# Map Builder

## Inleiding

De Map Builder is een losstaand deel van het project. Alles van de mapbuilder draait lokaal. Het is een onepager website die bestaat uit javascript, html en css. Aangezien dit de eerste iteratie is van een mapbuilder werd er gefocust op de functionaliteit. Met de mapbuilder kunnen alle teams (backbone, car, drone & robot) genieten van gegenereerde sql (in tekstvorm) die onderling op elkaar is afgestemd. Bijkomend wordt er een json-file gegenereerd die ter beschikking wordt gesteld voor de backbone (dit wordt door de MaaS opgehaald voor de visualisatie). Tijdens het bouwen van de map is er ook een grafische representatie ter beschikking.

## Technisch

Het project bestaat uit volgende delen:

* car.js, drone.js, robot.js & top.js: Hier komen de functies per type map.
* Event.js handelt de events van de index.html zoals de copy buttons, offsets savebuttons en headerbuttons.
* Script.js, dit is de start van de javascript.
* Sql.js, hier wordt de sql tekst gegenereerd voor de verschillende teams (top, car, drone & robot).
* Index.html, dit is de html van de pagina onderverdeeld in drie headers map, data en sql.
* Data.js: Hierin komen de variabele die nodig zijn voor het bouwen van de map.
* Style.css, dit is de stylesheet van het project.
* JQuery UI library onder /lib
* JQuery, bootstrap en fontawesome worden gebruikt door middel van een internetlink.
* /img folder voor een afbeelding die de verschillende robot tegels weergeeft met id.

### Car.js

Hieronder de verschillende functies voor de car-map:

* addCarNode, een car node wordt toegevoegd en meteen gelinkt aan alle andere nodes.
* deleteCarNode, een car node wordt gedelete en zijn links worden verwijderd en als dit een transit point is dan wordt zijn topnode verwijderd.
* updateCarNode, hierbij worden de x,y,z,w,mapname en isTransit van de node geüpdated. Zodra de isTransit van waarde veranderd, wordt de topmap geüpdated.
* createCanvasEventsCar, hier worden de volgende events vastgelegd:
  + Dubbelklikken op een lege plaats 🡪 nieuwe node toevoegen.
  + Dubbelklikken op een node 🡪 deze node wordt verwijderd.
  + Rechterklik op een node 🡪 dialoog verschijnt om waarde van deze node aan te passen.
* launchCarDialog, dit geeft een dialoog die vraagt naar de waarde van de node. Zodra deze dialoog sluit zal de desbetreffende node worden veranderd met de ingevulde gegevens.
* drawCanvasCar, er wordt een wit kader getekend, dan de links (drawCarLinks) en dan de nodes (drawCarNodes).

Speciaal: Car map heeft een een offset en een schaal. Omdat de gebruikte x en y in de databank kleinere waardere hebben dan de grote van de canvas. Er werd beslist om dit geschaalt te houden zodat het grafisch overeenkomt met de werkelijke positie.

### Drone.js

Dit is zeer gelijkaardig aan de car.js. Het enige verschil is dat je hier wel een node van plaats kan veranderen door te draggen. Bijkomend wordt er niet gevraagd naar de w-waarde en mapname van de node.

Ondanks dat er weinig verschil is met de car map en er dus veel code gemeenschappelijk is, werd er toch besloten om dit in een andere file te houden om het overzicht te behouden mocht er één van de teams hun map veranderen.

### Robot.js

Dit is de moeilijkste map en bestaat uit twee delen, een frontend en een backend. In de frontend worden er tegels gelegd aan de hand van het type tegel worden er nodes gelegd in de backend. In de backend kan je dan de nodes verbinden om zo tot een map te komen. Robot.js bestaat uit het volgende:

