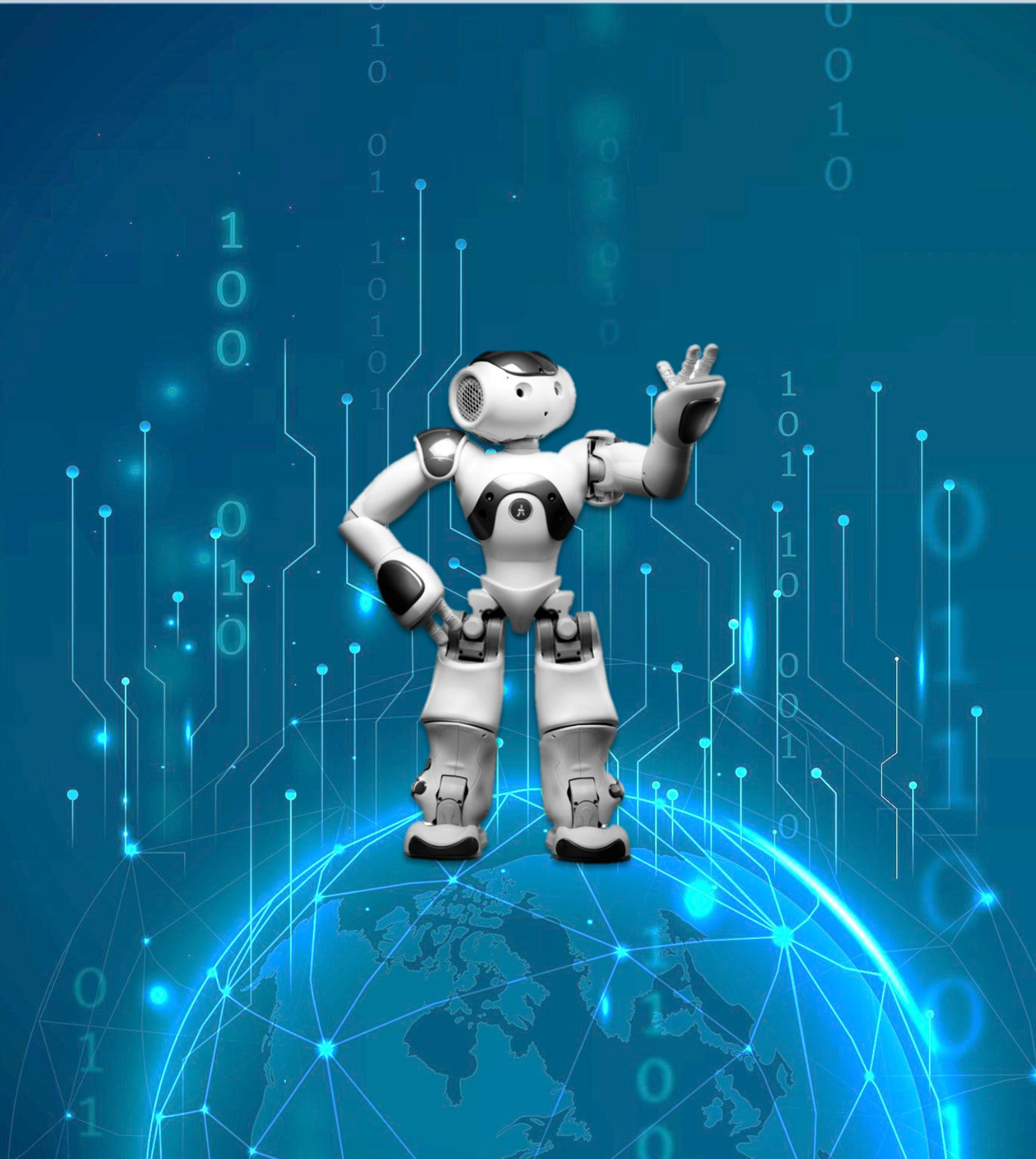
Robotikas rokasgrāmata



ROBOTIKAS ROKASGRĀMATA

Rokasgrāmata izstrādāta Interreg V-A Latvijas – Lietuvas pārrobežu sadarbības programmas 2014. – 2020. gadam projekta LLI-542 "IT programmēšanas un robotikas kompetenču attīstīšana pārrobežu reģiona skolās Latgalē, Visaginā un Ignalinā" ietvaros (akronīms: RoboNet)

Pasūtītājs: Latgales reģiona attīstības aģentūra

Izpildītājs: SIA Goodman Group

Autori: Mārtiņš Zemlītis

Jevgeņijs Kružkovs

Kopējās projekta LLI-542 "IT programmēšanas un robotikas kompetenču attīstīšana pārrobežu reģiona skolās Latgalē, Visaginā un Ignalinā" (akronīms: RoboNet) izmaksas ir 369 330 EUR. Projekta līdzfinansējums no Eiropas Reģionālās attīstības fonda ir 313 930 EUR. Šī rokasgrāmata ir sagatavots ar Eiropas Savienības finansiālo atbalstu. Par šīs rokasgrāmatas saturu pilnībā atbild Latgales reģiona attīstības aģentūra, un tas nekādos apstākļos nav uzskatāms par Eiropas Savienības oficiālo nostāju.

Rokasgrāmatas **elektroniskais formāts** pieejams:

http://visitlatgale.com/lv/latgale/lraa/projekts_robonet

Video materiāli šai grāmatai pieejami YouTube kanālā: **Robotika – RoboNet**

Detalizēta informācija par RoboNet projekta aktivitātēm pieejama projekta mājas lapā:

www.visitlatgale.lv

Saules iela 15, Daugavpils, Latvija

E-pasts: lraa@latgale.lv

EIROPAS SAVIENĪBA

SATURS

1. Ievads Programmēšanā

Dažkārt vislabākais veids, kā mācīties, ir uzdot jautājumus un pašam meklēt atbildes. Tāpēc pirmais jautājums, ko sev vajadzētu uzdot ir: "Kas ir programmēšana?", "Ko ES saprotu ar šo vārdu?" un "Kādu procesu tas apzīmē?" Tāpēc, pirms turpināt aplūkot tekstu tālāk, ir svarīgi par šo padomāt. Vislabāk ir **diskutēt grupās**. Varbūt šis vārds programmēšana vai programma ir pazīstams citā formā? Kādi vēl saistīti vārdi vai termini līdz šim ir dzirdēti?

Par datorprogrammēšanu sauc procesu, kurā datoram tiek dota virkne instrukciju, kuras tam ir jāveic. Biežāk vārda "datorprogrammēšana" vietā tiek lietots vārds "programmēšana". Cilvēkus, kuri nodarbojas ar programmēšanu sauc par programmētājiem.

Jautājums: Varbūt līdz šim ir dzirdēti vēl kādi programmētāja pienākumi, kas nav vienkārši programmas veidošana? Gadījumā, ja nav dzirdēts, būtu vērts padomāt, ar ko nodarbojas programmētājs un kādi varētu būt viņa pienākumi.

Programmētāji nodarbojas ne tikai ar programmas izveidi, bet arī ar programmas plānošanu un tās darbības pārbaudi vai testēšanu. Ir svarīgi ne tikai izveidot programmu, bet arī pirms tam izplānot tās darbības principu. Bieži vien, nedaudz veltot laiku programmas plānošanai, ir iespējams laicīgi pamanīt problēmas, kuras varētu palaist garām, ja uzreiz sāktu programmēt. Kā arī pēc programmas izveides, ir svarīgi pārliecināties, vai programma izstrādāta pareizi, aplūkojot pašu programmu vai aplūkojot robotu, vai tas darbojas pareizi ar šo programmu.

Jautājums: Tagad ir zināms, kas ir programmētājs un ko tas dara. Taču nav zināms, kas ir programma. Tas ir kaut kas, ko veido programmētājs, varbūt ir kādas idejas, kas tas varētu būt? Kā tas izskatās? Vai programmu varētu veidot dažādās formās?

Par programmu var saukt kādu instrukciju sarakstu, kuras ir jāveic datoram. Šīs instrukcijas var būt gan rakstiskā veidā vārdformās, gan grafiskā formā, kur katru komandu reprezentē kāda figūra. Ja tiek runāts par programmu, kas ir rakstīta vārdformās, tad ir vērts pieminēt šādu programmu dažādību. Tas nozīmē, programmu var uzrakstīt dažādās programmēšanas valodās. Līdzīgi kā sarunvalodā, var sarunāties dažādās valodās, tāpat arī ir programmēšanā. Var veidot programmas dažādās programmēšanas valodās.

Atšķirība ir tā, ka ikdienā sarunājoties var patvaļīgi mainīt valodu, kādā notiek saruna, taču programmējot, ja programmu izveido konkrētai programmēšanas valodai, tad šo valodu

arī ir jāizmanto šajā programmā. Kā arī, dažādām programmēšanas valodām var būt konkrēti pielietojumi. Tas ir, piemēram, programmēšanas valodu *R i*zmanto statistikai un Datu analīzei, bet *JavaScript* tiek lietots piemēram interaktīvu tīmekļa vietņu izveidē, savukārt programmēšanas valoda *Python* bieži tiek lietot mākslīga intelekta izstrādei. Taču tas nenozīmē, ka šie ir vienīgie katras programmēšanas valodas pielietojumi.

Runājot par programmēšanas valodu pielietojumu, dažas jau tika pieminētas. Piemēram, aplūkojiet savu datoru, uz tā jau strādā kāda programma. Tā tikpat labi var būt datora pašreizējā operētājsistēma, kuru darbina kāda iepriekš rakstīta programma. Taču tā var būt arī mājas lapa, kuru darbina kāda programma. Patiesībā, programmēšana ir lietota ļoti plaši.

2. Robota uzbūve

Lai piepazītos ar programmēšanu un iemācītos programmēt, mēs šajā grāmatā izmantosim robotu komplektu *LEGO Edication Spike Prime* un to grafisko programmēšanas vidi.

Iepazīsimies ar mūsu robotu. *Lego Spike Prime* komplektā ietilpst vairāki motori, sensori, galvenais bloks un vairākas LEGO detaļas. Ar LEGO detaļām, kas ietilpst komplektā ir iespējams iepazīties pašā komplektā, kuram līdzi nāk arī pieejamo detaļu saraksts. Šī nodaļa tiks veltīta nelielam sensoru un motoru apskatam.

Robotu komplektā ietilpst 3 sensori, 2 veidu motori un galvenais bloks ar bateriju.

2.1. Galvenais bloks

LEGO Eucation Spike Prime galvenais bloks (skat.att.2.1.) kalpo kā robota "Smadzenes". Tieši šis bloks vada mūsu robotu, izpilda nepieciešamās programmas un komandas. Pie tā ir iespējams pievienot citus komplekta elementus , tādus kā motorus un sensorus, izmantojot komplektā esošos speciāli tam paredzētos vadus, un pēc tam to programmēt. Pie galvenā bloka ir iespējams pievienot līdz 6 sensoriem vai motoriem. Bloka katrā pusē ir pa 3 speciāli tam paredzētiem portiem. Uz galvenā bloka virsmas ir pieejama 5x5 LED matrica un 3 pogas. Papildus galvenajā blokā ir iebūvētas arī citas ierīces, tādas kā skaļrunis un 6 asu žiroskops.

Att.2.1. Lego Spike Prime galvenais bloks

2.2. Krāsu sensors

Kā ir saprotams no šī sensora nosaukuma, tas spēj noteikt kāda objekta krāsu, bet ne tikai. Krāsu sensors (skat.att.2.2.) vēl var noteikt atstarošanos vai apkārtējo gaismu. Sensora vada garums ir 25cm .

Krāsu sensors spēj atšķirt, melnu, baltu, sarkanu, zilu, zaļu, dzeltenu, violetu vai gaiši zilu krāsu vai kad tas neredz nevienu.

Gaismas atstarošanos sensors vislabāk spēj noteikt 16mm no objekta, taču tas var mainīties no objekta izmēra un virsmas.

Att. 2.2. Krāsu sensors

2.3. Attāluma sensors

Ar attāluma sensoru (skat.att.2.3.) ir iespējams veikt attāluma mērījumus. Sensors spēj izmērīt attālumu līdz objektam vai virsmai, izmantojot ultraskaņas sensoru. Papildus sensoram ir gaismiņas ap "acīm", kas ir sadalīta divos segmentos ap katru "aci", kurus var aktivizēt atsevišķi. Sensora vada garums ir 25cm.

Att. 2.3. Attāluma sensors

Sensors var veikt "ātro" objektu noteikšanu 50mm-300mm un 50mm - 2000mm. Tas nozīmē, ka sensors nespēj noreaģēt uz objektiem, kas atrodas lielākā attālumā. Sensors spēj uztvert objektu 35 grādu leņķī.

2.4. Spēka sensors

Sensors (skat.att.2.4.) spēj noteikt vienkāršus pieskārienus un spēku, ar kādu uz to spiež. Sensora vada garums ir 25cm. Sensoram pieskaroties, tas spēj noteikt pieskārienu, ja uz sensoriem uzspiež no 0 līdz 2mm. Tas tiek uztverts, kā vienkārši pieskāriens. Spiežot tālāk, no 2mm līdz 8mm, ir jāpielieto attiecīgi spēks, kas ir no 2.5N līdz 10N. Sensors spēj nolasīt, kāds spēks tiek tam pielikts.

Att. 2.4. Spēka sensors

2.5. Motori

Robotu komplektā *LEGO Education Spike Prime* ietilps 3 motori: 2 vidēji (skat.att.2.5) un 1 liels motors (skat.att.2.6). Tie atbild par tādām kustībām, kā piemēram riteņu darbība.

Att. 2.5. Vidējais motors

Att. 2.6. Lielais motors

Abi var kalpot gan kā motors, gan kā sensors. Tie spēj noteikt, cik daudz tas ir pagriezts. Motora vada garums 25cm. Vidējais motors spēj veikt 185 apgriezienus minūtē, ja tam nav pievienota masa, lielais motors spēj veikt tikai 175 apgriezienus. Taču, pie maksimālās slodzes

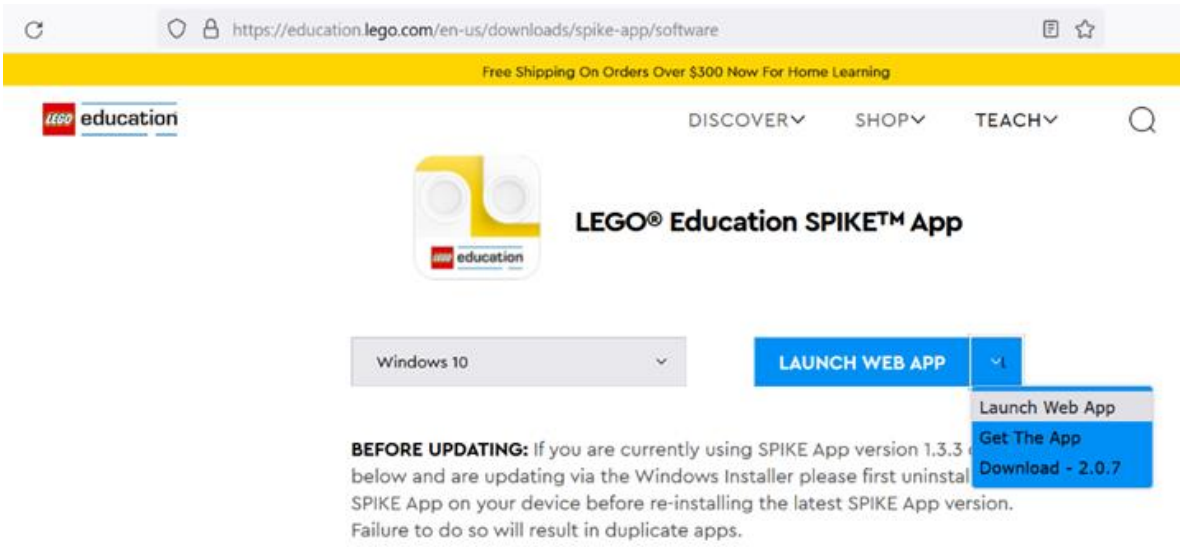
abi motori spēj veikt vienādu apgriezienu skaitu minūtē: 135 apgriezienus minūtē. Jāpiebilst, ka lielais motors ir “spēcīgāks” un spēj pavilkt lielāku masu.

