Kyvadlový provoz e-trucku v režimu českého dopravce

1.1 TEORETICKÝ ÚVOD

Reálný provoz nákladních vozidel je složitý: střídají se jízdy, nakládky/vykládky a (u elektro) nabíjení. Simulace nám umožní:

- přehrát celý cyklus „A ↔ B“ opakovaně,
- měřit podíl času v jednotlivých režimech,
- bezpečně zkoušet „co kdyby“ scénáře (výkon nabíječky, délka trasy, strategie nabíjení).

Cílem není „vychválit“ ani „potopit“ e-truck. Cílem je vyčíslit dopad nabíjení na využití vozidla v jednoduchém, ale transparentním modelu reálné linky A ↔ B, který vychází z reálné situace jednoho z největších českých dopravců a zároveň momentálně jedním z mála dopravců, který si díky své velikosti může dovolit e-truck provozovat, a to na pravidelné lince ze sídla v Jičíně do skladu vzdáleného cca 90 km, kdy nabíjení probíhá pouze na firemní nabíjecí stanici v Jičíně a e-truck se po této „otočce“ vrací s cca 30 % SOC (kapacity baterie). Teprve pak má smysl řešit náklady, emise a infrastrukturní nároky.

1.1.1 AnyLogic a Java

Java je staticky typovaný, objektově orientovaný jazyk, který se kompiluje do bytecode a běží na JVM (Java Virtual Machine). Tahle vrstva mezi vaším kódem a HW zajišťuje přenositelnost (stejný program spustíte na Windows, macOS i Linuxu) a bezpečnost (správa paměti přes garbage collector, sandboxing). V praxi píšete zdroják (.java), ten se přeloží kompilátorem javac do .class, a o běh se stará java (JVM). V AnyLogicu tohle vše běží „uvnitř“ – vaše skripty jsou Java a AnyLogic je normálně zkompiluje a spustí v rámci svého simulačního enginu.

Java staví na třídách a objektech. Třída (např. ETruck) je šablona; objekt je konkrétní instance v běhu. Třída obsahuje stav (proměnné/fields) a chování (metody). Konstruktory nastavují počáteční stav. Zapouzdření znamená, že vnitřní proměnné chráníme (typicky private) a ven pouštíme jen to, co chceme (přes public metody). Dědičnost umožní rozšířit existující třídu, ale v praxi dáváme přednost kompozici („má-vztah“) před dědičností („je-vztah“), protože je pružnější. Rozhraní (interface) definují smlouvy – co objekt umí – a přes polymorfismus můžeme psát kód, který pracuje s různými implementacemi stejným způsobem.

Jazyk je case-sensitive (proměnná soc není SOC), má primitivní typy (int, double, …) a referenční typy (třídy jako String, List, …). Číselné výpočty ve fyzice a simulacích se dělají typicky v double (rychlé a dostatečně přesné pro kWh, km, h), zatímco peníze patří do BigDecimal. Kolekce (List, Set, Map) jsou generické – např. List<Double> – a běžné operace nad daty umí i Stream API a lambdy (x -> x * 2). Ošetřování chyb je přes výjimky: některé jsou „checked" (musíte je zachytit nebo deklarovat), jiné „unchecked" (programové chyby). Čas a datum řeší moderní java.time (Duration, Instant), které je výrazně přesnější než staré Date.

Důležitá je životnost objektů a vláken. V čisté Javě spouštíte vlastní thready, v simulaci ale vlákna řídí AnyLogic Engine; vy běžně nepřidáváte svoje – používáte události, timeouty a stavové automaty (statecharts), které AnyLogic překládá do Javy za vás. AnyLogic také pracuje s modelovým časem. Pomocné funkce second(), minute(), hour() vracejí převody do jednotky modelu (abyste nemuseli ručně přepočítávat). Metody jako setSpeed() očekávají rychlost v metrech za modelovou sekundu; pokud máte rychlost v km/h, převedete ji násobkem 1000 / hour().

V kostce: Java nám dává robustní typy, OOP a bohatou standardní knihovnu; AnyLogic nad tím přidává notaci pro procesy, stavové automaty a čas. Výsledkem je, že model je jednak „naklikaný", ale uvnitř je to pořád čistá Java, kterou dokážete číst, ladit i rozšiřovat.