* drawCanvasRobot, tekent een wit vierkant op de frontend en backend en gaat vervolgens tegels (drawTiles), nodes (drawRobotNodes) en links (drawRobotLinks) tekenen.
* drawRobotLinks is de enige draw link functie waarbij de richting uitmaakt. Hiervoor worden er dus pijltjes gemaakt ipv lijnen. De punten om de pijl te tekenen zijn de randen van de node die door getBorderPoints worden bepaald in tegenstelling tot de x,y-waarden van de node. Bij het tekenen van de links worden er ook cirkels getekend in het midden van de link, deze cirkel posities worden bijgehouden om zo een link te kunnen selecteren.
* drawTiles, aan de hand van het type wordt het correcte getekend. Deze worden met kleine functies opgebouwd (drawUpBottom, drawUp, drawLight, …). Verkeerslichten hebben een cirkel als verkeerslicht.
* createCanvasEventsRobotFrontEnd
  + Linkerklik op een lege plaats 🡪 toevoegen van een tegel.
  + Linkerklik op een tegel 🡪 aanpassen van deze tegel.
  + Rechterklik op een tegel 🡪 deze tegel wordt verwijderd.
* createCanvasEventsRobotBackEnd
  + Rechterklik link 🡪 delete link
  + Linkerklik link 🡪 edit link
  + Drag van node tot node 🡪 link leggen tussen de nodes
* editRobotLink, hier worden de links aangepast. Als hetdezelfde tegel is dan is de lengte 0 en wordt er gevraagd naar de hoek. Zoniet dan wordt er gevraag naar de lengte en is de hoek 0. Bijkomend wordt er ook gevraagd naar het gewicht van de link.
* deleteRobotLink, de robotLink wordt verwijderd.
* addRobotLink, op zelfde wijze als editRobotLink wordt er naar de juiste gegevens gevraagd en een link gelegd tussen de begin -en eindnode.
* createTile, met de x en y-waarde van de geklikte locatie wordt de locatie van de nieuwe tegel bepaald. Het is de bedoeling dat iedere tegel aan elkaar aansluit. Om dit te bereiken wordt de dichtsbijzijnde tegel van de aangeklikte locatie gezocht en aan deze gevonden tegel, wordt de nieuwe tegel gevoegd.
* deleteTile, de tegel en zijn bijhorende nodes worden verwijderd.
* editTile, hier wordt de tegel aangepast, aan de hand van het canvastype wordt het tiletype en aantal nodes correct gezet. Kruispunten hebben 8 nodes, T-splitsingen hebben 6 nodes en endpoints & verkeerslichten hebben 1 node.
* launchTileDialog, er wordt gevraagd naar het type van de tegel en zijn rfid. Alleen tegels die nodes bezitten hebben rfid tags nodig (zie future works). Bijkomend wordt er een image getoond waarop de canvastypes worden getoond met bijhorend nummer.
* deleteRobotNodes, de robot nodes en hun links van een tegel worden verwijderd. Als de node een transit point is, dan wordt ook de corresponderende top node verwijderd.
* makeTileIds, dit is een functie die ids geeft aan de tiles die nodes hebben.
* createRobotNodes, tiles die nodes hebben, worden overlopen. Hier worden ook de x -en y-waarden bepaald van de nodes aan de hand van de frontend tegel. Voor een T-splitsing wordt er gekeken aan welke zijde er 2 nodes wegvallen. Als het type een endpoint is dan worden er top nodes aangemaakt.

### Top.js

Deze file bestaat uit volgende functies:

* addTopNode & deleteTopNode, maken en verwijderen top nodes. Deze functies worden uitgevoerd wanneer de gebruiker een transit maakt of verwijdert op een andere map.
* moveTopNode, hierbij krijgt de node een nieuwe positie.
* addTopLink, er wordt een link met gewicht 1 gelegd tussen twee nodes.
* deleteTopLink, de geselecteerde link wordt verwijderd.
* editTopLink, het gewicht van de link wordt aangepast.
* createCanvasEventsTop, de volgende events worden aangemaakt:
  + Linkerklik link 🡪 edit link.
  + Rechterklik link 🡪 verwijder link.
  + Drag node naar nieuwe locatie 🡪 move node.
  + Drag van een node naar een andere node 🡪 nieuwe link wordt aangemaakt.
* drawCanvasTop, er wordt een witte rechthoek getekend over de canvas en de links (drawTopLinks) en nodes (drawTopNodes) worden opnieuw getekend.