Motoru un sensoru precīzas specifikācijas un to parametrus var atrast oficiālajā LEGO mājas lapā www.lego.com

3. Programmēšana un vide

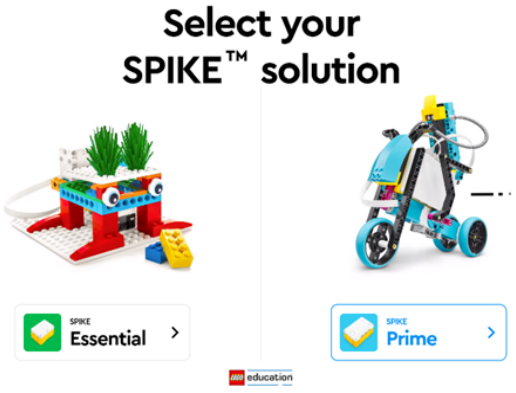
3.1. Programmēšanas vide

LEGO Education Spike prime robotu programmēšana tiek veikta lietotnē ar nosaukumu – *"LEGO Education SPIKE"*. Lietotnei ir pieejama versija interneta pārlūkā, taču to ir arī iespējams instalēt uz savas ierīces. Programmēšanas vides lapa ir pieejama pēc adreses http://education.lego.com/en-us/downloads/spike-app/software/ (skat.att.3.1)



Att. 3.1. Lego Spike programmēšanas vides izvēle

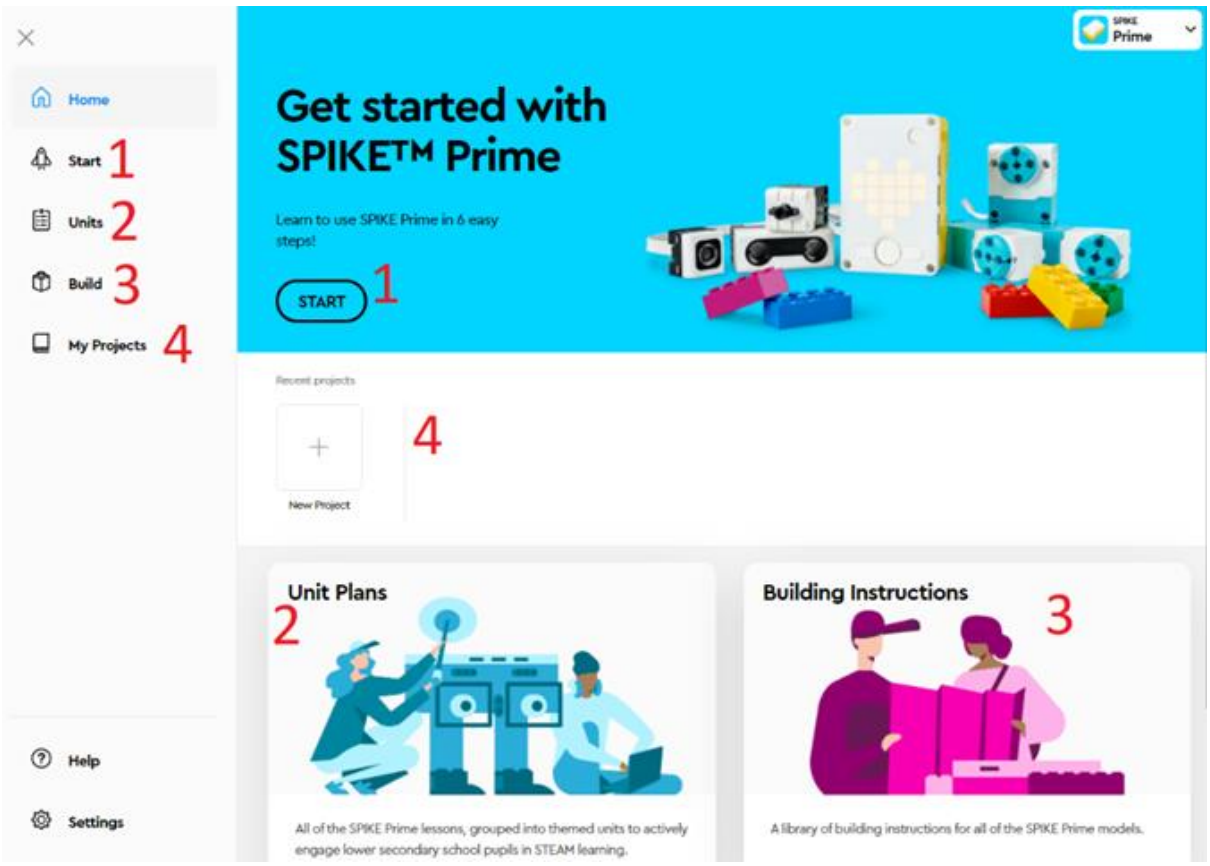
Izvēlamies kādu no metodēm, kura šķiet piemērotāka. Tālāk, kad lietotne ir vai nu instalēta, vai nu atvērta interneta pārlūkā, to atverot, parādās logs, kurā ir jāizvēlas vienu no divām opcijām atbilstoši Jums esošam komplektam (skat.att.3.2).



Att. 3.2. Lego Spike komplekta izvēle

Gadījumā, ja Jums ir pieejams komplekts *LEGO Spike Essential*, Jūs varat izmantot šo vidi un turpmāk programmēt robotu, kurš tiek būvēts ar šī komplekta palīdzību. Tā kā mēs šajā grāmatā izmantosim aprakstiem un uzdevumiem komplektu *LEGO Spike Prime*, tad no diviem piedāvātajiem variantiem izvēlamies *Spike Prime*.

Kad tika izvēlēta atbilstoša programmēšanas vide (*Spike Prime*), tad uz ekrāna parādīsies šāds logs (skat.att.3.3.)



Att. 3.3. Lego Spike Prime starta logs

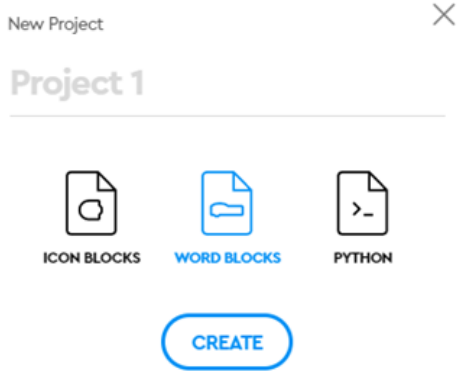
Atvērtais logs ir sadaļā *"Home"*. Attēlā (skat.att.3.3.) ar vienādiem cipariem ir atzīmētas sadaļas, kas veic vienu un to pašu darbību, tas nozīme, ka pārvieto lietotāju uz atbilstošu sadaļu. Katrs no Jums var izvēlēties sev piemērotāko metodi, kādā veidā var nokļūt atbilstošā sadaļā.

Sadaļa *START*, kas ir atzīmēta ar ciparu 1 (skat.att.3.3.) var atrast vairākas pamācības par robotu komplektu un iepazīties ar tā darbību. Šajās pamācībās ir iekļauti paši pamati *Lego Spike Prime* programmēšanā, kā arī sensoru un motoru lietošanas instrukcija. Tajās ir aprakstīts, kā strādā katrs no sensoriem un kā darbināt motoru. Katru no pamācībām var veikt pāris minūtēs, taču ir ieteicams turpināt izpētīt katra sensora vai motora darbību, turpinot mainīt dažādus parametrus, lai labāk izprastu darbības principu.

Sadaļā *UNITS*, kas ir atzīmēta ar ciparu 2 (skat.att.3.3.) ir iespējams atrast vairākas nodarbības un uzdevumus patstāvīgam darbam, kuras ir krietni lielākas, nekā *"START"* sadaļā. Tajā ir vairākas lielas nodaļas, kur katrā no tām ir vairākas nodarbības. Pie katras nodarbības ir sniegts arī aptuvenais laiks, cik tiks patērēts uz katru no tām. Svarīgi piebilst, ka šeit būs nepieciešams konstruēt robotu, kas katram uzdevumam var atšķirties. Lai gan daļai no uzdevumiem, konstrukcija tiks piedāvāta no vidē pieejamajām, nevienā no uzdevumiem nav aizliegt izveidot savu konstrukciju. Galvenais, lai tas netraucē veikt uzdevuma nosacījumus.

Sadaļā *BUILD* (skat.att.3.3.) ir pieejamas vairākas robotu būvēšanas instrukcijas. Daļa no šīm instrukcijām tiks izmantotas turpmāk tekstā. Atverot šo sadaļu vajadzētu parādīties vairākiem logiem uz kuriem būtu uzzīmēts robots un tā nosaukums. Uzklikšķinot virsū atveras izvēlētā robota būvēšanas instrukcija. Gadījumā, ja atverot šo sadaļu neparādās neviens logs ar robotu, nepieciešams atvērt sadaļu *UNITS* un noklikšķināt uz katru no logiem. Tas instalēs attiecīgo nodarbību un tajā iekļautās robotu būvēšanas instrukcijas.

Sadaļā *MY PROJECTS*, kas ir atzīmēta ar ciparu 4 (skat.att.3.3.), ir atrodami visi iesāktie projekti. Svarīgi piebilst, ka *HOME* sadaļā parādīsies tikai pēdējie projekti, taču *MY PROJECT* sadaļā būs atrodami visi projekti.



Att. 3.4. Jaunas programmas izveides logs

Tālāk mēs apskatīsim, kā izveidot jaunu projektu un *LEGO Spike* programmēšanas vidi. Lai uzsāktu darbu ar robotu un izveidotu projektu nepieciešams uzklikšķināt uz ikonu "*NEW PROJECT*" un tālāk atvērsies logs (skat.att.3.4)

Lai piešķirtu savam projektam nosaukumu, nepieciešams uzklikšķināt uz vārda *"Project 1"*, kas atrodas rakstlaukumā un tajā jāievada sava projekta nosaukums. Pretējā gadījumā projekts tiks saglabāts, ar nosaukumu *"Project 1"*. Nosaukumu būs iespējams arī vēlāk nomainīt. Nākošais solis - nepieciešams izvēlēties kādu no programmēšanas veidiem. Atkarībā no tā, kāds programmēšanas veids tiks izvēlēts, būs atkarīga arī programmēšanas vide. Katram programmēšanas veidam tās ir dažādas. Ir iespēja izvēlēties programmēšanu ar blokiem vai programmēšanu izmantojot programmēšanas valodu.

Mums ir iespēja izvēlēties kādu no veidiem:

1. *Icon block* - programmēšana ar blokiem, kad bloki tiek izvietoti horizontāli;
2. *Word bloks* - bloki tiek salikti vertikāli, viens zem otra;
3. Programmēšanas valodā ***Python.***

Mācību procesam vislabāk saprotams un uzskatāms ir *Word Block* programmēšanas veids, kad blokus savieno vertikāli, vienu zem otra un ir vairāk attēlota papildus informācija par bloku parametriem.

Kad ir izvēlēts programmēšanas veids, jānospiež poga *"CREATE"* tad tiks izveidots jauns projekts ar izvēlēto nosaukumu un tiks atvērta atbilstošā programmēšanas vide.

Programmēšanas vidē no kreisās puses tiks attēloti dažādu krāsu apļi (skat.att.3.5)

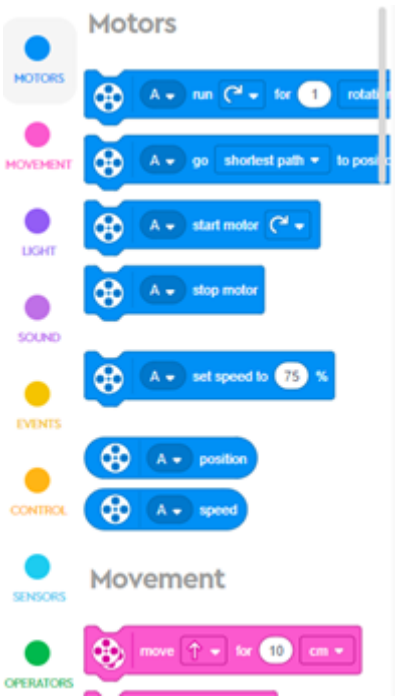
Katrs aplis nosaka kaut kādu bloku grupu un tiem ir dažādas nozīmes. Lai būtu ērtāk lietot bloku grupas un atšķirt tās, apļi, kas apzīmē grupas, un to bloki ir iekrāsoti dažādās krāsās.

Sadaļā *"MOTORS"* atrodas bloki, kuri atbild par motoru darbību.

Sadaļā *"MOVEMENTS"* atrodami bloki, kuri atbild par kustību.

Sadaļā *"LIGHT"* apvienoti bloki, kas atbild par gaismas paziņojumiem

"SOUND" sadaļā var atrast blokus, ar kuru palīdzību ir iespējams izvadīt skaņas paziņojumus vai veikt citas darbības ar skaņu.



Att. 3.5. Programmēšanas bloki

Nākošajā sadaļā *"EVENTS"* ir bloki, ar kuru palīdzību var apstrādāt dažādus notikumus.

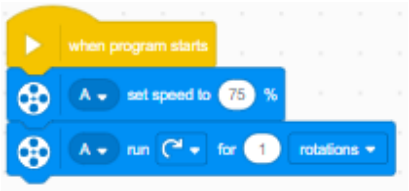
"CONTROLS" sadaļā ir iespējams atrast blokus, kuri atbild par robota kontroli.

Atsevišķā sadaļā *"SENSORS"* atrodas bloki, kas saistīti ar visu sensoru darbību un ar sensoru iegūto parametru nolasīšanu.

Sadaļā *"OPERATORS"* ir apkopoti dažādi operatori, kurus izmanto programmēšanā.

Ieteikums: Patstāvīgi detalizētāk aplūkot sadaļas programmēšanas vidē, mēģināt savienot dažādus blokus. Pievērst uzmanību ikonām, kuras ir uzzīmētas uz katra bloka. Saprast, kā tieši bloki vienojas viens ar otru kopā, kurus blokus var ievietot citos un kurus nē. Var arī pavirzīties soli tālāk un mēģināt izvietot programmu, uzinstalēt to uz robota un palaist to.

Jāņem vērā, ka tie, kuri jau iepriekš ir strādājuši vidē *Scratch*, šis programmēšanas veids varētu būt ļoti pazīstams. Zemāk izvietots programmas piemērs, kurā programmēšanas bloki tiek izvietoti viens zem otra (skat.att.3.6).



Att. 3.6. Piemērs, kur porgrammēšanas bloki atrodas viens zem otra

Kā mēs redzam šajā programmā ir izmantoti dažādu tipu bloki. Pirmais bloks ir no sadaļas *EVENTS* un tas apstrādā kādu notikumu. To forma ir tāda, ka tiem blokus var pievienot tikai apakšā. Izveidojot jaunu programmu, vajadzētu novērot, ka uz ekrāna atrodas viens bloks, kas ir no šīs sadaļas. Mūsu gadījumā tas ir

notikums *"When program starts"* - kad programma ir uzsākta. Ko tas nozīmē? Tas nozīmē, ka darbības tiks uzsāktas tad, kad robots tiks ieslēgts un starta programma tiks palaista automātiski. Tātad, šis un pārējie bloki, kas atrodas zem dzeltenā bloka, darbosies programmai sākoties.

Veidojot programmu ir iespējams izveidot vairākas kolonnas (programmas), kuriem augšā būs bloks no sadaļas *"EVENTS",* kas nozīme ka katra programma apstrādās savu notikumu un atbilstoši veiks nepieciešamās darbības. Jāpiebilst, ka bloki darbojas viens pēc otra nevis visi vienlaicīgi. Tas ir, ja kolonnā ir vairākas darbības, tās sekos viena pēc otras.

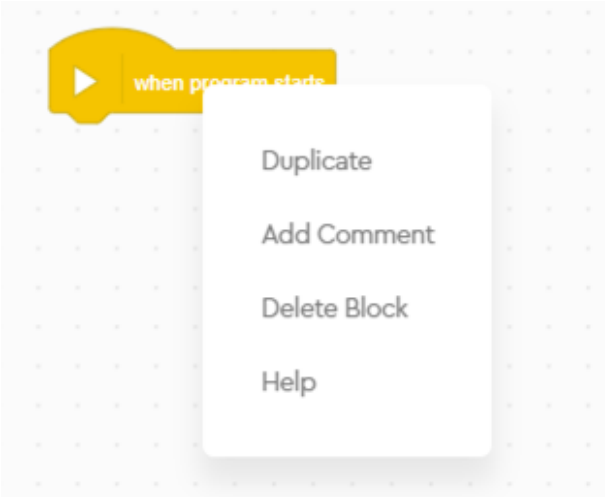
Tālāk seko divi bloki no sadaļas *"MOTORS",* ar kuru palīdzību notiek divas darbības. Motora A (motors, kar ir pieslēgts pie galvenā bloka A portā) ātrums tiek uzstādīts uz 75% un motora A viens pagrieziens pa labi.

Šajā piemērā var redzēt, kāpēc mēs izmantojam *Word Block* programmēšanas veidu. Katram modulim var redzēt aprakstu angļu valodā un precīzāk saprast, kas tā ir par darbību, kādi ir parametri, kas notiks un kāds būs rezultāts.

Att. 3.7. Piemērs, kur programmēšanas bloks ievietots citā blokā.

Dažos gadījumos, bloku nepieciešams ievietot citā blokā (skat.att.3.7.), taču šādi gadījumi tiks aplūkoti tālāk. Šāds programmēšanas veids ir draudzīgs iesācējiem, jo no programmas bloka formas ir iespējams pateikt, vai to ir iespējams savienot ar citu bloku.

Ja uzklikšķināt uz bloka ar labo peles klikšķi, tad tiks attēlota konteksta izvēlne, (skat.att.3.8.) kurā ir pieejamas vairākas funkcijas. Izvēloties funkciju *"HELP"* ir iespējams atrast aprakstu par šo bloku, kā to var izmantot. Funkcija *"DUBLICATE"* ļauj izvēlēto bloku dublēt, izveidojot bloka kopiju programmēšanas vidē. Komanda *"ADD COMMENT"* ļauj izveidot komentāru izvēlētajam blokam. Dažkārt ir nepieciešams atstāt komentārus savā programmā, lai citi, kuri aplūko šo programmu, spētu saprast, kas attiecīgajā fragmentā ir domāts vai gadījumā, lai pēc ilgāka laika atgriežoties pie šīs programmas, būtu vieglāk atcerēties, kas bija domāts attiecīgajā vietā, programmā. Funkcija *"DELETE BLOCK",* ļauj izdzēst izvēlēto bloku. Bloku izdzēst var arī ir nospiežot uz bloka ar



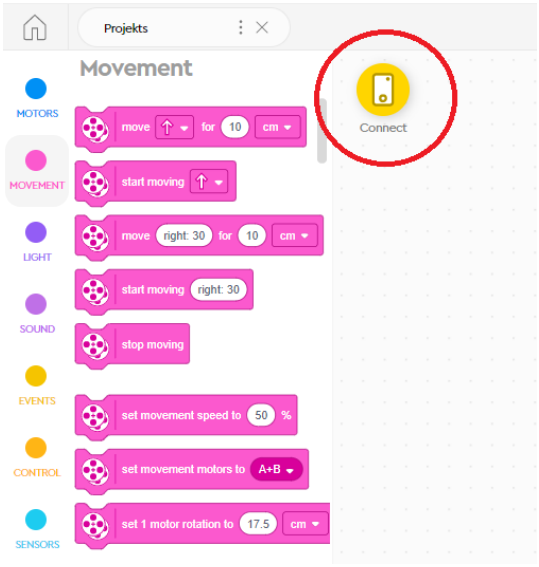
Att. 3.8. Konteksta izvēlne

kreiso peles taustiņu un uz tastatūras nospiest *"DELETE"* pogu. Funkcija *"HELP"* ļauj attēlot papildus informāciju par šo bloku.

3.2. Robota savienošana ar programmēšanas vidi

Lai varētu aplūkot programmu darbību, sākumā ir nepieciešams pievienot robotu un instalēt tajā programmu. Lai robotu pieslēgtu pie datora, ir divi veidi. Viens ir izveidot *Bluetooth* savienojumu, otrs ir savienot ar vadu.

Lai izveidotu savienojumu ar robotu, sākumā ir nepieciešams atvērt savu projektu un programmēšanas vidē, augšējā, kreisajā stūrī ir jāatrod pogu *"Connect"*. (skat.att.3.9)



Att. 3.9. Poga robota savienošanai ar ierīci

Ar vadu pieslēgt robotu pie datora ir pavisam vienkārši. Robota komplektā ietilpst *Micro-USB* vads, ar kuru iespējams pieslēgt robotu.

