Idealnie nadający się do tego celu tor to *none4002*. Tworzymy sobie zatem event:

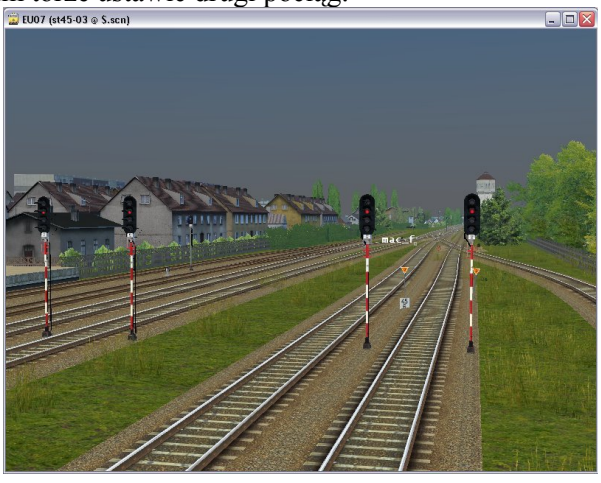
```
event none4002:event2 multiple 0 none macierzewo_wyjazd_osobowki endevent
event macierzewo_wyjazd_osobowki multiple 0 none macierzewo_wyjazd_cargo endevent
```

macierzewo_wyjazd_cargo to będzie nowa nazwa dla naszego dotychczasowego eventu keyctrl01:

```
//Wyjazd z Macierzewa
event macierzewo_wyjazd_cargo multiple 0 none macierzewo_ustaw_zwr macierzewo_ustaw_sem
          macprz2_zamykaj endevent
```

Teraz kombinacją klawiszy Shift+1 będziemy załączali wyjazd dla osobówki z Macierzewa. Musimy sprawdzić, jakie zwrotnice trzeba ustawić, semafory zapalić, oraz na jakim torze ustawić drugi pociąg:





A zatem dodatkowy pociąg wstawiamy na tor *macie1*, wyjazd następuje pod semaforem *mac_f*. Teraz po naciśnięciu kombinacji Shift+1 osobówka dostanie wyjazd:

```
event keyctrl01 multiple 0 none mac_os_zwr mac_os_sem endevent
event mac_os_zwr multiple 0 none mac_zwr7+ mac_zwr5+ mac_zwr4+ mac_zwr2+ endevent
event mac_os_sem multiple 5 none mac_f_S2 endevent
```

Do pliku calkowo_cargo.scn dopisujemy nasz pociąg osobowy. Nagłówek powinien mieć postać:

```
trainset rozklad macie1 10 0.1
```

Podana prędkość początkowa 0.1 oznacza, że pociąg w momencie uruchomienia symulacji będzie stał, ale zostanie automatycznie uruchomiony i przygotowany do jazdy. Jeżeli podamy mu semafor, to AI powinno ruszyć (oczywiście, jeżeli nie pousuwaliśmy na etapie rozdziału 2.3. eventów z torów! AI nie ruszy, jeżeli nie będzie przypisanego eventu *_sem_info*). Dobór składu pozostawiam użytkownikowi.

No dobrze, mamy dodatkowy pociąg, uruchamiamy wyjazd – osobówka rusza, wjeżdża na tor *none4002*, uruchamia wyjazd dla naszego pociągu, czyli możemy wyjeżdżać. Zadziała? Raczej zadziała, ale zwróćmy uwagę na drobny szczegół, w momencie gdy pociąg osobowy wjedzie na tor none4002 i uruchomi event wyjazdowy dla naszego pociągu, to przełoży zwrotnicę *mac_zwr4*. Jeżeli pociąg jest długi, to zostanie podcięty i będziemy mieli katastrofę symulatorową.

Można takiemu zdarzeniu zapobiec zwiększając opóźnienie załączenia eventu ale to niestety nie jest pewna metoda. W takiej sytuacji przychodzi nam z pomocą test na zajętość toru. Zmodyfikujmy nieznacznie nasze eventy:

```
event none4002:event2 multiple 0 none macierzewo_wyjazd_osobowki endevent
event macierzewo_wyjazd_osobowki multiple 0 mac_zwr4 macierzewo_wyjazd_cargo condition
          trackfree endevent
```

Tam gdzie wpisujemy *none* lub nazwę komórki wpisaliśmy nazwę krytycznej zwrotnicy i dodaliśmy warunek na zwolnienie toru. Wyobraźmy sobie teraz, że rusza z Macierzewa długi pociąg, uruchamia event *macierzewo_wyjazd_osobowki*, nie zdążył jeszcze zwolnić zwrotnicy *mac_zwr4*, toteż event się nie uruchamia i nie ma katastrofy. Niby fajnie, ale po chwili się również przekonamy, że utknęliśmy w Macierzewie na dobre.

Tutaj wykorzystamy poznaną w poprzednim podrozdziale możliwość rekurencyjnego wywoływania eventów. Wprowadźmy jeszcze jedną modyfikację:

```
event none4002:event2 multiple 0 none macierzewo_wyjazd_osobowki endevent
event macierzewo_wyjazd_osobowki multiple 6 mac_zwr4 macierzewo_wyjazd_cargo else
          macierzewo_wyjazd_osobowki condition trackfree endevent
```

Co się teraz stanie jak będzie wyjeżdżał długi pociąg? Nie wykona się event *macierzewo_wyjazd_cargo*, ale po 6 sekundach ponownie się uruchomi event *macierzewo_wyjazd_osobowki* i sprawdzi, czy zwrotnica jest pusta. I tak do momentu, aż zwrotnica faktycznie będzie zwolniona, wówczas dostaniemy wyjazd do Jarkawek.

4.7. Eventlauncher

Do tej pory poznaliśmy trzy sposoby uruchamiania eventów:

- Poprzez inny event,
- Poprzez najechanie pociągu na tor – event musi być wówczas przypisany do toru,
- Poprzez naciśnięcie kombinacji klawiszy Shift+cyfra.

Istnieje jeszcze jedna metoda, czyli stworzenie specjalnego obiektu, który będzie uruchamiał zdarzenia w określonej sytuacji. Najpierw przeanalizujemy jego budowę na poniższym przykładzie:

```
node -1 0 uruchomienie_scenariusza eventlauncher 1.0 1.0 1.0 -1 none 1201 macierzewo_wyjazd
          none end
```

- *node -1 0* – informacja, że zadeklarowaliśmy obiekt, który będzie zawsze widoczny,
- *uruchomienie_scenariusza* – nazwa obiektu,
- *eventlauncher* – typ obiektu, czyli „uruchamiacz" eventów,
- *1.0 1.0 1.0* – współrzędne, w których obiekt się znajduje. W odróżnieniu od komórek pamięci często współrzędne mają znaczenie, ale o tym niżej,

- *-1* – maksymalna odległość użytkownika od obiektu, aby zdarzenie mogło być uruchomione, -1 oznacza nieskończoność,
- *none* – aby event został uruchomiony, nie trzeba nacisnąć żadnego klawisza,
- *1201* – godzina, o której zostanie uruchomiony dalej podany event,
- *macierzewo_wyjazd* – nazwa eventu, który zostanie uruchomiony, gdy zostaną podane wcześniej warunki – w tym przypadku po prostu musi wybić godzina 12:01.
- *none* – parametr nieużywany, gdy event nie jest uruchamiany przez naciśnięcie klawisza.

W ten oto sposób pozbyliśmy się przykrej konieczności użycia kombinacji klawiszy Shift+1 po uruchomieniu scenariusza – event wyjazdowy zostanie uruchomiony przez eventlaunchera o godzinie 12:01.

W praktyce eventlaunchery mogą uruchamiać eventy także na inne sposoby. Można uruchamiać zdarzenie w sposób cykliczny:

```
node -1 0 eventlauncher1 eventlauncher 1.0 1.0 1.0 -1 none -5 zdarzenie1 none end
```

W powyższym przykładzie *zdarzenie1* będzie uruchamiane co 5 sekund. Zamiast godziny uruchomienia podajemy none, następnie podajemy interwał czasowy w sekundach i ze znakiem -.

Jeszcze inny sposób uruchomienia zdarzenia to użycie klawisza:

```
node -1 0 uruchomienie_scenariusza eventlauncher -8221.4 11.49 -15463.88 500 w 0
          macierzewo_wyjazd macierzewo_wyjazd end
```

- *-8221.4 11.49 -15463.88* – współrzędne, w których znajduje się obiekt,
- *500* – maksymalna odległość użytkownika od obiektu, czyli 500 metrów,
- *w* – zdarzenie się załączy, jeżeli użytkownik naciśnie klawisz w, i jednocześnie spełni warunek minimalnej odległości od obiektu.
- *0* – czas, co który jest sprawdzane naciśnięcie żądanego klawisza,
- *zdarzenie1* – zdarzenie, które będzie uruchamiane po naciśnięciu klawisza,
- *zdarzenie1* – zdarzenie, które będzie uruchamiane po naciśnięciu klawisza i przy jednoczesnym przytrzymaniu Shift.

W ten sposób mamy alternatywę do naciskania kombinacji klawiszy Shift+cyfra. Proszę zauważyć, że możemy to obwarować warunkami odległości, czyli jak przemieścimy się do Paszek i tam naciśniemy klawisz *w*, to nie otrzymamy wyjazdu z Macierzewa.

Aktualnie w scenariuszach jest spotykany głównie eventlauncher wykorzystujący uruchamianie o określonej godzinie. Uruchamianie klawiszem jest wykorzystywane w przejazdach niestrzeżonych, wówczas jak jedziemy szlakiem, widzimy W6 i naciśniemy klawisz dający sygnał Rp1, to nieświadomie uruchamiamy także poprzez eventlaunchera event zamykający najbliższy przejazd. Przykładem masowego wykorzystania w scenariuszu takiego sposobu uruchamiania eventów był *calkowo_sn61*, w najnowszej MaSzynie niedostępny.

Ważna informacja:

Uruchamianie eventów klawiszem niesie ze sobą pewne zagrożenie – mianowicie, eventlauncher działa tak szybko, że jak naciśniemy i puścimy klawisz, to w tym czasie zdąży kilka razy wywołać żądane zdarzenie. Należy albo zdarzenie wyposażyć w zabezpieczenie przed wielokrotnym uruchomieniem, albo pokombinować z liczbą znajdującą się za literą oznaczającą wciskany klawisz (w przykładzie jest to 0, może trzeba -0.5?) - do tej pory nikt nie przedstawił pełnego rozwiązania tego problemu.

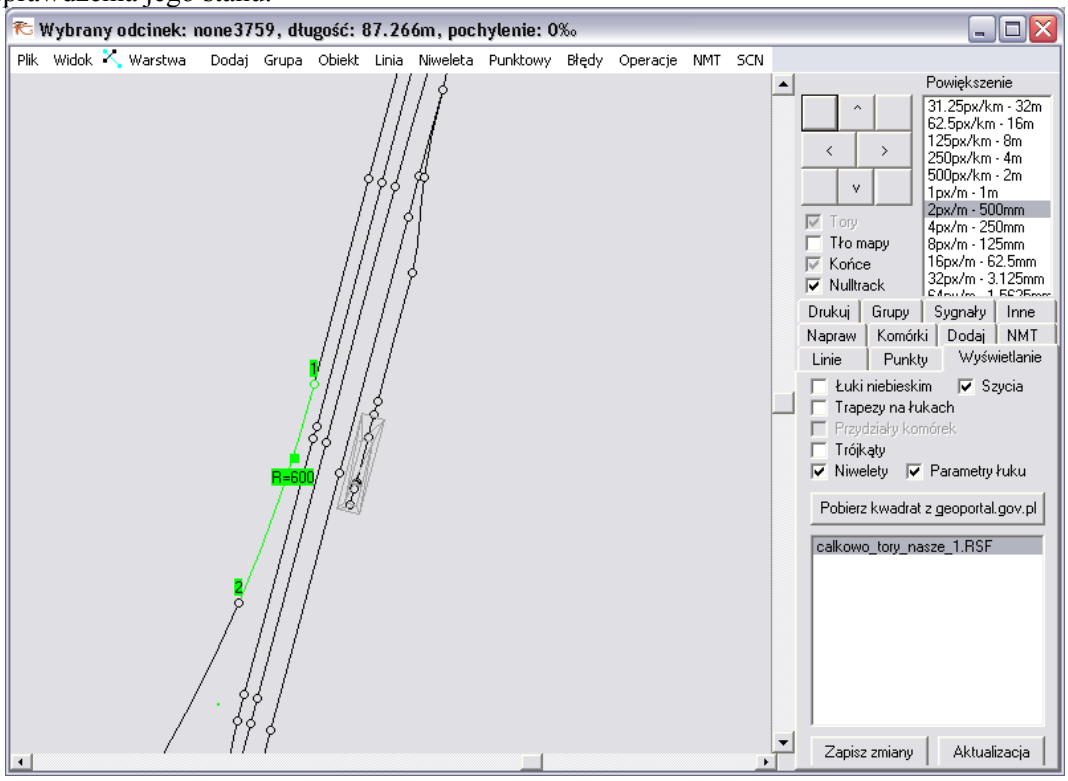
4.8. Odcinki izolowane

4.8.1. Definicja i wstawianie odcinków

Odcinek izolowany to próba zastąpienia eventów nowszym i lepszym systemem. Przynajmniej z założenia, gdyż mimo upływu wielu lat od ich wprowadzenia developerzy wciąż nie do końca wiedzą jak z nich korzystać, a niektórzy uważają, że powinna być nieznacznie zmieniona ich formuła. Mniejsza o to. Spójrzmy na mapę scenerii Całkowo i odszukajmy przy stacji Całkowo tor prowadzący do tartaku. Każdy z torów posiada w swojej definicji przypisanie odcinka izolowanego, np.:

```
node 1000 0 none3759 track normal 87.2664 1.435 0.25 25.0 20 0 flat vis
 rail_screw_rused1 4 tpd-oil3 0.2 0.5 1.1
-35.9481 0.200002 120.978 0.0
7.55865 0.0 -28.2093
-11.5098 0.0 26.6733
-7.29203 0.200002 38.5516 0.0
-600.0
isolated tor_w_tartaku
endtrack
```

Jak sprawdzimy pozostałe tory prowadzące do tartaku, to wszystkie będą miały ten sam wpis, po eventach, i przed informacją o prędkości szlakowej. Wszystkie te tory tworzą razem odcinek o nazwie *tor_w_tartaku*. Symulator automatycznie dla każdego odcinka udostępnia nam eventy uruchamiane w momencie wjazdu na odcinek izolowany, albo w momencie opuszczenia takiego odcinka. Można również skorzystać z odcinka jak z komórki pamięci do sprawdzenia jego stanu.



4.8.2. Korzystanie z odcinków izolowanych

Gdy skorzystamy z takiej składni:

```
event tor_w_tartaku:busy multiple 0 none zdarzenie1 endevent
```

to *zdarzenie1* będzie się uruchamiało w momencie, gdy jakiś pojazd wjedzie na odcinek izolowany. Jeżeli natomiast interesuje nas zdarzenie, które się wywoła w momencie opuszczenia takiego odcinka, to mamy do dyspozycji event o nazwie:

```
event tor_w_tartaku:free multiple 0 none zdarzenie2 endevent
```

Jeżeli wywołamy takie zdarzenie:

```
event zdarzenie1 multiple 0 tor_w_tartaku zdarzenie2 condition memcompare * * 0 endevent
```

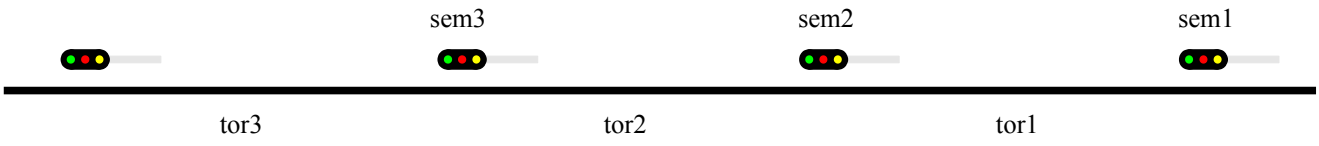
to *zdarzenie2* zostanie wywołane, jeżeli odcinek izolowany *tor_w_tartaku* będzie wolny. Jeżeli chcemy mieć zdarzenie z warunkiem, że odcinek izolowany jest zajęty, to składnia będzie następująca:

```
event zdarzenie1 multiple 0 tor_w_tartaku zdarzenie2 condition memcompare * * 1 endevent
```

Do czego stosować odcinki izolowane? Bardzo dobrze się sprawdzają przy sterowaniu przejazdami położonymi z dala od stacji, a szczególnie mocno zaleca się ich wykorzystywanie do sterowania semaforami SBL. Można również pisać całe scenariusze wyłącznie w oparciu o odcinki izolowane. Takie scenariusze to quarkmce2007, niektóre przebiegi na L61, również znajdujące się w testach (stan na 08.2017 r.) metro bałtyckie.

Przeanalizujemy proste wykorzystanie odcinków izolowanych do sterowania semaforami SBL. Zakładamy, że na danym torze pociągi mogą jeździć tylko w jednym kierunku:

Schemat trasy wyposażonej w SBL 3-stawną.



Przykładowa obsługa semaforów dla torów nr 2 i nr 3 może wyglądać następująco: w momencie wjazdu na odcinek *tor2* semafor *sem2* wskazuje S1, natomiast poprzedni semafor *sem1* zostaje ustawiony na S5, chyba, że na tym odcinku znajduje się inny pociąg. W momencie zwolnienia odcinka *tor2* semafor *sem2* zostaje ustawiony na S5, a semafor *sem1* na S2 (pod warunkiem, że odcinek *tor1* jest pusty).

Dla kolejnych odcinków zasada sterowania jest identyczna.

```
event tor2:busy multiple 5 none sem2_s1 tor2a endevent
          event tor2a multiple 0 tor1 sem1_s5 condition memcompare * * 0 endevent
event tor2:free multiple 5 none sem2_s5 tor2b endevent
          event tor2b multiple 0 tor1 sem1_s2 condition memcompare * * 0 endevent
event tor3:busy multiple 5 none sem3_s1 tor3a endevent
          event tor3a multiple 0 tor2 sem2_s5 condition memcompare * * 0 endevent
event tor3:free multiple 5 none sem3_s5 tor3b endevent
          event tor3b multiple 0 tor2 sem2_s2 condition memcompare * * 0 endevent
```

Oczywiście można to wszystko wykonać prościej na eventach przypisanych do torów. Dlaczego jednak odcinki izolowane są tak bardzo zalecane do SBL? Wyobraźmy sobie, że jedziemy szlakiem, gdzie semafory SBL są ustawione co kilometr, ich sterowanie jest oparte na eventach wpisanych w torowisko, a dodatkowo przed nami jedzie pociąg towarowy o długości 500 metrów. Jedziemy, widzimy sygnał S5, rozpoczynamy hamowanie wiedząc, że mamy kilometr na zmniejszenie prędkości do 20 km/h, i... rozbijamy się na ostatnim wagonie towarowego.

Taka sytuacja nie jeden raz zdarzyła się na szlaku, gdzie SBL były sterowane najeżdżaniem pociągu na event w torze (chociażby Drawinowo). Takiego niebezpieczeństwa nie ma, gdy zastosujemy sterowanie za pomocą odcinków izolowanych, gdyż S5 na poprzednim semaforze nie zostanie podany, dopóki cały pociąg nie zjedzie z odcinka między semaforami.

4.9. Przypisywanie semaforów do torów

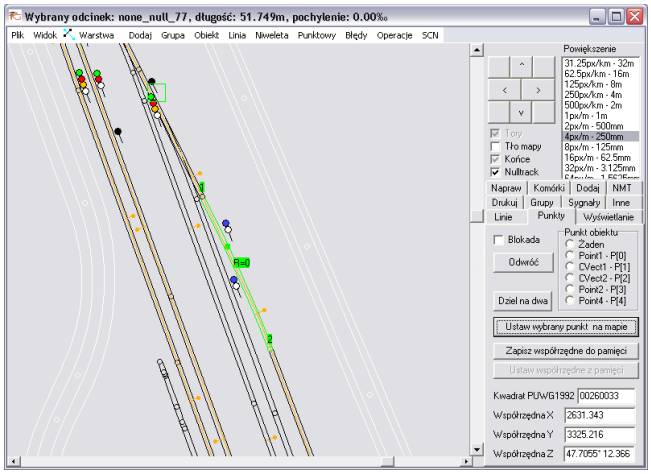
Bardzo przydatna sprawa, którą niestety nie wszyscy scenarzyści doceniają. Dzięki przypisaniu semaforów (i innych wskaźników) do torów możemy zrealizować bardzo zaawansowaną automatykę scenariusza. AI nie ruszy po podaniu S2 na semaforze, jeżeli ten semafor nie będzie przypisany do toru. Analogicznie AI nie zauważy przystanku osobowego i nie zatrzyma się przy nim, jeżeli nie będzie eventu sygnalizującego obecność W4.

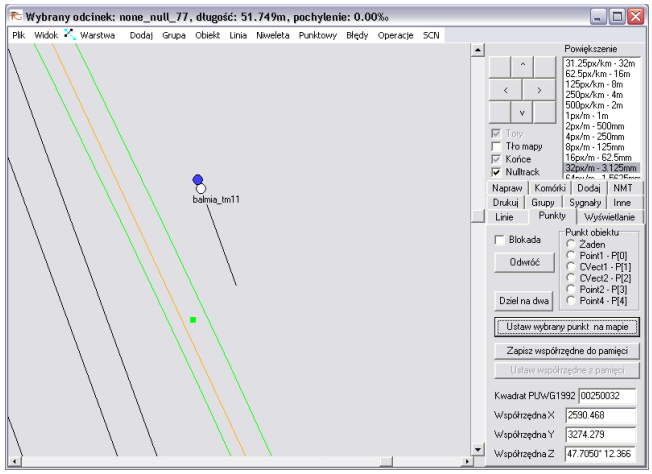
Aktualnie już prawie wszystkie scenerie posiadają przypisanie, jednak zawsze warto sprawdzić, czy przypadkiem ktoś po cichu nie pominął jakiś semaforów. Ponadto wiedza na temat przypisywania semaforów i wskaźników może być nam potrzebna, gdy zabierzemy się za scenariusze na dopiero powstających sceneriach.

W poniższym przykładzie posłużymy się stacją Bałtyk Miasto i przypiszemy wszystkie semafory oraz tarcze manewrowe do torów. Najpierw otworzymy w Rainsted układ torowy scenerii Bałtyk, w taki sposób aby były widoczne wskaźniki (sposób takiego załadowania scenerii został opisany w rozdziale 8.2.1.). Ponieważ jest kilka plików zawierających tory do scenerii Bałtyk, możemy wybrać *scenery/baltyk/baltyk_torowisko.scm*.

Zacznijmy od jednej z tarcz manewrowych znajdujących się przy zelektryfikowanej bocznicy. Najpierw sprawdzamy, jaki tor znajduje się na wysokości tarczy – jest to *none_null_77*. Sprawdzamy orientację toru – pociąg podjeżdżający do tarczy wjedzie najpierw na punkt 2 toru, toteż do toru wpiszemy *event1*. Następnie wykonujemy zbliżenie na tarczę aby odczytać jej nazwę – jest to *balmia_tm11*. W pliku zawierającym układ torowy odnajdujemy *none_null_77*, i uzupełniamy wpis.

```
node 1000 0 none_null_77 track normal 0.0 1.435 0.25 25.0 20 0 flat vis
 rail_screw_used1 4 tpd-oil2 0.2 0.5 1.1
-2574.28 -3.79986 3278.99 0.0 //point 1
-7.69409 0.0 -15.4365 //control vector 1
6.13452 0.0 16.1201 //control vector 2
-2595.43 -3.8 3231.76 0.0 //point 2
0
event1 balmia_tm11_sem_info
velocity 40.0
endtrack
```





Obok *balmia_tm11* znajduje się kolejna bocznica, zabezpieczona tarczą manewrową *balmia_tm2*, umieszczoną obok toru *none1753*. Tor wstawiony jest odwrotnie niż *none_null_77*. Wpis do toru będzie wyglądał tak:

```
node 1000 0 none1753 track normal 75.0 1.435 0.25 25.0 20 0 flat vis
 rail_screw_unused1 4 tpd-oil2 0.2 0.5 1.1
-2596.85 -3.79986 3211.73 0.0 //point 1
8.55029 0.0 23.4922 //control vector 1
-8.55054 0.0 -23.4924 //control vector 2
-2572.53 -3.79986 3278.19 0.0 //point 2
0
event2 balmia_tm2
endtrack
```

Pora na semafor wyjazdowy z Bałtyku Miasta w stronę Bałtyku Głównego. Semafor *balmia_c* stoi przy torze *balmia2*. Wpis do toru jest następujący:

```
node 1000 0 balmia2 track normal 90.0 1.435 0.25 25.0 20 0 flat vis
 rail_screw_used2 4 tpd-oil2 0.2 0.5 1.1
-2564.54 -3.79986 3247.86 0.0 //point 1
10.2607 0.0 28.1909 //control vector 1
-10.2605 0.0 -28.1907 //control vector 2
-2533.76 -3.79986 3332.43 0.0 //point 2
0
event2 balmia_c_sem_info
endtrack
```

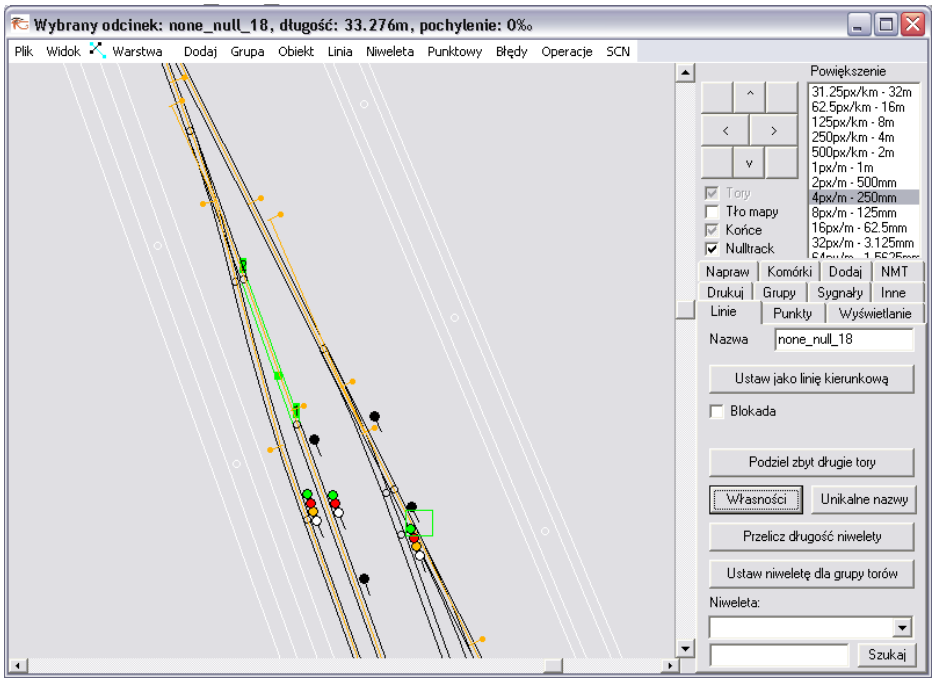
Chyba wszystko jest już jasne? W takim razie spróbuj samemu przypisać resztę semaforów i tarcz. Następnie możesz sprawdzić w poniższej tabelce, czy wszystko zostało zrobione prawidłowo:

Kierunek	Tor	Sygnalizator	Kierunek	Tor	Sygnalizator
event1	none_null_31	balmia_a	event1	balmia_3_a	balmia_e
event2	none_null_17	balmia_tm6	event1	balmia_2_a	balmia_f
event2	balmia1	balmia_d	event1	balmia_1_a	balmia_g
event2	balmia2	balmia_c	event1	none_null_25	balmia_tm3
event2	balmia3	balmia_b	event1	none1571	balmia_tm5
event2	none1753	balmia_tm2	event1	balmiaprz11	balmia_h
event1	none_null_77	balmia_tm11	event2	none1872	balmia_tm4

4.10. Zamykanie semaforów

Semafor / tarczę manewrową po przejechaniu pociągu należy koniecznie zamknąć. Nie ma sensu dla każdego scenariusza pisać osobnych eventów, zdecydowanie najlepszym rozwiązaniem jest wpisanie do torów eventów zamykających.

Uzupełnijmy zatem stację Bałtyk Miasto. Najpierw zapiszmy event zamykający semafor *balmia_c*. Następny tor za semaforem to *none_null_18*.



Wpis do toru będzie wyglądał następująco:

```
node 1000 0 none_null_18 track normal 0.0 1.435 0.25 25.0 20 0 flat vis
 rail_screw_used2 4 tpd-oil2 0.2 0.5 1.1
-2533.76 -3.79986 3332.43 0.0 //point 1
3.79443 0.0 10.4253 //control vector 1
-3.79443 0.0 -10.4253 //control vector 2
-2522.38 -3.79986 3363.7 0.0 //point 2
0
event2 balmia_c_S1
endtrack
```

Ważna informacja:

S1 zamyka semafor świetlny. Jeżeli zamykamy tarczę manewrową, to powinniśmy użyć _ms1. Semafory kształtowe zamykamy eventem _Sr1.