### Script.js

Hier worden enkele globale variabele gedeclareerd die je kan veranderen zonder de data.js aan te passen. Voor derest heb je volgende functies:

* $(document).ready, nadat de html is geladen worden de contexten voor de canvassen juist gezet, ook wordt hun grootte en font hier bepaald. Mocht de grootte hier aangepast worden, dan moet de drawCanvas functie van de desbetreffende canvas(sen) hun witte getekende rechthoek ook in grootte aanpassen.
* Main, hier wordt eenmalig de mapheaderbutton gehighlight, de map buttons toegevoegd, de canvasevents aangemaakt en de map geladen op de canvas.
* Refresh, deze functie wordt het beste uitgevoerd nadat er gegevens met betrekking tot de canvas (nodes, links, …) worden aangepast. Deze gaat namelijk de canvassen opnieuw tekenen en de nieuwe data en sql laten zien.
* addMapButtons, hier worden de map buttons toegevoegd, deze buttons krijgen ook een event mee.
* addMap, dit maakt een nieuwe map aan en de buttons worden opnieuw gemaakt.
* loadMap, als men op de buttons klikt om van map te veranderen, wordt deze nieuwe map geladen op de canvas en de juiste html elementen waaronder de canvas getoond.
* displayData, maakt de data.js file om de huidge map op te slaan. Bijkomend genereerd deze functie ook de geconverteerde json. Deze gegeneerde teksten worden weergegeven onder de header data.
* converDataToJson, dit wordt gebruikt in de displayData functie. Deze functie zal alle data omzetten in een json die aan de backbone wordt gegeven die dan kan worden opgehaald door de MaaS. Wees voorzichtig bij het aanpassen van deze funtie en stem dit af op de visualisatie van de MaaS. Lees future works voor tekortkomingen van deze functie op de robot map.
* displaySQL komt voor in de refresh functie en verwijst naar de sql.js.
* drawCanvas, aan de hand van het type wordt de juiste canvas functie aangeroepen.

Bijkomend zijn er extra functies om andere vooraf opgelijste functie te supporteren

* onlyUnique, zorgt voor unieke waarde in een array.
* getMousePos, geeft de x -en y waarden mee van de geklikte positie op een bepaalde canvas.
* getNewId, gaat een nieuwe id gegenereren voor een dataset, zodat deze oplopend blijven en gaten in opeenvolgingen wegwerkt.
* getItemWithId, gaat uit een lijst het item teruggeven met de overeenkomende id.
* deleteIdFromData, gaat uit een lijst het item verwijderen dat overeenkomt met de id.
* editIdFromData, gaat uit een lijst het item zoeken die overeenkomt met de id en deze vervangen door het gegeven object.
* toIntOrZero, dit probeert een input te parsen naar een integer en als dit niet lukt dan wordt het 0. Dit is vaak gebruikt in combinatie met prompt voor user input.
* getValuesWithMapId, filtert een dataset op de map id.
* getBorderPoint(s), dit zijn wiskundige methode om snijpunten te vinden met de rand van de nodes. Zodat er een pijl kan getekend worden bij de robot links.
* findElement, dit zoekt naar een element met bepaalde groote op de canvas in een lijst.

### Data.js

Hier worden alle variabele gezet die worden gebruikt om de mappen te bouwen. Als je aan de map gewerkt hebt end it wilt opslagen, moet je de data.js tekst kopiëren onder de data header en deze in de gebruikte datafile plakken. De gebruikte datafile wordt gelinkt in de index.html:

<script type="text/javascript" src="**data.js**"></script>

Dit zijn de verschillende variabele:

**Maps:** id, type {TOP, ROBOT, CAR, DRONE}, offsetX, offsetY (Offsets worden gebruikt voor het genereren van de json)

**Top\_nodes**: id, mapId, edgeId, x , y

**Top\_links:** id, start, stop, weight

**Car\_nodes:** id, x, y, car\_x, car\_y, car\_z, car\_w, isTransit, mapId, mapname (x, y hangen geschaald af van de gegeven car\_x, car\_y)

**Car\_links:** id, start, stop, mapId (Op de moment enkel gebruikt ter visualisatie.)

**Drone\_nodes:** id, x, y, drone\_x, drone\_y, drone\_z, isTransit, mapId (zelfde wijze als de car)

**Drone\_links:** id, start, stop, mapId (Ook op de moment enkel gebruikt ter visualisatie.)

**Robot\_tiles:** id, x, y, canvastype, mapId, isTile, rfid, tiletype {crossing, end, light}, nodes, tileId (Nodes, tiletype, isTile, rfid hangen af van het canvastype.)