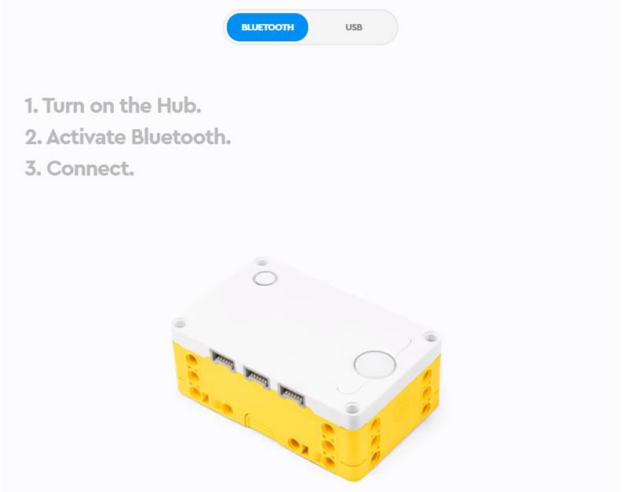
Svarīgi piebilst, ka pirmo reizi pieslēdzot robotu pie datora, to ir vēlams darīt ar USB vadu, jo būs nepieciešams veikt atjauninājumus robotam un to būs nepieciešams uzlādēt, ko ir iespējams izdarīt pieslēdzot USB vadu.

Lai izveidotu *Bluetooth* savienojumu ar robotu, sākumā ir nepieciešams ieslēgt galveno bloku un tad funkciju *Bluetooth* uz savas ierīces. Nākošajā solī ir jāatver savs projekts un programmēšanas vidē, augšējā, kreisajā stūrī ir jāatrod pogu *"CONNECT"*. (skat.att.3.9)

Uzklikšķinot uz pogas "CONNECT" tiks atvērts savienojuma izveides dialoga logs (skat.att.3.10):

Šajā logā ir jābūt izvēlētai funkcijai *Bluetooth*. Lai pareizi izveidotu savienojumu ar ierīci šajā logā tiks attēlota animācija ar kuras palīdzību varēs redzēt, kā izveidot savienojumu ar mūsu ierīci izmantojot izvēlēto savienojuma metodi.

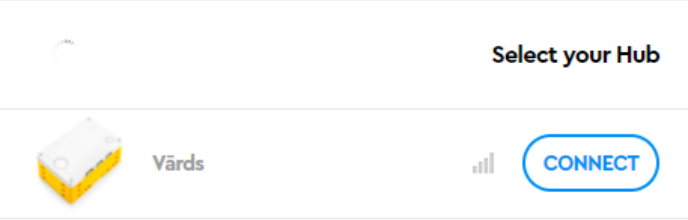
1. Ir jāieslēdz galveno bloku. To var izdarīt nospiežot ieslēgšanās pogu uz bloka virsmas (lielā poga).



Att.3.10. Robota savienošanai ar ierīci

2. Tālāk ir jānospiež mazā poga, uz kuras ir attēlots *Bluetooth* logo.

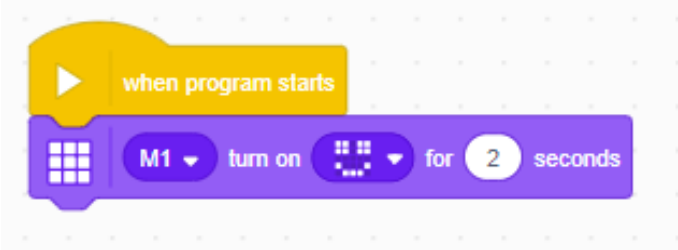
Ja viss ir izdarīts pareizi, tad labajā pusē būtu jāparādās logam, kurā būs saraksts ar visiem robotiem, ar kuriem ir iespējams izveidot *Bluetooth* savienojumu (skat.att.3.11)



Att. 3.11. Savienojuma izveide ar Bluetooth

Robotus var atšķirt pēc to nosaukumiem. Šajā gadījumā robots ir nosaukts: *"Robots"*. Lai pabeigtu savienošanas procesu atliek nospiest uz *CONNECT* pogas.

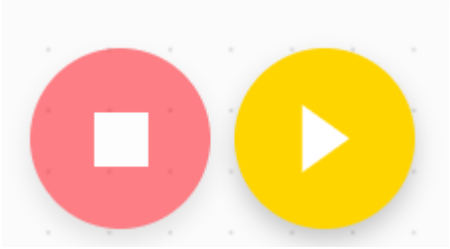
3.3. Programmas instalēšana un palaišana robotā



Att. 3.12. Programmas piemērs

Lai nodemonstrētu, kā instalēt programmu uz robota, izveidosim šādu programmu (skat.att.3.12)

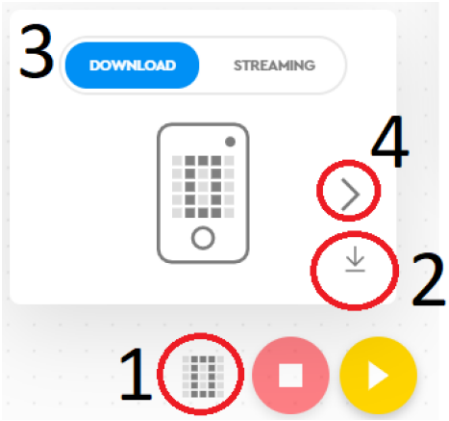
Palaižot šo programmu uz robota vadības bloka augšējas daļas jāparādās attiecīgajam zīmējumam no iebūvētām gaismas diodēm uz 2 sekundēm. Lai palaistu programmu, ir nepieciešams uzklikšķināt uz pogas ***PLAY*** (dzeltenā krāsā) (skat.att.3.13).



Att. 3.13. Pogas, PLAY un STOP

Programmai vajadzētu būt instalētai uz robota. Tā automātiski tiks palaista.

Klikšķinot uz pogas *STOP* (sarkanā poga) (skat.att.3.13.) ir iespējams apturēt programmu, ja tā ir palaista un robots ir savienots ar ierīci.



Att. 3.14. Programmu izvēle

Programmēšanas vides apakšējā daļā pie pogām *PLAY* un *STOP* atrodas ikona, (skat.att.3.14.) kas atzīmēta ar ciparu 1. Tā atver logu, kas parādīts tajā pašā attēlā. Ar ciparu 2 ir atzīmēta poga, kas ļauj programmu vienkārši instalēt uz robota atšķirībā no pogas *Play*, kas programmu instalē uz robota un uzreiz to palaiž. Pie cipara 3 ir poga, kas ļauj izvēlēties režīmu. Pēc noklusējuma ir iestatīta iespēja *Download*. Šis režīms ļauj vienkārši instalēt programmas uz robota. Otrs režīms, *Streaming* ļauj palaist programmu un

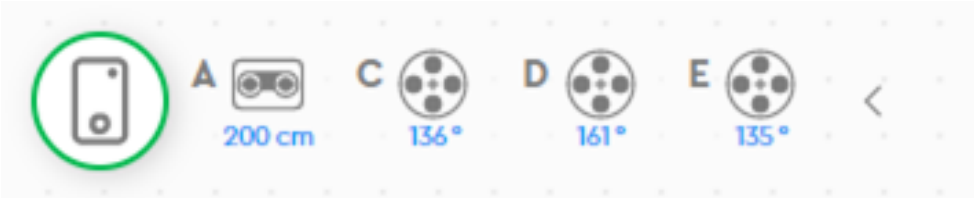
palaišanas laikā mainīt programmas parametrus. Šis režīms ir piemērots programmu testēšanai. Visbeidzot paliek poga, kas atzīmēta ar ciparu 4. Nospiežot šo pogu, nomainīsies cipars, kas ir redzams pa vidu šim logam. Pašlaik šis cipars ir nulle. Nospiežot instalēt programmu, kamēr ir izvēlēts kāds skaitlis, saglabās programmu "zem" izvēlētā cipara. Tas nozīmē, ka vienlaicīgi robotā var būt 20 programmas.

3.4. Motoru pievienošana

Pēc robota savienošanas ar ierīci, ja pie galvenā bloka ir pievienoti kādi motori/sensori, tad var novērot, ka pie šīs ikonas parādīsies kaut kādas vērtības (skat.att.3.15)

Att. 3.15. Pievienoto motoru stāvokļi

Šīs vērtības atbilst datiem, kurus dotajā brīdī nolasa sensori, vai cik daudz ir pagriezti motori. Piemēram, piestiprinot attāluma sensoru pie porta A, parādās jaunas vērtības (skat.att.3.16.).

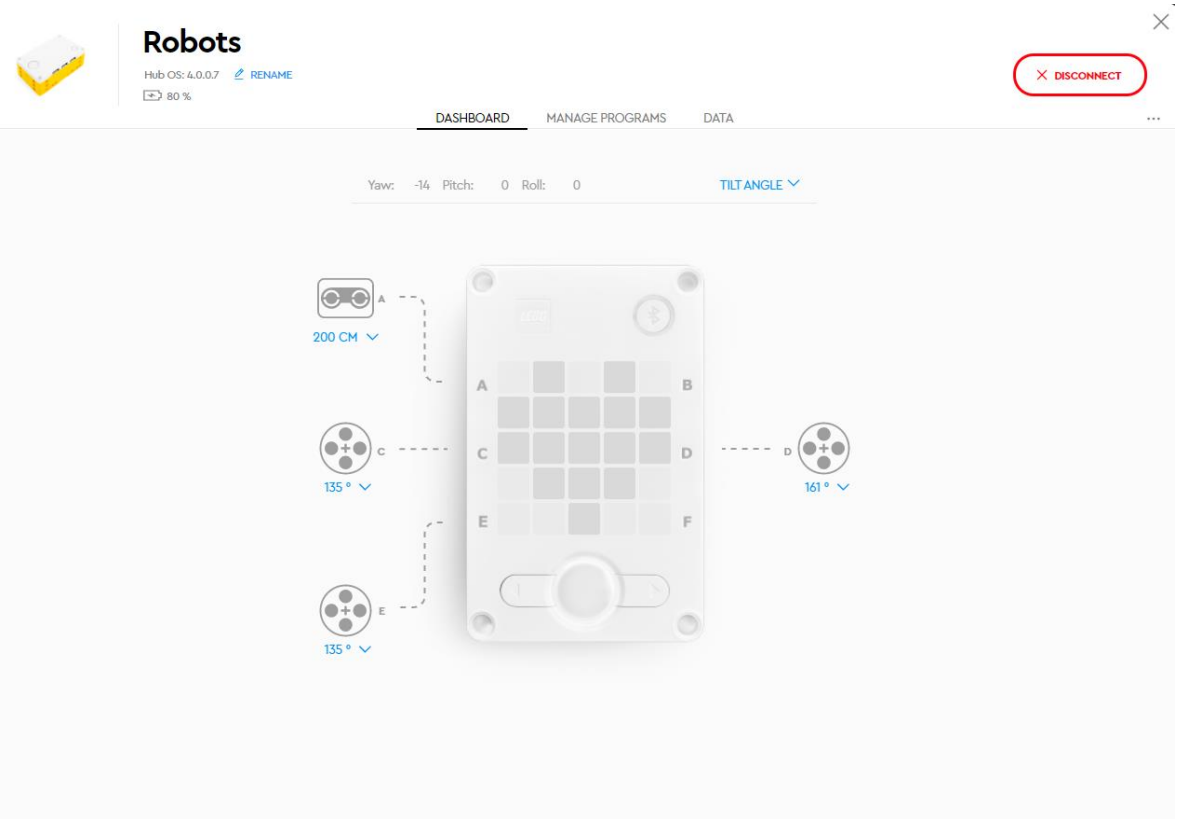


Att. 3.16. Pievienoto motoru stāvokļi un sensora dati

Var redzēt šī sensora lasījumus, šajā gadījumā sensoram priekšā neatrodas nekas, tāpēc tas redz maksimālo distanci. Ja tam priekšā pieliktu roku, vērtība no 200cm nomainītos uz attālumu, cik tālu no sensora ir līdz rokai, ja šis attālums nav lielāks par 200cm. Šāda opcija ir ļoti parocīga, ja ir nepieciešams sekot līdzi, vai robots dara to, ko robotam būtu jādara. Tas ir, testējot programmu, var sekot līdzi šiem lasījumiem, lai pateiktu, vai pie konkrētiem sensoru lasījumiem, robots izpilda pareizas darbības.

Att. 3.17. Poga robota pārskata atvēršanai

Uzklikšķinot uz robota pārskata ikonas (skat.att.3.17) atvērsies šāds logs (skat.att3.18).



Att. 3.18. Robota pārskata logs

Augšējā, kreisajā stūrī, ir redzams robota nosaukums, baterijas uzlādes līmenis un iespēja arī pārsaukt savu robotu kādā citā vārdā. Labajā pusē logam ir iespējams atvienot robotu no ierīces. Pa vidu ekrānam atkal ir iespējams redzēt sensoru un motoru vērtības.

Programmas izveide valodā Python.

Izveidojot jaunu projektu, kuru izvēlas veidot programmēšanas valodā *Python,* vajadzētu atvērties logam *LEGO Education Spike Prime Python* programmēšanas vide (skat.att.3.19)

Att. 3.19. Programmas piemērs programmēšanas valodā Python

Kas ir programmēšanas vide? Šeit programma tiek veidota šajā valodā. Izveidojot šo projektu, automātiski uz ekrāna parādās piemērs, kurā uz galvenā bloka LED diodēm iedegas smaidīga seja.

Var veikt izmaiņas programmā, kuras novedīs pie tā, ka programmā būs kļūda. Piemēram: (skat.att.3.20):

```
1 from spike import PrimeHub
2 from spike.control import wait_for_seconds, wait_until, Timer
3 from math import *
4 import time
5
6 hub = PrimeHub()
7
8 hub.light_matrix.show_image('')
9
```

Att. 3.20. Programma, kurā apzināti pieļauta kļūda

Ja pamēģināt šo programmu palaist uz robota, tad varētu novērot, ka poga uz galvenā bloka iemirgotos sarkanā krāsā, kas nozīmē, ka ir notikusi kļūda programmēšanas vidē, konsolē varēs redzēt kļūdas paziņojumu ar tās aprakstu. Mūsu gadījumā tas izskatīsies šādi: (skat.att.3.21):

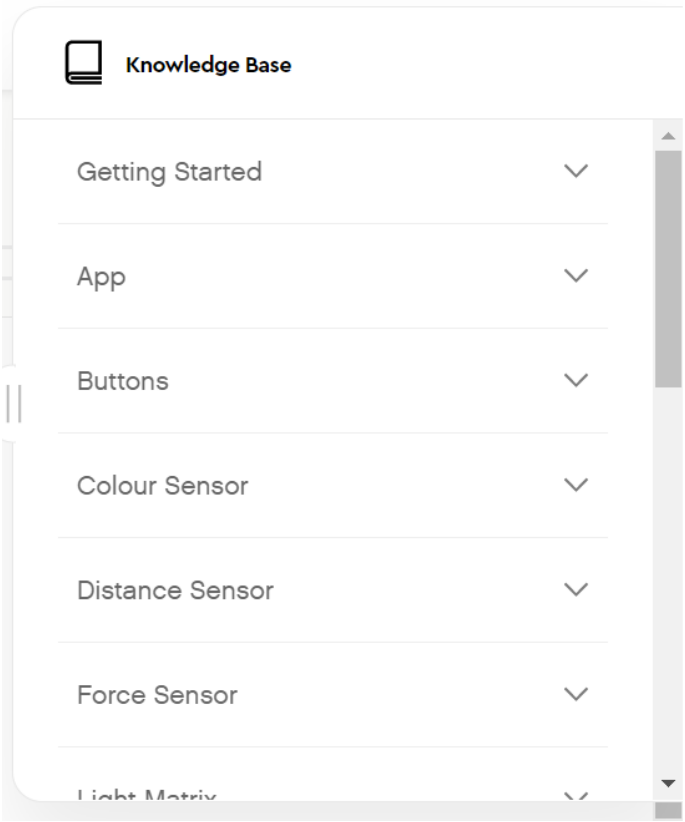
```
File "programrunner/__init__.py", line 1, in start_program
File "__init__.mpy", line 8
File "_api/lightmatrix.py", line 1, in show_image
ValueError: image is not one of the allowed values.
```

Att. 3.21. Kļūdas paziņojums mēģinot palaist kļūdaino programmu

Šīs paziņojums nozīmē, ka uz ekrāna mēģina uzzīmēt zīmējumu, kas nav piedāvāts sarakstā, no kura tos ņem.

Tā kā šajā izdevumā netiek aplūkota programmas izveide programmēšanas valodā *Python*, bet gan grafiskā programmēšanas valoda, tad vairāk par iespējam ar *Python* var atrast nospiežot *Python* programmēšanas vidē, labajā pusē uz ikonas (skat.att.3.22) .Pēc nospiešanas darba vides labajā malā atvērsies izvēlne, kurā var atrast informāciju par programmēšanu *Python* valodā, dažādas gatavas iespējas un funkcijas (skat.att.3.23).

Att. 3.22. Poga pamācību atvēršanai



Att 3.23. Pamācības izvēlne programmēšanas valodā Python

4. Algoritmu pamati un blokshēmas

Mūsdienās var vārdu algoritms var saskarties visai bieži, ne vienmēr to nosaucot par algoritmu. Piemēram, domājot par *Google* meklētāju, ir skaidrs, ka tas strādā pēc kāda algoritma, kas ļauj atrast nepieciešamo informāciju pēc ievadītā teksta fragmenta. Vai arī *Google Maps,* kas pēc kāda algoritma spēj izveidot maršrutus starp diviem vai vairākiem punktiem. Taču domājot par ikdienas dzīvi, darbadienas rītā, skolas laikā, arī var saskatīt algoritmu. Katrs skolēns no rīta pamostas, paēd brokastis, sakrāmē somu un dodas uz skolu. Šeit ir grūti pateikt, vai šis ir algoritms, jo tas nav definēts līdz šim, tāpēc -

Algoritms ir ļoti skaidrs un precīzs darbību saraksts kāda mērķa sasniegšanai.

Atgriežoties pie piemēra par rītu, var redzēt, ka tajā ir minēts kāds darbību saraksts, kuru mērķi var uzskatīt ierašanos skolā. Taču varētu rasties jautājums, vai dotās darbības ir skaidras un precīzas. Minētas darbības ir skaidras, taču nav precīzas. Piemēram, skolēns pamostas un uzreiz ēd. Šeit netiek precizēts, ko skolēns ēd brokastīs, kur skolēns ēd. Nav precizēts, ko skolēns ievieto somā, kā arī nav minēts skolēna celšanās laiks. Tāpēc to īsti nevar uzskatīt par algoritmu, jo ir nepieciešams precizēt vairākas papildus darbības.