Ta metoda nie zawsze jest możliwa do zastosowania. Wystarczy tylko spojrzeć na tarcze manewrowe *balmia_tm2* i *balmia_tm11*: torem następujący bezpośrednio za tarczą to zwrotnica wspólna dla obu bocznic. Nie możemy do wpisu toru dać dwóch wpisów *event1*, bo jeden z nich nie będzie wykonywany. W tej sytuacji należy do wpisu zwrotnicy *balmia_zw111* zapisać następujący event:

```
node 1000 0 balmia_zw111 track switch 34.0 1.435 0.24 15.0 20 2 flat vis
 rail_screw_used1 4 rail_screw_unused1 0.2 2.75 2.5
-2559.12 -3.79986 3309.42 0.1 //point 1
0.0 0.0 0.0 //control vector 1
0.0 0.0 0.0 //control vector 2
-2574.28 -3.79986 3278.99 -0.1 //point 2
0
-2559.12 -3.79986 3309.42 0 //point 1
-5.0564 0.0 -10.1445 //control vector 1
3.87671 0.0 10.6514 //control vector 2
-2572.53 -3.79986 3278.19 0 //point 2
-300.0
event1 wylacz_balmia_tm2_tm11
endtrack
```

Następnie w pliku .ctr definiujemy stworzony event:

```
event wylacz_balmia_tm2_tm11 multiple 1 none balmia_tm2_ms1 balmia_tm11_ms1 endevent
```

5. Ciekawostki dla bardziej zaawansowanych

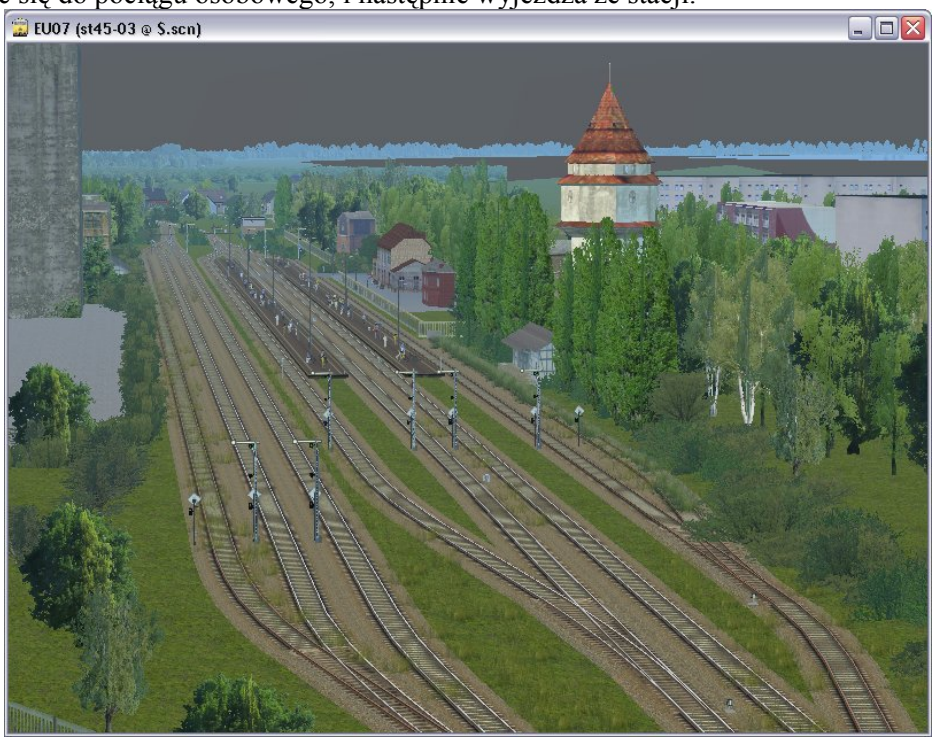
Umiemy już pisać bardzo ciekawe i nawet skomplikowane scenariusze, więc zaczynamy korzystać z narzędzi umożliwiających wyszlifowanie scenariusza „na wysoki połysk". W tym rozdziale dowiemy się, jak wydawać komendy manewrowe dla autopilota (AI), jak robić „choinki" na semaforach, a także w razie potrzeby wstawiać nowe obiekty.

5.1. Przekazywanie danych do pociągów

Scenariusz, w którym tylko jedziemy i nic się po za tym nie dzieje może być ciekawy tylko wtedy, jeżeli jest to scenariusz typu wycieczka zabytkowym pojazdem po zapomnianej linii. W pozostałych przypadkach lepiej się postarać wprowadzić jakiś ruch dodatkowych pociągów obsługujących przez autopilota (AI). Często zachodzi konieczność wykonania jakiś bardziej skomplikowanych manewrów, których nie da się przekazać wyłącznie za pomocą semaforów. Tutaj z pomocą przychodzą nam komendy do sterowania AI.

5.1.1. Oddziaływanie komórkami na pociągi

Od razu zacznijmy od przykładu: w naszym scenariuszu *calkowo_cargo* dojeżdżamy do Wilisia. Możemy dostać przelot, ale może zostaniemy przytrzymani na wjeździe przez manewrującą lokomotywę, która wyjeżdża z bocznicy (spod *wil_tm17*), zawraca za jedną ze zwrotnic (*wilis_zwr15*), po czym jedzie w peron (za semafor *wil_k*) aby podpiąć się do pociągu osobowego, i następnie wyjeżdża ze stacji.



Najpierw trzeba ustawić lokomotywę na torze, powiedzmy, że będzie to tor przed tarczą manewrową Tm17. Wstawiamy do pliku .scn kolejną lokomotywę:

```
trainset none wilis7 10 0
//$o -
node -1 0 SU42-510-SU dynamic PKP\SU42_V1 SU42-510-PR 6D-SU  0 headdriver 3 0 enddynamic
endtrainset
```

Lokomotywę ustawiliśmy na torze *wilis7* (znajdującym się przed tarczą manewrową Tm17). Po uruchomieniu symulatora lokomotywa będzie wygaszona – odpowiada za to ostatni parametr we wpisie *trainset* – gdyby wynosił 0.1, to lokomotywa zostałaby automatycznie uruchomiona wraz ze startem misji. W opisie składu znajduje się myślnik – oznacza to, że pociąg nie zostanie pokazany w okienku startowym Rainsted. Wstawiliśmy lokomotywę SU42.

No to teraz mamy pierwszy problem – jak uruchomić tę lokomotywę? Można było oczywiście podać w definicji *trainset* prędkość początkową niezerową, ale to może być z punktu widzenia scenariusza niewskazane. Na początek zdefiniujemy sobie komórkę pamięci:

```
node -1 0 wilis_kom memcell 1.0 1.0 1.0 Wait_for_orders 0 0 wilis7 endmemcell
```

Wygląda to jak zwyczajna komórka pamięci ale spójrzmy na sam koniec: tam gdzie dotychczas pisaliśmy *none*, wpisaliśmy nazwę toru. Oznacza to, że komórka wraz ze zmianą wartości, będzie oddziaływała na tor do którego jest przypisana, a w konsekwencji na pojazd, który na tym torze stoi.

Pora na event uruchamiający lokomotywę:

```
event uruchom_su42 updatevalues 0 wilis_kom Prepare_engine 1 0 endevent
```

W momencie wywołania tego eventu lokomotywa zostanie uruchomiona – komórka przekaże do lokomotywy polecenie uruchomienia.

Do listy wszystkich takich poleceń jeszcze wrócimy, najpierw przeanalizujemy inne możliwości podawania instrukcji do pociągów.

5.1.2. Eventy *getvalues* i *putvalues*

Udało się uruchomić lokomotywę stojącą pod tarczą Tm17. Wraz z eventem uruchamiającym możemy podać komendę *wil_tm17_ms2* albo *wil_tm17_m40*. Zwrotnice przestawiamy tak, aby lokomotywa dojechała za zwrotnicę wilis_zwr15. Po przejechaniu tej zwrotnicy powinna zawrócić.

Tor, który znajduje się bezpośrednio za zwrotnicą to *none4128*. Do niego przypiszemy event zawracający lokomotywę. Stwórzmy teraz komórkę:

```
node -1 0 wilis_kom2 memcell 1.0 1.0 1.0 Change_direction  0 0 none endmemcell
```

Komórka zawiera komendę zawracania. Trzeba ją teraz przesłać do pociągu ale nie możemy się posłużyć sposobem opisanym w poprzednim podrozdziale, gdyż tak można oddziaływać na pociągi stojące – a tutaj musimy lokomotywie podać komendę podczas jazdy.

Zróbmy zatem następujące zdarzenie:

```
event none4128:event2 getvalues 5 wilis_kom2 endevent
```

W momencie jak lokomotywa wjedzie na tor uruchomi się event, który po 5 sekundach przekaże do lokomotywy zawartość komórki *wilis_kom2*. Lokomotywa zawróci.

Istnieje również metoda alternatywna, ani lepsza ani gorsza, wybór należy do scenarzysty. Osobiście autor woli polecenie *putvalues*.

```
event none4128:event2 putvalues 5 none 1.0 1.0 1.0 Change_direction 0 0 endevent
```

Zysk jest taki, że nie musimy mieć odrębnej komórki. Strata jest taka, że polecenie wydawane do pociągu jest wpisane na sztywno i nie można go zmienić.

Skoro już zawróciliśmy na zwrotnicy wilis_zwr15, to lokomotywa jak już wjedzie pod semafor K, powinna się przypiąć do ustawionych na torze wagonów. Event przekazujący komendę doczepienia wpiszemy do toru *wilis2*:

```
event wilis2:event1 putvalues 1 none 1.0 1.0 1.0 Shunt -3 -3 endevent
```

Gdy takie polecenie zostanie przekazane do lokomotywy, wówczas lokomotywa będzie dalej jechać przed siebie, gdy dojedzie do wagonów sama się przypnie, zmieni kabinę, i zapali światła do jazdy pociągowej.

Komendy dla pojazdów to właściwie cała tajemnica sterowania AI. Wszystkie eventy dla semaforów kończące się *_sem_info*, to tak naprawdę eventy typu *getvalues*, a przekazywane komunikaty to najczęściej *SetVelocity -1 0*, *SetVelocity 40 0*, *ShuntVelocity 25 0*, itp.

5.1.3. Komendy do sterowania AI

Możemy już teraz wyczyniać z AI co się nam podoba, do dyspozycji mamy następujące komendy:

Komenda	Położenie XZY	Co robi komenda
Wait_for_orders	Nieistotne	Komenda, która nic nie robi. Czasami jej użycie jest potrzebne – tak jak mieliśmy to w podrozdziale 5.1.1. Przykłady zastosowania: **`Wait_for_orders 0 0`**
Prepare_engine	Nieistotne	Uruchamianie lub wyłączanie silnika. Przykłady: **`Prepare_engine 1 0`** – uruchamianie lokomotywy, jednakże nie zostaną zapalone żadne światła zewnętrzne, **`Prepare_engine 0 0`** – wyłączanie lokomotywy.
Shunt	Nieistotne	Komenda podłączania/rozłączania i manewrowania. Pierwsza liczba oznacza: 0, 1, 2, 3, ... - ilość wagonów, która ma zostać podpięta do lokomotywy; -1 – podpięcie do wagonów a potem jazda w kierunku zależnym od znaku drugiego parametru (dla wartości dodatnich skład będzie spychany), -2 – podpięcie do wagonów i zatrzymanie w oczekiwaniu na sygnał do dalszych manewrów (np.: na tarczy manewrowej), -3 – podpięcie do wagonów i włączenie trybu jazdy pociągowej. Druga liczba oznacza maskę sprzęgu, z którą lokomotywa się podepnie do wagonów. Przykłady zastosowania: **`Shunt 0 0`** – odpięcie lokomotywy i uruchomienie trybu manewrów, **`Shunt -2 -3`** – podpięcie lokomotywy do wagonów i pozostanie w trybie manewrowym **`Shunt -3 -3`** – podpięcie lokomotywy i przejście w tryb jazdy pociągowej (czyli zapali się trójkąt świetlny) **`Shunt 2 -3`** – lokomotywa podjedzie do wagonów, podczepi się i ponownie odczepi razem z 2 wagonami **`Shunt 3 -3`** – lokomotywa podjedzie do wagonów, podczepi się i ponownie odczepi razem z 3 wagonami **`Shunt -1 3`** – lokomotywa podjedzie do wagonów, podczepi się, po czym zacznie je spychać.
Change_direction	Można podać ale nieobowiązkowo	Przykłady zastosowania: **`Change_direction 0 0`** – pociąg (niezależnie czy w trybie manewrowym czy pociągowym) zmieni kierunek jazdy na przeciwny względem aktualnego, **`Change direction 1 0`** – pociąg zmieni kierunek jazdy w kierunku punktu wyznaczonego przez komórkę pamięci lub współrzędne w putvalues, **`Change direction -1 0`** – pociąg zmieni kierunek jazdy na przeciwny od punktu wyznaczonego przez komórkę pamięci lub współrzędne w putvalues.
Timetable:*	Nieistotne	Przypisywanie rozkładu jazdy. W miejsce * musimy podać ścieżkę dostępu do rozkładu jazdy. Pierwszy parametr liczbowy określa: • 0 – jeżeli pociąg ma do pierwszej stacji dojechać w trybie manewrowym, • 0.1 – jeżeli pociąg stoi w miejscu na pierwszej stacji z rozkładu, • -0.1 – jeżeli pociąg stoi w miejscu na pierwszej stacji z rozkładu, i ma jednocześnie zmienić kierunek na przeciwny, • Jeżeli pociąg ma od razu jechać z określoną prędkością, to ją podajemy np.: 40. Dla -40 będzie zmiana kierunku. Drugi parametr liczbowy określa ilość minut, którą symulator doda do rozkładu w podanym pliku – wykorzystywane w metrze bałtyckim, jest jeden rozkład, a każdy następny pociąg ma dodawane 5 minut względem poprzedniego. Przykłady zastosowania: **`Timetable:calkowo/ros419 0.1 0`** – w misji calkowo_noc, na stacji Macierzewo
Warning_signal	Należy podać współrzędne W6a	Uruchomienie Rp1. Pierwszy parametr oznacza długość trwania sygnału, drugi ilość powtórzeń. Komenda stosowana razem ze wskaźnikami W6a.
CabSignal	Należy podać współrzędne SHP	**`CabSignal -1 -1`** – efekt identyczny jak przejazd przez SHP Komenda stosowana razem z rezonatorami.
Emergency_brake	Nieistotne	**`Emergency_brake 1 1`** – uruchomienie radiostopu
Load=*	Położenie wagonu	Załadunek towaru *. Komenda praktycznie niestosowana, skomplikowana i nieefektowna obsługa.
UnLoad=*	Położenie wagonu	Rozładunek towaru *. Komenda po raz pierwszy zastosowana w scenariuszu Calkowo_lotos. Na jaw wyszły ewidentne błędy w kodzie symulatora związanym z jej obsługą. Aktualnie niezalecana.

Komenda	Położenie XZY	Co robi komenda
SetDamage	Nieistotne	Psucie lub naprawianie pociągu. Pierwszy parametr powinien być sumą liczb, które oznaczają konkretne usterki: • 1 – podkucie jednego z kół • 2 – zużycie jednego z kół • 4 – uszkodzenie łożyska • 8 – uszkodzenie sprzęgu • 16 – uszkodzenie wentylatorów (tylko lokomotywy elektryczne) • 32 – uszkodzenie silnika (tylko lokomotywy elektryczne) • 64 – uszkodzenie osi • 128 – wykolejenie Drugi parametr: 1 oznacza zepsucie, -1 naprawienie. Przykłady zastosowania: **`SetDamage 128 1`** – wykolejenie, stosowane w wykolejnicach **`SetDamage 33 1`** – podkucie koła i uszkodzenie silnika **`SetDamage 255 1`** – zepsucie wszystkiego i jeszcze wykolejenie...
SetVelocity	Położenie semafora	**Tzw. Komenda skanowana, tylko do użytku z getvalues!** Używane w semaforach. Jeżeli utworzymy komórkę o jakiś współrzędnych a następnie przypiszemy ją do toru poprzez event getvalues, to poprzez podawanie komendy SetVelocity będziemy mieli równoważnik semafora. Idealne do tworzenia ukrytych semaforów. Przykłady zastosowania: **`SetVelocity -1 -1`** – prędkość maksymalna a potem prędkość maksymalna **`SetVelocity 40 0`** – prędkość 40 km/h a potem zatrzymanie
ShuntVelocity	Położenie semafora lub tarczy ostrzegawczej	**Tzw. Komenda skanowana, tylko do użytku z getvalues!** Używane w semaforach i tarczach manewrowych. Działa identycznie jak SetVelocity, ale służy wyłącznie dla lokomotyw w trybie manewrowym.

Nie są to wszystkie dostępne komendy, podane zostały te najbardziej przydatne. Więcej informacji na temat komend można szukać na forum symulatora.

Na koniec niezwykle istotna uwaga: zarówno event *getvalues* jak i *putvalues* prawidłowo zadziała tylko wtedy, gdy zostanie uruchomiony przez pociąg, na który ma działać. Czyli np.: jeżeli umieścimy *putvalues* wewnątrz eventu o nazwie *keyctrl01*, to nie zauważymy efektu. Event prawidłowo zadziała, jeżeli go umieścimy w torze, uruchomimy poprzez inny event umieszczony w torze, lub wywołamy poprzez zajęcie lub zwolnienie odcinka izolowanego.

5.2. Grzebanie w semaforach

5.2.1. Eventy ukryte dla semaforów

Semafory mają wiele ciekawych wewnętrznych eventów. Nie sposób do wszystkich napisać instrukcję obsługi, ale można spróbować obejrzeć ich zawartość. Spróbujemy zatem odszukać, jakie tajniki w sobie kryje semafor wyjazdowy z Macierzewa, czyli *mac_d*. Najpierw odszukajmy miejsca, w którym znajduje się jego definicja w scenerii. Możemy przeszukać wszystkie zaincludowane do scenariusza pliki – dla ułatwienia powiem, że szukany semafor znajduje się w pliku *calkowo_wskazniki.scm*. Tam znajdziemy taką linię:

```
include ss4zcpbi.inc mac_d -8219.74 6.0 -15413.8 180.0 d-2m end
```

Szukamy w katalogu *scenery* pliku *ss4zcpbi.inc* – otwieramy go w notatniku. W środku oprócz definicji modelu semafora odnajdziemy również eventy sterujące samym semaforem. Znajdziemy także definicję eventów, które stosowaliśmy od samego początku naszej pracy przy scenariuszu:

```
event (p1)_s1 multiple 0 none (p1)_sem_ligh1 (p1)_sem_info_stop endevent
event (p1)_s2 multiple 0 none (p1)_sem_ligh2 (p1)_sem_info_vmax endevent
event (p1)_s3 multiple 0 none (p1)_sem_ligh3 (p1)_sem_info_vmax endevent
event (p1)_s4 multiple 0 none (p1)_sem_ligh4 (p1)_sem_info_vmax endevent
event (p1)_s5 multiple 0 none (p1)_sem_ligh5 (p1)_sem_info_vmax endevent
event (p1)_s10 multiple 0 none (p1)_sem_ligh10 (p1)_sem_info_v40 endevent
event (p1)_ms2 multiple 0 none (p1)_sem_lighs2 (p1)_sem_info_Shunt25 endevent
event (p1)_m40 multiple 0 none (p1)_sem_lighs2 (p1)_sem_info_Shunt40 endevent
event (p1)_sz1 multiple 0 none (p1)_sem_lighz1 (p1)_sem_info_v40 (p1)_wyg_sz endevent
```

```
event (p1)_wyg_sz multiple 90 (p1)_sem_mem (p1)_s1 condition memcompare SetVelocity 40 0
          endevent
```

Składnia (p1) oznacza, że w to miejsce wklejany jest tekst, będący pierwszym parametrem w danym *include* tego pliku. Spójrzmy jeszcze raz na zapis w pliku *calkowo_wskazniki.scm*. Pierwszym parametrem, zaraz po include i nazwie pliku, jest *mac_d*. Zatem nie tylko został utworzony semafor na scenerii, ale dodatkowo eventy: *mac_d_s1*, *mac_d_s2*, itd. Poniżej mamy definicję komórki, która zawiera informację dla AI:

```
//memcell do pamietania predkosci:
node -1 0 (p1)_sem_mem memcell (p2) (p3) (p4) SetVelocity 0.0 0.0 none endmemcell
```

jak również eventy wpływające na wartości tej komórki:

```
event (p1)_sem_info_stop updatevalues 0.0 (p1)_sem_mem SetVelocity 0.0 0.0 endevent
event (p1)_sem_info_vmax updatevalues 1.0 (p1)_sem_mem SetVelocity -1 * endevent
event (p1)_sem_info_v40 updatevalues 1.0 (p1)_sem_mem SetVelocity 40 * endevent
event (p1)_sem_info_v20 updatevalues 1.0 (p1)_sem_mem SetVelocity 20 0 endevent
```

Co z tego wszystkie wynika? Np.: jeżeli zamiast eventu *mac_d_S10* wywołamy *mac_d_sem_info_v40*, to wówczas AI zachowa się tak, jakby na semaforze został wyświetlony sygnał S10 – a w rzeczywistości semafor będzie czerwony. Taka sztuczka umożliwia nam wyjazd AI w różnych dziwnych sytuacjach, np.: była sytuacja losowa, że nie można podać zastępczego, został podyktowany rozkaz pisemny. Po zakończeniu nadawania tego rozkazu podajemy event *mac_d_sem_info_v40*. Jeżeli pociąg jest sterowany przez AI to da radę sam wyjechać ze stacji. Natomiast jeżeli pociągiem steruje użytkownik, to nie będzie blokady rozkładu jazdy – po przejechaniu S1 rozkład jazdy się zatrzymuje i nie jest dalej przewijany.

Wywołanie takiego eventu może być przydatne nie tylko gdy chcemy, aby dyktowany był rozkaz pisemny, ale również w sytuacji „choinki” na semaforze.

5.2.2. Event lights

Ten specjalny event pozwala nam na ręczne sterowanie światłami w semaforze. Składnia jest następująca:

```
event choinka lights 0 semafor 0 1 1 1 2 endevent
```

- *choinka* – nazwa eventu,
- *lights* – typ eventu,
- *0* – opóźnienie wykonania,
- *semafor* – nazwa semafora, który zamierzamy zepsuć. Najlepiej aby był to semafor świetlny,
- *0 1 1 1 2* – ręczne ustawienie światła dla każdej komory semafora. Ilość cyfr musi odpowiadać ilości komór semafora, natomiast użycie poszczególnych cyfr oznacza:
  - 0 – komora zgaszona,
  - 1 – komora zapalona,
  - 2 – komora migająca,
  - 3 – komora w dzień zgaszona, w nocy zapalona – automatyczne światło.
- *endevent* – zakończenie eventu.

Możemy dla przykładu w naszej misji *calkowo_cargo* dodać event o nazwie *mac_d_choinka*, który na semaforze wyjazdowym zapali nam wszystkie możliwe światła, a dolne białe będzie mrugające (czyli dostaniemy nietypowy zastępczy):

```
event mac_d_choinka lights 0 mac_d 1 1 1 2 endevent
```

W naszym scenariuszu *calkowo_cargo* wyjazd na choinkę może się odbyć przy użyciu następującej kombinacji eventów:

```
event macierzewo_ustaw_sem multiple 11 none mac_d_sem_info_v40 mac_d_choinka macierzewo_radio1
          else mac_d_S10 condition propability 0.2 endevent
```

5.3. Rozpoznawanie pociągów za pomocą eventu

Jeżeli na event znajdujący się w torze będzie najeżdżało wiele pociągów, to będziemy musieli jakoś je rozpoznawać. Z pomocą przychodzi nam event typu *whois*.

```
event zdarzenie1 whois 0 komorka1 7 endevent
```

- *zdarzenie1* – nazwa. Zdarzenie to musi być uruchomione przez pociąg, którego dane chcemy odczytać. Z tego powodu event musi być umieszczony w torze, albo odczyt musi znajdować się wewnątrz eventu wyzwalanego zajęciem lub zwolnieniem odcinka izolowanego.
- *whois* – słowo kluczowe,
- *0* – opóźnienie wykonania eventu,
- *komorka1* – komórka, do której event wpisze dane pobrane od pociągu,
- *7* – określenie, jakie parametry chcemy uzyskać.

Jeżeli sprawdzimy zawartość komórki po wykonaniu eventu, to w miejscu wartości tekstowej będziemy mieli numer rozkładu, np.: ROS419, TMS654079. Pierwsza wartość liczbowa będzie oznaczała ilość stacji do końca biegu, a druga wartość liczbowa postój (1) lub przelot (0) na najbliższej stacji. Za pomocą różnych wartości ostatniego parametru możemy pozyskać jeszcze inne dane:

Parametr	Wartość tekstowa	Pierwsza wartość liczbowa	Druga wartość liczbowa
7	Nazwa rozkładu	Ilość stacji do końca	1 = postój, 0 = przelot
15	Miejsce docelowe	Kierunek w składzie 1 lub -1	Moc silników
23	Nazwa ładunku	Ilość ładunku	Maksymalna ilość ładunku
31	Typ pojazdu	0	0

UWAGA!
Autor nie sprawdzał poprawności działania wszystkich parametrów. Powyższa tabela została spisana na podstawie materiałów dostępnych na rainsted.com.pl, gdzie była na dodatek adnotacja, że będą wprowadzane jeszcze zmiany.

W swoich scenariuszach autor używa parametru 1 – w efekcie do komórki zostaje zapisana tylko wartość tekstowa, czyli nazwa rozkładu.

5.4. Inne ciekawe eventy

5.4.1. Ręczna zmiana prędkości toru

Podczas wpisywania eventów do torów zapewne zauważyliśmy, że niektóre tory mają zdefiniowaną prędkość szlakową. Istnieje event, którym zmienimy prędkość szlakową wpisaną na stałe w torze:

```
event zdarzenie1 trackvel 0 tor 70 endevent
```

- *zdarzenie1* – nazwa zdarzenia,
- *trackvel* – słowo kluczowe, zmiana prędkości szlakowej,
- *0* – opóźnienie wykonania eventu,
- *tor* – nazwa toru, w którym będzie zmieniana prędkość szlakowa,
- *70* – nowa prędkość szlakowa w km/h.

Zdarzenie to możemy wykorzystać, gdy chcemy w scenariuszu wstawić czasowe ograniczenie prędkości, i chcemy aby AI reagowało na to ograniczenie.

5.4.2. Zmiana napięcia podstacji zasilającej

W sceneriach z siecią trakcyjną możemy dowolnie manipulować napięciem wyjściowym podstacji zasilającej. Do tego służy następujący event:

```
event zdarzenie1 voltage 0 podstacja 3000 endevent
```

- *zdarzenie1* – nazwa zdarzenia,
- *voltage* – słowo kluczowe, ustalenie wartości napięcia,
- *0* – opóźnienie wykonania eventu,
- *podstacja* – nazwa podstacji zasilającej, musi być zdefiniowana w scenerii,
- *3000* – wartość napięcia zasilającego w woltach.
  Możliwość stosowania tego eventu warunkuje poprawność przygotowania sieci trakcyjnej w scenerii.

5.4.3. Zmiana współczynnika tarcia

Jeżeli piszemy scenariusz w którym są opady deszczu, to warto wprowadzić utrudnienie dla mechanika w postaci zwiększonego prawdopodobieństwa wpadnięcia w poślizg. Zmiana współczynnika tarcia dotyczy całej scenerii – nie ma możliwości wpływania na konkretne tory:

```
event zdarzenie1 friction 0 none 0.5 endevent
```

- *zdarzenie1* – nazwa zdarzenia,
- *friction* – słowo kluczowe,
- *0* – opóźnienie wykonania eventu,
- *none* – po prostu wpisujemy none,
- *0.5* – wartość współczynnika tarcia, podajemy liczbę z przedziału <0;1>. Dla wartości 0 mamy poślizg idealny, dla 1 nie ma poślizgu.

Event został zaimplementowany w symulatorze w specyficznym czasie, kiedy wydawana była nowa paczka całościowa, oraz trwały burzliwe dyskusje między developerami. Z tego powodu event jest mało znany, właściwie niestosowany, i brakuje wiarygodnych danych na jego temat. Powyższe dane zostały ustalone przez autora metodą prób i błędów.