1.2 TVORBA MODELU

Budeme vytvářet agentově orientovaný (agent-based) model. Základní entitou je agent ETruck, který má vlastní stav (SOC, energie v kWh, rychlost), vnitřní chování řízené stavovým automatem (AtA → Charging → DriveToB → …), lokální rozhodování (odjezd/nabíjení podle SOC) a pohyb v prostoru mezi FirmA a FirmB. Agentový přístup volíme proto, že potřebujeme zachytit individuální heterogenitu (různé baterie/spotřeby/strategie nabíjení), mikro-dynamiku (čas ve stavech, trajektorie, skutečný profil SOC) a být připraveni model rozšířit na flotilu s interakcemi (fronty u nabíječek, priority, směny řidičů). Na rozdíl od čistě procesního DES („fronty a obsluhy") se nám zde jednodušeji vyjadřují prostorové a energetické rozhodovací situace a oproti systémové dynamice (SD) neztrácíme granularitu jednotlivých vozidel. Technicky vzato využíváme AnyLogic, který kombinuje ABM s diskrétními událostmi (timeouts v přechodech) — takže jde o ABM s DES prvky, ale primární perspektiva je „vozidlo jako autonomní agent".

1.2.1 Agent ETruck

Z palety **Agent** přetáhneme do hlavního prostředí nového Agenta **ETruck**.

Po dokončení průvodce začneme do plochy přidávat prvně parametry, které najdeme v paletě **Agent** pod názvem **Parameter** a budou určovat základní parametry fungování agenta ETruck. U všech parametrů nastavíme Type: double, názvy parametrů a default value viz tabulka:

Název	**Význam**	**Defaultní hodnota**
batteryCapacity_kWh	Kapacita baterie	300
soc	Počáteční SOC (0–1)	1.0
socMinToDepart	Minimální SOC pro odjezd (pod toto číslo nevyjede)	0.70
socTargetCharge	Cílový SOC při nabíjení (na kolik se má nabít vozidlo po návratu	1.0
consumption_kWh_per_km	Spotřeba kW na km	1.2
chargerPower_kW	Výkon nabíječky v kW	150
speed_kmh	Cestovní rychlost [km/h] (zvažujeme průměrnou rychlost e-trucku v rámci celé cesty)	60
distance_km	Vzdálenost A↔B (vzdálenost dvou firem)	90

Hodnoty SOC jsou v intervalu 0-1, což představuje 0–100 %. Všechny čísla je nutné psát (dle zásad jazyku Java) s desetinou tečkou, nikoliv čárkou. Názvy parametrů jsou case sensitive, tedy tak, jak je zvolíte, musíte poté psát i ve funkcích apod., např. socMinToDepart ≠socmintodepart.

Nyní přidáme několik **Variable**, taktéž z palety **Agent**, dle tabulky:

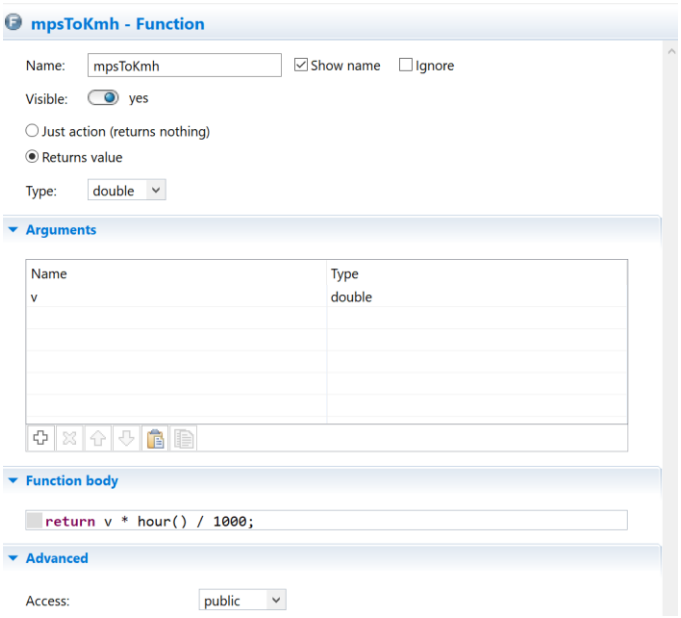
Name	***Type***	***Initial value***	***Poznámka***
statusText	*String*	`"Stav: inicializace"`	*Bude sloužit k ukládání momentálního stavu vozidla (nakládka, jízda…)*
driveSpeed_kmh	*double*	`speed_kmh`	*„cestovní" rychlost*
currentSpeed_kmh	*double*	`0`	*pro zobrazení v km/h*
_tLast	*double*	`time();`	*interní časovač energie*
energy_kWh	*double*	`energy_kWh = soc * batteryCapacity_kWh;`	*energie v baterii*
tCharge_h	*Double*	`0`	*Pomocné proměnné pro výpočty do grafů*
tDrive_h	*Double*	`0`	