UZDEVUMS: *Izveidot algoritmu savam rītam.*

Ir ļoti būtiski dot precīzas darbības algoritmā. Pretējā gadījumā, var rasties problēmas programmējot robotu, jo robots var kādu darbību nesaprast. Piemēram, cilvēks var parādīt ar pirkstu uz kādu objektu un palūgt citam to objektu pienest. Un cilvēks sapratīs šo komandu. Taču robotam būs jāsniedz precīzs darbību saraksts, lai tas spētu pienest šo objektu. Robotam būs precīzi jāpasaka, kā nokļūt līdz objektam, kurā brīdī un kā satvert objektu un kā atgriezties. Šeit var aplūkot sviestmaižu smērēšanu. Citam cilvēkam dodot pavēli uzsmērēt sviestmaizi. Cilvēkam pietiks ar šo komandu, lai to izdarītu. Taču robotam būs jāprecizē, kā to darīt, tas ir, kur un kā novietot nepieciešamās lietas uz galda, izņemt maizes šķēli no iepakojuma, cik daudz sviesta nepieciešams, kādām kustībām jāsmērē sviests.

UZDEVUMS: *Pāros izmēģiniet viens otram dot komandas, lai tiktu uzsmērēta sviestmaize***.**

Kā piemēru var minēt spēles vai rotaļas. Spēlēs ir kādi noteikumi, pēc kuriem ir jāspēlē katra spēle. Tie parasti ir skaidri definēti, lai katrs spēlētājs to saprastu un spētu spēlēt šo spēli? Varētu rasties jautājums: "Kas ir skaidri noteikumi? Kādiem kritērijiem ir jābūt, lai noteikumi būtu skaidri saprotami?" Kā piemēru var ņemt kādu iepriekš zināmu spēli un padomāt, vēlams pāros, kas tieši šīs spēles noteikumus padarīja saprotamus? Varbūt šīs spēles noteikumos var novērot kādas būtiskas lietas, kas būtu jāizmanto ikvienas spēles noteikumu aprakstā, lai tie būtu saprotami?

UZDEVUMS: *izmantot tikko iegūtās atbildes par skaidriem noteikumiem un radīt savu spēli. Spēlei nav jābūt sarežģītai, tā var būt vienkārša, piemēram, plaukstu sišanas spēle.*

Vēl piemērs no ikdienas varētu būt dejošana. Dejotājam ir iedots saraksts ar kustībām, kuras tam ir jāizpilda dejas laikā. Dodot neskaidras norādes, dejotājs var veikt nepareizo kustību nepareizajā laikā vai atrasties nepareizajā vietā. Tāpēc deju ir jāizmēģina vairākas reizes, lai deja tiktu izpildīta pareizi un neviens nekļūdītos. Līdzīgi ir robotam, tam iedod programmu, kuru tam ir jāizpilda. Un gadījumā, ja iedoto programmu izpildot robots izdara kaut ko nepareizi, programma ir jāpielabo un robotam ir jādod iespēja to izpildīt vēlreiz. Šādi varētu izskatīties programmas testēšana.

Kā piemēru var aplūkot daudziem zināmo Plaukstiņpolku. Lai labāk saprastu, kas tas ir, to ir ieteicams aplūkot internetā.

Bieži algoritmu vai robotu izstrādē ir vērts paskatīties internetā, vai iepriekš kāds kaut ko līdzīgu nav veidojies. Tas bieži vien ietaupa daudz laika vai ļauj laicīgi novērst kādas problēmas, kuras varētu novērot programmas izveides laikā.

Uzdevums: *ar vārdiem uzrakstīt algoritmu, kas atbilstu šai dejai.*

Plaukstiņpolkas algoritms:

1. Sasist plaukstas divas reizes
2. Sasist labās plaukstas
3. Sasist plaukstas 2 reizes
4. Sasit kreisās plaukstas
5. Sasist plaukstas 2 reizes
6. Sasist labās plaukstas
7. Sasist kreisās plaukstas
8. Sasit abas plaukstas 2 reizes
9. Sasist ar partneri rokas
10. Saķerties elkoņos
11. Ar 8 palēcieniem lec apkārt riņķī

Vai šāds algoritms ir pareizs? Vai nolasot šādas komandas ir skaidra deja? Vai ir kādas neprecizitātes, nepilnības? Ko vajadzētu uzlabot?

Tagad, kad ir sniegts ieskats, kas tad īsti ir algoritms, ir jāaplūko, kā algoritmus var pierakstīt. Protams, var katru komandu pierakstīt ar vārdiem un veidot garu virkni ar tādām komandām. Taču tādā veidā pierakstot, lielāki algoritmi kļūtu nepārskatāmi un iespējams sarežģītākus algoritmus būtu grūti pierakstīt. Viens no veidiem kā to ir iespējams pierakstīt ir pierakstīt algoritmus ar *Pseidokodu*. Tas ir pieraksta veids starp sarunvalodu un

programmēšanas valodu. Ar to piekrastīti algoritmi ir saprotamāki, nekā ar vārdiem pierakstīti. Piemēram:

Algoritms lielākā kopīgā dalītāja atrašanai

Kamēr A ≠ B :

Ja A > B :

B piešķir vērtību B-A

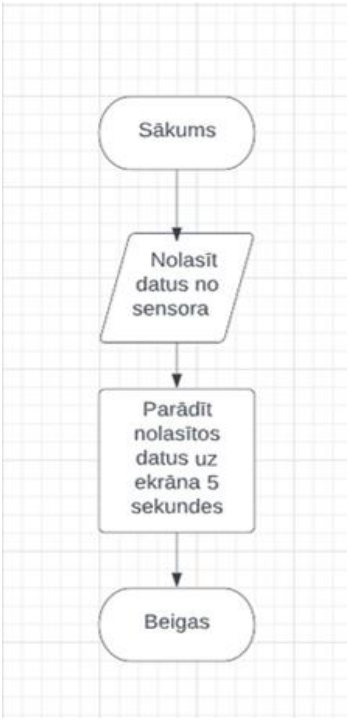
Pretējā gadījumā:

A piešķir vērtību A-B

Paziņot A

Cits algoritma pieraksta veids ir pierakstīt algoritmu ar blokshēmas palīdzību. Tas atgādina *Lego Spike Prime* programmēšanu. Ar figūru palīdzību tiek pierakstīts algoritms, kur katra figūra apzīmē savu darbību (skat.att.4.1).

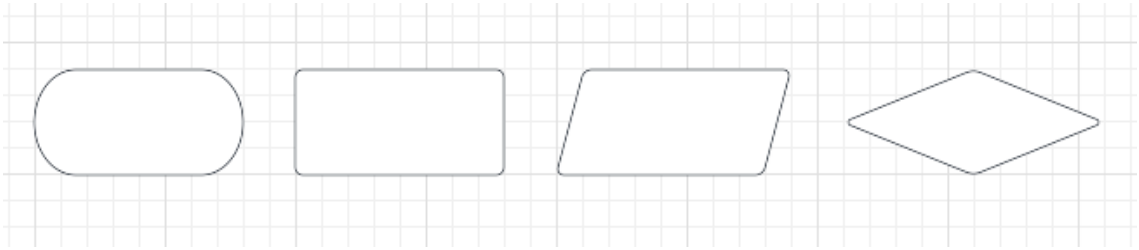
Ar blokshēmu palīdzību ir aprakstīts algoritms, kas sākas un uzreiz beidzas. Tātad, algoritma sākumu un beigas var apzīmēt ar ovālveidīgu formu. Apskatīsim kādu piemēru (skat.att.4.1.).



Att. 4.1. Blokshēmas piemērs

Šeit algoritms sākas, tad seko datu nolasīšana no sensora un tālāk seko darbība par nolasīto datu un to attēlošanu ekrānā uz piecām sekundēm. Šeit būtiskākais ir pievērst uzmanību figūru formām.

Visas figūras ir apkopotas zemāk (skat.att.4.2.) No kreisās puses, pirmā forma tiek lietota algoritmu sākumā un beigās, kā tas ir redzam iepriekšējos piemēros. No kreisās puses otrā figūra raksturo vienkāršas darbības. Tas var apzīmēt robota braukšanu uz priekšu vai datu parādīšanu uz ekrāna. Tālāk seko bloks, kas raksturo datu ievadi. Tas tiek lietots, lai apzīmētu datu ievadi, kā tas ir iepriekšējā piemērā, kur šī forma tika lietota vietā, kur notiek datu nolasīšana no sensora. Un visbeidzot seko forma, kas tiek lietota sazarojumos un ciklos. Sazarojumi un cikli tiks aplūkoti tālākās nodaļās.



Att. 4.2. Figūras blokshēmu veidošanai

Piemērs, kur robots pabrauc uz priekšu. Vispārīgā gadījumā var teikt, ka kvadrāts tiek lietots, kad ir darbības, kas veic kaut kādu izvadi. Un paralelogramu lieto, kad ir darbības, kas kaut ko ievada.

PIEZĪME: *Citos literatūras avotos ir jābūt uzmanīgiem, jo tajos var lietot citas, atšķirīgas figūras, lai apzīmētu vienu un to pašu.*

5. Lineāri algoritmi

Vieni no visvienkāršākajiem algoritmiem varētu būt lineārie algoritmi, jo tajos darbības seko viena pēc otras. Tieši šo vienkāršību piešķir tas, ka darbības ir viena pēc otras, neparādās kādas sarežģītas konstrukcijas algoritmos. Šos algoritmus arī ir viegli piekrastīt, ja gadījumā ir vairākas darbības, tad blokshēmā darbības elementi vienkārši sekotu viens zem otra. Lineāros algoritmus izmanto gadījumos, kad uzdevumu ir iespējams izpildīt veicot vairākus soļus pēc kārtas. Pārsvarā gan šāda veida algoritmi parādīsies kaut kādā kombinācijā ar cita veida algoritmiem, kas tiks aplūkoti turpmākajās nodaļās.

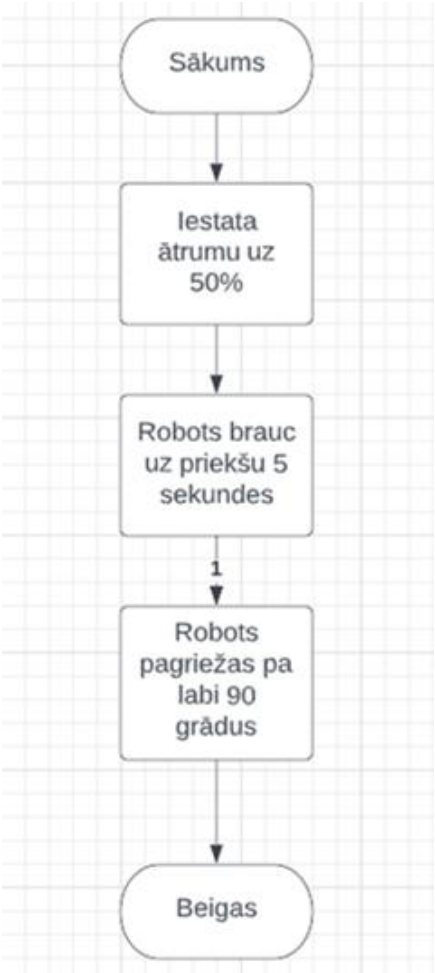
5.1. Vienkāršu algoritmu veidošana ar blokshēmām.

Šajā nodaļā būs aplūkota lineāru algoritmu veidošana ar blokshēmām. Taču vispirms, kas tad īsti ir lineārs algoritms. Lineārs algoritms un vienkāršs algoritms nav sinonīmi, taču ar to šajā nodaļā ir domāts viens un tas pats. Patiesībā, jau iepriekš parādītās blokshēmas reprezentēja lineārus algoritmus. Par lineāru algoritmu sauc tādu algoritmu, kuram darbības seko viena aiz otras. Var teikt, ka tajā ir iespējams visas darbības novietot vienu zem otras, uz vienas taisnes.

Piemēram, (skat.att.5.1) pirmajā darbībā robota ātrums tiek iestatīs uz 50%. Nākošā darbība liek robotam braukt uz priekšu 5 sekundes. Kad tas ir izdarīts, tad robotam trešā programmas darbība liek pagriezties pa labi par 90 grādiem. Šis ir lineārs algoritms, jo visas darbības atrodas uz vienas taisnes un katra no tām seko aiz iepriekšējas darbības, kad tā tika pilnīgi izpildīta.

Uzdevums: Uzzīmēt blokshēmu, kas atbilstu sviestmaižu smērēšanas algoritmam.

Uzdevums: Uzzīmēt blokshēmu, kas atbilst papīra lidmašīnas locīšanai.



Att. 5.1. Blokshēmas piemērs

5.2. Lineārais algoritms programmēšanas vidē

Veidot programmas, kas atbilst lineāriem algoritmiem ir salīdzinoši vienkārši. Tā kā šāda veida algoritmos darbības ir viena pēc otras, tad programmā vienkārši jāsaliek darbības

vienu pēc otras. Līdzīgi, kā norādes transporta maršrutam. Tajā parasti ir teikts, cik metrus jābrauc uz priekšu un kad jāpagriežas. Uzrakstot visas šīs komandas pēc kārtas, varētu iegūt lineāru algoritmu.

Tālāk tiks aplūkots vienkāršs piemērs lineāriem algoritmiem programmēšanas vidē, kurā vienkārši robotam būs jābrauc uz priekšu. Vispirms ir nepieciešams izveidot pašu robotu. Šajā uzdevumā tiek piedāvāts izmantot robotu, kura būvēšanas instrukciju var atrast *"Build"* sadaļā ar nosaukumu *"Driving Base 1"*.

PIEZĪME: *pēc robota izveides ir svarīgi pārliecināties, vai motori un sensori ir ievietoti pareizajos portos, atbilstoši instrukcijai. Nepareizi pievienoti motori un sensori var radīt problēmas dažos uzdevumos, jo robots nespēs izpildīt darbības atbilstoši piedāvātajai programmai.*

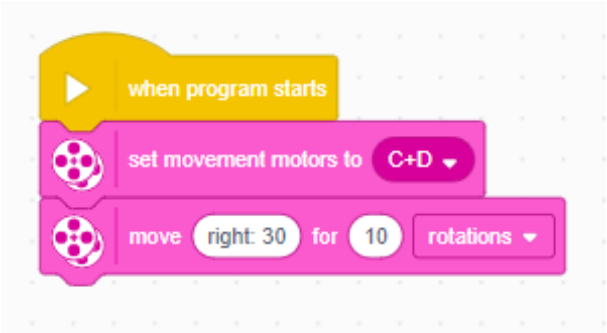
Tālāk pāriesim pie pašas programmas. Apskatīsimies vienu no variantiem (skat.att.5.2), kā likt robotam pārvietoties uz priekšu.

Šeit sākumā tiek parādīts, ka pie galvenā bloka portiem C un D ir pievienoti motori un tālāk šiem motoriem ir dota komanda pārvietoties uz priekšu 10 cm. Bloks, kurā parādās, ka "C+D" nozīmē, ka robotam tiek pateikts, ka pie šiem portiem ir pievienoti motori. Lai uzbūvētu nepieciešamo robotu "Build" sadaļā var atrast nosaukumu "Driving Base 1". Motori jau būs pievienoti pie šiem portiem. Gadījumā, ja konstruēšanas laikā uzreiz neizdodas atrast attiecīgos programmēšanas blokus, tos ērtāk var meklēt pēc to krāsas.

Att. 5.2. Programmas piemērs

Programmā netika norādīts, ne ātrums, ar kādu robotam vajadzēja braukt, ne arī, cik tālu vai cik ilgi vajadzēja braukt uz priekšu. Programmā tika norādīts tikai uz kādu attālumu pārvietoties, kas ir iestatīts pēc noklusējuma 10 cm uz priekšu. Robots pakustējās uz priekšu, līdz ar to var teikt, ka robots izpildīja doto komandu. Taču aplūkojot gadījumu, kad robotam ir jāpārvietojas uz priekšu, situācija kļūst nedaudz sarežģītāka.

Aplūkosim citu programmas piemēru (skat.att.5.3.)



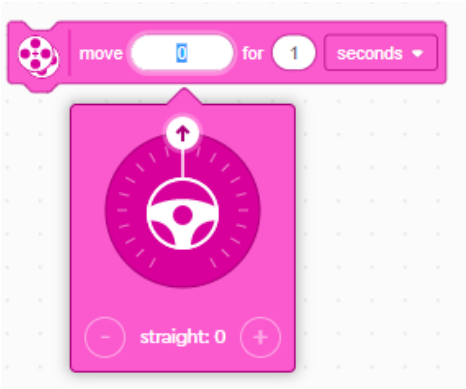
Att. 5.3. Programmas piemērs

Līdzīgi, kā iepriekš, tiek parādīts, ka portos C un D ir pievienoti motori un tālāk ir braukšana. Šeit šī braukšana nav taisni, bet ar leņķi. Tas nozīmē, robots braucot veidos arku. Šeit svarīgi saprast, ka nevar vienkārši dot komandu "Braukt pa labi", jo ir iespējami vairāki veidi, kā to izdarīt. Var ļoti strauji griezties pa labi, iestatot pirmo vērtību, piemēram, "*Right: 90*" vai arī minimālo, kustoties pa labi, iestatot to uz 10, kas būtiski atšķiras no kustības uz priekšu.

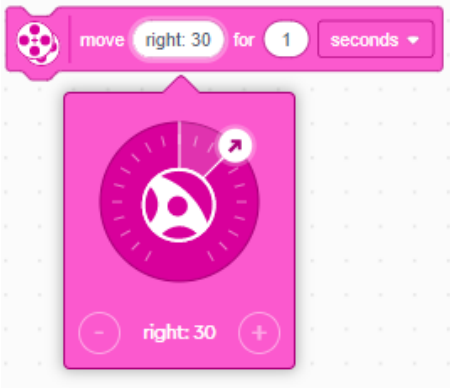
Lai labāk saprastu, ko nozīmē "braukt ar leņķi" izpildīsim šādu uzdevumu:

UZDEVUMS: Izmainīt vērtības un aplūkot, kā robots brauc pie dažādām vērtībām. Ieteicams arī mainīt kustības daudzumu.

Sākumā var aplūkot kustības bloku (skat.att.5.4 un 5.5.)