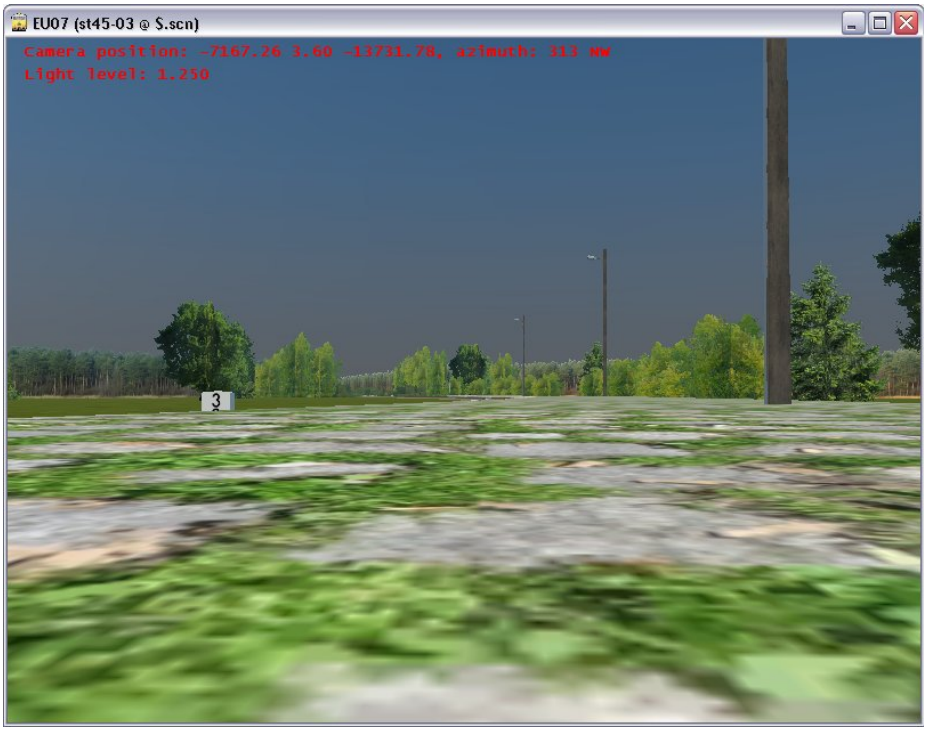
5.5. Wstawianie nowych obiektów do scenerii

To zagadnienie już wprawdzie zahacza o tworzenie nowych scenerii ale warto znać chociaż podstawy. Może zajść potrzeba umieszczenia w jakimś scenariuszu np.: dodatkowych pasażerów. Albo gdzieś ustawimy tymczasowe ograniczenie – trzeba będzie na scenerię wstawić wskaźniki W8.

Zacznijmy od najprostszej rzeczy – wstawienie dodatkowego pasażera. Najpierw musimy znać współrzędne, oraz orientację obiektu. W tym celu uruchamiamy symulację, wychodzimy z lokomotywy, idziemy do miejsca w którym chcemy wstawić pasażera, i naciskamy F3 aby odczytać współrzędne.

Na screenie widzimy peron przystanku Macierzewo Jezioro. Musimy się zniżyć aż do poziomu podłogi, aby wstawić człowieczka na właściwej wysokości. Z informacji wyświetlanych przez symulator wynika, że peron jest na wysokości 3.5 (na screenie jest pokazane 3.6 – w rzeczywistości screen został wykonany nieco powyżej płaszczyzny peronu, tak aby uwidocznić jego powierzchnię).

Mając współrzędne możemy odszukać ludzika (lub inny obiekt), którego chcemy wstawić. W tym celu musimy porozglądać się w katalogu *scenery* i poprzeglądać, co zawierają rozmaite pliki .inc. Ludziki odnajdziemy w podkatalogu *posers*.



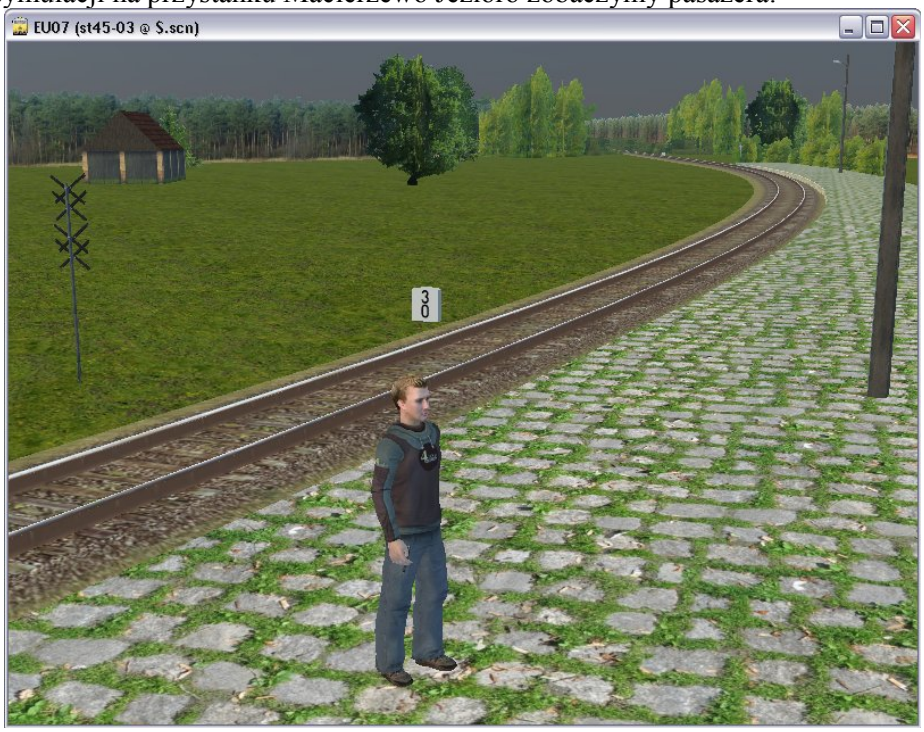
Zdecydowaliśmy się na ludzika *sport4_st.inc*. Otwieramy plik .inc i sprawdzamy jego zawartość:

```
origin (p2) (p3) (p4)
rotate 0 (p5) 0
node 500 0 (p1) model 0 0 0 0 posers/sport4_st.t3d none endmodel
rotate 0 0 0
endorigin
```

Z tego zapisu wynika, że musimy podać 5 parametrów: pierwszym będzie nazwa, kolejne trzy to współrzędne X Z Y, a ostatni parametr to obrót. Dopisujemy w pliku *events_cargo.ctr* następującą definicję:

```
include posers/sport4_st.inc ludek01 -7167.26 3.5 -13731.78 113 end
```

Po uruchomieniu symulacji na przystanku Macierzewo Jezioro zobaczymy pasażera:



Niestety kąt widoku prezentowany w symulatorze po naciśnięciu klawisza F3 nie odpowiada faktycznemu kątowi, który się podaje przy komendzie *include*. W takim przypadku można jedynie pokombinować z różnymi kątami wstawienia, i metodą prób i błędów dojść do właściwej wartości kąta.

Ważna informacja:

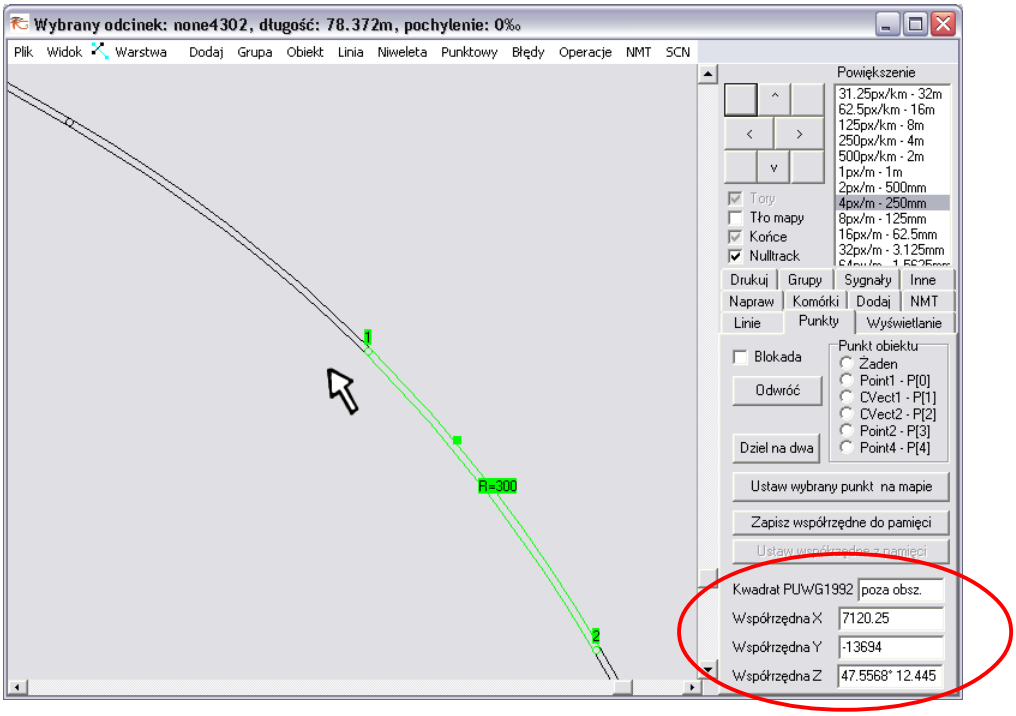
Jeżeli wstawiamy tylko jeden lub kilka obiektów, to można to zrobić z poziomu pliku z eventami. Jeżeli jednak chcemy umieścić na scenerii kilkadziesiąt nowych obiektów, to należy je wyodrębnić do jakiegoś nowego pliku z rozszerzeniem .scm, i zaincludować do pisanego scenariusza.

A teraz puśćmy wodze fantazji. Czy może by tak zasadzić rząd drzewek przy przystanku Macierzewo Jezioro? A czemu nie? Do spisywania współrzędnych najlepiej wykorzystać edytor Rainsted. Otwórzmy edytor, ustawmy kursor obok toru, i zobaczmy w okienku edytora jakie współrzędne zostały wyświetlone. Zapisujemy je, pamiętając, że musimy zmienić znak współrzędnej X na przeciwny. Edytor nie pokaże nam wprawdzie współrzędnej Z, jednakże możemy ją sobie sprawdzić po uruchomieniu symulacji, w sposób podany wyżej. Tak więc współrzędne na drzewka są następujące:

```
-7162.25 3.0 -13752.25
-7154.25 3.0 -13739.5
-7148 3.0 -13728.25
-7144 3.0 -13720
-7131.5 3.0 -13707.5
-7123.75 3.0 -13697.75
-7113.75 3.0 -13688
-7103.5 3.0 -13679.75
-7091.25 3.0 -13670.25
-7081.5 3.0 -13662
-7070.75 3.0 -13654.5
```

To dla urozmaicenia wstawmy jeszcze jakiś domek za drzewami. Współrzędne dla niego to:

```
-7116.25 3.0 -13721.25
```



Teraz w katalogu *scenery* szukamy drzewek i domków. W katalogu głównym znajduje się plik *tree.inc* – możemy dzięki niemu wstawiać drzewka. Obejrzymy jego zawartość – w nagłówku autor pliku powinien nam zostawić

wskazówki odnośnie użycia pliku:

```
//---------------drzewo---------------
//parametry: tekstura, x, y, z, kąt, wysokość, rozpiętość
```

czyli musimy podać 7 parametrów. O ile kąt, wysokość i rozpiętość jakoś sobie wymyślimy, to bez tekstury ani rusz. Otwieramy katalog symulatora i przechodzimy do podkatalogu *textures*. Musimy znaleźć jakieś ładne drzewka – proponuję od razu wejść do podkatalogu *l61_plants*. Proponuję wybrać teksturę *drzewo1024l.dds*. Ponieważ nie wszyscy mogą mieć zainstalowaną przeglądarkę plików .dds:

Ładna choineczka? To teraz czas na domek. Wracamy do katalogu *scenery* i wchodzimy do katalogu *mieszkalne*. Proponuję wybrać plik *dom_rozi_02.inc*.

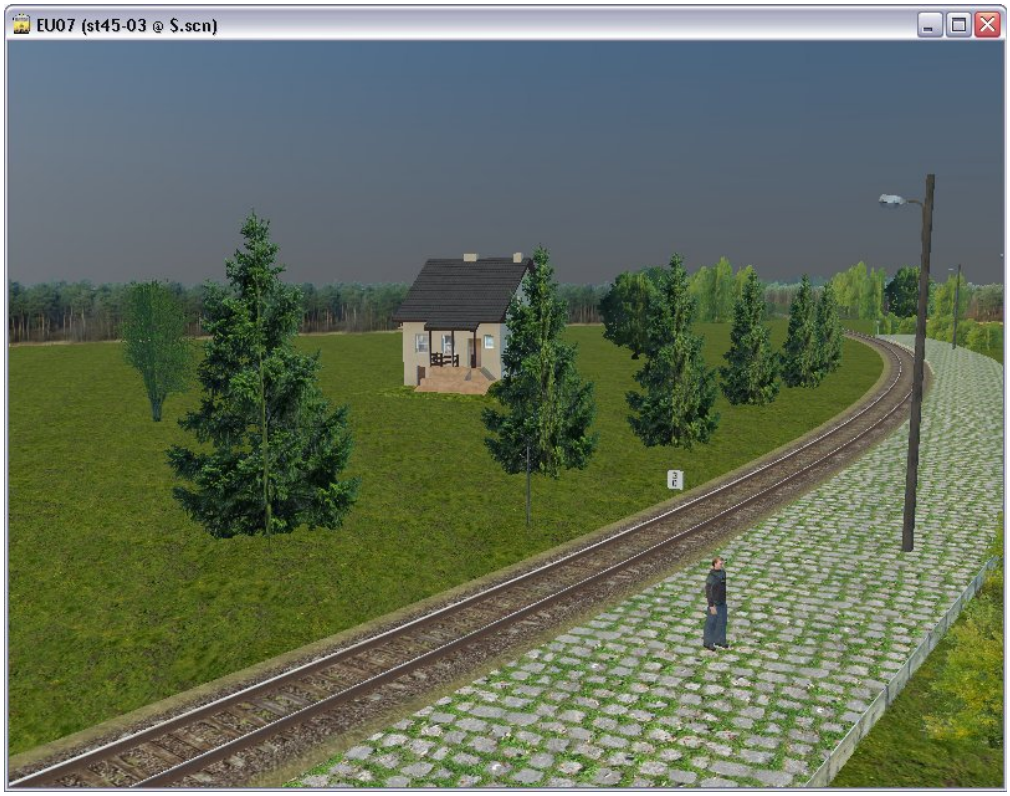
```
//script;size: 12.3 15.8;
origin (p2) (p3) (p4)
rotate 0 (p5) 0
node -1 0 none model 0 0 0 0 mieszkalne/dom_rozi_02.t3d none endmodel
rotate 0 0 0
endorigin
//autor modelu i tekstury: Rozi
//autor zdjęć na tekstury: GoogleSketchup
```

Tutaj sprawa jest znacznie prostsza, po prostu podajemy współrzędne i kąt. No to teraz czas zapisać definicje tych wszystkich drzewek i domku:

```
include tree.inc l61_plants/drzewo1024l -7162.25 3 -13752.25 0 10 3.5 end
include tree.inc l61_plants/drzewo1024l -7154.25 3 -13739.5 0 10 3.5 end
include tree.inc l61_plants/drzewo1024l -7148 3 -13728.25 0 9.5 3.2 end
include tree.inc l61_plants/drzewo1024l -7144 3 -13720 0 9.5 3.2 end
include tree.inc l61_plants/drzewo1024l -7131.5 3 -13707.5 0 9 3 end
include tree.inc l61_plants/drzewo1024l -7123.75 3 -13697.75 0 9 2.8 end
include tree.inc l61_plants/drzewo1024l -7113.75 3 -13688 10 8.5 2.6 end
include tree.inc l61_plants/drzewo1024l -7103.5 3 -13679.75 15 8.5 2.5 end
include tree.inc l61_plants/drzewo1024l -7091.25 3 -13670.25 20 8 2.5 end
include tree.inc l61_plants/drzewo1024l -7081.5 3 -13662 25 8 2.5 end
include tree.inc l61_plants/drzewo1024l -7070.75 3 -13654.5 30 7.5 2.5 end
include mieszkalne/dom_rozi_02.inc none -7116.25 3 -13721.25 45 end
```

Kąty dla drzewek to parametry zaraz po współrzędnej Y, wysokości zostały podane między 10 a 7,5 (wysokość podaje się w jednostce MaSzynowej, czyli w metrach), a rozpiętość drzewek między 2,5 a 3,5. Zwróćmy uwagę jeszcze raz na zapisy w *dom_rozi_02.inc*. Użyte są tam parametry p2, p3, p4, i p5 (czyli współrzędne i kąt). Nie ma nigdzie wspomnianego parametru p1 – tak czy inaczej, musimy go podać, w tym przypadku po prostu wpisujemy *none*.

Efekt końcowy wygląda następująco:



Umiemy wstawiać domki, drzewka, w zasadzie identyczny sposób możemy zasiać trawę, wstawić niemal dowolny obiekt, który posiada plik *.inc*. Można również wstawiać tzw. trójkąty, czyli teksturowane płaszczyzny. Ponownie rozszerzymy nasz przykład, tym razem dodamy płotek odgradzający domek od drzew i toru. Dla większej przejrzystości przyjmijmy, że płotek będzie jednolicie prosty. Za pomocą Rainsted lub w symulatorze odczytujemy współrzędne końców płotu: *(-7059.5; -13648)* i *(-7163.75; -13757)*. Grunt w tym miejscu jest na poziomie *3.0*, przyjmijmy również, że płot będzie miał wysokość 1,5 metra.

Poszczególne wierzchołki trójkąta wstawia się w następujący sposób:

```
node 1000 0 none triangles TEKSTURA
X1 Z1 Y1   AX AZ AY     TX1 TY1 end
X2 Z2 Y2   AX AZ AY     TX2 TY2 end
X3 Z3 Y3   AX AZ AY     TX3 TY3 endtri
```

- *node* – deklaracja obiektu,
- *1000 0* – tutaj warto zwrócić uwagę na wpisywane liczby, albowiem pierwsza wartość jest równa największej odległości, z której obiekt będzie widzialny. Jeżeli damy wszędzie -1, możemy niechcący pogorszyć wydajność (choć symulator chyba jest zabezpieczony na okoliczność takich sytuacji),
- *none* – nazwa, możemy podać jakąś nazwę, ale nie ma to większego znaczenia,
- *triangles* – słowo kluczowe informujące, że będziemy podawać wierzchołki trójkąta,
- *TEKSTURA* – podajemy ścieżkę dostępu i nazwę pliku zawierającego teksturę. Domyślnym katalogiem z teksturami jest *textures*.
- *X1 Z1 Y1, X2 Z2 Y2, X3 Z3 Y3* – wierzchołki trójkątów, współrzędne MaSzynowe,
- AX AZ AY – wektor normalny do powierzchni trójkąta,
- TX1 TY1, TX2 TY2 – ponieważ trójkąt jest płaski, musimy rozplanować na jego powierzchni współrzędne teksturowe. Przykładowo: jeżeli rozpinamy trójkąt, na którym będzie tekstura zwierzęcia lub kombajnu, to wartości te będą wynosiły tylko 0 lub 1. Jeżeli rozpinamy płot, to współrzędne TY1=0, TY2=1, TX1=0, TX2=długość płotu.

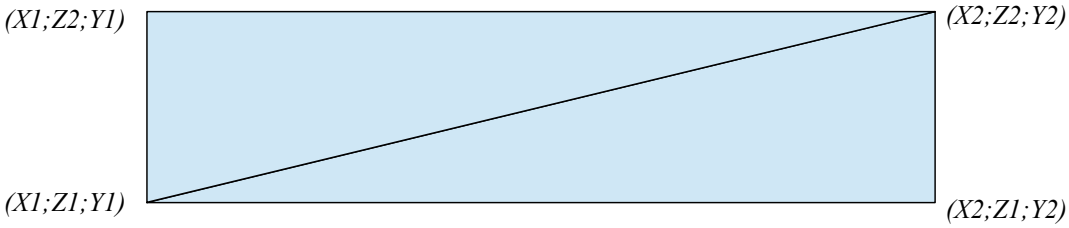
Jeśli przeanalizujemy dokładniej powyższy przykład, to stwierdzimy, że pojedynczy prostokąt musi się składać z dwóch trójkątów. Ale tutaj jeszcze czeka na nas drobny problem – każdy trójkąt jest widoczny tylko z jednej strony – czyli płotek zobaczymy od strony torów, ale od strony domku już nie. Możemy ten problem obejść rysując oba trójkąty powtórnie, ale tym razem ustawić wierzchołki w kierunku przeciwnym do ruchu wskazówek zegara.

Powyższa definicja może się wydawać strasznie zawiła i ręczne wstawianie trójkątów może się nam wydać

niemożliwe. Zwróćmy jednak uwagę na pewien istotny szczegół: chcemy wstawić płot, a więc obiekt, który będzie umieszczony w płaszczyźnie pionowej. Od razu podana wyżej definicja znacznie się uprości. Oczywiście jeżeli chcemy mieć płot w kształcie prostokąta, to musimy koniecznie wstawić 2 trójkąty:

```
node 1000 0 none triangles TEKSTURA
X1 Z1 Y1   AX AZ AY     TX1 TY1 end
X2 Z1 Y2   AX AZ AY     TX2 TY1 end
X2 Z2 Y2   AX AZ AY     TX2 TY2 end

X1 Z1 Y1   AX AZ AY     TX1 TY1 end
X2 Z2 Y2   AX AZ AY     TX2 TY2 end
X1 Z2 Y1   AX AZ AY     TX1 TY2 endtri
```

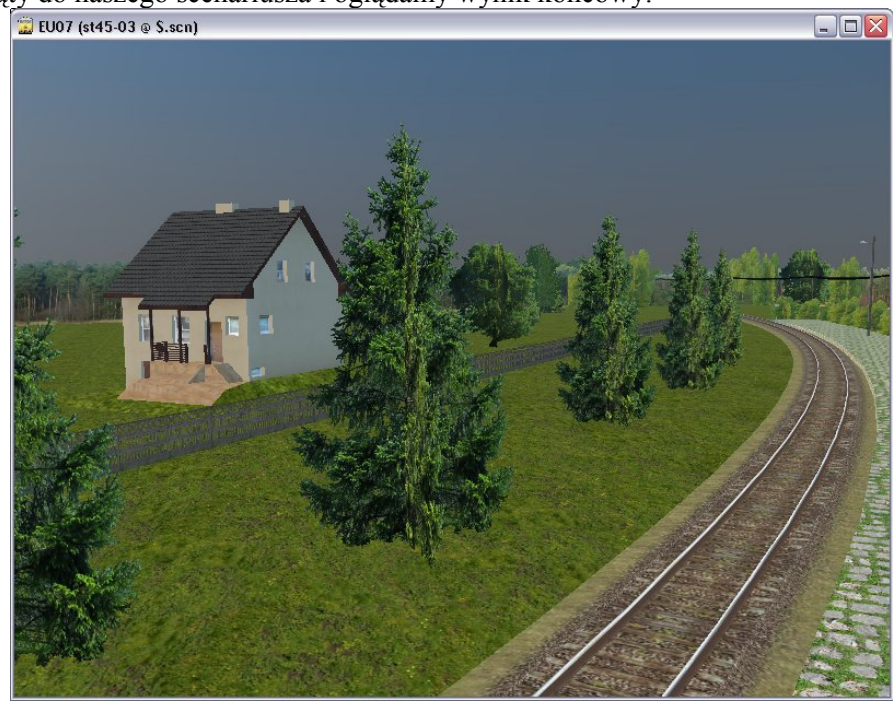


Po takim uproszczeniu wystarczają nam do zbudowania dwóch trójkątów zaledwie 2 punkty (początek i koniec płotu), oraz założona wysokość (poziom gruntu, oraz poziom gruntu + zakładana wysokość).

Skoro znamy już teorię, możemy sprawdzić ją praktycznie, czyli wstawić nasz płot. Wszystkie współrzędne są znane, wektory normalne możemy przyjąć *(0;0;0)*, natomiast współrzędne TX i TY będą równe: *TX1=0, TY1=0, TX2=75, TY2=1*. W katalogu *textures* znajduje się tekstura *fence-concrete1*, czyli betonowy płot.

```
node 1000 0 none triangles fence-concrete1
-7163.75 3.0 -13757     0 0 0  0 0 end
-7059.5 3.0 -13648      0 0 0  75 0 end
-7059.5 4.5 -13648      0 0 0  75 1 end
-7163.75 3.0 -13757     0 0 0  0 0 end
-7059.5 4.5 -13648      0 0 0  75 1 end
-7163.75 4.5 -13757     0 0 0  0 1 endtri
```

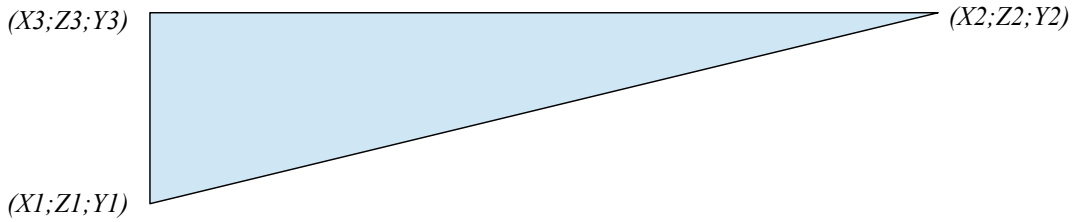
Kopiujemy te trójkąty do naszego scenariusza i oglądamy wynik końcowy:



Jedyną wadą takiego wstawiania jest brak wektora normalnego do powierzchni trójkąta. O ile jego brak nie stanowi problemu dla symulatora, o tyle jednak wczytanie scenerii z takim trójkątem w edytorze Rainsted zakończy się niepowodzeniem, podobnie jak w Szopa Track Viewer.

Skoro już jednak za coś się zabraliśmy, to zróbmy to porządnie – kto nam broni obliczyć wektor normalny ręcznie? W internecie możemy poszukać danych na temat obliczania wektora normalnego do płaszczyzny, dowiemy się, że płaszczyzna jest wyznaczana przez 2 wektory, a ich iloczyn daje właśnie wektor normalny. Zbyt długiego wywodu nie ma co przeprowadzać, od razu podam gotowy wzór:

$$AX = (y_3 - y_1)(z_2 - z_1) - (y_2 - y_1)(z_3 - z_1)$$
$$AY = (x_2 - x_1)(z_3 - z_1) - (x_3 - x_1)(z_2 - z_1)$$
$$AZ = (x_2 - x_1)(y_3 - y_1) - (x_3 - x_1)(y_2 - y_1)$$



To jeszcze nie wszystko, należy otrzymane wartości podzielić przez maksymalną wartość z trzech otrzymanych – tak aby największa wartość z tych trzech współrzędnych nie była większa od 1. Ponownie zapisujemy trójkąty do scenariusza:

```
node 1000 0 none triangles fence-concrete1
-7163.75 3.0 -13757     -1 0 0.956    0 0 end
-7059.5 3.0 -13648      -1 0 0.956    75 0 end
-7059.5 4.5 -13648      -1 0 0.956    75 1 end
-7163.75 3.0 -13757     -1 0 0.956    0 0 end
-7059.5 4.5 -13648      -1 0 0.956    75 1 end
-7163.75 4.5 -13757     -1 0 0.956    0 1 endtri
```

Można przygotować sobie skrypt w arkuszu kalkulacyjnym, który będzie automatycznie takie wektory obliczał. W niektórych szczególnych przypadkach nie musimy ich obliczać – dla powierzchni płaskich poziomych, takich jak droga, powierzchnia peronu, itp., wektor normalny wynosi *[0; 1; 0]*.

Porada:

Teraz możesz się zabrać za dekorowanie pustynnych scenerii. Developerzy przywitają Ciebie z otwartymi ramionami! ;-)

Podany sposób na dekorowanie scenerii jest niestety bardzo smutny, bo chociaż w założeniu niezwykle prosty, to jednak jest czasochłonny. Jeżeli jednak potrafimy w sprytny sposób wykorzystać excela, OOCalc (możemy także skorzystać z gotowych skryptów udostępnionych na forum MaSzyny – więcej w rozdziale 8.2.5.), to możemy naprawdę zdziałać cuda. Jako zachętę dodam, że wszystkie dekoracje miejskie w Macierzewie i Pomiankach (sceneria Całkowo) były wstawiane właśnie w taki sposób.

5.6. Animacje

Niektórymi obiektami można poruszać, np.: w Całkowie animacje wykorzystywane są do chowania i pokazywania pasażerów na peronach. Animacje są również wykorzystywane do poruszania ramion semaforów kształtowych, zamykania rogatek na przejazdach.

Jako przykład weźmy ludzika, którego wstawiliśmy na peron przystanku Macierzewo Jezioro. Na początek musimy obejrzeć jego plik .inc aby sprawdzić, czy w ogóle mamy możliwość jego animacji.

```
origin (p2) (p3) (p4)
rotate 0 (p5) 0
node 500 0 (p1) model 0 0 0 0 posers/sport4_st.t3d none endmodel
rotate 0 0 0
endorigin
```

3 linia mówi nam coś ważnego – ludzik jest modelem zawartym w pliku *sport4_st.t3d*, znajdującym się w katalogu *models/posers/*. Możemy teraz otworzyć ten model, choć zapewne napotkamy pewną trudność – w katalogu

z modelami będzie plik o takiej nazwie ale rozszerzeniu *.e3d*. Niestety tego pliku już tak łatwo nie obejrzymy, gdyż jest to plik binarny. Oryginalny model w formacie *.t3d* znajdziemy w repozytorium symulatora MaSzyna.