Name	***Type***	***Initial value***	***Poznámka***
tIdle_h	*Double*	*0*	
tTotal_h	*Double*	*0*	

Následně přidáme tři pomocné funkce, opět z palety **Agent** přidáme tři **Function**:

Název	**Akce / hodnota**	**Arguments**	**Function body**
applyCruiseSpeed	Just action (returns nothing)	-	`setSpeed( speed_kmh * 1000 / hour() );`
kmh	Returns value, type: double	Name v, type double	`return v * 1000 / hour();`
mpsToKmh	Returns value: type: double	Name v, type double	`return v * hour() / 1000;`

V případě funkcí je nutné, pokud vrací hodnotu, definovat její argument, viz níže ukázka u funkce kmh:



Nyní přidáme ještě události, které mají za úkol dělat předem nadefinované akce v určitých intervalech nebo v určitém času. Z palety **Agent** přidáme dvě události **Event**:

1) **Event** s názvem *tickUI*, který slouží k zjištění aktuální rychlosti
   - Trigger type: Timeout

- Mode: Cyclic (cyklicky se opakuje dle následujících časů)
- First occurrence time: 0 seconds (spouští se se startem modelu)
- Recurrence time: 0.2 seconds (opakuje se každé 0,2 sekudny)
- Action: `currentSpeed_kmh = mpsToKmh( getSpeed() );`

2) **Event** s názvem *tickEnergy*, který slouží k výpočtům spotřeby elektrické energie, nabíjení apod.
   - Trigger type: Timeout
   - Mode: Cyclic (cyklicky se opakuje dle následujících časů)
   - First occurrence time: 0 seconds
   - Recurrence time: 1 seconds
   - Action:

```
double t = time(); // deklarace proměnných t a dt, které se budou používat
double dt = t - _tLast;
if (dt <= 0) return;

double dt_h = dt / hour();

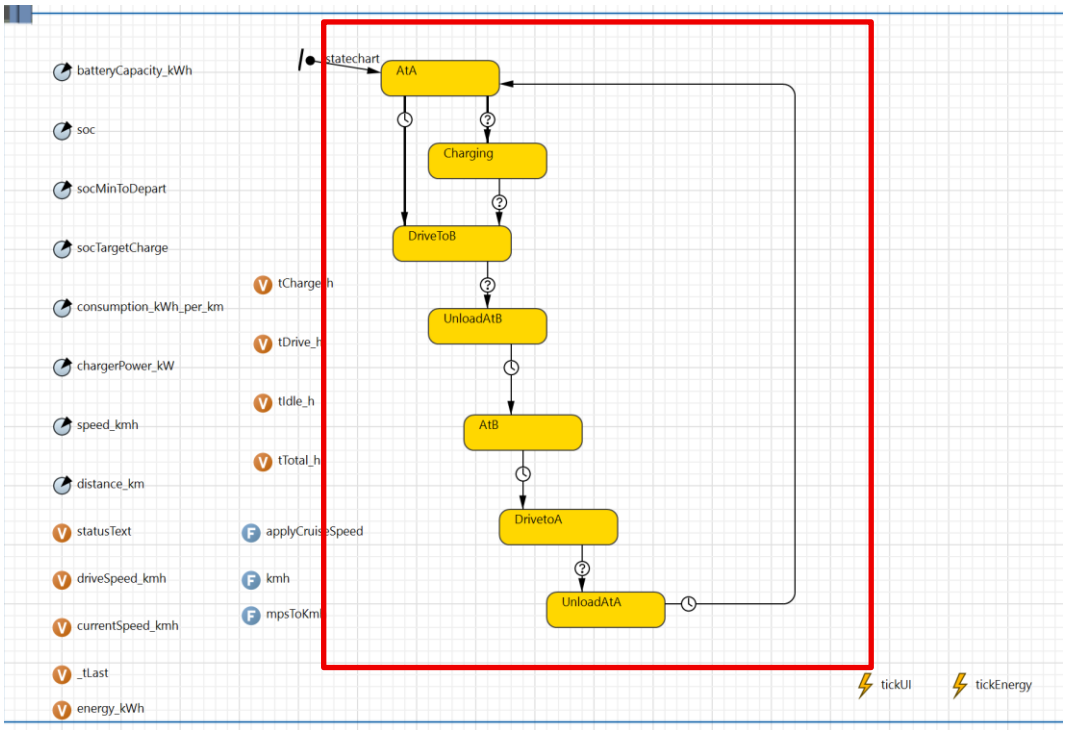
// jízda / nabíjení / čekání + energie
if (isMoving()) {
  tDrive_h += dt_h;
  double dist_km = getSpeed() * dt / 1000.0;   // m -> km
  energy_kWh -= dist_km * consumption_kWh_per_km;
} else if (inState(Charging)) {
  tCharge_h += dt_h;
  energy_kWh += chargerPower_kW * dt_h;       // kW * h = kWh
} else {
  tIdle_h += dt_h;
}
tTotal_h += dt_h;

// limity + SOC
energy_kWh = Math.max(0, Math.min(batteryCapacity_kWh, energy_kWh));
soc = energy_kWh / batteryCapacity_kWh;

_tLast = t;
```