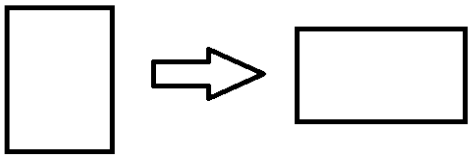


Att. 5.4. Rotācijas vērtība: 0



Att. 5.5. Rotācijas iestatīšana uz 30

Pozīcijā *"MOVE"* ierakstīta vērtība 0 (skat.att.5.4). Tas nozīmē, ka robots brauks uz priekšu 1 sekundi. Pamainīsim un 0 vietā iestatīsim *"Right 30"* (skat.att.5.5). Tas nozīmē, ka robots vairs nebrauks taisni, bet nedaudz uz labo pusi. To var interpretēt, kā braukt ar mašīnu. Ja stūri tur taisni, kas atbilstu pozīcijai: 0, tad mašīna brauc taisni. Ja stūri nedaudz pagriež pa labi, mašīna brauks mazliet uz labo pusi, kā tas ir gadījumā, kad pozīciju noregulē uz "*Right 30"*. Tad attiecīgi pozīcija: 100 atbilst pagriezienam, taču krietni straujākam, nekā uzstādot vērtību 30.



Att. 5.6. Pagrieziens par 90 grādiem

Att. 5.7. Programmas piemērs

Šeit varētu rasties jautājums. Kā likt pagriezties precīzi 90 grādu leņķī turpat uz vietas (skat.att.5.6.).

Viens variants ir lietot sensoru, taču tas tiks aplūkots turpmākajās nodaļās. Tāpēc paliek otrs variants - iestatīt maksimālo pagriezienu un atrast pareizo rotāciju skaitu vai kustības laiku, lai tiktu veiks 90 grādu pagrieziens.

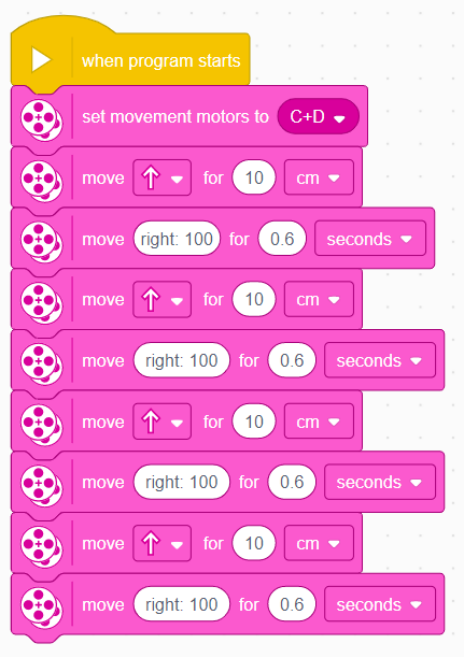
Tas nozīmē: tiek izvēlēta viena sekunde, kā kustības daudzums (skat.att.5.7). Palaižot programmu vajadzētu novērot, ka robots veic krietni lielāku pagriezienu nekā 90 grādus. Tas nozīmē, ka nepieciešams samazināt kustības laiku. Ievietojot laiku: 0.6 sekundes, vajadzētu iegūt aptuveni precīzu pagriezienu.

UZDEVUMS: Iepriekš apskatīto programmu pamēģināt palaist vairākas reizes. Pamēģināt to palaist uz dažādām virsmām un salīdzināt rezultātus. Vai ir iespējams novērot ko interesantu? Vai uz vienas virsmas robots veic paredzēto pagriezienu, bet uz citas nē? Kāpēc tā varētu notikt?

ATBILDE: Iespējams, mainot virsmas, pa kurām palaist robotu, nevarēja novērot nekādas atšķirības. Taču pastāv iespēja, ka varēja. Tātad, kāpēc tā? Ļoti iespējams, ka robots vienkārši noslīdēja. It īpaši, ja virsma bija putekļaina. Tad robotam ir lielāka varbūtība, ka tas noslīdēs.

Noslīdēšana varētu būt viens no iemesliem, kāpēc algoritmi dažkārt var būt neefektīvi vai neprecīzi. Jo izveidojot virkni komandu un izveidojot tām atbilstošu programmu, ir risks, ka robots vienkārši noslīdēs un visu sabojās. Jo neizpildot vienu komandu pareizi noslīdēšanas dēļ, nākamo komandu robots arī neizpildīt tā, kā bija plānots. Patiesībā, robots pareizi neizpildīs nevienu no nākamajām komandām. Tāpēc ir labi, ja var lietot sensoru palīdzību. Piemēram, nevis braukt kādu noteiktu laiku, kuru iepriekš iestata, bet braukt, līdz kamēr kādi konkrēti sensoru lasījumi ir spēkā, kā, piemēram, krāsa krāsu sensorā. Protams, arī sensori nav pilnībā precīzi, taču neliels puteklis, kas noved pie robota noslīdēšanas, ne obligāti izjauks visu tālāko programmu.

UZDEVUMS: Izveidot programmu, kurā robots izbrauc kvadrātu. Tas ir, robots brauc taisni, pagriežas 90 grādus, brauc atkal taisni, pagriežas un tā, līdz nonāk atpakaļ sākuma punktā. Šajā uzdevumā nav svarīgi, cik lielu kvadrātu. Piemēram, 10x10 cm kvadrāts šim uzdevumam. Vispirms ir nepieciešams izveidot pašu robotu, kas spētu šo kvadrātu izbraukt. Uzdevumā tiek piedāvāts izmantot to pašu robotu, ko iepriekš jeb *"Driving Base 1"*. Šī robota instrukciju var atrast *Build* sadaļā. Tālāk nepieciešams izveidot programmu. Var lietot jau iepriekš iegūtos rezultātus, iepriekšējā uzdevumā tika parādīts programmēšanas bloks, kas ļauj pagriezties robotam 90 grādu leņķī. To savienojot ar braukšanu taisni var iegūt šādu programmu (skat.att.5.8.).



Att. 5.8. Programmas piemērs

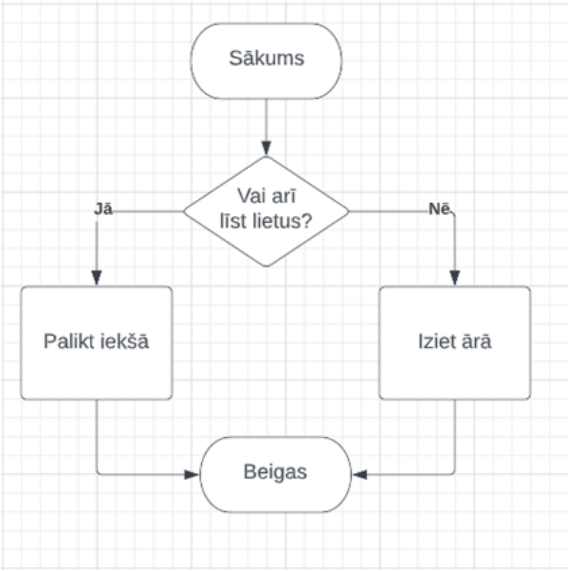
Šeit tiek norādīts, pie kuriem portiem ir pievienoti motori. Tālāk robots brauc un pagriežas. Pēc programmas koda, robotam vajadzētu apbraukt kvadrātu. Taču, iepriekš minētā robota noslīdēšana, var tam traucēt. Var redzēt, ka robotam noslīdot darbības sākumā, tas nespēs atgriezties turpat, kur sāka savu kustību. Katra nākamā darbība ir atkarīga no tā, ka iepriekšējā darbība tika izpildīta korekti. Tāpēc, piemēram, varētu pievienot krāsas sensoru, lai robots varētu sekot līnijai un spētu nobraukt šo taisno posmu precīzi taisni, vai arī žiroskopu, lai robots spētu noteikt, vai ir veicis precīzi šo 90 grādu pagriezienu.

JAUTĀJUMS: Kā varētu optimizēt šo programmu? Vai nav tā, ka viena un tā pati darbība tiek atkārtota vairākas reizes?

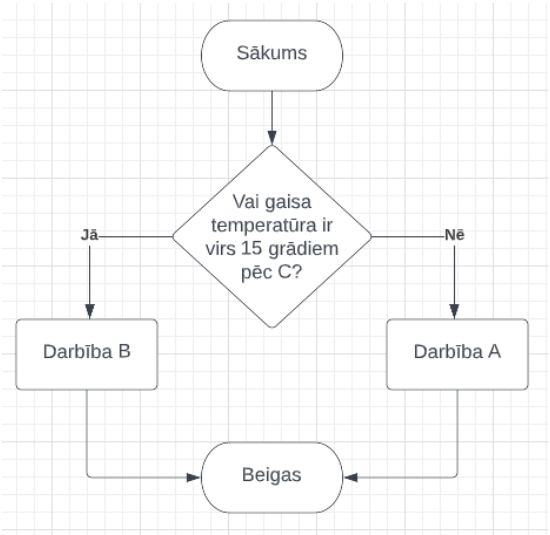
6. Sazarojumi

6.1. Sazarojumi

Ikdienas dzīvē sazarojumi ir mums visapkārt. Vienīgi, ikdienas dzīvē tos ne vienmēr var pamanīt, tās liekas ļoti pašsaprotamas darbības, izvēles. Bet kas tad īsti ir sazarojumi? Sazarojums ir līdzīgs lēmuma pieņemšanai. Piemēram, izskatīsim situāciju no dzīves: gadījumā, ja ārā līst lietus, būs nepieciešams palikt mājās, vai var iziet ārā pastaigāties. Ikdienā šis liekas diezgan pašsaprotams jautājums, taču programmējot robotu, tas nav pašsaprotams. Robotam ir jādefinē precīzu instrukciju un visus soļus vajag precīzi izskaidrot.



Att. 6.1. Sazarojuma algoritma blokshēmas piemērs



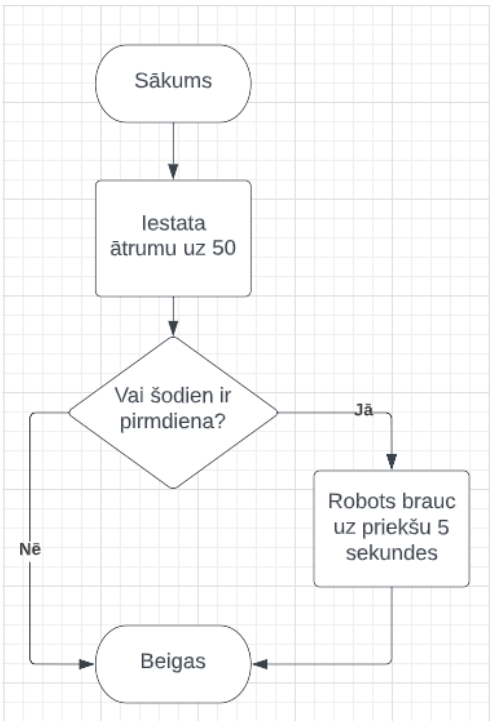
Att. 6.2. Sazarojuma blokshēmas piemers

Šo jautājumu, kā, piemēram, vai ārā līst lietus, uzskata par sazarojumu. To turpmāk sauks par apgalvojumu. Šis apgalvojums var pieņemt vai nu patiesas vai aplamas vērtības jeb tas vai nu ir spēkā vai nav spēkā. Ārā lietus vai nu līst vai nelīst.

Apskatīsim šīs situācijas blokshēmu, kurā ir redzams sazarojums (skat.att.6.1). Rombā ir ievietots mūsu jautājums: vai ārā līst lietus. Atkarībā no atbildes tiek pieņemts lēmums un tad notiek viena vai otra darbība.

Apskatīsim citu piemēru ar sazarojumu (skat.att.6.2), kur atkarībā no gaisa temperatūras, tiks veikta darbība A vai B. Sākumā tiek pārbaudīts vai gaisa temperatūra ir virs 15 grādiem. Darbības A un B var būt absolūti jebkas, tik pat labi, tajā vietā var būt arī vairākas darbības, piemēram, atsevišķs algoritms par sviestmaizes uzsmērēšanu, vai kaut kas cits. Attiecīgi pēc atbilstošas darbības A vai B izpildes algoritms ir pabeigts.

Nākošajā algoritma piemērā (skat.att.6.3) apskatīsim, kur sākumā tiek iestatīts robota ātrums un tad, atkarībā no nedēļas dienas, vai nu brauc uz priekšu vai arī algoritms beidzas.



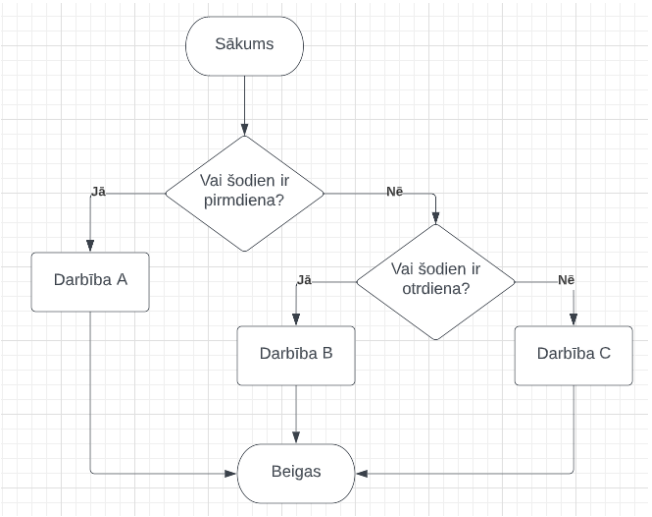
Att. 6.3. Blokshēmas piemērs

Att. 6.4. Blokshēmas piemērs

Varam papildināt šo piemēru, lai demonstrētu, ka var sekot arī vairākas darbības (sk.att.6.4.). Taču tad intuitīvi rodas jautājums: vai sazarojums var atrasties citā sazarojumā. Atbilde uz to ir jā, var: Apskatīsim šādu piemēru, ar vairākiem sazarojumiem (skat. att.6.5)

No sākumā tiek pārbaudīts, vai ir pirmdiena, gadījumā, ja atbilde ir nē, tad pārbauda, vai šodien ir otrdiena. Un tad atkarībā no atbildēm tiek veiktas darbības.

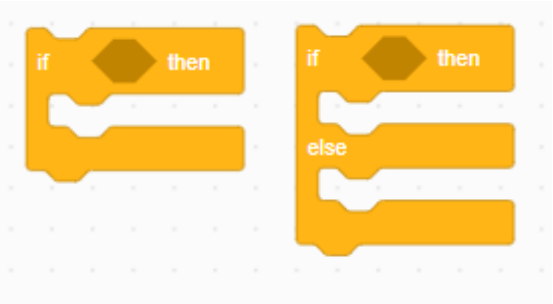
Var teikt, ka sazarojumi ir rīks, ar kura palīdzību, var veikt dažādas darbības atkarībā no kāda parametra vai stāvokļa. Piemēram, atkarībā no dažādiem sensoru mērījumiem veikt dažādas darbības. Pie dažādiem krāsu sensora lasījumiem parādīt dažādus attēlus uz ekrāna vai pie dažādiem attālumiem, ko sniedz attāluma sensors, mainīt robota braukšanas ātrumu.



Att. 6.5. Piemērs ar vairākiem sazarojumiem

6.2. Programmu piemēri sazarojumiem

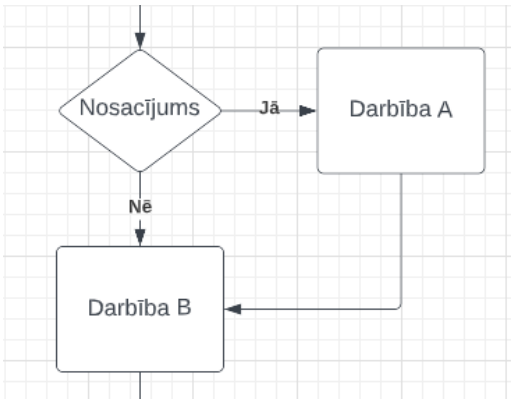
Lai varētu programmēšanas vidē *LEGO Education Spike Prime* izmanot sazarojumus ir nepieciešams sadaļā *CONTROL* atrast attiecīgus blokus. (skat.att.6.6).



Att. 6.6. Programmas bloku piemērs

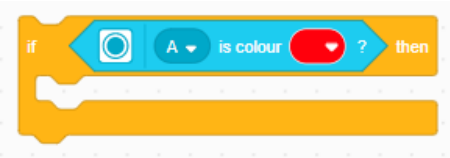
Abi bloki ir sazarojumu bloki un tos var pielietot veidojot sazarojumus savam robotam. Apskatīsim ar ko šie bloki atšķiras un kā tie darbojas.

Pirmajam blokam, kas novietots pa kreisi, varētu atbilst blokshēma (skat.att.6.7):



Att. 6.7. Blokshēmas fragments

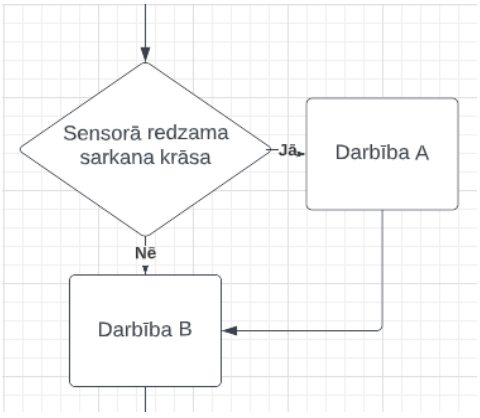
Šeit **darbības A un B** ir kādas patvaļīgas darbības vai darbību kopums un **Nosacījums** ir jebkurš nosacījums, kas mums atbilst noteiktā uzdevuma izpildei. Tātad, ja kāds dotais nosacījums ir spēkā, tad tiek izpildīta darbība A un tālāk seko darbība B, pretējā gadījumā, ja nosacījums netiek izpildīts, tad darbība A netiks izpildīta, bet uzreiz būs paveikta darbība B.



Att. 6.8. Programmas bloka piemērs

Šo konstrukciju pielieto, ja atkarībā no kāda nosacījuma ir nepieciešams izpildīt kādas darbības un tālāk turpināt programmas izpildi.

Aplūkojot programmas bloku var pamanīt, ka ir sešstūra formas tukšums. Tās ir paredzēts kādam nosacījumam, kas arī būs sešstūra formā. Tātad sešstūra vietā tiek ievietots kāds bloks, ka arī būs mūsu sazarojuma nosacījums (skat.att.6.8.)



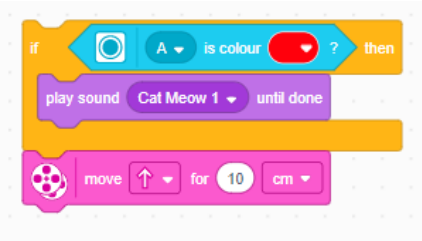
Att. 6.9. Blokshēmas fragments

Par nosacījumu paņemsim no sadaļas *SENSORS* bloku, kas pārbauda kādas krāsas ir objekts krāsu sensora priekša. Šim blokam blokshēma izskatās šādi (skat.att.6.9).