Kiedy już znajdziemy plik *.t3d* otwieramy go za pomocą notatnika:

```
//---------------------------------------------------------------------------
Parent: none
Type: Mesh
Name: banan
Anim: false
Ambient: 255 255 255
Diffuse: 255 255 255
Specular: 229.5 229.5 229.5
SelfIllum: false
Wire: false
WireSize: 1.0
Opacity: 100.0
Map: none
MaxDistance: 500
MinDistance: 0
Transform:
           1.0 0.0 0.0 0.0
           0.0 1.0 0.0 0.0
           0.0 0.0 1.0 0.0
           0.0 0.0 0.0 1.0
NumVerts: 0
//---------------------------------------------------------------------------
Parent: banan
Type: Mesh
Name: lp_stand
Anim: false
Ambient: 255 255 255
Diffuse: 255 255 255
Specular: 255 255 255
SelfIllum: false
Wire: false
WireSize: 1.0
Opacity: 100.0
Map: sportive04_m_25
MaxDistance: 500
MinDistance: 20
Transform:
           1.0 0.0 0.0 0.0
           0.0 1.0 0.0 0.0
           0.0 0.0 1.0 0.0
           0.0 0.0 0.0 1.0
NumVerts: 1320
1
0.0512946 -0.0418023 1.82036  0.566144 0.257494
0.0034449 -0.0714745 1.80372  0.498899 0.281643
0.00372945 -0.079366 1.77013  0.498439 0.225404
(...)

//---------------------------------------------------------------------------
Parent: banan
Type: Mesh
Name: sportive04_m_highpoly
Anim: false
Ambient: 255 255 255
Diffuse: 255 255 255
Specular: 255 255 255
SelfIllum: false
Wire: false
WireSize: 1.0
Opacity: 100.0
Map: sportive04_m_25
MaxDistance: 20
MinDistance: 0
Transform:
           1.0 0.0 0.0 0.0
           0.0 1.0 0.0 0.0
```

```
           0.0 0.0 1.0 0.0
           0.0 0.0 0.0 1.0
NumVerts: 14094
1
0.0344064 0.0719132 0.709778  0.952001 0.278686
0.0293222 0.0539116 0.69688   0.959155 0.275815
0.0121864 -0.00295512 0.73231 0.981887 0.288345
(...)
```

W tym pliku mamy zapisane wierzchołki stanowiące model, który jest porozdzielany na mniejsze submodele. Każdy model ma swojego rodzica, i w przypadku animacji modelu rodzica, submodel także jest animowany. Natomiast kiedy animujemy submodel, to nie animujemy jego rodzica. Zawiłe? No to przeanalizujemy powyższy plik:

- Model: banan, rodzic: brak,
- Model: lp_stand, rodzic: banan,
- Model: sportive04_m_highpoly, rodzic: banan.

Oznacza to, że przy animacji musimy wybierać model o nazwie *banan*. W przeciwnym wypadku animacja będzie niepełna (był swego czasu taki przykład w Całkowie – pasażer był animowany, ale gazeta którą czytał już nie, i w efekcie było widać na peronach lewitujące gazety).

Wiemy już jak nazywa się model, który mamy animować – pora zająć się eventami służącymi do animacji. Przypomnijmy sobie jeszcze wiersz, w którym wstawiliśmy ludka do scenerii:

```
include posers/sport4_st.inc ludek01 -7167.26 3.5 -13731.78 113 end
```

Nazwaliśmy go *ludek01* – bardzo ważne, aby obiekty animowane miały własne nazwy. Teraz wpiszemy event animujący:

```
event animacja1 animation 0 ludek01 translate banan 0 1 0 10 endevent
```

- *animacja1* – nazwa eventu,
- *animation* – słowo kluczowe,
- *0* – opóźnienie wykonania eventu,
- *ludek01* – nazwa obiektu, który chcemy animować,
- *translate* – operator, do wyboru mamy dwa typy animacji: translację o wektor (translate) i obrót o kąt (rotate),
- *banan* – model wewnątrz obiektu, który chcemy animować,
- *0 1 0* – wektor, o który przesuwamy ludka w formacie [X; Z; Y],
- *10* – szybkość wykonania animacji.

Uwaga! Wektor, który podajemy, musi być zawsze podany względem punktu w którym jest ludek umieszczony w scenerii. Co to oznacza? Że jeżeli chcemy cofnąć tę animację i wrócić ludka do położenia początkowego, to nie podajemy w kolejnej animacji wektora [0; -1; 0], tylko [0; 0; 0].

Teraz będziemy obracać ludzikiem:

```
event animacja2 animation 0 ludek01 rotate banan 0 90 0 3 endevent
```

Składnia jest niemal identyczna, tylko zamiast wektora przesunięcia podajemy wektor kątów [α; γ; β] wyrażony w stopniach. Analogicznie ostatni parametr określa szybkość animacji. W powyższym przykładzie ludek obróci się o 90° na peronie, ale nadal będzie na nim stał.

Inny przykład? Skopiujcie poniższy kod do swojego pliku *.ctr* scenariusza *calkowo_cargo*, ustawcie się przed semaforami wyjazdowymi z Całkowa w kierunku Wilisia, i naciśnijcie Shift+9:

```
event keyctrl09 multiple 0 none cal_g_ruch_opad cal_g_ruch_wahadlo cal_g_ruch_latarnia
          cal_g_ruch_wyzsze_ramie endevent
          event cal_g_ruch_opad multiple 0 none cgpp1 cgpp2 cgpp3 cgpp4 cgpp5 cgpp6 cgpp7
          endevent
          event cal_g_ruch_wahadlo multiple 2.1 none cgpr1 cgpr2 cgpr3 cgpr4 cgpr5 cgpr6
          endevent
          event cal_g_ruch_latarnia multiple 0 none cgpl1 cgpl2 endevent
          event cal_g_ruch_wyzsze_ramie multiple 0 none cgpg1 cgpg2 endevent
event cgpp1 animation 0.0 cal_g_sk2 rotate ramie_dol 0 -45 0 40 endevent
```

```
event cgpp2 animation 1.0 cal_g_sk2 rotate ramie_dol 0 -70 0 50 endevent
event cgpp3 animation 1.2 cal_g_sk2 rotate ramie_dol 0 -90 0 70 endevent
event cgpp4 animation 1.35 cal_g_sk2 rotate ramie_dol 6 -110 0 95 endevent
event cgpp5 animation 1.5 cal_g_sk2 rotate ramie_dol 0 -130 0 140 endevent
event cgpp6 animation 1.6 cal_g_sk2 rotate ramie_dol 2 -150 0 180 endevent
event cgpp7 animation 1.7 cal_g_sk2 rotate ramie_dol 5 -210 0 240 endevent
event cgpr1 animation 0.0 cal_g_sk2 rotate ramie_dol 5 -155 0 130 endevent
event cgpr2 animation 0.4 cal_g_sk2 rotate ramie_dol 5 -200 0 110 endevent
event cgpr3 animation 0.8 cal_g_sk2 rotate ramie_dol 5 -165 0 90 endevent
event cgpr4 animation 1.15 cal_g_sk2 rotate ramie_dol 5 -190 0 65 endevent
event cgpr5 animation 1.4 cal_g_sk2 rotate ramie_dol 5 -175 0 45 endevent
event cgpr6 animation 1.7 cal_g_sk2 rotate ramie_dol 5 -180 0 25 endevent
event cgpl1 animation 0.0 cal_g_sk2 rotate przeslona_02 0 -45 0 45 endevent
event cgpl2 animation 1.0 cal_g_sk2 rotate przeslona_02 0 0 0 240 endevent
event cgpg1 animation 0.0 cal_g_sk2 rotate ramie_gora 0 45 0 40 endevent
event cgpg2 animation 0.0 cal_g_sk2 rotate przeslona_01 0 -45 0 45 endevent
```

Co się wydarzyło? Wielka demolka ;-) Na marginesie: choć wygląda to dosyć obiecująco, to podobno w rzeczywistości takie zachowanie semafora nie jest możliwe nawet po zerwaniu pędni – wynika to z konstrukcji mechanicznej semaforów kształtowych.

Jakieś inne przykłady zastosowania animacji? Np.: dźwig budowlany na scenerii, będzie znacznie bardziej wiarygodny, jeżeli będzie się obracał co jakiś czas.

5.7. Polecenie config

Polecenia config i endconfig służą nie tyle do sterowania symulacją, co do konfiguracji parsera, i zmiany parametrów, które są zawarte w *eu07.ini*. Należy przy tym pamiętać, że nie na wszystkie parametry mamy wpływ – do tej grupy należy ustawiana w Rainsted rozdzielczość i tryb pełnoekranowy. Niektóre parametry możliwe do zmiany poprzez zastosowanie *config*:

- *movelight* – ustawienie dnia roku, opisane w rozdziale 2.1.
- *hiddenevents* – możliwość zdalnego przypisywania eventów do torów, do tego należy podać parametr *yes* lub *no*. W zależności od tego parametru zdalne przypisywanie jest włączane lub wyłączane.
- *Joinduplicatedevents* – w normalnej sytuacji nazwy eventów muszą być unikalne. Po użyciu tego polecenia z parametrem *yes* możemy stosować eventy o takich samych nazwach. W chwili wywołania eventu o nazwie np.: *zdarzenie1* uruchomione zostaną wszystkie eventy tak nazwane. **Uwaga! Komenda niezalecana, gdyż jej użycie może wprowadzić chaos!**
- *Livetraction* – komenda, która odpowiada za zasilanie lokomotyw elektrycznych tylko w sytuacji, gdy podniosą pantograf pod siecią. Użycie jej z parametrem *no* spowoduje, że będziemy mogli jeździć lokomotywą elektryczną po scenerii bez druta.
- *Enabletraction* – komenda, która odpowiada za łamanie pantografów lokomotyw elektrycznych pod nieprawidłowo ułożoną siecią trakcyjną. Użycie jej z parametrem *no* spowoduje, że nigdy nie połamiemy pantografów.

Omówiono tutaj tylko niektóre z dostępnych poleceń, polecam samemu poeksperymentować z przestawianiem komend z *eu07.ini*.

Porada:

Jeśli napisaliśmy jakąś nową scenerię, która będzie przeznaczona do jazdy lokomotywami elektrycznymi, a jeszcze nie rozwiesiliśmy na niej sieci, to możemy sobie dopisać w scenariuszu config livetraction no enabletraction no endconfig. *W efekcie zawsze będziemy mogli jeździć lokomotywą elektryczną, nawet jeśli zapomnimy odznaczyć właściwych opcji w Rainsted. Ta sztuczka została wykorzystana w testowym scenariuszu metra bałtyckiego.*

6. Coś dla ekspertów

Znamy już właściwie wszystkie eventy, komendy, obiekty... Możemy pisać bardzo rozbudowane i znakomite scenariusze. Ale... niestety sama znajomość języka eventów jeszcze nie gwarantuje że poradzimy sobie z naszym pomysłem na scenariusz. W tym rozdziale poznamy najbardziej zaawansowane konstrukcje, takie które pozwolą nam na realizacje niezwykle skomplikowanych przebiegów na stacjach.

6.1. Algorytm OS

Próbowaliście w tradycyjny sposób napisać eventy dla stacji, gdzie jeden pociąg przejeżdża, a w tle manewruje lokomotywa, aby podczepić się do składu? Proste, na pewno się uda. A co ze stacjami, gdzie pojawi się trzeci pociąg? Nagle kod eventów robi się skomplikowany. Strach pomyśleć, co by się stało, gdyby doszły następne pociągi... Nawet jeśli w końcu uda się nam napisać eventy w taki sposób, aby wszystko działało, to jeśli nie załatwią nas inni użytkownicy MaSzyny swoim sposobem jazdy, to wykończą nas zmiany w exe.

Autor niniejszego opracowania również stanął przed takim problemem. Na jednej ze stacji ówcześnie pisanego scenariusza (była to obsługa dla Wilisia, scenariusz *calkowo_turkol*) postanowił zastosować jakiś algorytm sterujący, który byłby jednorodny i w miarę przejrzysty, a jednocześnie byłby odporny na nieprzewidywalne sytuacje. W ten sposób powstały podwaliny algorytmu, nazwanego później algorytmem OS (od słów: obsługa stacji). Już pierwsze testy pokazały jak bardzo algorytm ułatwia pisanie skomplikowanych przebiegów na stacjach – wystarczy wspomnieć, że potem autor wszystkie swoje wcześniej napisane w tradycyjny sposób scenariusze napisał jeszcze raz, z wykorzystaniem algorytmu OS.

6.1.1. Budowa OS

Najważniejszą zaletą OS jest to, że niezależnie od tego czy piszemy obsługę dla jakieś prostej mijanki, czy dla giga-skomplikowanych manewrów na rozbudowanej stacji, za każdym razem obsługę się pisze na jedno kopyto – czyli założenia są identyczne, a jedynie wydłuża się nam ilość kodu potrzebna do obsługi stacji.

Pierwszą rzeczą jest przypisanie dla każdego pociągu / lokomotywy osobnej komórki, którą nazwiemy *flagą pojazdu*. Wagony stojące luzem nie potrzebują takich flag. Na początku definiujemy komórkę i eventy, które będą zmieniać jej zawartość:

```
node -1 0 stacja1_lok1_stat memcell 1.0 1.0 1.0 a 0 * none endmemcell
event stacja1_lok_stat_b updatevalues 0 stacja1_lok1_stat b * * endevent
event stacja1_lok_stat_c updatevalues 0 stacja1_lok1_stat c * * endevent
event stacja1_lok_stat_1 updatevalues 0 stacja1_lok1_stat a 1 * endevent
event stacja1_lok_stat_2 updatevalues 0 stacja1_lok1_stat a 2 * endevent
```

Tak może wyglądać flaga i eventy zmieniające jej zawartość np.: dla pociągu osobowego przejeżdżającego z zatrzymaniem przez stację. Co która wartość oznacza?

- Wartość liczbowa 0 oznacza, że pociąg znajduje się w etapie dojeżdżania do stacji, jest przed semaforem wjazdowym,
- Wartość liczbowa 1 oznacza, że pociąg znajduje się między semaforem wjazdowym, a semaforem wyjazdowym,
- Wartość liczbowa 2 oznacza, że pociąg otrzymał wyjazd ze stacji i wyjeżdża lub wyjechał.

Wartości liczbowe komórki tylko z grubsza definiują stan pociągu, wartość tekstowa zawiera doprecyzowanie:

- Wartość a – pociąg wykonuje dany manewr lub przejazd, blokuje szlak lub zwrotnice,
- Wartość b – pociąg wykonał manewr lub przejazd, nie blokuje zwrotnic, ale nie może wykonać następnego ruchu, bo np.: wysadza pasażerów w peronie, lokomotywa podczepia wagony, itp.
- Wartość c – pociąg wykonał manewr lub przejazd, jest gotowy do wykonania następnego ruchu.

Rozpiszmy teraz jak będzie się zmieniała wartość komórki pociągu osobowego przejeżdżającego z zatrzymaniem przez nieskomplikowaną stację:

- *a 0 ** - pociąg dojeżdża do stacji, jest daleko przed semaforem wjazdowym,
- *c 0 ** - pociąg dojechał lub już stoi pod semaforem wjazdowym, można podać wjazd,
- *a 1 ** - pociąg otrzymał wjazd i wjeżdża w peron, blokuje rozjazdy,
- *b 1 ** - pociąg wjechał w peron, już nie blokuje rozjazdów (inne pociągi mogą przejeżdżać) ale nie może

otrzymać wyjazdu bo wsiadają i wysiadają pasażerowie,

- *c 1 ** - pociąg zakończył wymianę pasażerów, można podać wyjazd,
- *a 2 ** - pociąg wyjeżdża ze stacji, blokuje rozjazdy,
- *c 2 ** - pociąg wyjechał na dobre ze stacji, rozjazdy zwolnione.

Na ogół ostatni stan zamiast literki *c* często ma literkę *d*. Ma to uzasadnienie czysto wydajnościowe, z analizy dalszych przykładów zobaczymy, dlaczego warto stosować takie oznaczenie. Warto – ale nie koniecznie trzeba, początkujący mogą to pominąć.

Dalej procedurę obsługi stacji można podzielić na grupę eventów odwzorowującą dyżurnego, oraz grupę odwzorowującą maszynistów:

- Eventy dyżurnego sprawdzają aktualny stan pociągów i jeżeli stwierdzą, że dany pociąg jest w stanie gotowości do następnego ruchu, oraz żaden inny pociąg nie blokuje drogi, to ustawiana jest flaga następnego ruchu – zawsze jest to *a* z odpowiednim numerem,
- Eventy maszynisty to eventy umieszczone w torach – pociąg wjeżdżający na tor ustawia sobie odpowiednią flagę: *b* lub *c*, i jednocześnie wywołuje event obsługi stacji, czyli powiadamia dyżurnego, że może mu ustawić dalszą drogę.

Tak wygląda w dużym skrócie budowa algorytmu. Mało zrozumiałe? Pora na konkretne przykłady.

6.1.2. Przykład 1 – Paszki Wielkie

Spróbujmy napisać obsługę stacji dla stacji Paszki Wielkie. Założenie jest takie, że mijamy się z pociągiem osobowym. My naszym pociągiem podjeżdżamy do semafora F, następnie jedziemy pod semafor B, i stamtąd możemy otrzymać wyjazd do Całkowa. Pociąg osobowy dojeżdża pod semafor A, następnie wjeżdża pod semafor E, następuje wymiana pasażerów, po czym wyjeżdża do Macierzewa.

Układ torowy stacji Paszki Wielkie.

Zatem zaczynamy od zdefiniowania flag pociągów:

```
node -1 0 paszki_osobowy_stat memcell 1.0 1.0 1.0 a 0 * none endmemcell
event paszki_osobowy_stat_b updatevalues 0 paszki_osobowy_stat b * * endevent
event paszki_osobowy_stat_c updatevalues 0 paszki_osobowy_stat c * * endevent
event paszki_osobowy_stat_1 updatevalues 0 paszki_osobowy_stat a 1 * endevent
event paszki_osobowy_stat_2 updatevalues 0 paszki_osobowy_stat a 2 * endevent

node -1 0 paszki_towarowy_stat memcell 1.0 1.0 1.0 a 0 * none endmemcell
event paszki_towarowy_stat_c updatevalues 0 paszki_towarowy_stat c * * endevent
event paszki_towarowy_stat_1 updatevalues 0 paszki_towarowy_stat a 1 * endevent
event paszki_towarowy_stat_2 updatevalues 0 paszki_towarowy_stat a 2 * endevent
```

Warto sobie rozpisać dokładnie wszystkie ruchy pociągów, można to zrobić metodą „kartka i długopis”, można przygotować tabelę w Wordzie, Excelu, innym programie – wszystko jedno jak to zrobimy, ale musimy mieć jasno i czytelnie napisane, która flaga oznacza dane położenie.

Dla pociągu towarowego:

- *a 0 ** - dojeżdżamy do stacji Paszki,
- *c 0 ** - zgłosiliśmy się pod semaforem F, wołamy dyżurnego,
- *a 1 ** - otrzymaliśmy wjazd pod semafor B, wjeżdżamy – w tym stanie blokujemy zwrotnicę wyjazdową z Paszek,
- *c 1 ** - wjechaliśmy pod semafor B, zwrotnica nr 3 jest zwolniona, wołamy dyżurnego,
- *a 2 ** - otrzymaliśmy wyjazd do Całkowa, wyjeżdżamy, blokujemy zwrotnice nr 1,
- *c 2 ** - wyjechaliśmy ze stacji.

Dla pociągu osobowego:

- *a 0 ** - pociąg osobowy zmierza do stacji Paszki, jeszcze jest daleko,
- *c 0 ** - pociąg podjechał do semafora wjazdowego A, zgłasza to dyżurnemu,
- *a 1 ** - pociąg osobowy wjeżdża na stację, blokuje zwrotnice nr 1 i 2,
- *b 1 ** - pociąg już wjechał pod semafor E, nie blokuje zwrotnic, ale trwa wymiana pasażerów,
- *c 1 ** - pasażerowie już wsiedli / wysiedli, pociąg zgłasza dyżurnemu gotowość do dalszej jazdy,
- *a 2 ** - wyjazd do Macierzewa, blokuje zwrotnice nr 3,
- *c 2 ** - pociąg wyjechał ze stacji Paszki Wielkie.

Możemy sobie to wszystko tak rozpisać, ewentualnie jeżeli szkoda nam papieru lub klawiatury, to możemy sobie zrobić takie tabelki:

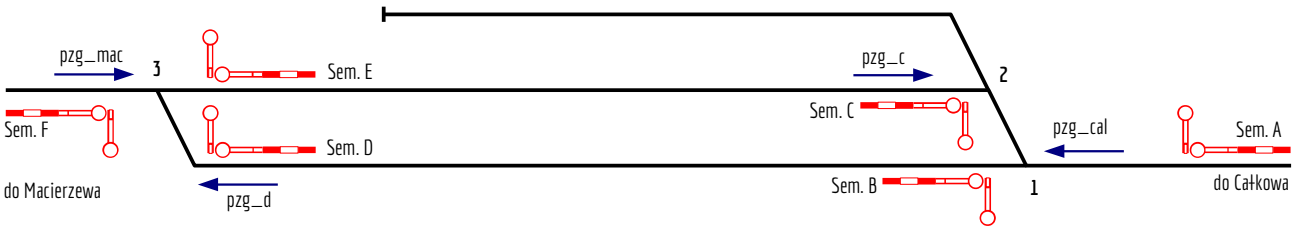
Towarowy

0 – 1	Od semafora F (wjazd od Macierzewa) pod semafor B
1 – 2	Od semafora B – wyjazd do Całkowa

Osobowy

0 – 1	Od semafora A (wjazd od Całkowa) pod semafor E
1 – 2	Od semafora E – wyjazd do Macierzewa

Teraz napiszemy procedurę zgłaszania pociągów przez maszynistów:



Układ torowy stacji Paszki Wielkie – przypisanie eventów zgłaszających.

Na początku przypisujemy do torów eventy zgłaszające. Aby zapewnić prawidłową obsługę potrzebujemy 6 eventów. Na powyższym rysunku zaznaczone jest umieszczenie 4 z nich, pozostałe 2 umieszczamy przed semaforami wjazdowymi, najlepiej na wysokości tarczy ostrzegawczej.

```
event none3830:event2 multiple 0 none pzg_a endevent //semafor wjazdowy A
event none_null_046:event1 multiple 0 none pzg_c endevent //wjazd pod semafor E (od strony C)
event none_null_049:event2 multiple 0 none pzg_d endevent //wjazd pod semafor B (od strony d)
event none4262:event2 multiple 0 none pzg_f endevent //semafor wjazdowy F
event none3834:event2 multiple 0 none pzg_mac endevent //wyjazd do Macierzewa
event paszprz:event2 multiple 0 none pzg_cal endevent //wyjazd do Calkowa
```

Mamy przypisane eventy do torów, możemy pisać obsługę zgłaszania. Najpierw semafory wjazdowe:

```
event pzg_a multiple 0 paszki_osobowy_stat paszki_osobowy_stat_c paszki_obsluga_stacji
          condition memcompare a 0 * endevent
event pzg_f multiple 0 paszki_towarowy_stat paszki_towarowy_stat_c paszki_obsluga_stacji
          condition memcompare a 0 * endevent
```

Po najechaniu na event zgłaszający pociąg ustawia swoją flagę na *c **, po czym powiadamia dyżurnego (poprzez wywołanie eventu *paszki_obsluga_stacji*). Teraz czas na eventy znajdujące się w obrębie stacji:

```
event pzg_c multiple 5.5 pasz_zwr2 pzg_c1 else pzg_c condition trackfree endevent
 event pzg_c1 multiple 0 paszki_osobowy_stat paszki_osobowy_stat_b paszki_obsluga_stacji
          pzg_c1x condition memcompare a 1 * endevent
 event pzg_c1x multiple 60 none paszki_osobowy_stat_c paszki_obsluga_stacji endevent
event pzg_d multiple 5.5 pasz_zwr3 pzg_d1 else pzg_d condition trackfree endevent
```

```
 event pzg_d1 multiple 0 paszki_towarowy_stat paszki_towarowy_stat_c paszki_obsluga_stacji
          condition memcompare a 1 * endevent
event pzg_mac multiple 5.5 pasz_zwr3 pzg_mac1 else pzg_mac condition trackfree endevent
 event pzg_mac1 multiple 0 paszki_osobowy_stat paszki_osobowy_stat_c paszki_obsluga_stacji
          condition memcompare a 2 * endevent
event pzg_cal multiple 5.5 pasz_zwr1 pzg_cal1 else pzg_cal condition trackfree endevent
 event pzg_cal1 multiple 0 paszki_towarowy_stat paszki_towarowy_stat_c paszki_obsluga_stacji
          condition memcompare a 2 * endevent
```

Najechanie na event zgłaszający powoduje najpierw uruchomienie testu zwolnienia ostatniej zwrotnicy, tak aby można było jak najszybciej zgłosić zmianę flagi, ale jednocześnie nie podciąć wjeżdżającego pociągu. Test nie ma sensu przy semaforach wjazdowych, wszędzie indziej usilnie zaleca się wykonanie testu na zwolnienie ostatniej zwrotnicy.

Kiedy test zwolnienia wypadnie pozytywnie, uruchamiane jest zdarzenie zmieniające flagę pociągu. Zaprezentowany przykład wydaje się niepotrzebnie skomplikowany, ale w sytuacji, gdyby jeszcze jakieś inne pociągi były przewidziane do wjazdu na ten sam event zgłaszający, to zwyczajnie dopisujemy kolejny event z testem (czyli: gdyby kilka pociągów wjeżdżało pod semafor d, to dopisujemy kolejne eventy *pzg_d2*, *pzg_d3*, ...).

W przypadku pociągu osobowego, kiedy wjedzie pod semafor E (event nazywa się *pzg_c*, gdyż wjeżdża od strony semafora C), najpierw zmienia swoją flagę na *b * **, odczeka 60 sekund (przewidziane na wymianę pasażerów), a potem ustawia flagę *c * **.

A teraz gwóźdź programu! Napiszemy event *paszki_obsluga_stacji*:

```
node -1 0 pos memcell 1.0 1.0 1.0 * 0 0 none endmemcell
event pos_0 updatevalues 0 pos * 0 0 endevent
event pos_1 updatevalues 0 pos * 1 1 endevent

event paszki_obsluga_stacji multiple 5.5 pos pos_next pos_1 else paszki_obsluga_stacji
          condition memcompare * 0 0 endevent

node -1 0 paszki_obsluga_stacji_stat memcell 1.0 1.0 1.0 * 0 0 none endmemcell
event paszki_obsluga_stacji_stat_up addvalues 0 paszki_obsluga_stacji_stat * 1 1 endevent
event paszki_obsluga_stacji_stat_0 updatevalues 0 paszki_obsluga_stacji_stat * 0 0 endevent

event pos_next multiple 0 paszki_obsluga_stacji_stat paszki_obsluga_stacji_osobowy
          paszki_obsluga_stacji_stat_up else pos_next1 condition memcompare * 0 0 endevent
event pos_next1 multiple 0 paszki_obsluga_stacji_stat paszki_obsluga_stacji_towarowy
          paszki_obsluga_stacji_stat_up else paszki_obsluga_stacji_stat_0 pos_0 condition
          memcompare * 1 1 endevent
```

Na początek sterta kodu, która może niewiele nam mówi. Otóż wywołujemy obsługę stacji, która ma zabezpieczenie przed jednoczesnym wywołaniem przez dwa pociągi jednocześnie. Po to są właśnie te wszystkie komórki i dodatkowe eventy. Prześledźmy zatem co się stanie, jeżeli pociąg wywoła event *paszki_obsluga_stacji*:

- Event sprawdzi czy komórka *pos* ma wartości równe ** 0 0*. Jeżeli nie, to event będzie się powtarzał tak długo, dopóki warunek nie zostanie spełniony. Jeśli test wypadnie pozytywnie, to zmieniamy wartości w komórce *pos* na ** 1 1* (czyli blokujemy event obsługi stacji przed wywołaniem go przez inny pociąg, zanim sami nie zakończymy cyklu obsługi), oraz wywołujemy event *pos_next*.
- Event *pos_next* to cała pętla obsługi dla wszystkich pociągów przewidzianych na stacji. Do realizacji tej pętli mamy pomocniczą komórkę *paszki_obsluga_stacji_stat*. Gdy rozpoczynamy obsługę stacji, event *pos_next* sprawdza wartość tej pomocniczej komórki, i jeżeli test wypadnie pozytywnie, to inkrementuje jej wartość (dodaje ** 1 1*), oraz wywołuje event obsługujący konkretny pociąg, Jeżeli test wypadnie negatywnie to wywoływane jest następne zdarzenie (w tym przypadku *pos_next1*),
- Po przejściu przez całą pętle i obsłużeniu wszystkich pociągów odbezpieczamy komórkę *pos* ustawiając jej wartość na ** 0 0*, oraz zerujemy pomocniczą komórkę *paszki_obsluga_stacji_stat*.