**Robot\_nodes:** id, tileId, x, y, mapId, isTransit (TileId is gelinkt aan robot\_tiles)

**Robot\_links:** id, start, stop, startTile, stopTile, angle, weight, length, mapId

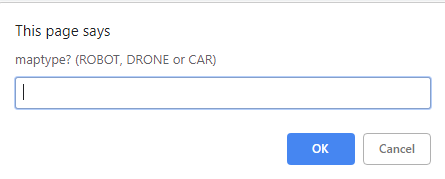
## User manual

Van een lege map starten doe je door dit in de data.js file te hebben: var maps = [{"id":0,"type":"TOP}] en derest = [].

Vervolgens druk je op:



Dan komt er een popup waar je het type meegeeft:



Vervolgens komt de nieuwe map tussen de lijst:



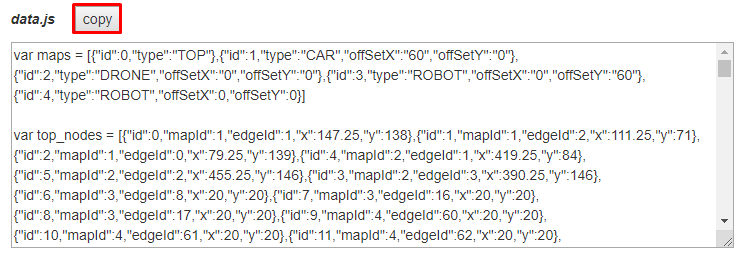
Door op de verschillende buttons te tikken, switch je van map.

Bij iedere map staat er instructies hoe je tiles, nodes of links aanmaakt, aanpast of verwijderd.

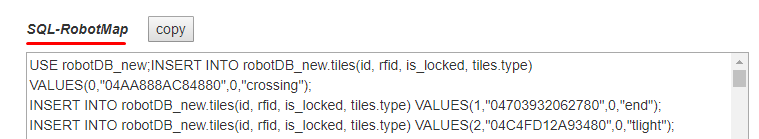
Bij de robot map moet je echter opletten om eerst de tegels te maken en dan pas de linken op het tweede canvas te leggen om eventuele bugs die nog aanwezig zijn te vermijden.

Nadat je deze map hebt gemaakt zullen er op de top map nodes bijgekomen zijn van de transit points van de net gecreërde maps. Vervolgens kan je de top map links aanpassen.

Als alle maps zijn gemaakt, kan je deze map opstelling bewaren door de data.js tekst te kopiëren naar de gebruikte datafile.



Als laatste komt het verspreiden van de gekregen tekst files, 1 json (onder de tab data) en 4 sql teksten (onder de tab sql). Bij de SQL staat er voor welk team dit is:



## Restricties

In de variabele maps van de data.js moet de eerste map op het type top staan en er mag slechts 1 top map zijn.

Bij het maken van de robotmap moeten eerst de tiles getekend worden en vervolgens de links tussen de points gelegd. Dit is belangrijk om bugs te vermijden die voorkomen bij het maken van de robot points. (zie future works)

Er zijn enkele variabele die je kan veranderen zoals de grootte van nodes. Vooraleer je deze veranderd moet de datafile op de variabele maps na leeggemaakt worden. Want zodra er variabele veranderen, zullen er problemen komen voor de x -en y-posities.

## Conclusie

Er zitten nog enkele restricties in die de functionaliteit van de map builder kunnen belemmeren, maar als er een juiste workflow gebruikt wordt zijn deze te vermijden. Aangezien het de eerste iteratie is van de map builder, is er een grote oppurtiniteit om dit te verbeteren (beschreven in future works). Ondanks dat het de eerste iteratie is, is de functionaliteit waarvoor de map builder is gemaakt reeds aanwezig.