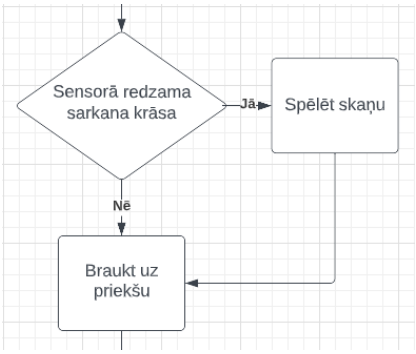
Tātad, ja sensorā ir redzama sarkanā krāsa, tiek izpildīta darbība A un tikai tad seko

darbība B. Turpinot piemēru papildināsim mūsu sazarojuma bloku ar darbībām.

Pievienosim bloku no sadaļas *SOUNDS*, kas varētu atskaņot kādu skaņas paziņojumu un ievietosim to uzreiz kā piemērā (skat.att.6.10.). Un par darbību B izvēlēsimies robota kustību uz priekšu par 10 cm no sadaļas *MOVEMENTS*. Rezultātā mēs saņemsim programmu, kas veiks skaņas signālu, ja robots ar krāsu sensora palīdzību "redzēs" sarkano objektu un tad sekos braukšana uz priekšu. Programmas blokshēma ir attēlota zemāk (skat.att.6.11.).



Att. 6.10. Programmas fragments



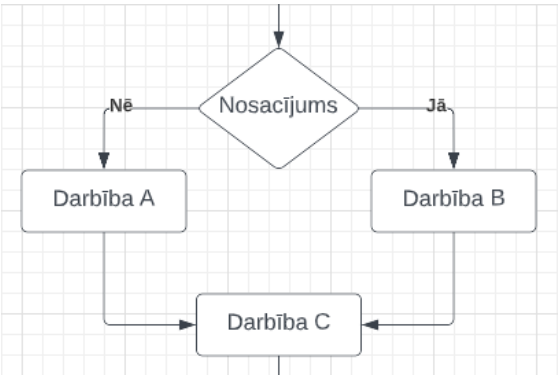
Att. 6.11. Blokshēmas fragments

Gadījumā, ja sensorā nav sarkanās krāsas, robots vienkārši brauc uz priekšu. Blokshēma:

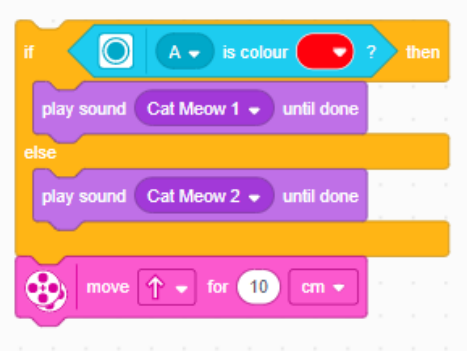
Šo bloku var saukt par "Ja, tad", jo gadījumā, ja izpildās nosacījums, tad tiek izpildīta attiecīgā darbība.

Tālāk apskatīsimies otru sazarojuma veida bloku un tā blokshēmu. (skat.att.6.12.).

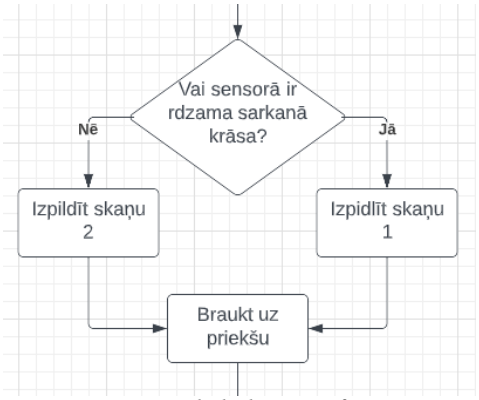
Patiesībā, šī blokshēma atbilst aplūkoto piemēru sazarojumiem. Gadījumā, ja tiek izpildīts nosacījums, seko darbība B, pretējā gadījumā seko darbība A un pēc tam ir darbība C neatkarīgi no iepriekšējās darbības. Ņemot nosacījumu no iepriekšējā piemēra par krāsu sensoru izpildīsim darbības ar līdzīgiem blokiem (skat.att.6.13) un arī šīs programmas fragmenta blokshēmu (skat.att.6.14.).



Att. 6.12. Blokshēmas fragments



Att. 6.13. Programmas fragments



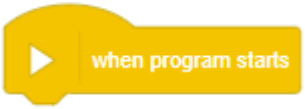
Att. 6.14. Blokshēmas fragments

Šeit, gadījumā, ja sensorā ir redzama sarkanā krāsa, tad tiek atskaņota skaņa "*Cat Meow 1"*, pretējā gadījumā tiek atskaņota skaņa: *"Cat Meow 2*". Un tad seko robota kustība uz priekšu.

6.3. Notikumi

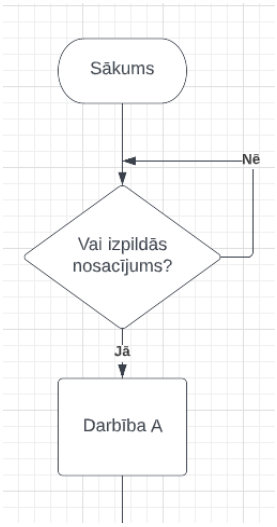
Lai mēs varētu veidot pilnvērtīgu programmu, mums ir nepieciešams programmas sākums. Lai to paveiktu ir sadaļa *"EVENTS"*, kurā ir vairāki bloki, kas ļauj mums to izdarīt. Apskatīsim šo sadaļu.

Šis bloks (skat.att.6.15.) uzsāk darbību brīdī, kad programma tiek palaista. Šo var uzskatīt par programmas sākumu, taču šim blokam ne obligāti ir jāparādās, kā tas jau tika novērots iepriekš. Pārsvarā šo bloku lieto tad, ja sākoties programmai vajag norādīt, pie kuriem portiem ir pieslēgti motori un to ātrumu iestatīšana vai citas tamlīdzīgas funkcijas. Šādus blokus var pievienot vairākus, taču jāskatās, vai vienādas darbības savā starpā nekonfliktē. Tas nozīmē, ka divus dažādus zīmējumus uz ekrāna vienlaicīgi nevar ieslēgt. Tas pats ar motoru kustību dažādos virzienos vienlaicīgi.



Att. 6.15. Sadaļas EVENT bloks

Šie bloki (skat.att.6.16.) jau iepriekš tika aplūkoti, bet apskatīsim tos detalizētāk. Kā arī citi *EVENT* sadaļas notikumi tie gaida, kamēr tiek izpildīts minētais nosacījums, kurš tiek iedarbināts ar kādu no sensoriem un tad uzsāk darbību. Tiem varētu atbilst blokshēma (skat.att.6.17.)



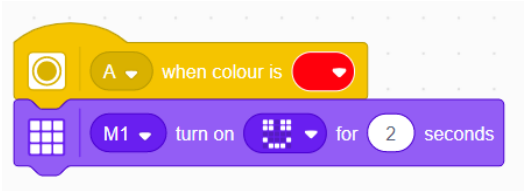
Att. 6.16. Programmas bloku piemēri *Att. 6.17. Blokshēmas fragments*

Var pamanīt, ka notikums gaida, līdz stāsies spēkā nosacījums, kas ir saistīts ar kādu no sensoriem. Tiklīdz tas ir spēkā, tad seko darbība, kas atrodas zem šī bloka.

Jautājums: vai šīs blokshēmas fragments ir attēlots pareizi?

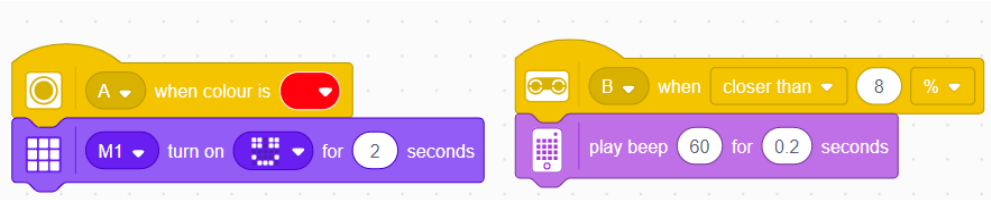
Atbilde: Ne līdz galam. Lai labāk saprastu, var aplūkot piemēru.

Kur nosacījums ir sarkanā krāsa sensorā un darbība A ir izgaismot zīmējumu uz galvenā bloka (skat.at..6.18.). Ja sensorā nav sarkanās krāsas, tad robots to gaida, pretējā gadījumā izpilda darbību A. Tik tālu, dotā blokshēma ir pareiza. Taču izpildot darbību A, blokshēmā bultiņa aizgāja tālāk un neatgriezās atpakaļ augšā. Taču realitātē, ja izpildās nosacījums un darbība A, sensorā var atkārtoti parādīt sarkano krāsu un iegūt darbību A. Blokshēmā šāda atkārtošanās netiek parādīta.



Att. 6.18. Programmas bloku piemērs

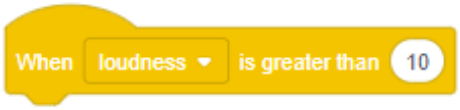
Otrkārt, ko vajadzētu aplūkot ir gadījumi, kad programmā parādās vairāki šādi bloki (sk.att.6.19.).



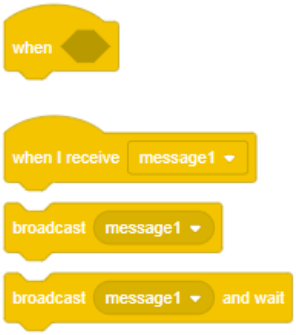
Att. 6.19. Programmas piemērs

Šeit mēģinot uzzīmēt blokshēmu, varētu rasties problēmas, jo ar blokshēmām nevar modelēt vairākas darbības, kas var tikt veiktas paralēli. Šādos gadījumos risinājums varētu būt vairāku blokshēmu izveide. Pareizāk, katrai kolonnai izveidot savu blokshēmu.

Att. 6.20. Programmas bloki



Att. 6.21. Programmas bloka piemērs

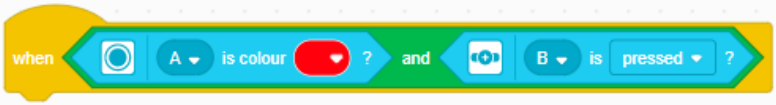


Att. 6.22. Programmas bloku piemēri

Turpinot aplūkot blokus sadaļā *"EVENTS"*, var redzēt, ka arī citiem blokiem (skat.att.6.20) darbības princips ir vienāds, vienīgi programma gaida nevis kādus sensora lasījumus, bet gan kādus notikumus no galvenā bloka vai darbības ar to, jo tajā arī ir iebūvēti sensori un tie arī padod signālu galvenajam blokam, ka notiek kaut kādas izmaiņas ar to.

Līdzīgi ir ar bloku, kas atskaņo trokšņus. Tas gaida, kamēr apkārtējais troksnis pārsniegs kādu iepriekš norādītu skaļumu (skat.att.6.21).

Sadaļā *"EVENTS"* ir arī citi bloki (skat.att.622.), kuri izskatās citādi. Uz tām nav nevienas ikonas, lai gan tie strādā pēc līdzīga principa - gaida, līdz būs spēkā nosacījums un tad turpinās zem tiem esošās darbības. Pirmajā blokā pašam ir iespējams ievietot kādu darbību. Tajos var ievietot sensoru lasījumus, taču tas būtu identiski kā lietot jau veselu bloku, kas sāk darboties pie kāda nosacījuma saistībā ar sensoru. Taču tajā ir iespējams ievietot krietni specifiskākus sensoru lasījumus, piemēram, gadījumus, kad vairāki nosacījumi tiek izpildīti vienlaicīgi. (skat.att.6.23)



Att. 6.23 programmas bloku piemērs

Gadījums, kad krāsu sensorā ir redzama sarkanā krāsa un spiedpoga ir nospiesta. Tas nozīmē, ka darbības zem šī bloka sāksies tikai tad, kad vienlaicīgi tiks izpildīti abi nosacījumi.

Uzdevums: uzzīmēt šim blokam atbilstošu blokshēmu.

Nākamie trīs bloki (skat.att.6.22.) ir savstarpēji saistīti. Pirmais ir "*When I recieve message1*", darbības, kas novietotas zem šī bloka sāk darboties tad, kad tiek saņemta īsziņa. Šo ziņu var nosūtīt izmantojot pārejos divus blokus. Viens no tiem vienkārši nosūta ziņu, tikmēr otrs nosūta ziņu un pēc tam gaida.

Šāda struktūra ir parocīga, kad vairākās vietās programmā ir nepieciešams veikt vienas un tās pašas darbības. Tad katru reizi nav jāveido programma, kas veic šo darbību, bet katrā vietā ievietot šo nosūtīt ziņu bloku ar vienu un to pašu ziņu.

7. Sensori un to programmēšana

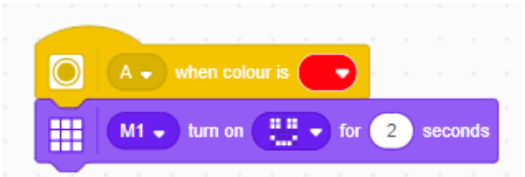
Grāmatas pirmajā sadaļā tika aplūkoti sensori un to darbība. Tagad mēs aplūkosim, kā un kādā veidā tos ir iespējams programmēt.

7.1. Krāsu sensors

Aplūkosim piemēru, kurā tiks demonstrēta krāsu sensora darbība. Lai varētu uzsākt, krāsu sensoru pievienosim galvenajam blokam portā A.

Uzdevums: Nepieciešams izveidot programmu, ar kuras palīdzību robots, nosakot sarkano krāsu uzsmaidīs. Šajā uzdevumā robotam nebūs vajadzība pārvietoties, tāpēc ka robots šeit var kalpot vienkārši - pie galvenā bloka pievienots krāsu sensors. Tālāk programmēšanas vidē izveido programmu (skat.att.7.1.).

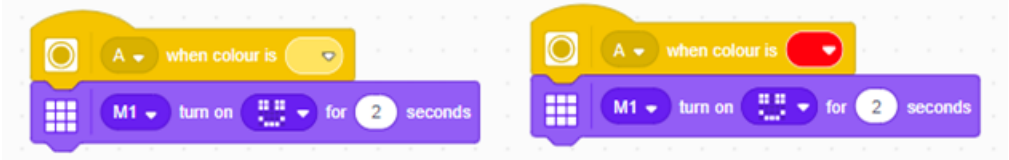
Sākuma bloku paņemsim no sadaļas *"EVENTS"*, kas nozīme "Notikumi" un otro no sadaļas *"LIGHT"* - "GAISMAS". Ja mēs pievēršam uzmanību pirmajam blokam, tad varēsim redzēt ikonu no bloka kreisās puses, kas nozīmē, ka šī darbība attiecas krāsu sensoram. Nākošajā posmā pēc programmas izveides ir nepieciešams robotā instalēt doto programmu un to palaist. Vajadzētu novērot, ka pietuvinot sarkano objektu pret sensoru, gaismām vajadzētu iedegties uz robota galvenā bloka un pēc 2 sekundēm nodzist. Taču pietuvinot jebkuras citas krāsas objektu, uz ekrāna nekas neiedegas. Tas nozīmē, ka augšējais bloks gaida notikumu, kas šajā gadījumā ir sarkanās krāsas objekta atpazīšana, un tad izpildās darbības, kas seko pēc izvēlētā notikuma. Izmainot programmas blokā krāsu uz, piemēram, dzeltenu varēsim sasniegt to pašu rezultātu, bet ar dzeltenās krāsas objektiem (skat.att.7.2.).



Att. 7.1. Programmas piemērs

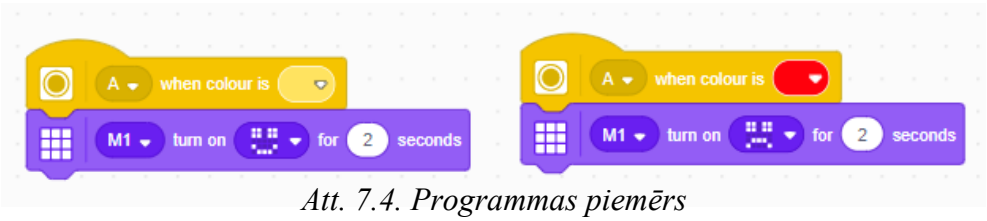
Att. 7.2. Programmas piemērs

Programmai vajadzētu darboties līdzīgi, bet notikums dotajā gadījumā ir dzeltenās krāsas objekts, nevis sarkanais objekts, ko nolasa sensors. Pamēģināsim apvienot abas programmas (skat.att.7.3.)



Att. 7.3. Programmas piemērs

Uz ekrāna vajadzētu iedegties gaismām, ja viena no šīm krāsām tiek novietota pie sensora. Rodas jautājums, kā izdarīt tā, lai atkarībā no krāsas, tiktu parādīts cits zīmējums, piemēram, bēdīga seja. Tam ir nepieciešams izveidot šādu programmu. (skat.att.7.4)



Att. 7.4. Programmas piemērs

Pamēģinām vienu pēc otra pie sensora pielikt klāt kādu sarkano objektu un tad dzeltenās krāsas objektu.

UZDEVUMS: *Aplūkot, kas notika šajā situācijā. Apspriest rezultātus***.**

Piemēram, ja pirmo pievieno dzelteno un tad sarkano, vajadzētu novērot, ka uz nelielu mirkli iedegās smaidīgā seja, kuru nomainīja bēdīgā seja. Pavisam vienkārši, sensorā parādoties noteiktai krāsai, notiek darbība, kas atrodas zem tās. Nav svarīgi, cik ātri mainās krāsa sensorā. Taču tā kā nav iespējams vienlaicīgi izpildīt šīs darbības, tiks izpildīta pēdējā.

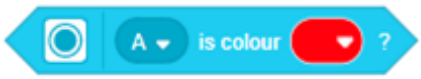
Pamēģināsim nedaudz pārveidot programmu, lai nosakot sarkano objektu, robots izpildītu skaņas paziņojumu. Lai to izdarītu ir nepieciešams bloku ar gaismām nomainīt ar bloku no skaņas *"SOUND*" sadaļas**.** (skat.att7.5.)