Mało zrozumiałe? No to aby było bardziej obrazowo, przebieg obsługi stacji będzie wyglądał tak:

1. Wywołany został event obsługi stacji, robiony jest test na dostępność, jeżeli wypada negatywnie, to po 5,5 sekundy ponownie jest uruchamiany event obsługi stacji, jeżeli test wypadnie pozytywnie -> punkt 2,
2. Uruchamiany jest *pos_next*. Robiony jest test na wartość komórki pomocniczej (która aktualnie wynosi ** 0 0*). Ponieważ jest to pierwszy cykl, test jest pozytywny, wartość komórki pomocniczej jest zmieniana na ** 1 1*, oraz wywoływany jest event *paszki_obsluga_stacji_osobowy*,
3. Event *paszki_obsluga_stacji_osobowy* sprawdza przebiegi, ustawia zwrotnice i semafory, a na koniec działania wywołuje *pos_next*,

4. Znowu jest uruchamiany *pos_next*. Tym razem test wypada negatywnie, bo wartość komórki pomocniczej wynosi ** 1 1*, wywoływany jest *pos_next1*.
5. Uruchamiany jest *pos_next1*, test wypada pozytywnie, toteż wartość komórki pomocniczej jest zmieniana na ** 2 2*, i wywoływany jest event *paszki_obsluga_stacji_towarowy*,
6. Event *paszki_obsluga_stacji_towarowy* sprawdza przebiegi, ustawia zwrotnice i semafory, a na koniec działania wywołuje *pos_next*,
7. Znowu jest uruchamiany *pos_next*. Test wypada negatywnie, bo wartość komórki pomocniczej wynosi ** 2 2*, wywoływany jest *pos_next1*.
8. Uruchamiany jest *pos_next1*, test wypada negatywnie, toteż zerowana jest komórka pomocnicza oraz komórka *pos*. Tutaj kończy się obsługa stacji.

Mamy już naszą pętlę wywołującą obsługę dla poszczególnych pociągów. Teraz możemy napisać eventy *paszki_obsluga_stacji_osobowy* i *paszki_obsluga_stacji_towarowy*.

```
event paszki_obsluga_stacji_osobowy multiple 0 paszki_osobowy_stat pos_osobowy_1 else pos_next
          condition memcompare c * * endevent
```

Pierwszy test – czy pociąg jest w stanie gotowości do dalszego ruchu (czyli czy ma flagę *c*), jeżeli nie jest, to dalsza jego obsługa nie ma sensu, i wracamy do pętli (wywołujemy *pos_next*).

```
event pos_osobowy_1 multiple 0 paszki_osobowy_stat pos_osobowy_1a else pos_osobowy_2 condition
          memcompare c 0 * endevent
event pos_osobowy_1a multiple 0 none paszki_osobowy_stat_1 pasz_zwr1- pasz_zwr2-
          paszprz01_zamykaj pos_osobowy_1sem pos_next endevent
event pos_osobowy_1sem multiple 12 none pasz_a_sr3 pasz_toa_od2 endevent
```

Kolejny event i kolejny test: jeżeli flaga pociągu ma wartości *c 0 ** to wywoływany jest event *pos_osobowy_1a*, jeżeli nie, przechodzimy do *pos_osobowy_2*.

Event *pos_osobowy_1a* odwzorowuje działanie dyżurnego: maszynista zgłosił, że jest pod semaforem wjazdowym (flaga *c 0 **), dyżurny sprawdza, czy może podać wjazd. W tym evencie nie ma żadnego warunku – w sumie to po co, nie ma żadnych przeciwwskazań, aby pociąg otrzymał wjazd. Dlatego od razu ustawiamy zwrotnice i podajemy semafor, a także – czego nie wolno nam zapomnieć – ustawiamy flagę pociągu na *a 1 **, oraz wracamy do pętli obsługi (wywołujemy *pos_next*).

```
event pos_osobowy_2 multiple 0 paszki_osobowy_stat pos_osobowy_2a else pos_next condition
          memcompare c 1 * endevent
event pos_osobowy_2a multiple 0 paszki_towarowy_stat pos_next else pos_osobowy_2b condition
          memcompare * 0 * endevent
event pos_osobowy_2b multiple 0 paszki_towarowy_stat pos_next else paszki_osobowy_stat_2
          pasz_zwr3+ pos_osobowy_2sem pos_next condition memcompare a 1 * endevent
event pos_osobowy_2sem multiple 5 none pasz_e_Sr2 endevent
```

Kolejny event jest znacznie bardziej ciekawy. Tak samo jak przy poprzednim, najpierw jest sprawdzenie czy flaga ma wartość *c 1 **, jeżeli nie to wracamy do pętli obsługi stacji, gdyż nie ma już innych stanów pociągu, które trzeba analizować.

Tym razem nie ma bezwarunkowego podania semafora. W pierwszym evencie *pos_osobowy_2a* sprawdzamy flagę pociągu towarowego, jeżeli jest równa ** 0 **, to przerywamy obsługę, wracamy do pętli (*pos_next*). Chyba logiczne – nie możemy dać osobowemu wyjazdu, jeżeli jeszcze nie przyjechał towarowy na mijankę. W drugim evencie *pos_osobowy_2b* ponownie test, obsługę przerywamy, jeżeli towarowy ma flagę *a 1 ** - również logiczne, pociąg towarowy może już wjeżdżać na stację, ale nie możemy wyjechać, bo towarowy blokuje nam zwrotnicę nr 3. Dopiero teraz mamy pewność, że szlak jest pusty, i osobowy może dostać wyjazd – czyli ustawiamy zwrotnicę, podajemy semafor, oraz ustawiamy flagę pociągu na *a 2 **, i wracamy do pętli.

W ten sposób zakończyliśmy pisanie obsługi stacji dla pociągu osobowego. Widzimy już ważną cechę – sprawdzanie możliwości ustawienia przebiegu odbywa się na podstawie testów „negatywnych", czyli na sprawdzeniu, czy inne pociągi nie blokują nam szlaku. Przy sprawdzaniu stanów innych pociągów przydatne będą dla nas tabelki, które sobie porozpisywaliśmy. Możliwe są również inne warunki ale o tym później. Najpierw napiszmy obsługę dla pociągu towarowego:

```
event paszki_obsluga_stacji_towarowy multiple 0 paszki_towarowy_stat pos_towarowy_1 else
          pos_next condition memcompare c * * endevent
event pos_towarowy_1 multiple 0 paszki_towarowy_stat pos_towarowy_1a else pos_towarowy_2
          condition memcompare c 0 * endevent
```

```
 event pos_towarowy_1a multiple 0 none pasz_zwr3- pos_towarowy_1sem paszki_towarowy_stat_1
          pos_next endevent
 event pos_towarowy_1sem multiple 4 none pasz_f_Sr3 pasz_tof_od2 pasz_f1_sp4 endevent
event pos_towarowy_2 multiple 0 paszki_towarowy_stat pos_towarowy_2a else pos_next condition
          memcompare c 1 * endevent
 event pos_towarowy_2a multiple 0 paszki_osobowy_stat pos_next else pos_towarowy_2b condition
          memcompare * 0 * endevent
 event pos_towarowy_2b multiple 0 paszki_osobowy_stat pos_next else pos_towarowy_2zwr
          pos_towarowy_2sem paszki_towarowy_stat_2 pos_next condition memcompare a 1 *
          endevent
 event pos_towarowy_2zwr multiple 0 none pasz_zwr1+ paszprz01_zamykaj endevent
 event pos_towarowy_2sem multiple 12 none pasz_b_Sr2 endevent
```

I tutaj się kończy cała procedura obsługi dla stacji Paszki Wielkie.

Jak wrażenia? Może się ten algorytm wydawać niepotrzebnie skomplikowany i rozdmuchany. Dla małych stacji i niewielkich przebiegów może faktycznie tak jest, ale jego możliwości uwidaczniają się przy obsłudze skomplikowanych przebiegów na większych stacjach: zasada pisania jest wciąż taka sama, myśleć i męczyć się przy pisaniu zbyt wiele nie trzeba, właściwie tylko podczas wymyślania przebiegów pociągów, i podczas pisania obsługi dla pociągów, które flagi innych pojazdów są zabronione dla danego ruchu. Ponadto jest ten algorytm odporny na zdarzenia losowe – wszystko jedno, co przyjedzie najpierw, czy towarowy, czy osobowy, czy wjadą jednocześnie – obsługa stacji i tak sobie z tym poradzi.

6.1.3. Przykład 2 – rozwinięcie stacji Paszki

Skoro znamy już absolutne podstawy, to opracujemy 2 rozwinięcia: pierwsze to będzie napisanie obsługi dla przejazdu kolejowego w Paszkach – przejazd ten to koszmar scenarzystów, bo znajduje się na terenie stacji. W naszym przykładzie nie jest jeszcze źle, ale spróbujcie uruchomić manewry na tej stacji... sterowanie przejazdem w tradycyjny sposób może się okazać problemem nie do przeskoczenia. Drugie rozwinięcie: czy towarowy, w sytuacji gdy osobowy już wjechał na stację, nie mógłby otrzymać przelotu? Spróbujemy zrobić przelot.

Na początek obsługa przejazdu. Przejazd potraktujemy jak jeszcze jeden pociąg:

```
event pos_next multiple 0 paszki_obsluga_stacji_stat paszki_obsluga_stacji_osobowy
          paszki_obsluga_stacji_stat_up else pos_next1 condition memcompare * 0 0 endevent
event pos_next1 multiple 0 paszki_obsluga_stacji_stat paszki_obsluga_stacji_towarowy
          paszki_obsluga_stacji_stat_up else pos_next2 condition memcompare * 1 1 endevent
event pos_next2 multiple 0 paszki_obsluga_stacji_stat paszki_obsluga_przejazdu
          paszki_obsluga_stacji_stat_up else paszki_obsluga_stacji_stat_0 pos_0 condition
          memcompare * 2 2 endevent
```

Dodaliśmy jeszcze jeden obiekt obsługiwany przez pętlę obsługi.

```
event paszki_obsluga_przejazdu multiple 0 paszprz01_syg_mem pos_prz1_1 else pos_next condition
          memcompare * 0 0 endevent
event pos_prz1_1 multiple 0 paszki_osobowy_stat pos_next else pos_prz1_2 condition memcompare
          a 1 * endevent
event pos_prz1_2 multiple 0 paszki_towarowy_stat pos_next else paszprz01_otwieraj1 pos_next
          condition memcompare a 2 * endevent
```

Na początku event obsługi sprawdza, czy przejazd w Paszkach jest zamknięty – jeżeli nie, to następuje powrót do pętli eventowej, gdyż cała obsługa przejazdu polega na jego otwieraniu. Zamykanie – jak zapewne zauważyliśmy – jest wykonywane przez eventy obsługujące poszczególne pociągi.

W dalszej kolejności sprawdzamy, czy pociąg osobowy nie ma flagi *a 1 **, wówczas przejazd nie może zostać otwarty, bo wjazd osobówki następuje przez przejazd. Kolejnym warunek, to aby towarowy nie miał flagi *a 2 **, albowiem przy wyjeździe towarowy blokuje przejazd. Jeżeli te 2 warunki zostały spełnione, przejazd zostanie otwarty.

Teraz czas na zrobienie przelotu pociągu towarowego. W pierwszej kolejności aktualizujemy możliwe flagi dla tego pociągu:

```
node -1 0 paszki_towarowy_stat memcell 1.0 1.0 1.0 a 0 * none endmemcell
event paszki_towarowy_stat_c updatevalues 0 paszki_towarowy_stat c * * endevent
event paszki_towarowy_stat_1 updatevalues 0 paszki_towarowy_stat a 1 * endevent
event paszki_towarowy_stat_2 updatevalues 0 paszki_towarowy_stat a 2 * endevent
```

```
event paszki_towarowy_stat_3 updatevalues 0 paszki_towarowy_stat a 3 * endevent
```

Aktualizujemy tabelkę:

Towarowy

0 – 1	Od semafora F (wjazd od Macierzewa) pod semafor B
1 – 2	Od semafora B – wyjazd do Całkowa
0 – 3	Od semafora F, przelot pod semaforem B, wyjazd do Całkowa

Jak widać, przelot opisujemy flagą ** 3 **.

Po zmianie flagi musimy uzupełnić eventy zgłaszające pociągi – konkretniej event zgłaszający wyjazd do Całkowa.

```
event pzg_cal multiple 5.5 pasz_zwr1 pzg_cal1 pzg_cal2 else pzg_cal condition trackfree
          endevent
 event pzg_cal1 multiple 0 paszki_towarowy_stat paszki_towarowy_stat_c paszki_obsluga_stacji
          condition memcompare a 2 * endevent
 event pzg_cal2 multiple 0 paszki_towarowy_stat paszki_towarowy_stat_c paszki_obsluga_stacji
          condition memcompare a 3 * endevent
```

Może w tym momencie niektórzy się zastanawiają: „A do czego w ogóle potrzebne zgłaszanie pociągów na wyjeździe?” W tym przykładzie nie jest to aż tak potrzebne, w zasadzie jest to używane tylko do otwarcia przejazdu po wyjeździe towarowego do Całkowa. Jednakże na ogół informacja o wyjeździe jest bardzo potrzebna.

Aktualizujemy obsługę stacji dla towarowego:

```
event paszki_obsluga_stacji_towarowy multiple 0 paszki_towarowy_stat pos_towarowy_1 else
          pos_next condition memcompare c * * endevent
event pos_towarowy_1 multiple 0 paszki_towarowy_stat pos_towarowy_1a else pos_towarowy_2
          condition memcompare c 0 * endevent
 event pos_towarowy_1a multiple 0 paszki_osobowy_stat pos_towarowy_1wjazd else pos_towarowy_1b
          condition memcompare * 0 * endevent
 event pos_towarowy_1b multiple 0 paszki_osobowy_stat pos_towarowy_1wjazd else
          pos_towarowy_1przelot condition memcompare a 1 * endevent
 event pos_towarowy_1wjazd multiple 0 none pasz_zwr3- pos_towarowy_1wjazdsem
          paszki_towarowy_stat_1 pos_next endevent
 event pos_towarowy_1wjazdsem multiple 4 none pasz_f_Sr3 pasz_tof_od2 pasz_f1_sp4 endevent
 event pos_towarowy_1przelot multiple 0 none pasz_zwr3- pasz_zwr1+ paszprz01_zamykaj
          pos_towarowy_1przelotsem paszki_towarowy_stat_3 pos_next endevent
 event pos_towarowy_1przelotsem multiple 12 none pasz_f_Sr3 pasz_tof_Od2 pasz_f1_Sp4
          pasz_b_Sr2 pasz_tob_Od2 endevent
event pos_towarowy_2 multiple 0 paszki_towarowy_stat pos_towarowy_2a else pos_next condition
          memcompare c 1 * endevent
 event pos_towarowy_2a multiple 0 paszki_osobowy_stat pos_next else pos_towarowy_2b condition
          memcompare * 0 * endevent
 event pos_towarowy_2b multiple 0 paszki_osobowy_stat pos_next else pos_towarowy_2zwr
          pos_towarowy_2sem paszki_towarowy_stat_2 pos_next condition memcompare a 1 *
          endevent
 event pos_towarowy_2zwr multiple 0 none pasz_zwr1+ paszprz01_zamykaj endevent
 event pos_towarowy_2sem multiple 12 none pasz_b_Sr2 endevent
```

Aby mieć przelot, na samym wjeździe musimy wykonać test, czy osobowy już przyjechał i wjechał w peron. Jeżeli tak, to uruchamiamy event, który ustawi przelot (i flagę pociągu towarowego na *a 3 **), jeżeli nie, to tylko dajemy wjazd na stację.

To jeszcze nie wszystko, musimy zmienić obsługę pociągu osobowego tak, aby nie dostał wyjazdu, gdy towarowy będzie realizował przelot:

```
event pos_osobowy_2 multiple 0 paszki_osobowy_stat pos_osobowy_2a else pos_next condition
          memcompare c 1 * endevent
event pos_osobowy_2a multiple 0 paszki_towarowy_stat pos_next else pos_osobowy_2b condition
          memcompare * 0 * endevent
event pos_osobowy_2b multiple 0 paszki_towarowy_stat pos_next else pos_osobowy_2c condition
          memcompare a 1 * endevent
event pos_osobowy_2c multiple 0 paszki_towarowy_stat pos_next else paszki_osobowy_stat_2
          pasz_zwr3+ pos_osobowy_2sem pos_next condition memcompare a 3 * endevent
event pos_osobowy_2sem multiple 5 none pasz_e_Sr2 endevent
```

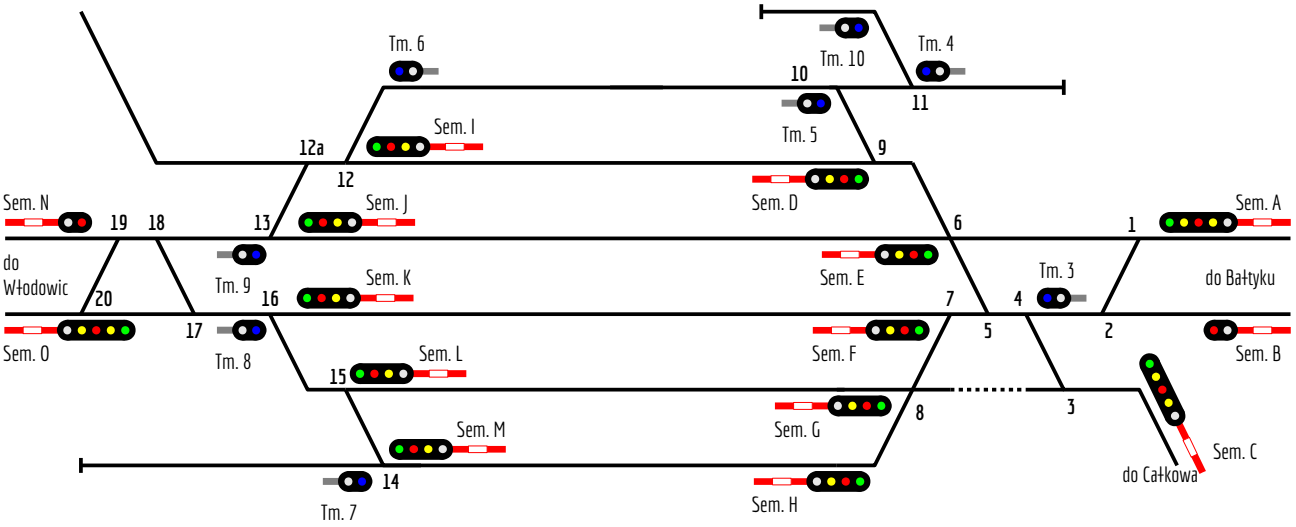
Dopisaliśmy jeszcze jeden test – wyjazdu nie będzie, kiedy towarowy będzie miał flagę *a 3 **, czyli będzie w trakcie robienia przelotu.

Dopisanie tego warunku było bardzo proste, żadnych skomplikowanych warunków, dodatkowych komórek pamięci, po prostu dopisaliśmy jeszcze jeden event testowy. To bardzo użyteczna cecha algorytmu OS: w razie konieczności wstawienie jeszcze jednego pociągu na stację nie zburzy nam całkowicie wcześniej wypracowanej koncepcji, po prostu dopiszemy kolejny warunek do listy.

6.1.4. Przykład 3 – Macierzewo

Był prosty przykład, pora na coś bardziej skomplikowanego. Najpierw sobie pomarzmy, jak mógłby wyglądać początek misji *calkowo_cargo* w Macierzewie:

- Możemy zaczynać misję pod tarczą manewrową Tm7, następnie pojechać pod semafor H i tam dostać Ms2, zawrócić za tarczą Tm3, wjechać pod semafor D gdzie czekają na nas wagony towarowe, podczepić się, po czym wyjechać do Jarkawek,
- W międzyczasie może przejeżdżać od semafora O w stronę Bałtyku pociąg towarowy,
- Również w międzyczasie może przejechać pociąg pośpieszny od semafora A, pod semaforem J, i do Włodowic.
- Oczywiście fajnie by było, gdyby te pociągi przyjeżdżały o losowym czasie, tak aby nam losowo przeszkodziły w manewrach, albo w wyjeździe. A może by nam się poszczęściło, i jakoś byśmy się między nimi prześlizgnęli.
  Brzmi ambitnie? Jeżeli zastosujemy algorytm OS to nic trudnego.



Układ torowy stacji Macierzewo.

Zaczynamy od napisania flag pociągów, tak aby można było spełnić podane założenia:

```
node -1 0 mac_osobowy_stat memcell 1.0 1.0 1.0 a 0 * none endmemcell
event mac_osobowy_stat_b updatevalues 0 mac_osobowy_stat b * * endevent
event mac_osobowy_stat_c updatevalues 0 mac_osobowy_stat c * * endevent
event mac_osobowy_stat_d updatevalues 0 mac_osobowy_stat d * * endevent
event mac_osobowy_stat_1 updatevalues 0 mac_osobowy_stat a 1 * endevent
event mac_osobowy_stat_2 updatevalues 0 mac_osobowy_stat a 2 * endevent

node -1 0 mac_kontener_stat memcell 1.0 1.0 1.0 a 0 * none endmemcell
event mac_kontener_stat_c updatevalues 0 mac_kontener_stat c * * endevent
event mac_kontener_stat_d updatevalues 0 mac_kontener_stat d * * endevent
event mac_kontener_stat_1 updatevalues 0 mac_kontener_stat a 1 * endevent
event mac_kontener_stat_2 updatevalues 0 mac_kontener_stat a 2 * endevent
event mac_kontener_stat_3 updatevalues 0 mac_kontener_stat a 3 * endevent
```

```
node -1 0 mac_cargo_stat memcell 1.0 1.0 1.0 a 0 * none endmemcell
event mac_cargo_stat_b updatevalues 0 mac_cargo_stat b * * endevent
event mac_cargo_stat_c updatevalues 0 mac_cargo_stat c * * endevent
event mac_cargo_stat_d updatevalues 0 mac_cargo_stat d * * endevent
event mac_cargo_stat_1 updatevalues 0 mac_cargo_stat a 1 * endevent
event mac_cargo_stat_2 updatevalues 0 mac_cargo_stat a 2 * endevent
event mac_cargo_stat_3 updatevalues 0 mac_cargo_stat a 3 * endevent
event mac_cargo_stat_4 updatevalues 0 mac_cargo_stat a 4 * endevent
```

mac_osobowy_stat to komórka zawierająca flagę pociągu osobowego z Bałtyku do Włodowic, pociąg zatrzymuje się w peronach. *mac_kontener_stat* oznacza pociąg towarowy przejeżdżający z Włodowic do Bałtyku, z opcją przelotu. *mac_cargo_stat* to komórka z flagą naszej lokomotywy. Teraz rozpisujemy sobie tabelki:

Osobowy

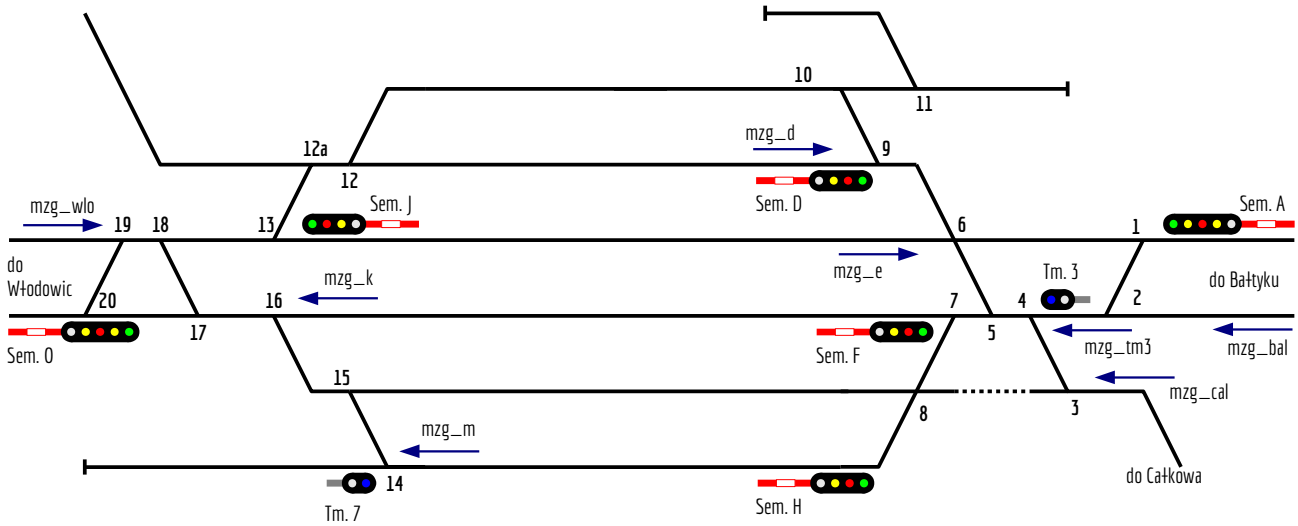
0 – 1	Od semafora A (wjazd od Bałtyku) pod semafor J
1 – 2	Od semafora J – wyjazd do Włodowic

Kontener

0 – 1	Od semafora O (wjazd od Włodowic) pod semafor F
1 – 2	Od semafora F – wyjazd do Bałtyku
0 – 3	Od semafora O, przelot pod semaforem F, wyjazd do Bałtyku

Cargo

0 – 1	Postój pod Tm7, przejazd pod semafor H
1 – 2	Od semafora H pod Tm3
2 – 3	Od Tm3 pod semafor D, przyczepienie wagonów
3 – 4	Wyjazd spod semafora D do Jarkawek



Układ torowy stacji Macierzewo – niebieskimi strzałkami zaznaczono eventy zgłaszające.

Kolejny etap to eventy zgłaszające:

```
event none3958:event1 multiple 0 none mzg_a endevent
event none3984:event1 multiple 0 none mzg_o endevent
event none_null_074:event2 multiple 0 none mzg_d endevent
event none_null_072:event2 multiple 0 none mzg_e endevent
event none3968:event2 multiple 0 none mzg_k endevent
event none3964:event2 multiple 0 none mzg_m endevent
event none3998:event2 multiple 0 none mzg_tm3 endevent
event none4002:event2 multiple 0 none mzg_bal endevent
event none3955:event2 multiple 0 none mzg_cal endevent
```

```
event none_null_069:event2 multiple 0 none mzg_wlo endevent
```

Po przypisaniu eventów do torów wypisujemy eventy zgłaszające, pomagając sobie w razie potrzeby tabelkami:

```
//Wjazd pod semafor wjazdowy od strony Bałtyku – zbliża się osobowy
event mzg_a multiple 0 mac_osobowy_stat mac_osobowy_stat_c mac_obsluga_stacji condition
          memcompare a 0 * endevent
//Wjazd za semafor D – nasza lokomotywa podjechała podpiąć wagony
event mzg_d multiple 5.5 mac_zwr9 mzg_d1 else mzg_d condition trackfree endevent
 event mzg_d1 multiple 0 mac_cargo_stat mac_cargo_stat_b mac_obsluga_stacji mzg_d1x condition
          memcompare a 3 * endevent
 event mzg_d1x multiple 250 none mac_cargo_stat_c mac_obsluga_stacji endevent
//Wjazd za semafor E – osobowy zatrzymuje się w peronie
event mzg_e multiple 5.5 mac_zwr6_d mzg_e1 else mzg_e condition trackfree endevent
 event mzg_e1 multiple 0 mac_osobowy_stat mac_osobowy_stat_b mac_obsluga_stacji mzg_e1x
          condition memcompare a 1 * endevent
 event mzg_e1x multiple 60 none mac_osobowy_stat_c mac_obsluga_stacji endevent
//Wjazd za semafor K – kontener
event mzg_k multiple 5.5 mac_zwr16 mzg_k1 else mzg_k condition trackfree endevent
 event mzg_k1 multiple 0 mac_kontener_stat mac_kontener_stat_c mac_osluga_stacji condition
          memcompare a 1 * endevent
//Wjazd za semafor M – nasza lokomotywa manewruje
event mzg_m multiple 5.5 mac_zwr14 mzg_m1 else mzg_m condition trackfree endevent
 event mzg_m1 multiple 0 mac_cargo_stat mac_cargo_stat_c mac_obsluga_stacji condition
          memcompare a 1 * endevent
//Wjazd za Tm3 – nasza lokomotywa manewruje, następuje zmiana kabiny
event mzg_tm3 multiple 5.5 mac_zwr4 mzg_tm3_1 else mzg_tm3 condition trackfree endevent
 event mzg_tm3_1 multiple 0 mac_cargo_stat mac_cargo_stat_c mac_obsluga_stacji condition
          memcompare a 2 * endevent
//Wyjazd do Bałtyku – kontener wyjeżdża po S2 na semaforze F lub po przelocie
event mzg_bal multiple 5.5 mac_zwr2 mzg_bal1 mzg_bal2 else mzg_bal condition trackfree
          endevent
 event mzg_bal1 multiple 0 mac_kontener_stat mac_kontener_stat_d mac_obsluga_stacji condition
          memcompare a 2 * endevent
 event mzg_bal2 multiple 0 mac_kontener_stat mac_kontener_stat_d mac_obsluga_stacji condition
          memcompare a 3 * endevent
//Wyjazd do Jarkawek – nasz pociąg
event mzg_cal multiple 5.5 mac_zwr3 mzg_cal1 else mzg_cal condition trackfree endevent
 event mzg_cal1 multiple 0 mac_cargo_stat mac_cargo_stat_d mac_obsluga_stacji condition
          memcompare a 4 * endevent
//Wyjazd do Włodowic – wyjazd osobowego ze stacji
event mzg_wlo multiple 5.5 mac_zwr19 mzg_wlo1 else mzg_wlo condition trackfree endevent
 event mzg_wlo1 multiple 5 mac_osobowy_stat mac_osobowy_stat_d mac_obsluga_stacji condition
          memcompare a 2 * endevent
```

Flagi *b* użyliśmy w dwóch sytuacjach:

- Po wjeździe osobowego w peron przed semafor J, pociąg ma 60 sekund na wymianę pasażerów,
- Po wjeździe naszej lokomotywy cargo za semafor D, na podpięcie i próbę hamulca mamy 250 sekund.