## Future Work

Herschrijven hoe robot nodes worden aangemaakt. Op dit moment wordt dit iedere keer aangemaakt nadat de robot canvas wordt getekend en uit de getekende robot tegels worden steeds de nodes aangemaakt. Dit zorgt voor de volgende bugs:

* Geen meerdere mappen van het type robot mogelijk.
* Het deleten van robot tegels nadat er links zijn gelegd, zorgt voor verkeerde links.
* Top nodes die van de robot map komen worden steeds opnieuw aangemaakt, waardoor ook hun x,y terug op de startpositie komen.

Bekijken om de map builder te integreren op een server. Op dit moment is het een los staand programma, waar tekst (sql/json) uit komt die dan wordt doorgegeven aan de desbetreffende teams. Idealiter zou de map gemaakt worden en de databanken van de teams rechtstreeks updaten met de nieuwe map.

Save functie verbeteren. Op dit moment kan je een map maken, maar zodra je de browser sluit ben je alle gegevens kwijt. Om dit te vermijden kan je de data.js file in tekst ophalen op de webpagina en manueel zelf de datafile updaten. Van zodra de map geïntegreerd wordt, zou het mooi zijn mocht deze automatisch een data.js file updaten die online beschikbaar is op de server.

De website gebruiksvriendelijker te maken. Doordat er gefocust werd op de functionaliteit is de website niet zo gebruiksvriendelijk. Dit zou opgelost kunnen worden door betere events (zoals een zoom-functie) en/of betere user interface. Een voorbeeld van betere events, is het selecteren van het canvastype van de robottegel. het zou aangenamer zijn om de juiste tegel te selecteren in plaats van een id te type. Een voorbeeld van betere user interface is dat de gebruiker bepaald welke gegevens hij/zij wilt zien (zoals het kunnen wegvinken van de display van de linkweights). Ook de gegevens die worden weergegven zijn niet dynamisch genoeg weergegeven.

Het maken van de robot map gebeurt door tegels te leggen en dan de nodes van deze tegels op een ander canvas te verbinden. Het zou aangenamer zijn mochten er alleen tegels gelegd worden, die achterliggend de nodes correct verbinden. Om dit te verwezelijken zullen er meerdere tegels moeten geïntroduceerd worden waarop een richting staat aangegeven.

Alleen robot tegels die nodes genereren hebben een rfid tag nodig, toch wordt er bij het veranderen van andere tegels gevraagd naar een rfid, die achterliggend niet gebruikt worden. In de toekomst zou er een methode nodig zijn waar je alleen vraagt naar het rfid als dit ook effectief nodig is om verwarring te vermijden.

Mocht er een top, car, drone of robot map veranderen met hun databank, moet de map builder meeveranderen naar de nieuwe versie van deze map.

In de converDataToJson functie (in script.js) wordt de robot pointlist op andere wijze gegenereerd dan de andere maps. Er is een optie gehouden om de tileIds of pointIds terug te geven. De tileIds werken correct, maar is niet geïmplementeerd op de robots zelf. De pointIds worden alleen gebruikt van verkeerslichten en endpoints, omdat deze tegels 1 node hebben, waardoor we de buren kunnen bepalen. Deze oplossing is onvoldoende 1 van de 2 opties moet gekozen worden en dit moet gezamelijk afgestemd worden op de MaaS, backbone en robots. Als er voor tildIds gekozen wordt in de pointlist dan moet de robot kunnen rijden van tile tot tile. Anders zal je alle buren van iedere points moeten bepalen en dan moet de visualisatie van de MaaS dit opvangen door tegels te voorzien. Tot op heden hebben beide teams niet de capaciteit om dit probleem op te lossen dus vandaar deze onvoldoende oplossing.

Offsets verbeteren, ideaal zou zijn mochten de maps hun transit points elkaar overlappen in plaats van offsets te geven aan de maps. Bespreek dit met de visualisatie van de MaaS om dit deel te verbeteren.