Att. 7.5. Programmas piemērs

UZDEVUMS: *Pārbaudīt, kā darbojas robots pietuvinot dažādas krāsas objektus. Pamēģināt objektus mainīt ātri. Vajadzētu novērot, ka iedegas gan smaidīga seja, gan tiek izpildīts skaņas signāls.*

Ir pieejami arī citi bloki, kas ir saistīti ar krāsu sensoru (skat.att.7.6.) Šos blokus ir iespējams atrast sadaļā *"SENSORS"* - SENSORI.

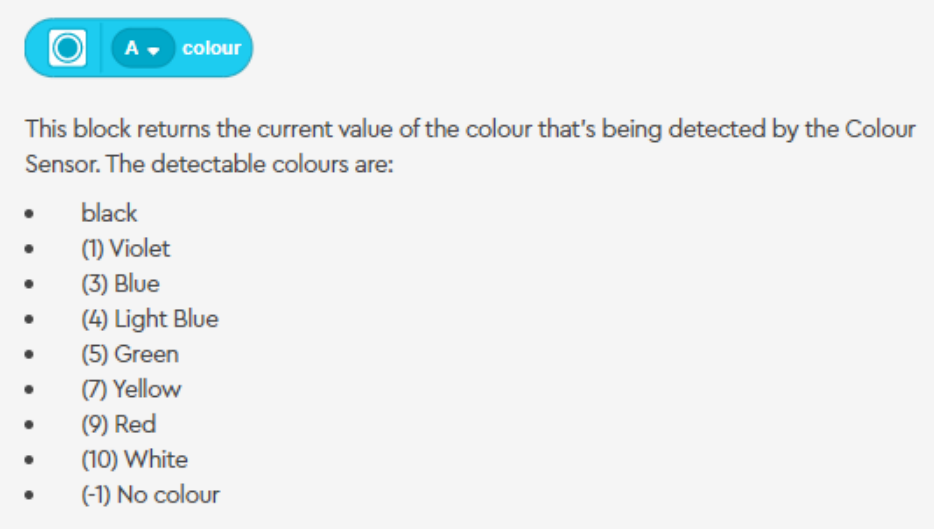
Šo bloku var uzskatīt par loģisko bloku. To var lietot veidojot sazarojumus. Šis bloks var pieņemt vērtību, kas ir vai nu patiess vai aplams. Pēc paša bloka var redzēt, ka tas jautā, vai krāsu sensorā ir redzama krāsa (šeit sarkanā). Ja tā ir, tad šis bloks pieņem patiesu vērtību. Pretējā gadījumā aplamu. Aplūkojot iepriekš zīmētās blokshēmas ar



Att. 7.6. Programmas bloks krāsu sensoram

sazarojumiem, šos blokus lieto sazarojumu vietās. To, kā tas izskatās programmā, tiks aplūkots tālāk.

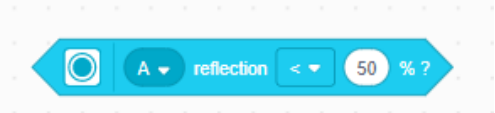
Nākošais bloks nedaudz atšķiras no pirmā. Šis bloks pavērš atpakaļ nevis patiess vai aplams vērtības, bet gan katrai krāsai piešķir savu ciparu un to vērš atpakaļ. Tas ir, sensorā parādoties sarkanajai krāsa, tiek pavērsta atpakaļ vērtība 9. Šīs vērtības ir atrodamas klikšķinot ar labo pogu uz šī bloka un atverot *HELP* sadaļu (skat.att.7.7.).



Att. 7.7. Programmēšanas bloka vērtības atkarībā no krāsas

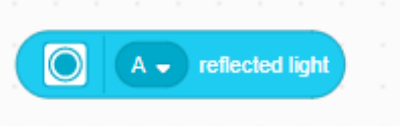
Šajā aprakstā nav pievienots, taču parādoties melnajai krāsai, šis bloks pieņems vērtību 0.

Nākamie divi bloki ir ļoti līdzīgi pirmajiem diviem blokiem. Piemēram



Att. 7.8. Programmas bloks, kura vērtības ir patiesas vai aplamas

Šis bloks (skat.att.7.8.) mēra atstarojošās gaismas intensitāti. Atšķirībā no pirmā bloka tas arī pavērš atpakaļ vērtību patiess vai aplams, bet šeit ir iespējams vienādību nomainīt ar nevienādībām un nomainīt vēlamo atstarojošās gaismas intensitāti.



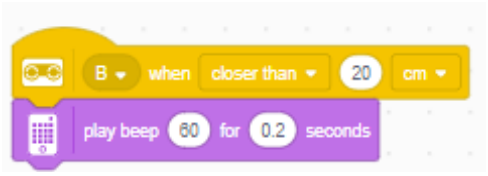
Att. 7.9. Programmas bloks, kas pavērš atpakaļ gaismas intensitāti

Šis bloks (skat.att.7.9.) pavērš atpakaļ skaitlisko atstarotās gaismas intensitātes vērtību. Šī skaitliskā vērtība ir procentos, līdzīgi, kā iepriekš.

7.2. Attāluma sensors

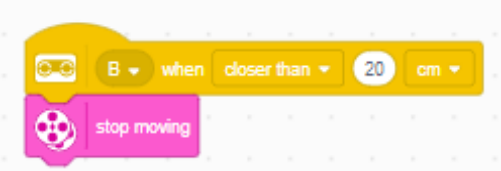
Tālāk tiks aplūkot piemērs, kurā tiks parādīta attāluma sensora darbība. Pirmais solis būtu izveidot pašu robotu, kas spētu realizēt šo uzdevumu. Tiek piedāvāts izveidot robotu ar nosaukumu "*DELIVERY CART*", kuru var atrast *BUILD* sadaļā. Tālāk tiek izveidota programma (skat.att.7.10.), kas robotam liek paziņot, ja kāds objekts tam tuvojas.

Augšējais bloks atrodas sadaļā *"EVENTS"* un otrs pie *"SOUNDS"*. Jāpievērš uzmanība tam, ka izvēlētais bloks no sadaļas *"EVENTS"* ir ar attāluma sensora ikonu. Palaižot programmu, vajadzētu novērot, ka pietuvinot kādu objektu tuvāk par 20 cm attāluma sensoram, robotam vajadzētu atskaņot signālu. Šo programmas fragmentu var savienot ar robota kustību, lai tas izdod skaņu, kad kāds objekts parādās tam priekšā. Kaut ko līdzīgu var saskatīt reālajā dzīvē, kad autovadītāji signalizē trauksmes gadījumā.



Att. 7.10. Programmas piemērs

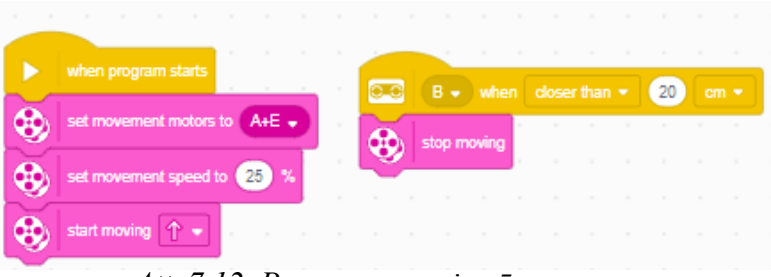
Taču padomājot par reālo dzīvi, iespējams būtu loģiskāk nevis sākt taurēt, bet gan apstādināt robotu, ja kāds objekts tam ir priekšā. Lai to izdarītu nomainīsim bloku no sadaļas *"SOUNDS"* ar bloku no sadaļas *"MOVEMENTS*" - "KUSTĪBAS", kas apstādina robotu *"STOP MOVING"* (skat.att.7.11).



Att. 7.11. Programmas piemērs

Taču šajā piemērā būtu nepieciešams pievienot papildus blokus, kas sākoties programmai robotam norāda, pie kuriem portiem ir pievienoti motori, kā arī kustības sākums. Risinājums varētu izskatīties šādi:

Attēlā (skat.att.7.12) redzamā programma sastāv no divām kolonnām. Kolonna labajā pusē ir palikusi nemainīga, tāpēc atliek aplūkot izmaiņas kolonnā pa kreisi. Vispirms var minēt, ka visi trīs bloki ir atrodami sadaļā "*MOTORS*". Pirmais bloks pasaka, ka pie portiem A un E ir pievienoti motori. Otrais bloks pasaka, ka šo motoru ātrums tiks iestatīts uz 25%. Tas nozīmē, ka robotam pasakot, ka jākustina motori, tas zinās, ka tie jākustina ar šādu ātrumu. Un tad pēdējais bloks pasaka to, ka robotam ir jāsāk braukt uz priekšu.



Att. 7.12. Programmas piemērs

Taču palaižot programmu, varētu novērot problēmu. Robots uzsāk kustību, līdz priekšā pamana objektu un apstājas. Bet apstājoties, robotam nekur nav dota komanda turpināt darbību.

Uzdevums: Veikt labojumus programmā, lai robotam būtu arī iespējams atsākt kustību, ja priekšā nav neviena objekta.

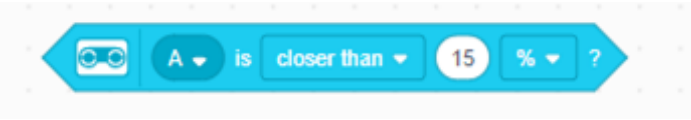
Kā risinājumu var piedāvāt šādu programmu (skat.att.7.13.).

Att. 7.13. Programmas piemērs

Šeit, sākoties programmai, joprojām tiek parādīts, kuros portos ir motori un ar kādu ātrumu ir jāpārvietojas. Taču kustība uz priekšu ir novietota zem bloka, kas sāk darboties tikai tad, ja attālums sensorā ir virs 20 cm. Tas nozīmē, ka jebkurā laika momentā, ja sensora lasījumi ir virs 20 cm robots kustas uz priekšu, pretējā gadījumā robots stāv uz vietas.

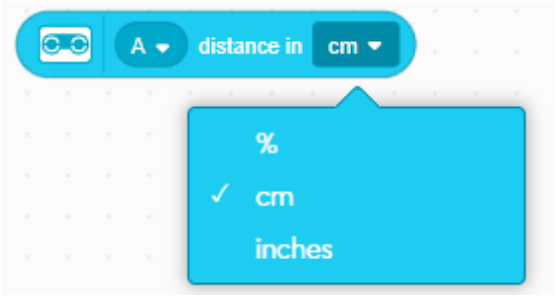
JAUTĀJUMS: *Kas notiek, ja roku pietuvina pie paša sensora? Apspriest rezultātus.*

Aplūkojot sadaļu "Sensors", saistībā ar attāluma sensoru ir šie bloki:



Att. 7.14. Programmas bloka piemērs, kas atgriež vērtības patiess/aplams

Att. 7.14. ir attēlots loģiskais bloks, kas salīdzina, vai dotajā brīdī sensora lasījumi ir lielāki vai mazāki nekā norādītā vērtība. Šajā gadījumā norādītā vērtība ir mazāka nekā 15% . Šie 15 % ir no maksimālās distances, ko sensors ir spējīgs uztvert, kas ir 30cm. Tātad, ja sensora maksimālā distance, kuru tas uztver ir 200cm , tad 50% no tā būs vienkāršs puse. Lai iegūtu 15%, maksimālo distanci var vienkārši reizināt ar 0.15.



Att. 7.15. Programmas bloks ar iespēju veikt mērījumus dažādās mērvienībās

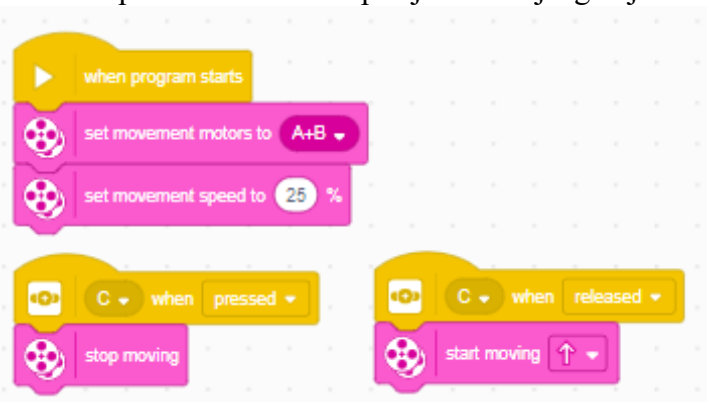
Ar šī bloka palīdzību ir iespējams saņemt sensora lasījumus dažādās mērvienībās (skat.att.7.15.). Līdzīgi var izvēlēties arī pirmajā blokā. Tātad, ja sensors uztver objektu 50cm attālumā no sensora, tad šī bloka vērtība būs 50. Šis bloks ļauj lietot sensora lasījumus kā skaitlisko vērtību.

7.3.Pieskāriena sensors

Apskatīsim nākamo sensoru - pieskāriena sensors. To var saukt arī par pogu vai spiedpogu. Tā darbības princips ir pavisam vienkāršs, to var nospiest vai atlaist. To ir iespējams nospiest divos dažādos veidos, tas ir, vienkārši nospiest vai "iespiest dziļāk". Pamēģinot paspaidīt, vajadzētu sajust, ka līdz pusei pogu ir diez gan vienkārši nospiest, taču tālāk ir nepieciešams nedaudz vairāk spēka. Kā tas iepriekš tika aprakstīts, šis sensors ir spējīgs uztvert spēku no 2 līdz 10 Ņūtoniem.

Tālāk tiks aplūkots piemērs ar pieskāriena sensoru. Šeit katrs skolēns var izveidots savu robota konstrukciju vai lietot *BUILD* sadaļā atrodamu konstrukciju *"RHINO"*. Tiek izveidota programma, kur robots brauc, ja sensors nav nospiests un pretēji - apstājas:

Ir pirmā kolonna (skat.att.7.16.), augšā, kas iestata motorus un to ātrumu. Pa kreisi, apakšā, kolonna darbojas, kad sensors tiek nospiests un robots apstājās. Pretējā gadījumā darbojas kolonna apakšējā, labajā stūrī, kas nosaka, ka robots brauc uz priekšu, ja poga nav nospiesta.



Att. 7.16. Programmas piemērs

Lai apskatītos, kā darbojas šī programma, vislabāk ir to izveidot un palaist uz sava robota. Programmas darbības princips ir šāds: ja poga ir nospiesta, motors sāk griezties un ja šo pogu atlaiž, motors apstājas.

7.4. Žiroskops

Vēl viens sensors, kuru mēs apskatīsim ir žiroskops. Kā jau sākumā tika aprakstīts, šis sensors nav kā atsevišķs sensors, bet gan atrodas robota galvenā bloka iekšpusē. Programmēt šo sensoru ir pavisam vienkārši. Atliek galveno bloku pievienot pie savas ierīces un izveidot programmu žiroskopam (skat att. 7.17.).

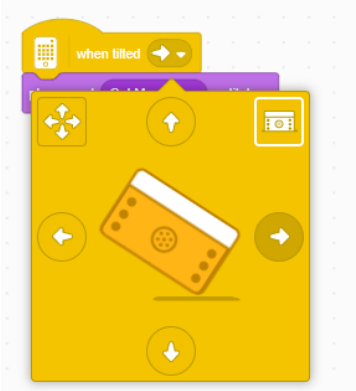
Att. 7.17. Programmas piemērs

Programmas izveidei izmantojam bloku no sadaļas "*EVENTS*" - "NOTIKUMI", kur izvēlamies bloku ar žiroskopa ikonu un mums nepieciešamo darbību. Lai konstatētu žiroskopa darbību izmantosim vēl vienu bloku ar skaņas paziņojumu. Rezultātā mēs saņemsim programmu, kas noskaņos izvēlēto signālu ja mēs sasvērsim galveno bloku pa labi.

Svarīgi, ka šī skaņa atskanēs uz ierīces, pie kuras ir pievienots robots nevis uz paša robota. Tāpēc jāpārliecinās vai nav izslēgta skaņa.

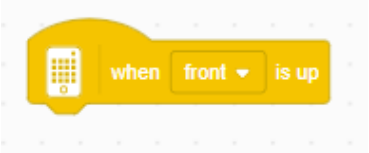
Šajā programmā izmantotais *"EVENTS"* bloks (skat.att.7.18.) ļauj uztvert dažādas galvenā bloka kustības.

Ir iespējams uztvert tā svārstības uz visām pusēm. Piemēram, šajā kodā tas uztver to mirkli, kad robots sasvērās uz labo pusi. Nospiežot blokā uz bultiņas parādās logs, kurā noklikšķinot uz attiecīgās bultiņas, tiks parādīts kāda kustība tiek uztverta. Augšējā, kreisajā stūrī ir opcija, kas ļaus uztvert visas četras kustības vienlaicīgi. Un labajā, augšējā, stūrī ir variants, kas uztver tieši to mirkli, kad robota galvenais bloks ir miera stāvoklī, nav sagrozīts.



Att. 7.18. Programmas bloka piemērs

UZDEVUMS: Lai labāk saprastu katru no šiem variantiem, pamēģiniet pamainīt šajā blokā uztveramo galvenā bloka stāvokli un skatīties, kā strādā programma to pamainot. Rezultātus apspriest grupās.

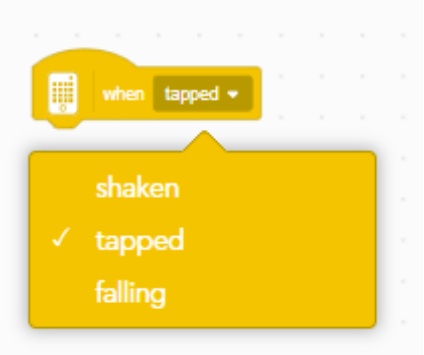


Att. 7.19. Programmas bloka piemērs

PIEZĪME: Ja robots tiek sasvērts uz labo pusi un paliek tādā stāvoklī, tad šis bloks to neuztvers kā atkārtotu kustību pa labi.

Apskatīsim citu bloku (sk.att.7.19.)

Tas uztver, kura no galvenā bloka malām ir augšā. Ja aplūkosim robotu konstrukciju *"Driving base 1"*, kas ir pieejama programmēšanas vidē, tad robotu novietojot uz riteņiem, uz augšu būtu tā galvenā bloka mala, uz kuras ir pogas un LED gaismas. Ja galvenais bloks nemaina savu orientāciju, tad šis bloks atkārtoti neuzsāks savu darbību.



Att. 7.20. Programamas bloka piemērs

Vēl šis sensors ļauj robotam uztvert cita veida kustības, kā piemēram, pieskārienu, krišanu vai kratīšanu (skat.att.7.20.).