Wszystkie pociągi na wyjeździe ze stacji otrzymują flagę *d*. Jak spojrzymy do eventu obsługującego stację, rozpatrywane są tylko pociągi mające flagę *c*. Zatem ta dodatkowa flaga ma służyć do szybkiej identyfikacji pociągu, który już przejechał przez stację, jak również do zmniejszenia ilości przebiegów w pętli obsługi stacji.

A teraz uwaga: nasza lokomotywa wykonuje manewry, czyli musimy wpisać komendy sterujące AI, które umożliwią zawrócenie pod Tm3, podpięcie pod pociąg, oraz przypisanie rozkładu jazdy. Najpierw sobie zdefiniujemy eventy, które nam to zrobią:

```
event mac_cargo_zawroc putvalues 2 none 1.0 1.0 1.0 Change_direction 0 0 endevent
event mac_cargo_podepnij putvalues 2 none 1.0 1.0 1.0 Shunt -3 -3 endevent
event mac_cargo_rozklad putvalues 210 none 1.0 1.0 1.0 Timetable:TMS886453 0.1 0 endevent
```

I kolejna zaleta algorytmu OS: komendy dla AI (czyli te, które wykonuje maszynista danego pociągu) wpisujemy tam, gdzie jest obsługa ze strony maszynisty. Wszystkie eventy zgłaszające będą na pewno uruchomione przez pociąg, któremu zmieniamy flagę. Możemy uzupełnić 2 eventy zgłaszające:

```
//Wjazd za semafor D – nasza lokomotywa podjechała podpiąć wagony
event mzg_d multiple 5.5 mac_zwr9 mzg_d1 else mzg_d condition trackfree endevent
 event mzg_d1 multiple 0 mac_cargo_stat mac_cargo_stat_b mac_obsluga_stacji mzg_d1x
```

```
          mac_cargo_podepnij mac_cargo_rozklad condition memcompare a 3 * endevent
 event mzg_d1x multiple 250 none mac_cargo_stat_c mac_obsluga_stacji endevent
//Wjazd za Tm3 - nasza lokomotywa manewruje, następuje zmiana kabiny
event mzg_tm3 multiple 5.5 mac_zwr4 mzg_tm3_1 else mzg_tm3 condition trackfree endevent
 event mzg_tm3_1 multiple 0 mac_cargo_stat mac_cargo_stat_c mac_obsluga_stacji
          mac_cargo_zawroc condition memcompare a 2 * endevent
```

Teraz piszemy obsługę stacji. Główna pętla obsługi wygląda niemal identycznie jak ta w Paszkach Wielkich:

```
node -1 0 mos memcell 1.0 1.0 1.0 * 0 0 none endmemcell
event mos_0 updatevalues 0 mos * 0 0 endevent
event mos_1 updatevalues 0 mos * 1 1 endevent

event mac_obsluga_stacji multiple 5.5 mos mos_next mos_1 else mac_obsluga_stacji condition
          memcompare * 0 0 endevent

node -1 0 mac_obsluga_stacji_stat memcell 1.0 1.0 1.0 * 0 0 none endmemcell
event mac_obsluga_stacji_stat_up addvalues 0 mac_obsluga_stacji_stat * 1 1 endevent
event mac_obsluga_stacji_stat_0 updatevalues 0 mac_obsluga_stacji_stat * 0 0 endevent

event mos_next multiple 0 mac_obsluga_stacji_stat mac_obsluga_stacji_osobowy
          mac_obsluga_stacji_stat_up else mos_next1 condition memcompare * 0 0 endevent
event mos_next1 multiple 0 mac_obsluga_stacji_stat mac_obsluga_stacji_kontener
          mac_obsluga_stacji_stat_up else mos_next2 condition memcompare * 1 1 endevent
event mos_next2 multiple 0 mac_obsluga_stacji_stat mac_obsluga_stacji_cargo
          mac_obsluga_stacji_stat_up else mac_obsluga_stacji_stat_0 mos_0 condition
          memcompare * 2 2 endevent
```

I znowu ważna cecha algorytmu OS: bardzo łatwo możemy ustawić priorytety dla poszczególnych pociągów. Ten, który jest wywoływany jako pierwszy w pętli obsługi stacji ma najwyższy priorytet, ten na końcu – najniższy. W tym przykładzie najwyższy priorytet ma osobówka, potem kontener, a na końcu cargo (czyli nasza lokomotywa). Nie będzie wprawdzie tutaj aż tak bardzo widać tych priorytetów ale zdarzają się stacje i sytuacje, gdzie ma to znaczenie.

Pisanie eventów obsługi dla poszczególnych pociągów rozpoczniemy od naszej lokomotywy:

```
event mac_obsluga_stacji_cargo multiple 0 mac_cargo_stat mos_cargo_1 else mos_next condition
          memcompare c * * endevent
event mos_cargo_1 multiple 0 mac_cargo_stat mos_cargo_1a else mos_cargo_2 condition memcompare
          c 0 * endevent
 event mos_cargo_1a multiple 0 none mac_zwr14+ mos_cargo_1sem mac_cargo_stat_1 mos_next
          endevent
 event mos_cargo_1sem multiple 4 none mac_tm7_m40 endevent
event mos_cargo_2 multiple 0 mac_cargo_stat mos_cargo_2a else mos_cargo_3 condition memcompare
          c 1 * endevent
 event mos_cargo_2a multiple 0 mac_kontener_stat mos_next else mos_cargo_2b condition
          memcompare a 2 * endevent
 event mos_cargo_2b multiple 0 mac_kontener_stat mos_next else mos_cargo_2zwr mos_cargo_2sem
          mac_cargo_stat_2 mos_next condition memcompare a 3 * endevent
 event mos_cargo_2zwr multiple 0 none mac_zwr8bd mac_zwr7- mac_zwr5+ mac_zwr4+ endevent
 event mos_cargo_2sem multiple 5 none mac_h_m40 endevent
event mos_cargo_3 multiple 0 mac_cargo_stat mos_cargo_3a else mos_cargo4 condition memcompare
          c 2 * endevent
 event mos_cargo_3a multiple 0 mac_osobowy_stat mos_next else mos_cargo_3zwr mos_cargo_3sem
          mac_cargo_stat_3 mos_next condition memcompare a 1 * endevent
 event mos_cargo_3zwr multiple 0 none mac_zwr4+ mac_zwr5- mac_zwr6ac mac_zwr9+ endevent
 event mos_cargo_3sem multiple 4 none mac_tm3_ms2 endevent
event mos_cargo_4 multiple 0 mac_cargo_stat mos_cargo_4a else mos_next condition memcompare c
          3 * endevent
 event mos_cargo_4a multiple 0 mac_osobowy_stat mos_next else mos_cargo_4b condition
          memcompare a 1 * endevent
 event mos_cargo_4b multiple 0 mac_kontener_stat mos_next else mos_cargo_4c condition
          memcompare a 2 * endevent
 event mos_cargo_4c multiple 0 mac_kontener_stat mos_next else mos_cargo_4zwr mos_cargo_4sem
          mac_cargo_stat_4 mos_next condition memcompare a 3 * endevent
 event mos_cargo_4zwr multiple 0 none mac_zwr9+ mac_zwr6ac mac_zwr5- mac_zwr4- mac_zwr3-
          macprz2_zamykaj endevent
 event mos_cargo_4sem multiple 13 none mac_d_S10 endevent
```

Warto przeanalizować wszystkie zapisane warunki:

- Przy wyjeździe spod tarczy Tm7: żaden inny pociąg nie przejedzie tamtym torem, więc wyjazd następuje bez warunków – lokomotywa ustawi flagę *c* i wywoła obsługę stacji – zawsze otrzyma białe światło (m40 oznacza zgodę na manewry z prędkością 40 km/h – można tak, ponieważ lokomotywa dostaje wjazd na wolny tor),
- Przed wyjazdem za semafor H pojawiają się warunki: kontener nie może wyjeżdżać ze stacji (flaga *a 2 **), nie może również mieć przelotu (*a 3 **),
- Po dojechaniu za tarczę Tm3 wjazd pod semafor D może być podany, pod warunkiem, że na stację nie wjeżdża osobowy (flaga *a 1 **),
- Wyjazd do Jarkawek może być podany pod warunkiem, że osobowy nie wjeżdża (flaga *a 1 **), kontener nie wyjeżdża (flaga *a 2 **), oraz nie ma przelotu (flaga *a 3 **).

Dalej piszemy warunki dla pociągu osobowego – tutaj jedyny ruch, przy którym może wystąpić kolizja z naszą lokomotywą, to wjazd na stację. Nie może być podany, jeżeli przejeżdżamy spod Tm3 pod semafor D, oraz jeśli wyjeżdżamy ze stacji (czyli kiedy nasza lokomotywa ma flagi *a 3 ** oraz *a 4 **):

```
event mac_obsluga_stacji_osobowy multiple 0 mac_osobowy_stat mos_osobowy_1 else mos_next
          condition memcompare c * * endevent
event mos_osobowy_1 multiple 0 mac_osobowy_stat mos_osobowy_1a else mos_osobowy_2 condition
          memcompare c 0 * endevent
 event mos_osobowy_1a multiple 0 mac_cargo_stat mos_next else mos_osobowy_1b condition
          memcompare a 3 * endevent
 event mos_osobowy_1b multiple 0 mac_cargo_stat mos_next else mos_osobowy_1zwr
          mos_osobowy_1sem mac_osobowy_stat_1 mos_next condition memcompare a 4 * endevent
 event mos_osobowy_1zwr multiple 0 none mac_zwr1+ mac_zwr6bd endevent
 event mos_osobowy_1sem multiple 5 none mac_a_S5 endevent
event mos_osobowy_2 multiple 0 mac_osobowy_stat mos_osobowy_2a else mos_next condition
          memcompare c 1 * endevent
 event mos_osobowy_2a none mos_osobowy_2zwr mos_osobowy_2sem mac_osobowy_stat_2 mos_next
          endevent
 event mos_osobowy_2zwr multiple 0 none mac_zwr13+ mac_zwr18+ mac_zwr19+ macprze1_zamykaj
          endevent
 event mos_osobowy_2sem multiple 12 none mac_j_S2 endevent
```

I na koniec warunki dla kontenerowca – kontener nie wyjedzie ze stacji, jeżeli przejeżdżamy spod semafora H pod tarczę Tm3 (flaga *a 2 **), gdy stoimy pod tarczą Tm3 (flaga *c 2 **), przejeżdżamy spod Tm3 pod semafor D (flaga *a 3 **), oraz kiedy wyjeżdżamy do Jarkawek (flaga *a 4 **):

```
event mac_obsluga_stacji_kontener multiple 0 mac_kontener_stat mos_kontener_1 else mos_next
          condition memcompare c * * endevent
event mos_kontener_1 multiple 0 mac_kontener_stat mos_kontener_1a else mos_kontener_2
          condition memcompare c 0 * endevent
 event mos_kontener_1a multiple 0 mac_cargo_stat mos_kontener_1wjazd else mos_kontener_1b
          condition memcompare * 2 * endevent
 event mos_kontener_1b multiple 0 mac_cargo_stat mos_kontener_1wjazd else mos_kontener_1c
          condition memcompare a 3 * endevent
 event mos_kontener_1c multiple 0 mac_cargo_stat mos_kontener_1wjazd else
          mos_kontener_1przelot condition memcompare a 4 * endevent
 event mos_kontener_1wjazd multiple 0 none mac_zwr20+ mac_zwr17+ mac_zwr16+ macprze1_zamykaj
          mos_kontener_1wjazdsem mac_kontener_stat_1 mos_next endevent
  event mos_kontener_1wjazdsem multiple 12 none mac_o_S5 endevent
 event mos_kontener_1przelot multiple 0 none mos_kontener_1wjazd mac_zwr7+ mac_zwr5+ mac_zwr4+
          mac_zwr2+ mos_kontener_1przelotsem mac_kontener_stat_3 mos_next endevent
  event mos_kontener_1przelotsem multiple 14 none mac_o_S2 mac_f_S2 endevent
event mos_kontener_2 multiple 0 mac_kontener_stat mos_kontener_2a else mos_next condition
          memcompare a 1 * endevent
 event mos_kontener_2a multiple 0 mac_cargo_stat mos_next else mos_kontener_2b condition
          memcompare * 2 * endevent
 event mos_kontener_2b multiple 0 mac_cargo_stat mos_next else mos_kontener_2c condition
          memcompare a 3 * endevent
 event mos_kontener_2c multiple 0 mac_cargo_stat mos_next else mos_kontener_2zwr
          mos_kontener_2sem mac_kontener_stat_2 mos_next condition memcompare a 4 * endevent
 event mos_kontener_2zwr multiple 0 none mac_zwr7+ mac_zwr5+ mac_zwr4+ mac_zwr2+ endevent
 event mos_kontener_2sem multiple 5 none mac_f_S2 endevent
```

I to już właściwie wszystko – to co sobie wymarzyliśmy, jest realizowane w powyższym kodzie. Jak wrażenia? Pomimo, że przejazd jest znacznie bardziej skomplikowany niż w Paszkach, to formuła sterowania jest identyczna, doszło tylko kilka wierszy kodu.

W Macierzewie są 2 przejazdy, dopisanie ich obsługi pozostawiam czytelnikowi. Skupimy się na jeszcze jednej funkcjonalności – nasz pociąg nie powinien dostać wyjazdu przed godziną zapisaną w rozkładzie. Najpierw napiszemy sobie dodatkową komórkę i eventlauncher:

```
node -1 0 mac_cargo_godzodjazdu memcell 1.0 1.0 1.0 * 0 0 none endmemcell
event mac_cargo_godzodjazdu_up updatevalues 0 mac_cargo_godzodjazdu * 1 1 endevent

node -1 0 mac_cargo_odjazd eventlauncher 1.0 1.0 1.0 -1 none 1210 mac_cargo_odblokuj_wyjazd
          none end
event mac_cargo_odblokuj_wyjazd multiple 0 none mac_cargo_godzodjazdu_up mac_obsluga_stacji
          endevent
```

O godzinie odjazdu (12:10) eventlauncher uruchomi zdarzenie, które zmieni wartość komórki pomocniczej, oraz wywoła obsługę stacji. Teraz uzupełnimy eventy obsługujące nasz pociąg:

```
event mos_cargo_4 multiple 0 mac_cargo_stat mos_cargo_4a else mos_next condition memcompare c
          3 * endevent
 event mos_cargo_4a multiple 0 mac_osobowy_stat mos_next else mos_cargo_4b condition
          memcompare a 1 * endevent
 event mos_cargo_4b multiple 0 mac_kontener_stat mos_next else mos_cargo_4c condition
          memcompare a 2 * endevent
 event mos_cargo_4c multiple 0 mac_kontener_stat mos_next else mos_cargo_4d condition
          memcompare a 3 * endevent
 event mos_cargo_4d multiple 0 mac_cargo_godzodjazdu mos_next else mos_cargo_4zwr
          mos_cargo_4sem mac_cargo_stat_4 mos_next condition memcompare * 0 0 endevent
 event mos_cargo_4zwr multiple 0 none mac_zwr9+ mac_zwr6ac mac_zwr5- mac_zwr4- mac_zwr3-
          macprz2_zamykaj endevent
 event mos_cargo_4sem multiple 13 none mac_d_S10 endevent
```

W ten sposób nie dostaniemy wyjazdu wcześniej jak o 12:10, nawet jeśli uda nam się ukończyć manewry wcześniej.

6.1.5. OS – pytania i odpowiedzi

Do czego służy nieużywana wartość w komórce zawierającej flagę?

Ta wartość jest zostawiona do wykorzystania przez użytkownika. W scenariuszu *calkowo_lotos*, podczas manewrów na stacji Wiliś, ostatnia wartość jest używana do określenia priorytetu pociągu. Można również tę wartość wykorzystać w inny sposób, np.: jeżeli lokomotywa manewrująca po stacji kilkanaście razy podjeżdża w jeden punkt, to pociąg przejeżdżający przez ten punkt nie musi sprawdzać kilkunastu warunków, można we flagach lokomotywy manewrującej przejazd przez ten punkt oznaczyć jakąś wartością, co nam ograniczy test do jednego warunku.

Czy jest możliwa dynamiczna zmiana priorytetu?

Jest to możliwe, zrobiono to w wyżej wspomnianym scenariuszu *calkowo_lotos*, na stacji Wiliś. Nie jest to zbyt przemyślny mechanizm, po prostu wykorzystano nieużywaną wartość z flag, oraz dodano kilka wywołań obsługi pociągów w pętli obsługi, niektóre zależne od ostatniej wartości w komórce flagi.

Czy jest możliwe używanie klawisza „w” do zgłaszania pociągu?

Jest to nawet bardzo proste: wystarczy napisać eventlaunchera, który wywoła event zmieniający dla naszego pociągu flagę na *c*, pod warunkiem, że aktualnie mamy flagę *b*.

Czy jest możliwe zastosowanie wspólnych torów dla wielu scenariuszy opartych na algorytmie OS?

Czy jest możliwe? Nie dość że jest to łatwe, to jeszcze nam zmniejszy ilość pracy. Jedynymi eventami w układzie torowym są eventy zgłaszające pociągi.

Czy zamiast eventów zgłaszających da się użyć odcinków izolowanych?

Na pewno jest to możliwe, zostało to użyte w scenariuszu w metrze bałtyckim. Jednakże z uwagi na mały stopień skomplikowania stacji metra, nie ma jeszcze pełnego rozwiązania tego zagadnienia.

Czy jest możliwe pisanie przebiegów alternatywnych?

Jest możliwe i często pojawia się w całkowskich scenariuszach.

Czy możliwe są zdarzenia losowe?

Ten algorytm powstał przede wszystkim po to, aby można było wstawiać zdarzenia losowe, przy czym

głównie takim zdarzeniem jest losowe opóźnienie przyjeżdżających na stacje pociągów – aczkolwiek to wystarczająco dużo, aby zrobić scenariusz praktycznie nieprzewidywalnym.

Czy diagnostyka takiego scenariusza jest łatwa?

Diagnostyka scenariuszy opracowanych na algorytmie OS jest opisana w rozdziale 7.5. Diagnostyka nie jest trudna, pod warunkiem, że będziemy przeglądając *log.txt* używać kombinacji Ctrl+F do wyszukania, czy uruchomione zostały odpowiednie eventy.

Czy jest możliwe napisanie takiej obsługi stacji, aby zamiast położeń innych pociągów była sprawdzana zajętość torów?

Oczywiście że coś takiego jest możliwe, próba napisania czegoś takiego była w nieopublikowanym (w czasie pisania tego dokumentu, sierpień 2017 r.) metrze bałtyckim. Z uwagi na mały stopień skomplikowania stacji metra nie ma jeszcze pełnego rozwiązania tego zagadnienia.

Struktura OS jest bardzo jednorodna. Czy istnieje jakiś edytor pozwalający na automatyzację pisania OS?

Niewątpliwie obsługę stacji w tym algorytmie – niezależnie od tego czy jest to zwykła mijanka czy potężna stacja z manewrami – pisze się tak samo, a to podstawa do stworzenia automatyzacji. Niemniej jednak do tej pory nikt za taki program / edytor się nie wziął. Ale kto wie... polecam śledzić wątki na forum symulatora.

6.2. Zaawansowany radiotelefon

W prostych scenariuszach istniejący mechanizm obsługi radiotelefonu jest całkowicie wystarczający. Jeśli jednak mamy bardziej skomplikowany scenariusz, w którym dostępnych jest kilka misji, to sprawa się komplikuje. Istnieją scenariusze, w których jest więcej niż jedna misja – Bałtyk, Krzyżowa, Quarkmce2007 – ale misje te są ubogie w rozmowy radiotelefoniczne. Możemy napotkać na następujące problemy:

- Rozmowy radiotelefoniczne są słyszalne w promieniu nawet kilkunastu kilometrów od stacji, choć im dalej tym ciszej. Może się zdarzyć sytuacja, że przejeżdżamy przez jakąś stację, a w tle słyszymy rozmowy na drugiej stacji,
- Im więcej rozmów, tym większe ryzyko że się naniosą, a jest to sytuacja niedopuszczalna.

Poniżej prezentuję propozycję obsługi radiotelefonu. Pierwsza rzecz, to zdefiniowanie punktów dla poszczególnych stacji. Zróbmy przykład dla scenariusza *calkowo_cargo*: będą 4 radiostacje, w Macierzewie, Paszkach, Całkowie, i w Wilisiu. Definicja radiostacji dla Macierzewa będzie wyglądała następująco:

```
node -1 0 radiodetektor_mac eventlauncher -8234.7 6.0 -15145.8 5000 none -60 radio_macierzewo
          radio_macierzewo end

node -1 0 radio_mac memcell 1.0 1.0 1.0 * 0 0 none endmemcell
event radio_mac_0 updatevalues 0 radio_mac * 0 * endevent
event radio_mac_1 updatevalues 0 radio_mac * 1 * endevent
event radio_mac_blokuj updatevalues 0 radio_mac * * 1 endevent
event radio_macierzewo multiple 0 none radio_mac_1 radio_pasz_0 radio_cal_0 radio_wil_0
          endevent
```

Eventlauncher wykrywający znalezienie się w rejonie stacji Macierzewo uruchamiany jest co 60 sekund. W momencie znalezienia się w zasięgu eventlaunchera załączana jest radiostacja w Macierzewie, i wyłączane są radiostacje na innych stacjach (wywołanie eventów *radio_mac_1*, oraz *radio_pasz_0*, *radio_cal_0*, *radio_wil_0*). Analogicznie wygląda załączanie radiostacji dla innych miejscowości.

Rozwijamy dalej zrobioną w rozdziale 6.1.4. obsługę stacji w Macierzewie – dodamy 3 rozmowy:

- W momencie kiedy dojeżdżający do stacji pociąg osobowy zgłosi się do dyżurnego, odtworzone zostanie nagranie *calkowo_tartak2_g23.wav* (*„Dzień dobry 17005!”*),
- Jeżeli dyżurny nie będzie mógł podać wjazdu, to da informację *„Wjazd podam jak skończą się manewry”* (*calkowo_tartak2_g26.wav*),
- Kiedy dyżurny poda wjazd, poinformuje przez radio: *„Wjazd podany”* (*calkowo_tartak2_g24.wav*).

Na początek odtworzymy wytypowane pliki *.wav* i zapiszemy długość każdego z nich – kolejno 6 sekund, 5 sekund, oraz 7 sekund.

```
node -1 0 mac_radmor1 sound -8234.7 6.0 -15145.8 calkowo_tartak2_g23.wav endsound event
          mac_radmor1w sound 0 mac_radmor1 1 endevent
 event mac_radmor1 multiple 0 radio_mac mac_radmor1x condition memcompare * 1 * endevent
 event mac_radmor1x multiple 1 radio_mac radio_mac_blokuj mac_radmor1w mac_radmor1r else
          mac_radmor1y condition memcompare * * 0 endevent
```

```
 event mac_radmor1y multiple 1 radio_mac radio_mac_blokuj mac_radmor1w mac_radmor1r else
          mac_radmor1x condition memcompare * * 0 endevent
 event mac_radmor1r updatevalues 6 radio_mac * * 0 endevent

node -1 0 mac_radmor2 sound -8234.7 6.0 -15145.8 calkowo_tartak2_g26.wav endsound event
          mac_radmor2w sound 0 mac_radmor2 1 endevent
 event mac_radmor2 multiple 0 radio_mac mac_radmor2xtest condition memcompare * 1 * endevent
 event mac_radmor2xtest multiple 0 mac_radmor2test mac_radmor2x mac_radmor2test_1 condition
          memcompare * 0 0 endevent
  node -1 0 mac_radmor2test memcell 1.0 1.0 1.0 * 0 0 none endmemcell
  event mac_radmor2test_1 updatevalues 0 mac_radmor2test * 1 1 endevent
 event mac_radmor2x multiple 1 radio_mac radio_mac_blokuj mac_radmor2w mac_radmor2r else
          mac_radmor2y condition memcompare * * 0 endevent
 event mac_radmor2y multiple 1 radio_mac radio_mac_blokuj mac_radmor2w mac_radmor2r else
          mac_radmor2x condition memcompare * * 0 endevent
 event mac_radmor2r updatevalues 5 radio_mac * * 0 endevent

node -1 0 mac_radmor3 sound -8234.7 6.0 -15145.8 calkowo_tartak2_g24.wav endsound event
          mac_radmor3w sound 0 mac_radmor3 1 endevent
 event mac_radmor3 multiple 0 radio_mac mac_radmor3x condition memcompare * 1 * endevent
 event mac_radmor3x multiple 1 radio_mac radio_mac_blokuj mac_radmor3w mac_radmor3r else
          mac_radmor3y condition memcompare * * 0 endevent
 event mac_radmor3y multiple 1 radio_mac radio_mac_blokuj mac_radmor3w mac_radmor3r else
          mac_radmor3x condition memcompare * * 0 endevent
 event mac_radmor3r updatevalues 7 radio_mac * * 0 endevent
```

Zasada działania radia jest następująca:

- Po otrzymaniu ze scenariusza polecenia odtworzenia dźwięku sprawdzana jest dostępność kanału, czy nie jest w danym momencie odtwarzany inny dźwięk, jeżeli tak, to poprzez event rekurencyjny następuje czekanie na zwolnienie kanału,
- Gdy kanał zostanie zwolniony, zakładana jest blokada na kanał, odtwarzane jest nagranie, a po interwale czasowym równym długości nagrania, blokada jest zdejmowana,

Pogrubioną czcionką zaznaczono miejsce wpisania nazwy pliku dźwiękowego, oraz długości jego trwania. Pierwsze i trzecie nagranie różni się od drugiego – dodatkowo występuje warunek, że nagranie *mac_radmor2* możemy odtworzyć tylko raz.

Teraz pora uzupełnić event zgłaszający pociągu osobowego. W momencie gdy pociąg zmieni swoją flagę na *c * **, odtworzone zostanie nagranie *mac_radmor1*:

```
//Wjazd pod semafor wjazdowy od strony Bałtyku - zbliża się osobowy
event mzg_a multiple 0 mac_osobowy_stat mac_osobowy_stat_c mac_radmor1 mac_obsluga_stacji
          condition memcompare a 0 * endevent
```

Wstawiamy pozostałe nagrania:

```
event mac_obsluga_stacji_osobowy multiple 0 mac_osobowy_stat mos_osobowy_1 else mos_next
          condition memcompare c * * endevent
event mos_osobowy_1 multiple 0 mac_osobowy_stat mos_osobowy_1a else mos_osobowy_2 condition
          memcompare c 0 * endevent
 event mos_osobowy_1a multiple 0 mac_cargo_stat mos_next mac_radmor2 else mos_osobowy_1b
          condition memcompare a 3 * endevent
 event mos_osobowy_1b multiple 0 mac_cargo_stat mos_next mac_radmor2 else mos_osobowy_1zwr
          mos_osobowy_1sem mac_osobowy_stat_1 mac_radmor3 mos_next condition memcompare a 4 *
          endevent
 event mos_osobowy_1zwr multiple 0 none mac_zwr1+ mac_zwr6bd endevent
 event mos_osobowy_1sem multiple 5 none mac_a_S5 endevent
```

Jeżeli dajemy komunikat w radiu informujący o braku możliwości podania wjazdu (tutaj *mac_radmor2*), to koniecznie musi być przygotowane zabezpieczenie przed kilkukrotnym uruchomieniem tego komunikatu.

W zasadzie to wszystko: konstrukcja umożliwia wyłączenie odtwarzania nagrań na oddalonych stacjach, automatyczne przełączanie radiostacji, zabezpieczenie przed nakładaniem się rozmów. Mechanizm ten został zastosowany w scenariuszach Odysei Spalinowej, jednakże dopiero od scenariusza nr 7.