Pokračujeme vytvořením stavového automatu, tedy **statechart**, který popisuje chování agenta, resp. budoucí populace agentů ETruck. Pro vytváření používáme bloky s chováním z palety **Agent blok State**, pro řízená spojení mezi bloky činností používáme **Transition**, kterými bloky spojíme. Neméně důležitá je šipka **Inital State Pointer**, kterou vložíme na první blok a ukazuje, kde má cyklus fungování daného agenta začít.

Bloky vložíme a propojíme dle schématu:


batteryCapacity_kWh
soc
socMinToDepart
socTargetCharge
consumption_kWh_per_km
chargerPower_kW
speed_kmh
distance_km
statusText
driveSpeed_kmh
currentSpeed_kmh
_tLast
energy_kWh
tCharge_h
tDrive_h
tIdle_h
tTotal_h
applyCruiseSpeed
kmh
mpsToKmh
statechart
AtA
Charging
DriveToB
UnloadAtB
AtB
DrivetoA
UnloadAtA
tickUI
tickEnergy

Blok	Význam	Entry action	Exit action
AtA	„At A“ – vozidlo se nachází na firmě A, do bloku vede šipka statechart, kdy zde cyklus začíná, probíhá nakládka	`statusText = "Nakládka v A";` `setSpeed(0);`	

DriveToB	Vozidlo jede na firmu B, pokud jsou splněny podmínky dostatečného nabití baterie, pokud ne, „jede“ přes blok Charging	`statusText = "Jede do B";`
		`applyCruiseSpeed();`
		`moveTo( main.FirmB );`
			`setSpeed( driveSpeed_kmh * 1000 / hour() );`
			`// km/h -> m/s`
			`moveTo( main.FirmB );`
			
			`// Odečti energii za cestu A→B`
			`double tripEnergy = consumption_kWh_per_km * distance_km; // kWh`
			`soc = Math.max(0, soc - tripEnergy / batteryCapacity_kWh);`

Charging	Vozidlo se před výjezdem z bloku A nabíjí, dokud nejsou splněny podmínky na dostatečný stav (nastavené v parametrech)	`statusText = "Stav: Nab + nakl v A";` `setSpeed(0);`	`soc = Math.min(1.0, socTargetCharge);` `statusText = String.format("Nabíjení v A (%.0f %% → %.0f %%)", soc, socTargetCharge);`

UnloadAtB	Vozidlo vykládá ve firmě B	`setSpeed(0);` `statusText = "Vykládka v B";`	
AtB	Vozidlo je ve firmě B, probíhá nakládka	`setSpeed(0);` `statusText = "Nakládka v B";`	

DrivetoA	Vozidlo jede do firmy A	`statusText = "Jede do A";`
		`applyCruiseSpeed();`
		`moveTo( main.FirmA`
			`double tripEnergy = consumption_kWh_per_km * distance_km;`
			`soc = Math.max(0, soc - tripEnergy / batteryCapacity_kWh);`

UnloadAtA	Vozidlo vykládá ve firmě A	`setSpeed(0);` `statusText = "Vykládka na A";`	

Nyní je nutné správně nastavit přechody (transition) mezi bloky, které zajišťují svými podmínkami přechod na další blok:

Transiton	**Triggered by**	**Guard**	**Funkčnost**
Statechart → AtA	Defaultní nastavení		
AtA → DriveToB	Timeout, 30 minutes	`soc >= socMinToDepart`	Do B jede po 30 min. vykládky, hlídá stav nabití baterie
AtA → Charging	Condition, true	`soc < socMinToDepart`	Nabíjí (s nakládkou), dokud vozidlo nemá min. kapacitu k dobití
Charging → DriveToB	Condition, true	`soc >= socMinToDepart`	Po nabití jede do firmy B
DriveToB → UnloadAtB	Condition, !isMoving()		Jede na firmu B
UnloadAtB → AtB	Timeout, 30 minutes		Vykládá na B po dobu 30 minut
AtB → DrivetoA	Timeout, 30 minutes		Nakládá na B po dobu 30 minut
DrivetoA → UnloadAtA	Condition, !isMoving()		Jede na firmu A
UnloadAtA → AtA	Timeout, 30 minutes		Vykládka na A po dobu 30 minut

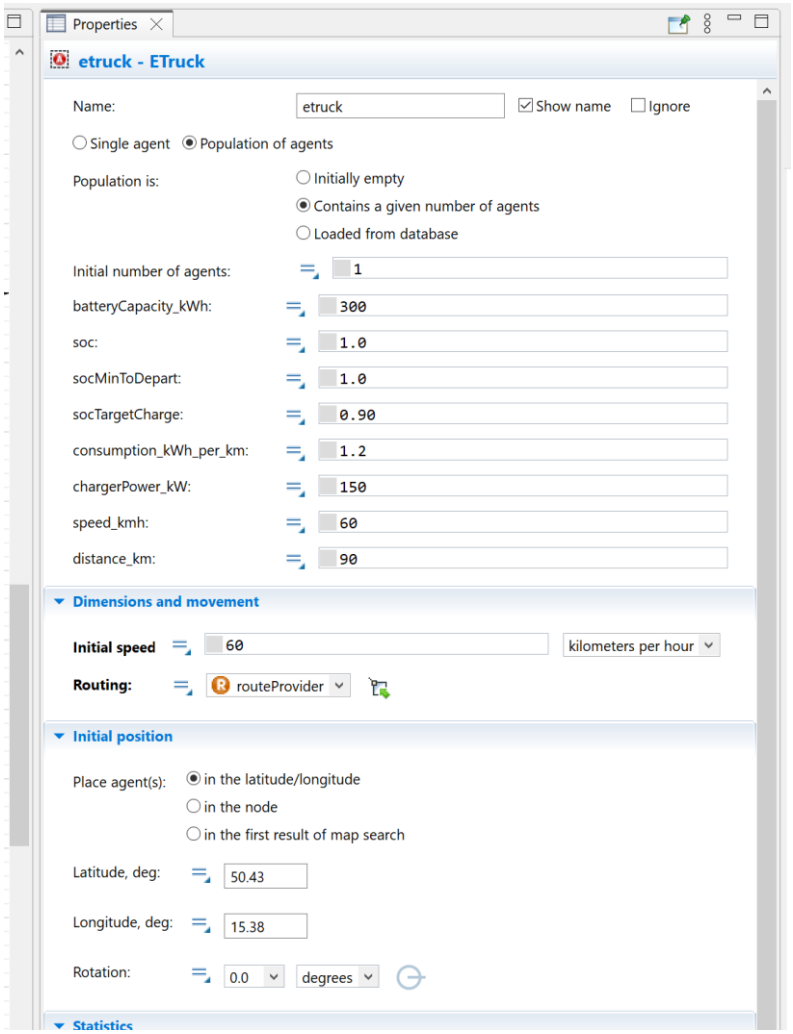
1.2.2 Prezentace v Main

Nyní máme připraveného agenta ETruck, který je nutné implementovat do hlavní části našeho modelu.

Pro prezentaci v prostředí České republiky přetáhneme na hlavní plochu z palety **Space Markup GIS Map** a **Route Provider**. Na mapu poté klikneme pravým tlačítkem, zvolíme Edit Map a nastavíme mapu přibližně tak, abychom viděli Prahu a Jičín, případně manuálně nastavíme Latitude: 50.27329 a Longitude: 14.85537 a Scale: 1:100000. V Route Provider poté nastavíme Routers are: Requested from OSM server, Routing method: Fastest a Road type: Car.