6.3. Algorytm stosowany w Quarkmce2007

O tym algorytmie należy wspomnieć z kronikarskiego obowiązku, choć z uwagi na jego poziom skomplikowania, a przede wszystkim brak porządnej dokumentacji, algorytm zostanie omówiony bardzo skrótowo.

Quarkmce2007 był pierwszym scenariuszem wykonanym wyłącznie przy użyciu odcinków izolowanych. Zasada jego działania jest, w dość dużym uogólnieniu, bardzo podobna do algorytmu OS – występuje mechanizm odwzorowujący zgłaszanie się do stacji mechanika, oraz mechanizm obsługi stacji, który stara się jak najwierniej odtwarzać realne urządzenia SRK. Dodatkowo algorytm jest tak skonstruowany, że w dodawanie przebiegu dla kolejnych pociągów to dopisywanie raptem kilku linijek kodu. Obsługa stacji jest w pełni zautomatyzowana i przebieg ustalany sprawdzanie zajętości torów.

Zasada działania dla wszystkich stacji jest mniej więcej taka:

1. Pociąg zbliżający się do stacji zgłasza się za pomocą eventu *whois*,
2. Eventy rozpoznają pociąg, uruchamiana jest pętla eventowa, która działa tak długo, dopóki nie zostanie znaleziony wolny tor do wjazdu lub przelotu,
3. Kiedy zostanie ustalony wstępny przebieg, wywoływana jest procedura obsługi stacji: do odcinków izolowanych toru stacyjnego oraz głowicy stacji, zapisywana jest wartość ** * 1*, oznaczając w ten sposób zarezerwowanie toru,
4. Uruchomione zostają eventy, które sprawdzają możliwość ustawienia przebiegu, czy jakiś inny pociąg już wcześniej nie zarezerwował sobie przebiegu. Eventy są uruchamiane w pętli, tak aby czekać do momentu, aż uda się utwierdzić przebieg.
5. Następnie wywoływane są eventy ustawiające zwrotnice, po ich przestawieniu, przebieg zostaje utwierdzony – wartości w odcinkach izolowanych zmieniane są na ** * 2*.
6. Po utwierdzeniu przebiegu otwierane są semafory,
7. W momencie wjechania pociągu na odcinek izolowany głowicy stacyjnej zamykane są wszystkie semafory (wjazdowe i wyjazdowe) powiązane z tą głowicą – tutaj jest jeszcze inna metoda zamykania semaforów, w ogóle nie ma eventów zamykających umieszczonych w torach za semaforami.

Trudno wskazać, który algorytm daje większe możliwości – czy ten z Quarkmce2007, czy algorytm OS. Poniżej próba porównania:

- Quarkmce2007 jest zbudowany wyłącznie z wykorzystaniem odcinków izolowanych, w przeciwieństwie do Odysei Spalinowej. Z drugiej strony – istnieje możliwość zbudowania OS na odcinkach izolowanych – udało się to zrobić w metrze bałtyckim.
- W Quarkmce2007 należy zdefiniować wszystkie przebiegi stacyjne, aby móc z nich korzystać. W OS nic takiego nie jest potrzebne, z drugiej strony, trzeba się mocno nagimnastykować aby móc zrobić w OS alternatywne przebiegi pociągów.
- W Quarkmce 2007 występuje pełna automatyzacja stacji. W OS automatyzacja jest połowiczna. Z drugiej strony algorytm umożliwia przygotowanie pełnej automatyzacji – zamiast kontrolować flagi poszczególnych pociągów, można kontrolować stan torów stacyjnych.

Nasuwa się ostateczny wniosek – oba algorytmy, pomimo braku podobieństwa w kodzie, działają właściwie na bardzo podobnej zasadzie. Wykorzystanie jednego lub drugiego zależy od pomysłu i potrzeb autora scenariusza.

7. Usuwanie usterek

Do tej pory omijaliśmy ten temat szerokim łukiem ale najwyższy czas się za to wziąć. W tym rozdziale poznajemy mechanizmy i sposoby ułatwiające diagnostykę scenariuszy. Przydatne będą dla nas niezależnie od stopnia zaawansowania.

Nie załamuj się nerwowo:
Jeżeli dopiero zaczynasz przygodę z pisaniem scenariuszy to pierwszym sukcesem będzie zakończona sukcesem próba uruchomienia symulatora – choć potem się okaże, że to tylko początek.
Usuwanie błędów i usterek to nieodłączny element pisania scenariuszy, więc nie poddawaj się po pierwszych niepowodzeniach, szukaj odpowiedzi w poradniku, na forum (ale nie poprzez zadawanie pytań „Co mam zrobić? Poprowadźcie mnie za rączkę! Pliiiiiizz!”).

7.1. Pierwsze uruchomienie – errors.txt

Scenariusz jest gotowy, lub doprowadzony do jakiegoś ważnego etapu. Uruchamiamy symulator – wysyp! Uruchamiamy symulator – sukces! Nie zależnie czy mamy przypadek pierwszy czy drugi, przy tym drugim od razu kończymy symulację, w obu otwieramy w katalogu głównym symulatora pliku *errors.txt*.

Dla przykładu: napisaliśmy scenariusz z takim oto plikiem .ctr:

```
//Wyjazd z Macierzewa
event keyctrl01 multiple 0 none macierzewo_ustaw_zwr macierzewo_ustaw_sem macprz2_zamykaj
          endevent
event macierzewo_ustaw_zwr multiple 0 none mac_zwr9+ mac_zwr6ac mac_zwr5- mac_zwr4- mac_zwr3-
          endevent
event macierzewo_ustaw_sem multiple 11 none mac_d_Sz1 else mac_d_S10 condition propability 0.2
          endevent
//Przelot przez Paszki Wielkie
event keyctrl02 multiple 0 none paszki_ustaw_zwr1 paszki_ustaw_sem1 else paszki_ustaw_zwr2
          paszki_ustaw_sem2 condiiton propability 0.5 endevent
event paszki_ustaw_zwr1 multiple 0 none pasz_zwr3- pasz_zwr1+ paszprz01_zamykaj endevent
event paszki_ustaw_sem1 multiple 11 none pasz_f_sr3 pasz_tof_od2 pasz_f1_sp4 pasz_b_sr2
          pasz_tob_od2 endevent
event paszki_ustaw_zwr2 multiple 0 none pasz_zwr3+ pasz_zwr2- pasz_zwr1- paszprz01_zamykaj
          endevent
event paszki_ustaw_sem2 multiple 11 none pasz_f_sr2 pasz_tof_od2 pasz_f1_sp2 pasz_c_sr3
          pasz_tob_od2 endevent
//Przelot przez Całkowo
event keyctrl03 multiple 0 none calkowo_ustaw_zwr calkowo_ustaw_sem cal_prz1_zamykaj
          randomdleay 30 endevent
event calkowo_ustaw_zwr multiple 0 none cal_zwr1+ cal_zwr2+ cal_zwr4+ cal_zwr6+ endevent
event calkowo_ustaw_sem multiple 11 none cal_a_sr2 cal_toa_od2 cal_f_sr2 cal_tof_od2 endevent
```

Uruchamiamy symulator. Niby sukces bo symulator się włączył, ale naciskamy F10, wychodzimy, i otwieramy *errors.txt*. Może się nam wyświetlić bardzo pokaźna lista błędów i nie będzie wiadomo od czego zacząć: są zgłoszenia o błędnych torach, modelach i zdarzeniach. Ponieważ zajmujemy się scenariuszem, interesują nas głównie zdarzenia dotyczące eventów:

```
Missed event: randomdleay in multiple keyctrl03
Missed event: 30 in multiple keyctrl03
```

Wczytujemy się dokładnie w listę błędów: zrobiliśmy błąd przy wpisywaniu randomdelay. Poprawiamy i dalej patrzymy w errors.txt. I znowu:

```
Missed event: condiiton in multiple keyctrl02
Missed event: propability in multiple keyctrl02
Missed event: 0.5 in multiple keyctrl02
```

Co tutaj było źle? Poprawiamy condition, żadnych więcej zażaleń już nie ma. Ponownie uruchamiamy

scenariusz i tym razem jest dużo lepiej, w errors.txt nie ma żadnego błędnego eventu.

Być może jednak wyświetlonych zostanie wiele innych błędów, które mogą nas zaniepokoić:

- Bad track – błąd związany z torami, albo w torze znajduje się jakiś event, który nie jest zdefiniowany, albo są jakieś inne błędy związane z torem – ale takie nas nie dotyczą, gdyż zajmowaliśmy się tylko scenariuszem,
- Bad model – jakaś niedoróbka w dekoracjach, to też możemy pominąć,
- i wiele innych możliwych, których nie sposób wymienić – pamiętajmy jednak, że jeżeli będziemy mieli porządek z naszymi eventami, to scenariusz nam zadziała nawet pomimo tego, że symulator zgłosił błędy. Powyższy przykład z założenia jest ubogi, żaden podręcznik nie opisze wszelkich możliwych błędów.

7.2. Korzystanie z log.txt

Są takie błędy w scenariuszu, które dla parsera wcale nie są błędami. Przeanalizujmy taki przykład:

```
node -1 0 mac_komorka1 memcell 1.0 1.0 1.0 * 0 0 none endmemcell
event mac_komorka1_add addvalues 0 mac_komorka1 * 1 1 endevent
event mac_sprawdz_komorke multiple 6 mac_komorka1 mac_wyjazd else mac_komorka1_add
          mac_sprawdz_komorke condition memcompare * -1 -1 endevent
```

Oczekujemy w powyższej sekwencji na uruchomienie eventu *mac_wyjazd*. Kiedy ten event zostanie uruchomiony? NIGDY! Ale parser MaSzyny nie zauważy błędu, bo składnia jest tutaj elegancka i wszystko może być bez problemów wykonane – niestety parser nie zauważy, że kod nie ma sensu.

Kolejny przykład w naszym scenariuszu *calkowo_cargo*:

```
//Przelot przez Paszki Wielkie
event keyctrl02 multiple 0 none paszki_ustaw_zwr1 paszki_ustaw_sem1 paszki_ustaw_zwr2
          paszki_ustaw_sem2 condition propability 0.5 endevent
event paszki_ustaw_zwr1 multiple 0 none pasz_zwr3- pasz_zwr1+ paszprz01_zamykaj endevent
event paszki_ustaw_sem1 multiple 11 none pasz_f_sr3 pasz_tof_od2 pasz_f1_sp4 pasz_b_sr2
          pasz_tob_od2 endevent
event paszki_ustaw_zwr2 multiple 0 none pasz_zwr3+ pasz_zwr2- pasz_zwr1- paszprz01_zamykaj
          endevent
event paszki_ustaw_sem2 multiple 11 none pasz_f_sr2 pasz_tof_od2 pasz_f1_sp2 pasz_c_sr3
          pasz_tob_od2 endevent
```

Tutaj z kolei możemy nie dostać przelotu przez Paszki – bo zapomnieliśmy o słowie kluczowym *else*. Pomyłka się zdarzyła w tak pechowym miejscu, że bez słowa kluczowego wszystko jest poprawne z punktu widzenia składni.

W takich sytuacjach MaSzyna udostępnia nam narzędzia, które mogą nam pomóc w odnalezieniu takiego fragmentu kodu. Po pierwsze: przed uruchomieniem scenariusza przechodzimy w Rainsted do karty *Ustawienia* i zaznaczamy opcję *Tryb testowy (debugmode)*, oraz *Przebieg symulacji w „log.txt”*, a także *zapis pliku „log.txt”*. Następnie możemy uruchomić scenariusz – zostańmy przy powyższym przykładzie, czyli przy przelocie przez Paszki. Dojeżdżamy, naciskamy Shift+2, nic się nie dzieje. W takiej sytuacji przechodzimy do eksploratora plików, otwieramy *log.txt* i sprawdzamy zawartość ostatnich jego linii:

```
Key pressed: [Shift]+[2]
EVENT ADDED TO QUEUE: keyctrl02
EVENT LAUNCHED: keyctrl02
Random integer: 0.808741/0.500000
```

Nacisnęliśmy Shift+2. Zdarzenie *keyctrl02* zostało dodane do kolejki i od razu uruchomione (z uwagi na to, że opóźnienie wykonania ma równe 0). Symulator nas poinformował o wyniku losowania i na tym zapis się urywa – a to oznacza, że problem występuje na etapie losowania. W ten sposób nasza uwaga skupia się na jednej linii kodu, gdzie musimy dopatrzeć się błędu.

Po wnikliwej analizie uzupełniliśmy słowo kluczowe *else*. Zobaczmy jak teraz będzie wyglądał *log.txt* po naciśnięciu Shift+2:

```
Key pressed: [Shift]+[2]
EVENT ADDED TO QUEUE: keyctrl02
EVENT LAUNCHED: keyctrl02
Random integer: 0.001251/0.500000
Multiple passed
```

```
EVENT ADDED TO QUEUE: paszki_ustaw_zwr1
EVENT ADDED TO QUEUE: paszki_ustaw_sem1
EVENT LAUNCHED: paszki_ustaw_zwr1
Multiple passed
EVENT ADDED TO QUEUE: pasz_zwr3-
EVENT ADDED TO QUEUE: pasz_zwr1+
EVENT ADDED TO QUEUE: paszprz01_zamykaj
EVENT LAUNCHED: pasz_zwr3-
Multiple passed
```

Teraz wygląda to zupełnie inaczej – widać, że dalsze eventy zostały uruchomione.

7.3. Event logvalues

Jeżeli problem jest ukryty gdzieś głęboko w komórkach, to możemy się wspomóc eventem użytecznym dla celów diagnostycznych. Dopiszmy do naszego pliku .ctr:

```
event keyctrl00 logvalues 0 komorka1 endevent
```

Podczas wykonywania scenariusza, w momencie gdy naciśniemy kombinację Shift+0, do pliku *log.txt* zostaną zapisane wszystkie 3 wartości komórki o nazwie *komorka1*.

Jeżeli zamiast nazwy komórki podamy *none*, to do *log.txt* zostaną zapisane wartości wszystkich komórek, które występują w scenariuszu. Opcja bardzo ciekawa ale wbrew pozorom jest podchwytliwa – zapisane będą wartości wszystkich komórek, nawet tych, które są poukrywane w semaforach i innych obiektach. Jeśli użyjemy w taki sposób eventu *logvalues*, to nie zdziwmy się, jak będziemy mieli wypisane w *log.txt* około setki wartości, i wśród tego będziemy musieli szukać 2 czy 3 własnych komórek.

7.4. Gotowe widoki z kamer

Zdarzyła się wam sytuacja, że po uruchomieniu scenariusza znajdujecie się na jednym krańcu scenerii a chcecie przemieścić się do drugiego krańca? Przefruwanie nad scenerią, nawet przy bardzo dużym przyspieszeniu, jest uciążliwe i czasochłonne. Z pomocą przychodzi nam możliwość zdefiniowania widoków z kamer zewnętrznych.

Wyjdźmy z lokomotywy, ustawmy kamerę w żądanej pozycji, następnie naciśnijmy jeden z klawiszy od 0 do 9. Po naciśnięciu zajrzyjmy do *log.txt*:

```
Key pressed: [0]
camera -2530.627000 8.542000 -6444.643000 -4.387000 19.347000 0.000000 0 endcamera
```

Drugą linię możemy w całości przekopiować do pliku *.ctr* albo *.scn*. Przy następnym uruchomieniu symulatora, jak wyjdziemy z lokomotywy i naciśniemy klawisz 0, to kamera zostanie ustawiona w zapisanej pozycji.

Porada:

Warto ustawić sobie widoki z kamer na poszczególne stacje w scenerii. W ten sposób ułatwimy sobie życie podczas testów scenariusza.
Niektóre scenerie posiadają na wstępie posiadają zdefiniowane takie widoki, np.: Quarkmce2007. Natomiast w Całkowie widoki są zdefiniowane ale zakomentowane – aby ich użyć wystarczy usunąć znaki // z początku linii.

7.5. Diagnostyka scenariusza OS

Jeżeli napisaliśmy scenariusz opisany na algorytmie OS, i na stacji stanęła nam lokomotywa, nie ma podawanej dalszej drogi, to oczywiście otwieramy *log.txt*, ale raczej nie znajdziemy rozwiązania w ostatniej linii pliku.

Wróćmy zatem do przykładu opisanego w rozdziale 6.1.4. Przypuśćmy, że nasza lokomotywa stanęła pod

tarczą manewrową Tm3 i nie dostaje Ms2. Otwieramy *log.txt* na samym początku, szukamy wywołania fragmentu obsługi, który odpowiada za otwarcie tej tarczy: *mos_cargo_3a*. Jak go znajdziemy, patrzymy czy test w evencie wypadł pozytywnie, następnie patrzymy czy uruchomione zostało *mos_cargo_3b*, itd. Może się zdarzyć, że w ogóle nie znajdziemy w logu wywołania eventu obsługi, wówczas musimy sprawdzić, czy lokomotywa została zgłoszona – być może błąd jest w evencie zgłaszającym.

7.6. Problemy i odpowiedzi

Wstawiłem pociąg na scenerię a on ustawił się na odwrót niż chciałem.

Niestety nie ma możliwości ustawiania pociągu inaczej jak czołem w kierunku punktu 1 toru. Można ten problem naprawić dwojako: albo odwrócić tor, albo ustawić w Rainsted cały pociąg na odwrót – choć to drugie rozwiązanie może nie wyglądać zbyt elegancko.

AI nie reaguje na semafory.

Aby AI poprawnie jeździło należy przypisać wszystkie semafory do torów. No i należy pamiętać o tym, że jeżeli w torze damy jakiś event warunkowy i dopiero w nim będzie *_sem_info*, to AI tego nie zauważy. Event *_sem_info* musi być wpisany bezpośrednio w tor.

Przypisałem zdalnie event do toru o nazwie none_null_077 i nie działa.

Odkryłeś problem, którego nie rozwiązali developerzy MaSzyny. Zdalne przypisanie może nie zadziałać, jeżeli w nazwie toru są podkreślniki. Zmień nazwę toru (jeśli to możliwe) na nonenull077.

Odczytałem w Rainsted współrzędne do umieszczenia dźwięku, dałem do scenerii i go nie słychać.

Rainsted ma inny układ współrzędnych niż MaSzyna – należy zmienić znak współrzędnej X odczytanej w Rainsted. Jeżeli korzystasz z Szopa Track Viewer, to musisz zmienić znak w obu współrzędnych.

Przy wjeździe na uporek lokomotywa wylatuje z torów.

Każdy tor ślepo zakończony musi kończyć się torem z prędkością szlakową = 0 km/h. Należy zapisać do toru końcowego *velocity 0*.

Wyskoczył w errors.txt błąd, który nie jest tu opisany.

Polecam otworzyć forum symulatora MaSzyna, wejść do działu *Symulator*, i przeczytać przyklejony wątek *Wyskoczył nieoczekiwany błąd – zajrzyj – spis errorów.*

8. Inne tematy związane z scenariuszami

8.1. Prace wykończeniowe

Najbardziej zaawansowany scenariusz nie będzie dobrze wyglądał, jeżeli nie będzie ukończony „na wysoki połysk”. Potraktujmy go jak naszą wizytówkę marketingową – musi przyciągać użytkownika MaSzyny ciekawym opisem, zdjęciami, szatą graficzną... a przede wszystkim musi być skończony.

8.1.1. Obrazek do scenariusza

Przygotowujemy obrazek, najlepiej w rozdzielczości 225×169, w formacie .jpg (warto zapisać obrazek z najniższym stopniem kompresji, aby był w jak najwyższej jakości, przy tym rozmiarze waga i tak będzie znikoma). Co będzie na tym obrazku – zależy tylko od nas, wśród twórców scenariuszy panuje niepisana reguła, że powinien być to screen ze scenariusza z naniesioną nazwą scenerii.

Przygotowany obrazek wgrywamy do katalogu *scenery/images*.

Następnie w pliku *.scn* uaktualniamy jedną z linii:

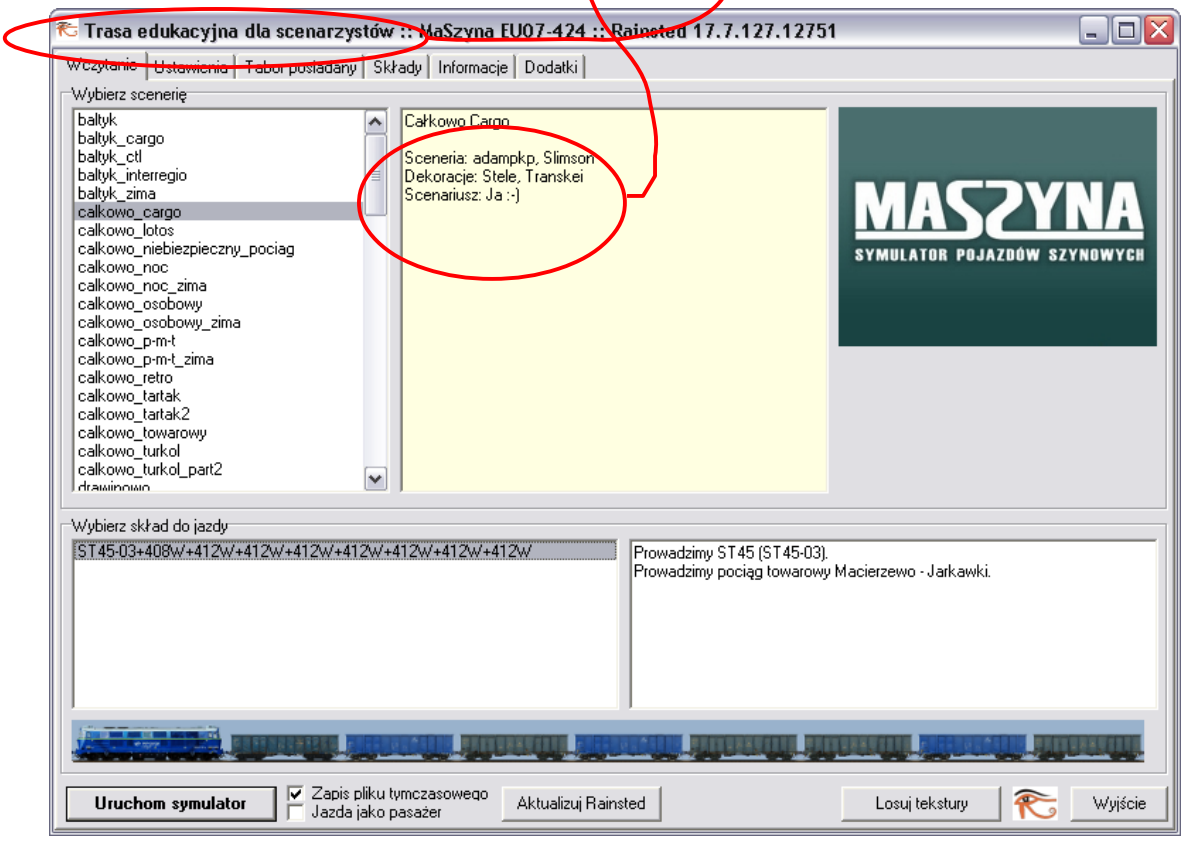
```
//$i nasz_obrazek.jpg
```

8.1.2. Opis scenariusza w oknie Rainsted

Aktualizujemy również najwyższe linie pliku *.scn*:

```
//$n Trasa edukacyjna dla scenarzystów
//$d Całkowo Cargo
//$d
//$d Sceneria: adampkp, Slimson
//$d Dekoracje: Stele, Transkei
//$d Scenariusz: Ja :-)
```

Kiedy otworzymy Rainsted i wybierzemy nasz scenariusz, to wygląd okna będzie następujący:



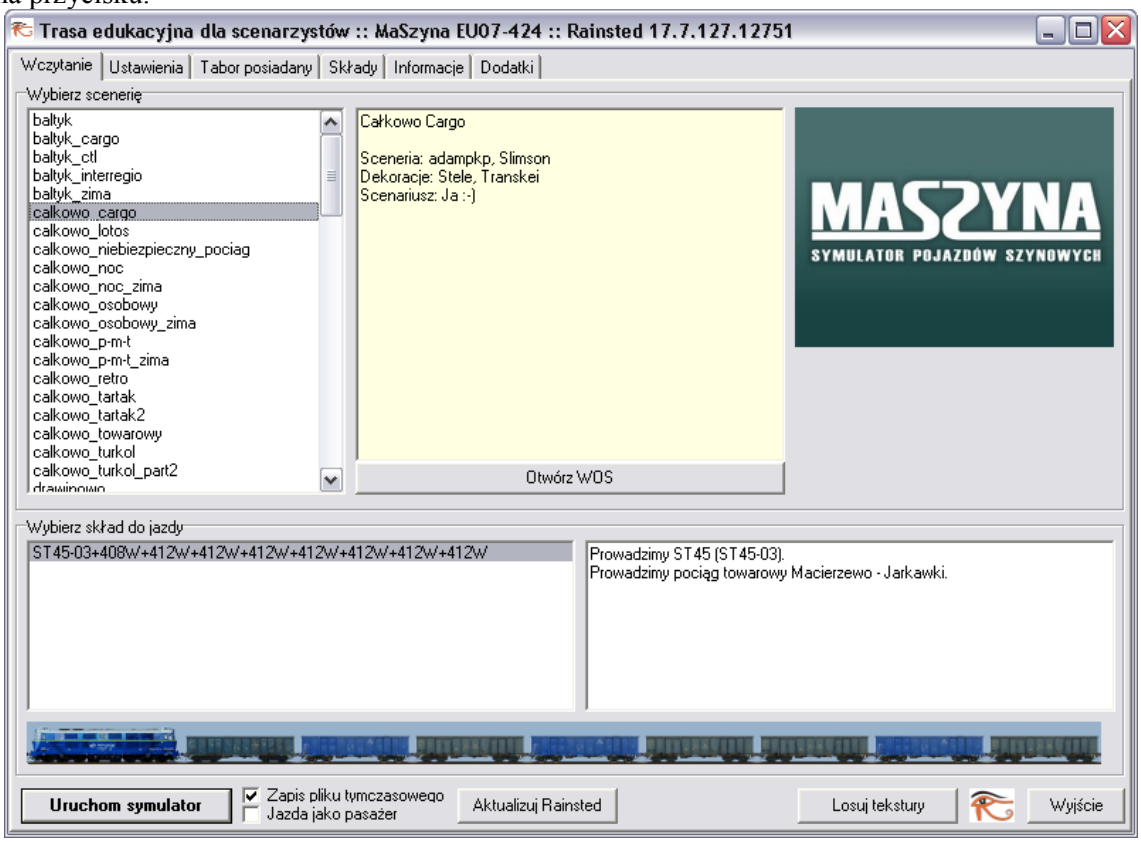
8.1.3. Dodatkowe opisy w html lub pdf

Czasami nie uda się opisać scenariusza za pomocą krótkiego tekstu podawanego przy opisie pociągu. Wtedy warto przygotować bardziej szczegółowy opis scenariusza. Może być to plik *.pdf* albo *.html*. Dopuszczalne są pliki różnych typów, musimy tylko brać pod uwagę, abyśmy nie wybrali takiego formatu, którego jakiś użytkownik nie będzie w stanie otworzyć. Zamiast opisu scenariusza można umieścić np.: wykaz ostrzeżeń stałych (WOS).

Kiedy już przygotujemy potrzebne pliki, wgrywamy je do katalogu *inne* – możemy tam utworzyć własny podkatalog, lub skorzystać z istniejącego. Następnie w pliku *.scn* uzupełniamy wpisy:

```
//$f pl inne/wos_calkowo.PDF Otwórz WOS
```

Po znaczniku *//$f* umieszczamy ścieżkę dostępu i nazwę pliku, następnie piszemy tekst, który wyświetlony zostanie na przycisku.



W zależności od potrzeb, możemy umieścić maksymalnie 3 przyciski z linkami do dodatkowych plików.

8.1.4. Rozkłady jazdy

Wbrew pozorom nie jest to żaden banalny dodatek do scenariusza a coś bardzo istotnego. Rozkład jazdy piszemy w notatniku i korzystamy z jakiegoś innego rozkładu dla danej scenerii, a jeżeli sceneria dla której tworzymy scenariusz jest nowa, to szablon jest na tyle uniwersalny, że możemy go właściwie wziąć z dowolnej innej scenerii. Wówczas musimy jednak prawidłowo przypisać kilometry i oznaczenia stacji. Nie martwmy się, jeżeli zrobimy to źle – dla symulatora istotny jest numer pociągu, seria i obciążenie lokomotywy, stacja początkowa i docelowa, nazwy poszczególnych stacji, godziny przyjazdu i odjazdu. Dla pociągów towarowych bezpiecznie jest podawać tylko godzinę odjazdu, bez godziny przyjazdu – jeżeli przyjedziemy na daną stację przed czasem, to nie będziemy musieli czekać na niej aż wybije godzina odjazdu.