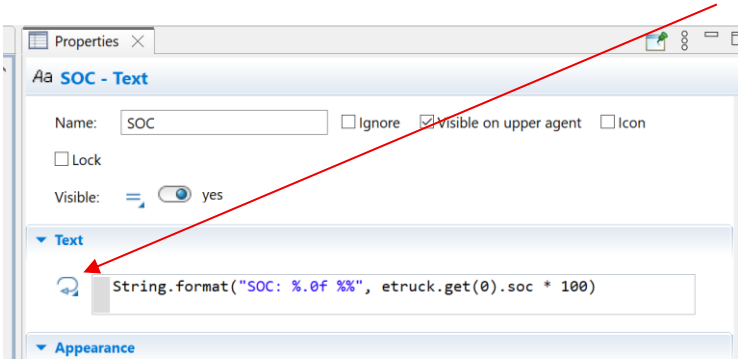
Ze stejné palety (**Space Markup**) přidáme přetažením do mapy dva **GIS Point**, které budou reprezentovat naše dvě firmy. První bod, FirmA bude v Jičíně, v sídle dopravce, s parametry Latitude: 50.43 a Longitude: 15.38. Druhý bod, FirmB, bude v odpovídající vzdálenosti umístěn na předměstí Prahy, Latitude: 50.13 a Longitude: 14.48.

Dále do **Main** z palety **Agent** vložíme nového agenta, nastavíme populaci agentů, pojmenujeme jej etruck a přidáme existujícího agenta ETruck (tedy máme populaci agentů etruck, ve které jezdí jednotliví agenti ETruck). Zde nastavíme inital number of agents: 1, tedy bude nám jezdit jeden e-truck. Rovněž nastavíme počáteční bod totožný s firmou A, tedy Latitude: 50.43 a Longitude: 15.38. Zbytek parametrů by se měl přenést sám z parametrů agenta, případně nastavíme viz níže:



1.2.3 Výpis aktuálních hodnot

V Mainu přímo pod mapu vložíme z palety **Presentation** třikrát text, který bude sloužit k výpisu hodnot. Pokud nechceme statický text, ale dynamický, vycházející z našich funkcí, případně aby sám o sobě text obsahoval funkcí, musíme prvně nastavit, že se jedná **o dynamic value:**

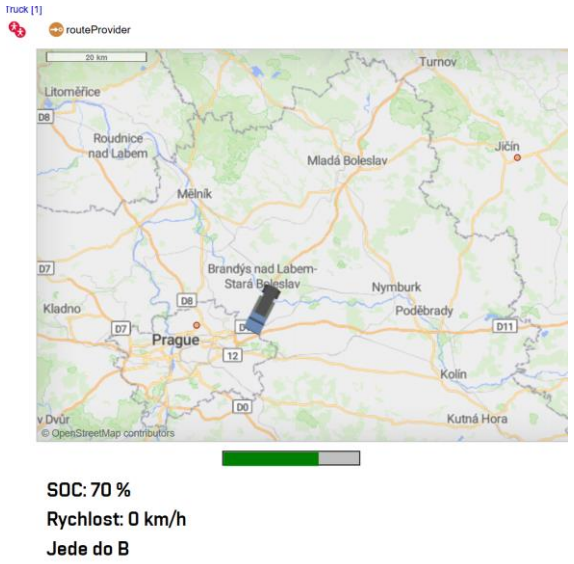


Takto nyní můžeme vložit všechny tři texty, které budou zobrazovat stav nabití baterie, rychlost a aktuální stav vozidla:

Text	Hodnota
SOC	`String.format("SOC: %.0f %%", etruck.get(0).soc * 100)`
SPEED	`etruck.size()>0 ? String.format("Rychlost: %.0f km/h", etruck.get(0).currentSpeed_kmh) : "–"`
ACTUAL	`etruck.size() > 0 ? etruck.get(0).statusText : "–"`

1.2.4 Ukazatel nabíjení

Pro lepší prezentaci stavu nabití baterie nyní přidáme grafický ukazatel.



Z palety **Presentation** přidáme **Rectangle**, který pojmenujeme barBg. Nastavíme mu šedou barvu, černý rámeček a pevnou velikost, Width: 140 a Height: 14.

Nyní vložíme další **Rectangle** (zde na názvu již nezáleží), nastavíme mu zelenou výplň bez orámování. Jeho pozice musíme přepnout na dynamické hodnoty, stejně jako u textu a nastavíme:

```
X:        barBg.getX() +1
Y:        barBg.getY() + 1
Width:    (barBg.getWidth() - 2) * ( etruck.isEmpty() ? 0 : Math.min(1,
          Math.max(0, etruck.get(0).soc)) )

Z-Height: barBg.getHeight() - 2
```

Toto nastavení zapříčiní to, že zelený čtverec bude nad šedým čtvercem a jeho velikost se bude měnit právě dle proměnné soc, která obsahuje hodnotu o nabití baterie.

1.2.5 Graf stavu baterie v čase

Z palety **Analysis** přidáme **Time Plot** s názvem socPlot. Do Data vložíme jednu **Value**, která bude mít Title: SoC (%) a bude obsahovat value:

```
etruck.isEmpty() ? Double.NaN : 100 * etruck.get(0).soc
```

Dále si opět nastavíme first update time na 0 seconds, tedy start modelu a Recurrence time např. na 1 minutu. Výsledný graf je vzhledem k délce času lepší prezentovat po vykopírování hodnot (po proběhu modelu kliknutím na jeho pravý horní roh) v MS Excel.

1.2.6 Graf nabíjení kW a spotřeby kW při jízdě

Z palety **Analysis** přidáme **Time Plot** s názvem powerPlot. Do Data vložíme Value, která bude mít Title: Charging kW a bude obsahovat value:

```
etruck.isEmpty() ? Double.NaN : ( etruck.get(0).inState(ETruck.Charging) ? etruck.get(0).chargerPower_kW : 0 )
```

Následně vložíme druhou Value, která bude mít Title: Drive kW a bude obsahovat value:

```
etruck.isEmpty() ? Double.NaN : (
  etruck.get(0).isMoving()
    ? etruck.get(0).consumption_kWh_per_km * etruck.get(0).currentSpeed_kmh
    : 0
)
```

Dále si opět nastavíme first update time na 0 seconds, tedy start modelu a Recurrence time např. na 1 minutu. Výsledný graf je vzhledem k délce času lepší prezentovat po vykopírování hodnot (po proběhu modelu kliknutím na jeho pravý horní roh) v MS Excel.

1.2.7 Graf vytížení

Závěrem si do modelu přidáme komplexní graf, který nám bude zobrazovat vytížení e-trucku po celou dobu provozu. Bude tedy obsahovat sloupec s nakládkou, sloupec s jízdou a sloupec s nabíjením.

Pro spuštění tohoto grafu je nejprve nutné přidat „sběrné" funkce (**Function**), které z palety **Agent** přetáhneme přímo do **Main**. Všechny funkce budou mít nastavení Returns value a type double. Jejich názvy a Function body budou následující:

Název	**Function body**

utilIdle	`double tot = 0, id = 0;`
	`for (ETruck e : etruck) {`
	`  tot += e.tTotal_h;`
	`  id  += e.tIdle_h;`
	`}`
	`return tot > 0 ? 100.0 * id / tot : 0.0;`

utilDrive	`double tot = 0, dr = 0;`
	`for (ETruck e : etruck) {`
	`  tot += e.tTotal_h;`
	`  dr  += e.tDrive_h;`
	`}`
	`return tot > 0 ? 100.0 * dr / tot : 0.0;`

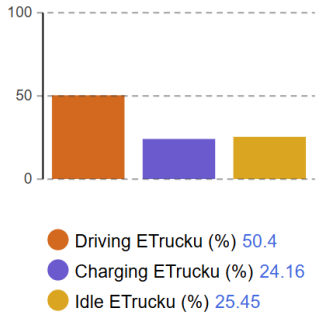
utilCharge	`double tot = 0, ch = 0;`
	`for (ETruck e : etruck) {    // etruck = populace`
	`  tot += e.tTotal_h;`
	`  ch  += e.tCharge_h;`
	`}`
	`return tot > 0 ? 100.0 * ch / tot : 0.0;`

Z palety **Analysis** poté vložíme **BarChart**, který pojmenujeme Vytížení, nastavíme mu pevné měřítko od 0 do 100, nastavíme First update time na 0 seconds a Recurrance time na 1 minutes.

Do dat grafu poté vložíme tři sloupce:

Název sloupce	**Value**
Driving ETrucku (%)	`utilDrive()`
Charging ETrucku (%)	`utilCharge()`
Idle ETrucku (%)	`utilIdle()`

Tímto získáváme komplexní časový diagram o provozu e-trucku.