Szczegóły dotyczące tworzenia rozkładów jazdy można wyszukać na forum, gdyż był to temat dość obszernie omawiany.

```
[ Rodzaj i numer pociągu                                | TMS886453   ]
[_______________________________________________________|_____________]
[                                                       | Macierzewo  ]
[ Relacja pociągu                                       |    Jarkawki ]
[_______________________________________________________|_____________]
[ Wymagany % ciężaru hamującego                         | 41%         ]
[_______________________________________________________|_____________]
[ Seria i obciążenie lokomotywy                         |  ST45 1200  ]
[_______________________________________________________|_____________]
[   0.0 |      | Macierzewo                        1        |     ]
[       |      | pp,M,R1,H                         1  12.10 |     ]
[       | 60   |-----------------------------------1--------|-----]
[  3.05 |      | Macierzewo_Jezioro                1     |    | 3   ]
[       |      | po                                1  12.13 |     ]
[       |      |-----------------------------------1--------|-----]
[  8.32 |      | Chrosciele                        1     |    | 6   ]
[       |      | po,pp,R1                          1  12.19 |     ]
[       |      |-----------------------------------1--------|-----]
[  12.3 |      | Paszki_Male                       1     |    | 6   ]
[       |      | po                                1  12.25 |     ]
[       |      |-----------------------------------1--------|-----]
[ 13.68 |      | Paszki_Wielkie                    1     |    | 2   ]
[       |      | pp,M,R1,H                         1  12.27 |     ]
[       |      |-----------------------------------1--------|-----]
```

Jeżeli mamy w scenariuszu przewidzianą lokomotywę jadącą „luzem”, to wpisujemy obciążenie 1. Parser symulatora nie odczyta rozkładu, w którym nie ma obciążenia.

Warto zwrócić uwagę na prędkość rozkładową – jeżeli nawet prędkość szlakowa jest równa 70 km/h, a w rozkładzie jest podana prędkość 60 km/h, to AI będzie jechało z prędkością rozkładową, czyli 60 km/h.

Przy pisaniu rozkładów jazdy należy także zwrócić uwagę na dobór numeru pociągu, wymagany % ciężaru hamującego, oraz oznaczenia stacyjne. Ustalanie tych parametrów nie jest szybkie i proste, ale w internecie można znaleźć liczne materiały, które pozwolą nam dobrać odpowiednie parametry.

8.1.5. Transkrypcje nagrań

Jeżeli do scenariusza nagraliśmy własne rozmowy radiotelefoniczne, to należy przygotować dla nich transkrypcje – czyli zapis rozmowy z pliku *.wav* w pliku *.txt*. Przykładowo: otwórzmy z katalogu *sounds* plik *004_mamy_wolne.wav*. Transkrypcja dla tego pliku będzie wyglądała następująco:

```
[14][30]004, mamy wolne.
[49][60]No jedziemy.
```

Parser MaSzyny zinterpretuje ten zapis tak, że podczas odtwarzania nagrania, między 1 sek. 400 milisek. a 3 sekundą wyświetli napis: *004, mamy wolne*, a między 5 a 6 sekundą wyświetli: *No jedziemy*. Jeżeli wyświetlany tekst jest bardzo długi, to można go podzielić na kilka wierszy, wstawiając co jakiś czas znak | (Shift+\).

W przypadku bardzo krótkich nagrań, np.: *mozna_luzowac.wav*, możemy w ogóle nie podawać sekund, w których ma być wyświetlany napis – symulator po prostu wyświetli go przez cały czas odtwarzania pliku. Czyli dla wspomnianego nagrania transkrypcja wystarczy że będzie:

```
Można luzować!
```

Transkrypcje tworzymy po polsku, jednakże jeśli możemy się wykazać znajomością języków obcych, to możemy również utworzyć transkrypcje w innych językach. Transkrypcje zostały bowiem wprowadzone właśnie z myślą o zagranicznych użytkownikach symulatora.

Gotową transkrypcje zapisujemy w pliku *.txt*, którego nazwa musi być identyczna jak nazwa nagrania, z przyrostkiem *-pl*. Dla wymienionych wyżej nagrań nazwy plików będą następujące: *004_mamy_wolne-pl.txt*, oraz *mozna_luzowac-pl.txt*. Przyrostek może być inny, jeżeli transkrypcja jest w innym języku.

8.2. Narzędzia dla scenarzystów

W poprzednich rozdziałach wszystkie użyteczne programy zostały opisane w sposób skrótowy. Pora podsumować wszystkie użyte i wspomniane narzędzia, oraz niektóre omówić nieco szerzej. A zatem:

- MaSzyna – sam symulator może nam pomóc w pisaniu scenariuszy. Nie jest to jednak narzędzie dedykowane do pisania, toteż pomoc jest konkretna ale niewielka. W sposób opisany w poprzednich podrozdziałach możemy sprawdzić nazwy torów i semaforów, a także odczytać współrzędne miejsca, w którym się aktualnie znajdujemy. **Większość wymienionych funkcji jest dostępna w symulatorze od wersji 17.07!**
- Rainsted – nazwa to kompilacja słów: RA – nick autora, INstalator – możliwość instalowania nowych dodatków, STarter – to co najczęściej używamy, EDytor – to co nam ułatwi pisanie scenariuszy.
- Szopa Track Viewer – alternatywa dla Rainsted. Ma kilka zalet i wad. Program dołączony do każdej kopii symulatora.
- Notatnik – standardowe narzędzie załączane do każdego Windowsa (w Win10 jest nazwany edytorem kodu). Pomimo, że jest to bardzo proste narzędzie, można w nim zrobić praktycznie wszystko. Wadą jest to, że prostota graniczy z prymitywnością, co przy większych projektach przeszkadza. Użytkownikom Linuxa polecam edytor Kate.
- Notepad++ - niektórym nie będzie pasować prostota notatnika. Nie ma co ukrywać – mają rację. Darmowy edytor Notepad++ to znakomite narzędzie do edycji kodu, oferujące bardzo duże możliwości. W Linuxie jest dostępny jego odpowiednik Notepadqq. Istnieją również inne edytory kodu, chociażby Vim.
- EventGenerator – program dołączony do każdej kopii symulatora, najbardziej wyspecjalizowany program do pisania scenariuszy.
- EventoUsuwacz – program pomagający usunąć zbędne eventy z układu torowego. Dołączony do każdej kopii symulatora. Do programu dołączona jest instrukcja obsługi, możemy również przygotować listę, które eventy mają pozostać (czyli te, które zawierają s*em_info*, *lineinfo*, itd.) W sumie cały efekt jego działania można zrobić samemu w ok. 5 minut w Notepad++, z użyciem funkcji wyszukiwania i zamieniania.
- Edytor tekstu, edytor grafiki – głównie dla celów przygotowania opisu scenariusza i obrazka.

8.2.1. Podgląd scenerii w Rainsted

W rozdziale 2.2. pokazano jak w prosty sposób przygotować sobie podgląd torów w scenariuszu. Jednakże taki podgląd jest bardzo ubogi, bardzo trudno się zorientować gdzie są stacje, przystanki, przejazdy, semafory... Można zaimportować do edytora całą scenerię, ale rzadko kiedy się to udaje, a nawet jeśli, to importujemy także wiele nieprzydatnych obiektów, które tylko zaciemniają nam obraz.

Spróbujmy otworzyć scenerię całkowską w edytorze Rainsted. Muszą być w niej tory, drogi, i semafory. Najpierw skopiujmy sobie nasz scenariusz *calkowo_cargo*, nadajmy kopii nazwę np.: *podglad_calkowo.scn*. Następnie edytujmy jego zawartość:

```
include slimson/calkowo_tory_nasze.scm end
//include slimson/teren.scm end
//include slimson/pozostale_os.scm end
//include slimson/tri_os.scm end
//include slimson/zielen_os.scm end
//include slimson/pola_c1.scm end
include slimson/drogi3_os.scm end
//include slimson/slupy_os.scm end
//include slimson/druty_os.scm end
//include slimson/elementy_lampy.scm end
//include slimson/hekt.scm end
//include slimson/calkowo_posers.scm end
//include slimson/calkowo_lawki.scm end
include slimson/calkowo_wskazniki.scm end
//include calkowo/events_cargo.ctr end
```

Stawiając znaki // unieważniliśmy instrukcje dołączenia większości plików scenerii, pozostawiliśmy jedynie tory, drogi, oraz plik zawierający wskaźniki – w tym semafory. Otwieramy teraz Rainsted, uruchamiamy

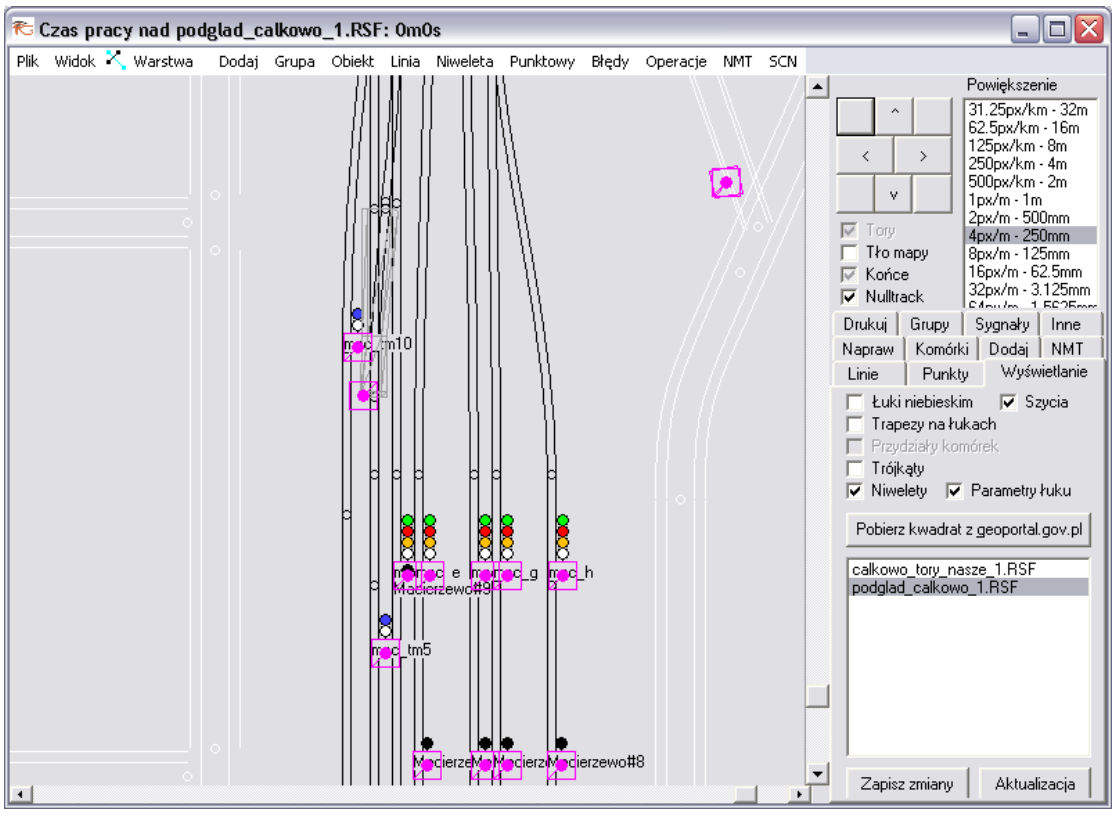
w razie potrzeby tryb pracy specjalny dla trasopisarzy, zaznaczamy w oknie *Wczytanie* nowo utworzony *podglad_calkowo*, i przechodzimy do karty *Debugger*, klikamy kolejno przyciski *Eksport scenerii do RSF* (może to spowodować chwilowy „zwis" edytora), a potem *Edytor scenerii RSF*.

Jeżeli w otworzonym edytorze pojawi się nam możliwość wyboru scenerii *podglad_calkowo*, oznacza to, że konwersję udało się przeprowadzić.

W oknie oprócz torów pojawią się drogi oraz semafory. Prawda, że teraz wygląda to znacznie czytelniej?

Ważna informacja:

W plikach innych scenerii semafory i wskaźniki mogą być zaszyte wewnątrz plików z dekoracjami. Wówczas będziemy musieli albo wyodrębnić wskaźniki do osobnego pliku, albo po prostu wczytać wszystkie dekoracje.



Edytor Rainsted posiada wiele innych funkcji, które mogą się okazać dla nas przydatne:

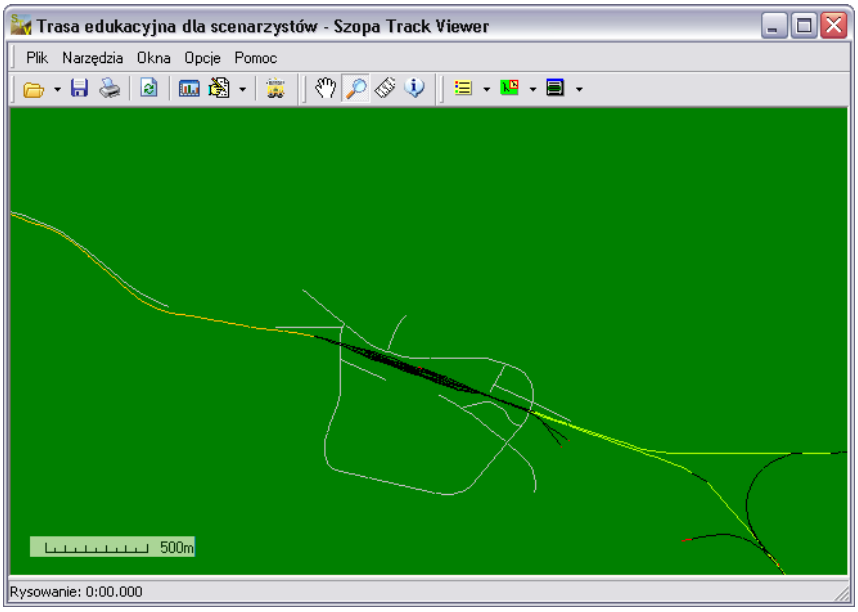
- Przypisywanie sygnalizatorów do torów,
- Definiowanie odcinków izolowanych,
- Wstawianie nowych torów, obiektów, itp.

Zainteresowanych wykorzystaniem tych funkcji odsyłam do instrukcji korzystania z Rainsted.

8.2.2. Szopa Track Viewer

Niestety program wymaga instalacji. Należy uruchomić plik znajdujący się w katalogu *programy_na _potrzeby_symulatora/podglad_scn*. Kiedy już zainstalujemy program, możemy kliknąć na stworzony w rozdziale 8.2.1. plik prawym przyciskiem myszy, wybrać opcję *Otwórz za pomocą...* i otworzyć plik w programie Szopa Track Viewer.

Ukaże się następujący widok:



Program jest prymitywniejszy niż Rainsted ale zawiera w zamian wiele interesujących funkcji. Możemy za pomocą lupki powiększyć widok, następnie z menu *Narzędzia* wybrać *Wskaźnik*. Po kliknięciu na tor ukaże się nam okienko z danymi: współrzędne toru, długość, nachylenie, prędkość szlakowa, przypisane eventy, pociągi stojące na tym torze.

Inne przydatne opcje to:

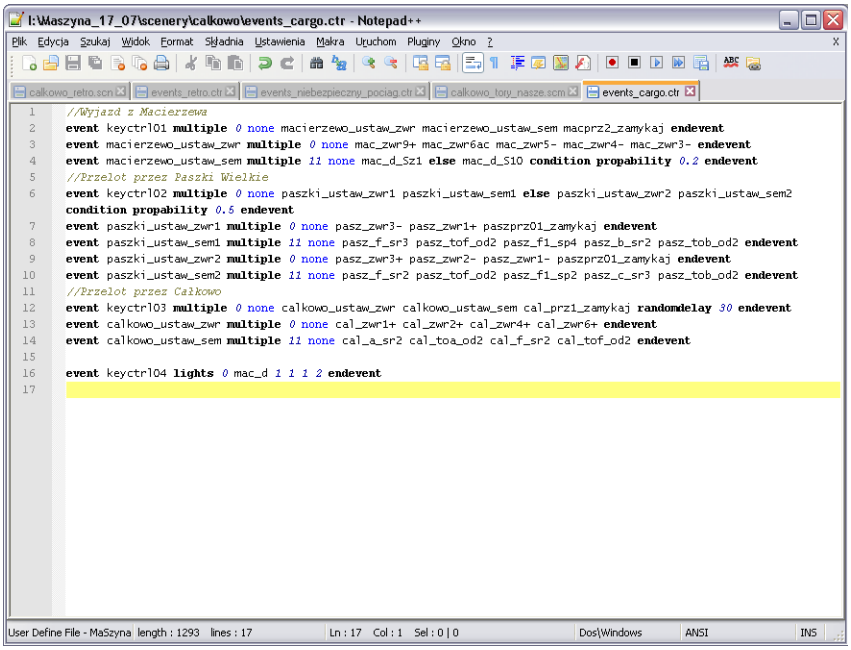
- Możemy wybrać różne typy mapy:
  - Prędkość szlakowa – mamy wizualną informację, czy na szlaku są jakieś ograniczenia, możemy sprawdzić czy wszystkie ślepe tory są zakończone krótkim torem o prędkości szlakowej 0... Kolor czerwony oznacza prędkość 0, przez żółty, zielony, do czarnego – prędkość maksymalna.
  - Hipsometryczna – jeżeli interesuje nas wysokość bezwzględna toru,
  - Wzniesieniowa – bardzo przydatna mapa, gdy chcemy sprawdzić, czy na szlaku występują wzniesienia.
- Program może rysować następujące elementy:
  - Drogi, rzeki,
  - Zaznaczać sieć trakcyjną – bardzo przydatne, gdy nie chcemy wjechać lokomotywą elektryczną na tor bez druta,
- Narzędzie miarki pozwala na szybkie zmierzenie odległości między dwoma punktami,
- Lista nazwanych torów – dzięki temu możemy szybko zidentyfikować, gdzie znajduje się interesujący nas tor.

Niestety program bardzo często się wysypuje przy wczytywaniu scenerii. O ile bez problemu powinniśmy otworzyć specjalnie przygotowany plik *podglad_calkowo.scn*, to próba otwarcia w tym programie pliku *calkowo_cargo.scn* zakończy się wysypem programu.

8.2.3. Notepad++

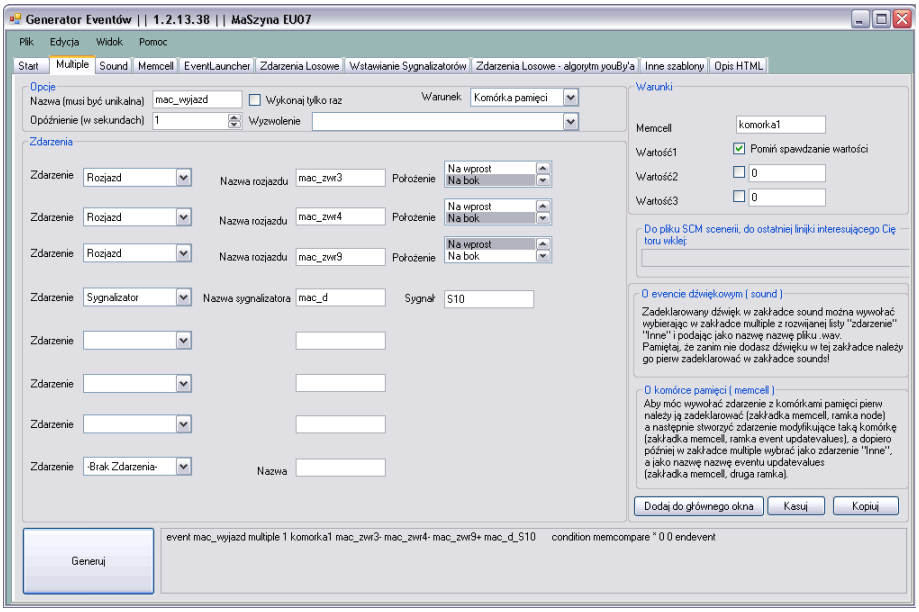
Program o ogromnych możliwościach, zwłaszcza w zakresie otwierania wielu dokumentów jednocześnie, wyszukiwania i zastępowania fraz tekstu, możliwość użycia znaków zastępczych... W internecie dostępna jest obszerna pomoc do tego programu.

Warto wspomnieć o możliwości podświetlania składni dla wielu języków programowania. Wprawdzie nie ma domyślnie wbudowanego podświetlenia dla języka eventów, jednakże mamy możliwość ręcznego wprowadzenia skryptu, który podświetli nam odpowiednio słowa kluczowe eventów.. Najlepiej jednak zaimportować skrypt z podświetleniem z forum MaSzyny w wątku w którym developerzy udostępnili swoje własne skrypty podświetlające składnie eventów (Dział *Pomoc w tworzeniu*, wątek *Style języków symkowych do notepada++*).



8.2.4. Generator Eventów

Program zasłużony dla symulatora MaSzyna, był pierwszą bardzo poważną próbą stworzenia narzędzia przeznaczonego specjalnie do pisania scenariuszy. Na pewno bardzo ułatwi wdrożenie osobom początkującym, dla osób obeznanych z pisaniem scenariuszy może być przydatny jako interaktywny help (autor niniejszego opracowania tak właśnie używa tego generatora).



Na powyższym obrazku widzimy istotę generatora – wybieramy kartę odpowiadającą zdarzeniu lub obiektowi, który chcemy utworzyć. Wypełniamy formularz, następnie klikamy przycisk *Generuj*. W polu obok przycisku otrzymujemy zapis, który musimy wkleić do pliku scenariusza. Jednym zdaniem: bardzo przydatny program dla osób chcących poeksperymentować ze składnią eventów *multiple*, *sound*, wstawianiem komórek, oraz eventlauncherów. Są również dostępne szablony do tworzenia zdarzeń losowych.

Czy program ma jakieś istotne wady? Nie był aktualizowany od nowości, a w czasie gdy powstał nie było czegoś takiego jak *else*, *randomdelay*, *whois*, *addvalues* – bardzo przydatnych słów kluczowych.

Do programu została załączona książka opisująca bardzo dokładnie szczegóły działania – zainteresowanych generatorem odsyłam do lektury.

8.2.5. Narzędzia dostępne na forum

Forum symulatora to wielka i czynna kopalnia rozmaitych pomysłów, w tym miejsce publikacji wielu ciekawych narzędzi. Poniżej przedstawiam narzędzia i wątki, w których są lub będą – może w przyszłości – dostępne takowe narzędzia:

- Dział *Symulator*, wątek przyklejony *Różne mariuszowe narzędzia*. Pakiet programów oraz arkuszy kalkulacyjnych z zestawem makr. Wprawdzie w pisaniu scenariuszy mogą nie być szczególnie użyteczne, ale przydają się, gdybyśmy chcieli wprowadzić jakieś niewielkie korekty w scenerii – dostępny jest nawet prosty edytor. Inne możliwe operacje to: odrutowanie słupów telegraficznych, osadzanie na podłożu „latających" obiektów w scenerii, automatyczne wstawianie latarni, segregacja i porządkowanie plików. W niektórych przypadkach wymagany MS Excel.
- Dział *Symulator*, wątek przyklejony *Programy na użytek symulatora. [Szczawik]*. Programy służące wprawdzie głównie do edycji scenerii, jednakże można wstawiać za jego pomocą odcinki izolowane.
- Dział *Na warsztacie*, wątek *Edytor plików scenerii ScnEdit, alpha testy*. Ciekawie zapowiadający się edytor dla piszących scenariusze – oprócz podświetlania składni miał mieć różne ciekawe funkcje, aktualnie nie słychać o postępach prac, a do pobrania są jedynie nieskompilowane źródła.
- W dziale *Na warsztacie* niektórzy przedstawili swoje własne edytory scenerii, niektóre wyglądające na bardzo zaawansowane. Stan prac nieznany, linków do pobrania brak. Ale kto wie, może ktoś... kiedyś... Reasumując: polecam śledzenie rozwoju wydarzeń na forum.

8.3. Trendy w pisaniu scenariuszy

Skoro już wiemy wszystko, to warto abyśmy zastosowali się do bieżących wymogów stawianych twórcom na forum eu07.pl. Trendy te na bieżąco się zmieniają, dlatego warto śledzić wątki na forum, oraz pojawiające się tam dodatki. Poniżej podaję kilka najważniejszych spostrzeżeń, aktualnych podczas pisania niniejszego poradnika (Sierpień 2017 r.):

- Niegdyś wszystkie scenariusze były pisane w taki sposób, że jechał jeden pociąg, a reszta stanowiła tylko dekorację, w dosłownym niemal tego słowa znaczeniu (np.: jak mijaliśmy na stacji jakiś pociąg, to najczęściej jechał on sprzed stacji za stację, lub z niej w ogóle nie wyjeżdżał). Obecnie ta tendencja zanika. Nawet jeśli pociągi są tylko dekoracyjne, to najczęściej pokonują jakieś sensowne trasy.
- Należy unikać naciskania w scenariuszu kombinacji klawiszy Shift+cyfra. Nie znaczy, że jest to zakazane, ale powinno być zostawione tylko dla detali nie mających wpływu dla przebiegu (np.: wyłączenie odgłosów deszczu),
- Przygotowałeś/aś scenariusz z odrębnym układem torowym? Nie jest to zalecane, ale akceptowalne.
- Poświęć trochę czasu – jeżeli w układzie torowym, którym dysponujesz są jakieś nie przypisane wskaźniki do torów (semafory, W4), to je przypisz. Ułatwi to prace innym a nawet Tobie.
- Scenariusz powinien zostać napisany tak, aby był w pełni przejezdny przez AI. Wymaga trochę więcej sprytu podczas pisania, ale ułatwi Tobie testy – będzie można uruchomić scenariusz, włączyć szybkie tempo, nacisnąć Ctrl+Q, usiąść z założonymi rękami – scenariusz sam się przetestuje. Ponadto jeżeli tworzysz scenariusz dla pociągu osobowego, to będziesz mógł się przejechać jako pasażer.
- Zadbaj o udźwiękowienie, cisza w eterze nie poprawi atrakcyjności scenariusza. Mile widziane będzie ćwierkanie ptaszków, szum morza (jeśli jeździmy po scenerii Bałtyk i dojechaliśmy do stacji Bałtyk Plaża), odgłosy stacji, cykanie świerszczy w nocy, hałasy z zakładów przemysłowych, itp.
- Bez rozkładu jazdy nawet nie pokazuj publicznie scenariusza, kiedyś to była nowość, potem fajna rzecz, potem przydatna rzecz, teraz to obowiązek.
- Staraj się pisać scenariusze tak, aby nie było konieczności tworzenia kolejnych układów torowych dla dalszych scenariuszy.
- Odcinki izolowane zostały stworzone z myślą, aby wyeliminować konieczność przypisywania eventów do torów, i tym samym ujednolicić układy torowe. Pisanie scenariusza z użyciem odcinków izolowanych tylko dlatego, że jest presja aby tak robić może nie mieć w sobie celu – pojawił się niegdyś w testach scenariusz, który jest oparty tylko na odcinkach izolowanych, a i tak z uwagi na sposób ich zbudowania wymagał odrębnego układu torowego. Zatem stosuj odcinki izolowane ale nie na siłę, ich stosowanie samo w sobie może nic nie dać.
- Planujesz scenariusz 6-godzinny? Nie będzie zbyt wielu chętnych do jazdy. Jeśli naprawdę planujesz taką misję, to podziel ją na odcinki. Scenariusz będzie optymalny, jeżeli nie będzie trwał dłużej niż 2 godziny.

- Niektórzy nie lubią manewrów tylko wolą jeździć, inni uważają je za bardzo ciekawe urozmaicenie – jakikolwiek scenariusz nie napiszesz, wszystkim nie dogodzisz, ale na pewno znajdą się osoby, które uznają scenariusz za ciekawy. Autor sam woli krótsze scenariusze, gdzie jest głównie jazda i mało manewrów, ale raz na jakiś czas lubi też uruchomić jakąś misję manewrową.
- Niezależnie od tego, czy piszesz malutki czy ogromny scenariusz – zadbaj o to, aby w sposób wyczerpujący opisać kod źródłowy za pomocą komentarzy umieszczanych w pliku .ctr, tak aby ułatwić w przyszłości developerom ewentualne modyfikacje.
- **Scenariusz musi być zgodny z przepisami obowiązującymi na kolei! Nie ma wyjątków od tej reguły!** Przepisy są dostępne w internecie, można również przejrzeć zawartość katalogu *przepisy_kolejowe* dołączonego do każdej kopii symulatora. Jeżeli w Twoim scenariuszu występuje jakaś bardzo nietypowa sytuacja, z którą nie wiesz jak sobie poradzić, to zadaj pytanie na forum – osoby pracujące na co dzień na kolei na pewno udzielą odpowiedzi.
- **Często twórcy coś zaczną, doprowadzą do stanu bardzo zaawansowanego, ale nie dokańczają na 100%. Nie wpisuj się w ten trend, jak coś zaczniesz – doprowadź do końca! Albowiem łatwo jest coś zacząć, trudniej jest to skończyć. Żeby nie było, dotyczy to również autora niniejszego opracowania.**

Indeks