1.3 EXPERIMENTOVÁNÍ

V tomto modelu stojí za to pracovat hlavně s parametry agenta `ETruck` a sledovat, jak se mění časové rozdělení mezi jízdou, nabíjením a stáním. Nejdůležitějším ukazatelem je vytížení jízdou (poměr času v pohybu vůči nabíjení a neproduktivním stavům), dále průběh SOC(t) v čase a počet dokončených cyklů A↔B.

Začněte kapacitou baterie (`batteryCapacity_kWh`). Větší kapacita obvykle prodlouží dojezd a sníží četnost nabíjení, což se projeví nárůstem podílu „Jízda %“. Pokud ale spotřeba na trasu zůstává vysoká, mohou se přínosy snižovat – sledujte zejména SOC při příjezdu a odjezdu a zda se vozidlo nepřibližuje k minimu.

Druhou důležitou skupinu tvoří prahy SOC: počáteční `soc`, minimální SOC pro odjezd `socMinToDepart` a cílový SOC při nabíjení `socTargetCharge`. Vyšší minimální SOC zvyšuje bezpečnost odjezdu, ale může prodloužit nabíjení; vyšší cílový SOC zase zvyšuje jistotu dojezdu dalšího legu, ale stojí další čas na kabelu. Při změnách těchto prahů sledujte délku stavu „Charging“, podíl „Nabíjení %“ a také případné nouzové zastavení (když SOC klesne na nulu).

Spotřeba na kilometr (`consumption_kWh_per_km`) reprezentuje zimu, protivítr, těžší náklad apod. Je-li vyšší, zvedá energii na jeden leg a nutnost častěji nabíjet. V kombinaci se vzdáleností (`distance_km`) určuje, zda je vůbec možné dojet A→B bez přestávky. Při těchto změnách se dívejte na SOC po příjezdu do B/A a na počet nabíjecích cyklů v horizontu dne.

Výkon nabíječky (`chargerPower_kW`) má přímý vliv na dobu nabíjení. Silnější nabíječka obvykle posouvá podíl „Nabíjení %“ dolů a „Jízda %“ nahoru. Pokud však vozidlo i tak tráví hodně času manipulací nebo jízdou na dlouhé trase, mohou být přínosy výkonu nad určitou hranici menší než očekávání.

Rychlost (`speed_kmh`) v našem zjednodušeném modelu neovlivňuje energii na kilometr, ale mění dobu jízdy. Vyšší rychlost zkracuje stavy „DriveToA/B“ a mění časové poměry ve vytížení. Na výsledku se velmi projeví i časy nakládky/vykládky (časové přechody, např. 30 min). Delší manipulace zvyšují „Stání %“ bez ohledu na baterii.

Při každém experimentu sledujte sadu metrik: SOC(t) a Speed(t) v čase (pro kontrolu, kdy se jede a kdy stojí), Vytížení E-trucku (Jízda/Nabíjení/Stání v %), počet cyklů A↔B za den a celkovou energii [kWh] za den. Hodí se také průměrný SOC při odjezdu a příjezdu pro posouzení bezpečnostních rezerv a evidence výskytu nouzových zastavení.

Doporučené pořadí experimentů je jednoduché. Nejprve si změřte baseline s výchozími parametry (např. 300 kWh, 1.2 kWh/km, 150 kW, 90 km, 60 km/h) a uložte si vytížení a počty cyklů. Poté vyzkoušejte silnější nabíječku a porovnejte, o kolik klesne podíl „Nabíjení %“. Následně otestujte jinou strategii nabíjení – zkuste snížit `socTargetCharge` a mírně zvednout `socMinToDepart`, čímž získáte kratší, ale častější nabíjení; porovnejte, zda se zvýší vytížení jízdou bez poklesu spolehlivosti. Dále simulujte zátěž/zimu navýšením `consumption_kWh_per_km` a delší trasu zvýšením `distance_km`; v obou případech sledujte, jestli jsou nastavené prahy SOC a nabíjecí výkon stále vhodně dimenzované. Nakonec přidejte srovnání s dieselovým truckem (bez nabíjení, případně s krátkým tankováním 1× denně) a vyhodnoťte rozdíl ve vytížení jízdou a v počtu cyklů.

Aby byly závěry srozumitelné, vždy měňte jen jeden parametr najednou, hodnoty SOC udržujte v intervalu 0–1 (tj. 0–100 %) a zapisujte časy konzistentně v minutách/hodinách podle nastavení modelu. Pokud se později rozhodnete model rozšířit o více vozidel, lze stejnou sadu metrik sledovat agregovaně a porovnat rozptyl mezi jednotlivými e-trucky.