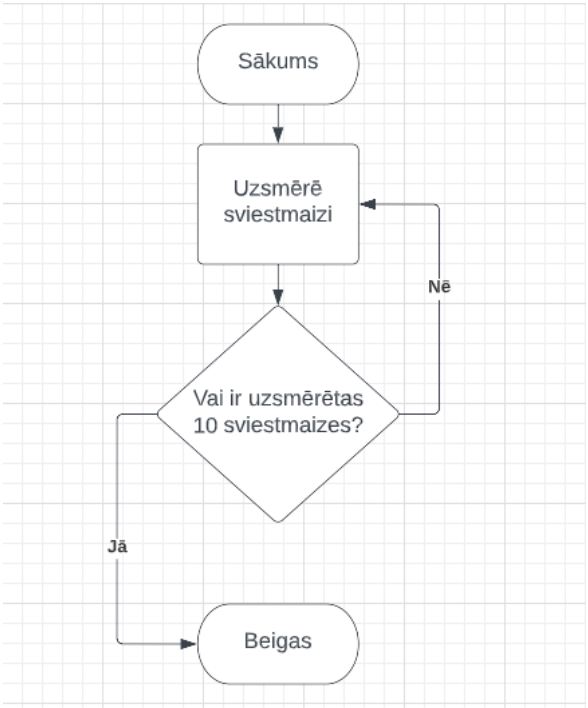
PIEZĪME: Šeit gan jābūt uzmanīgiem, jo sensors var neuztvert salīdzinoši mazus pieskārienus vai kustības.

8. Cikli

Cikli jeb cikliskas darbības. Iespējams ikdienā tam netiek pievērsta tik liela uzmanība, bet arī cikliskas darbības var novērot ikdienā. Var aplūkot fiziskās aktivitātes. Piemēram, apskriet stadiona apli. Ja tas ir jāveic vairākas reizes, to var nosaukt par ciklisku darbību. Cits piemērs varētu būt naglas iedzīšana ar āmuru. Kamēr nagla nav iesista, nepieciešams turpināt sist ar āmuru.

Ciklus izmanto tad, kad ir kāda darbība vai darbību kopums, kuras jāatkārto vairākas reizes. Piemēram, naglas sišana, šī darbība būtu āmura atvēzēšana un sišana pa naglu. Un šīs darbības tiek atkārtotas, līdz nagla ir pilnībā iesista. Šajā piemērā parādās arī sazarojuma algoritms, beigu nosacījums ciklam: naglas iesišana līdz galam. Tas nozīmē, ja nagla nav iesista, ir nepieciešams atkārtot darbību. Ja nagla ir iesista, ciklu nepieciešams pārtraukt. Aplūkojot algoritmus, šāda konstrukcija bieži parādīsies, cikli būs kopā ar sazarojumiem, bet ne vienmēr. Dažreiz ir nepieciešams izveidot ciklu, kuram nav šāda beigu nosacījuma, tas izpildās mūžīgi.

Cits piemērs varētu būt, lai uzsmērētu 10 sviestmaizes, desmit reizes tiek atkārtota sviestmaizes smērēšanas darbība. Lai izveidotu algoritmu, pēc kura robots spētu uzsmērēt 10 sviestmaizes, vajag izveidot tikai algoritmu, kas uzsmērē vienu sviestmaizi un šo darbību atkārto 10 reizes. Līdzīgi ir ar 15, 20 vai 100 sviestmaizēm. Dotā algoritma blokshēma ir attēlota zemāk (skat.att.8.1.)



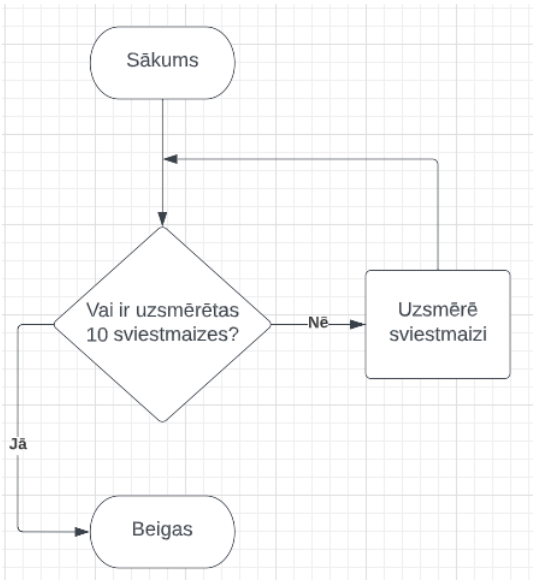
Att. 8.1. Blokshēma ar ciklu

Šeit, tiek uzsmērēta viena sviestmaize un pārbaudīts, vai ir uzsmērētas 10. Gadījumā, ja ir, tad algoritms beidz darbību, pretējā gadījumā uzsmērē vēl vienu sviestmaizi un atkārtoti pārbauda, vai ir 10 sviestmaizes kopā. Taču šim algoritmam ir viens trūkums. Gadījumā, ja vajag uzsmērēt nulle sviestmaizes, viena sviestmaize tik un tā tiks uzsmērēta.

JAUTĀJUMS: *Kā izlabot doto algoritmu?*

ATBILDE: Nepieciešams samainīt vietām darbības - uzsmērēto sviestmaižu skaita noteikšanu ar sviestmaizes smērēšanu. Tas ir, sākumā pārbauda, vai ir uzsmērēts vajadzīgais sviestmaižu skaits un tad uzsmērē sviestmaizi (skat.att.8.2.)

Ciklu, kuram cikla izpildes nosacījums tiek pārbaudīts, kad jau darbība tika veikta sauc par **ciklu ar pēc-nosacījumu.** Savukārt ciklu, kuram pirms darbību izpildes tiek pārbaudīts cikla izpildes nosacījums, sauc par **ciklu ar priekšnosacījumu.** Tas nozīmē, ja mēs izmantosim ciklu ar priekšnosacījumu, tad netiks uzsmērēta neviena sviestmaize, ja būs jāuzsmērē 0 sviestmaizes. Svarīgi šeit zīmēt, ka atkarībā no uzdevuma, variants, kurā izdara darbību un pēc tam veic salīdzināšanu vai otrādi var būt pareizi.



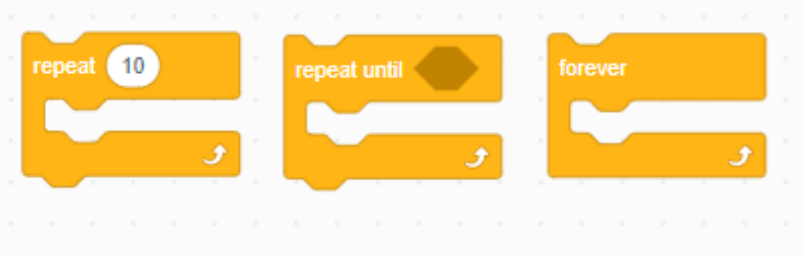
Att. 8.2. Blokshēma ar ciklu

Līdzīgi, kā ar sazarojumiem, arī ciklu ir iespējams ievietot citā ciklā.

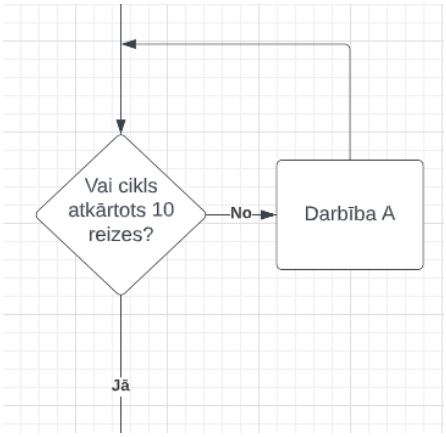
UZDEVUMS: izdomāt piemēru, kurā būtu nepieciešams ciklu ievietot ciklā.

8.1. Programmas apskats cikliem

Veidojot programmu, kas lietos ciklus, tiks lietoti šie bloki (skat.att.8.3.)



Att. 8.3. Programmas bloku piemēri

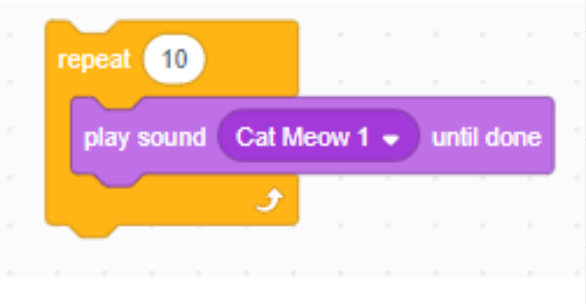


Att. 8.4. Blokshēmas fragments

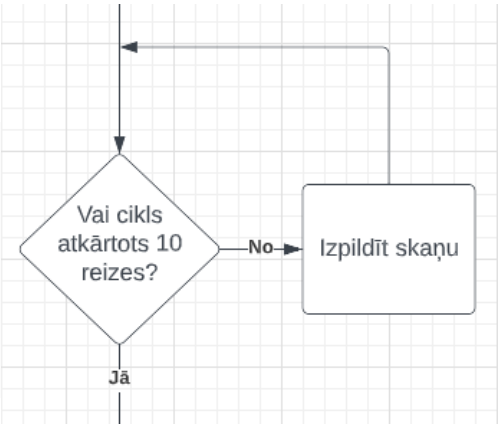
Pirmais bloks raksturo ciklu, kas tiek atkārtots noteiktu skaitu reižu. Šajā gadījumā, ciklā ievietotās darbības tiks atkārtotas 10 reizes. Tam varētu atbilsts blokshēma (skat.att.8.4.)

Šeit, darbība A tiek atkārtota 10 reizes. Skaitli 10 var aizstāt ar jebkuru citu pozitīvu skaitli.

Tālāk darbības A vietā var ievietot skaņas atskaņošanu (skat.att.8.5.)

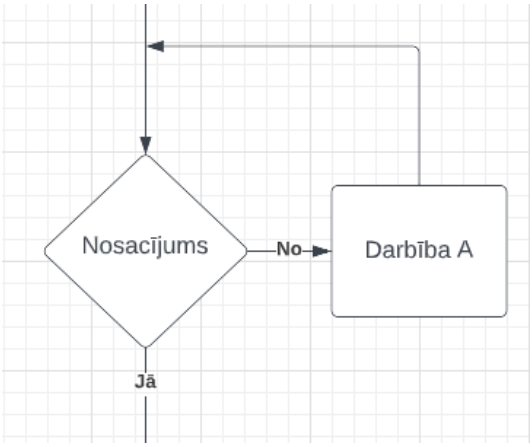


Att. 8.5. Programmas bloku piemērs



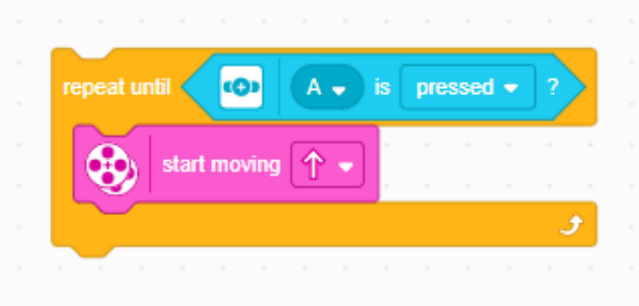
Att. 8.6. Blokshēmas fragments

Šajā programmas fragmentā, skaņa ”Cat Meow 1” tiek atkārtota 10 reizes. Tam atbilstu blokshēma (skat.att.8.6.)



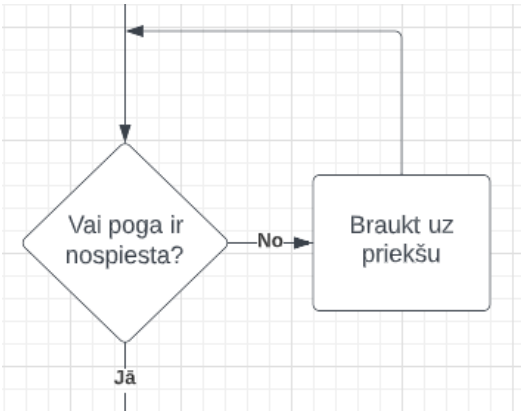
Att. 8.7. Blokshēmas fragments

Tālāk aplūko otru cikla bloku. Atšķirībā no pirmā bloka, šeit ir iespējams ievietot patvaļīgu beigu nosacījumu, tas ir, nevis atkārtot 10 reizes, bet, piemēram, cikls darbību beidz, kad poga tiek nospiesta. Blokshēmas piemērs blokam (skat.att.8.7.)



Att. 8.8. Programmas bloku piemērs

Ir kaut kāds nosacījums. Gadījumā, ja tas nav spēkā, tiek veikta darbība A un atkārtoti pārbauda, vai šis nosacījums ir spēkā. Gadījumā, ja nosacījums ir spēkā, cikls savu darbību beidz un programma izpilda darbības, kas seko pēc tās. Turpinot piemēru ar spiedpogu, programmas kods varētu izskatīties (skat.att.8.8.)



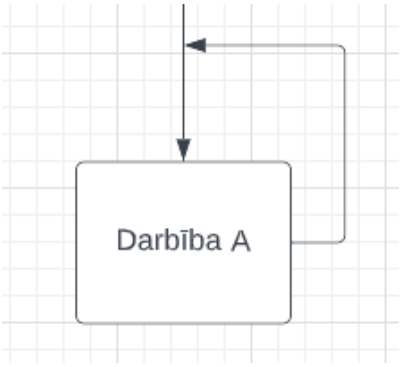
Att. 8.9. Blokshēmas piemērs

Kamēr poga nav nospiesta, robots turpina braukt uz priekšu. Blokshēma dotajam programmas fragmentam (skat.att.8.9.).

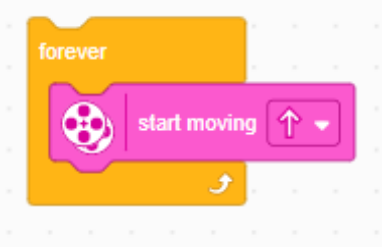
Tālāk atliek aplūkot pēdējo bloku. Šo bloku var saukt par mūžīgo ciklu. Tam sākoties, tas nebeidz savu darbību. Aplūkojot pašu bloku, tam arī nav redzamas iespējas ievietot kādu beigu nosacījumu. Tāpēc ar šo bloku ir jāuzmanās, nevietā

ievietots, tas programmai var likt darboties pilnībā savādāk, kā tas bija plānots. Šī bloka blokshēma izskatās šādi (skat.att.8.10.).

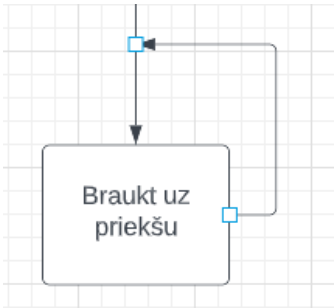
Šeit darbība A tiek atkārtota bezgalīgi daudz reižu. Piemēram, programmas fragments, kas liktu robotam mūžīgi braukt uz priekšu izskatītos šādi (skat.att.8.11.), kā arī tās blokshēma (skat.att.8.12.).



Att. 8.10. Blokshēmas piemērs



Att. 8.11. Programmas bloku piemērs



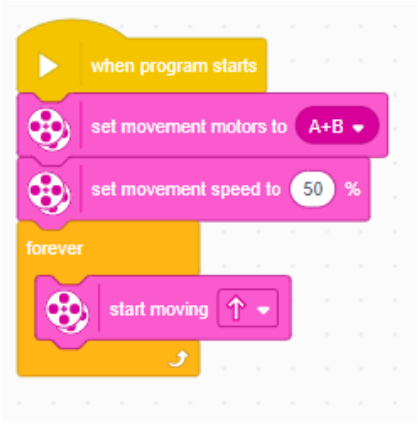
Att. 8.12. Blokshēmas fragments

Pie sazarojumiem varēja novērot, ka sensoru darbība tiek lietota nedaudz savādāk, nekā pirms tam. Pirms tam sensorus darbināja lietojot blokus sadaļā *"EVENTS"*. Var teikt, ka bloki, kas atrodas tajā sadaļā strādā kā sazarojuma bloki. Tie gaida, kad notiks nosacījums, kas raksturo doto bloku un tad izpildīsies darbība zem tā.

8.2. Piemēri ar kodu

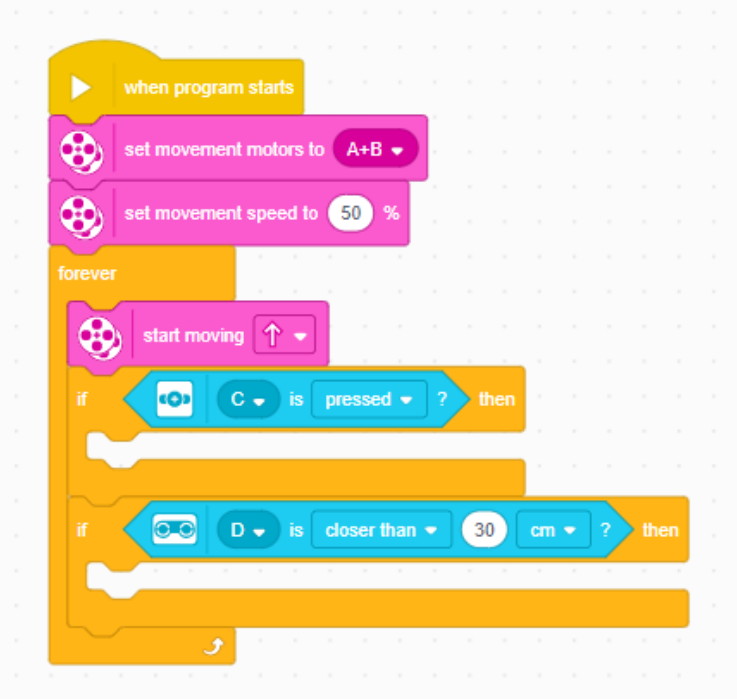
Šajā uzdevumā tiks aplūkots, kā sazarojumus var lietot kopā ar sensoriem. Šajā uzdevumā nav dota robota konstrukcija, kādu būtu jāizmanto, taču var lietot robotu *"Driving Base 1"* ar pievienotu pieskāriena sensoru. Ir nepieciešams izveidot programmu, kas liek robotam braukt atpakaļ gaitā, pagriezties un turpināt braukt taisni, kad attāluma sensors uztver distanci zem 30 cm vai arī tiek nospiesta poga. Pretējā gadījumā robots vienkārši brauc uz priekšu.

Arī šoreiz ir vieglāk gala programmu sadalīt vairākos soļos, kurus saliekot kopā tiks iegūts vajadzīgais rezultāts. Tas ir, sākumā jāizveido programma, kas robotam liek vienkārši braukt uz priekšu pēc tā palaišanas (skat.att.8.13.)



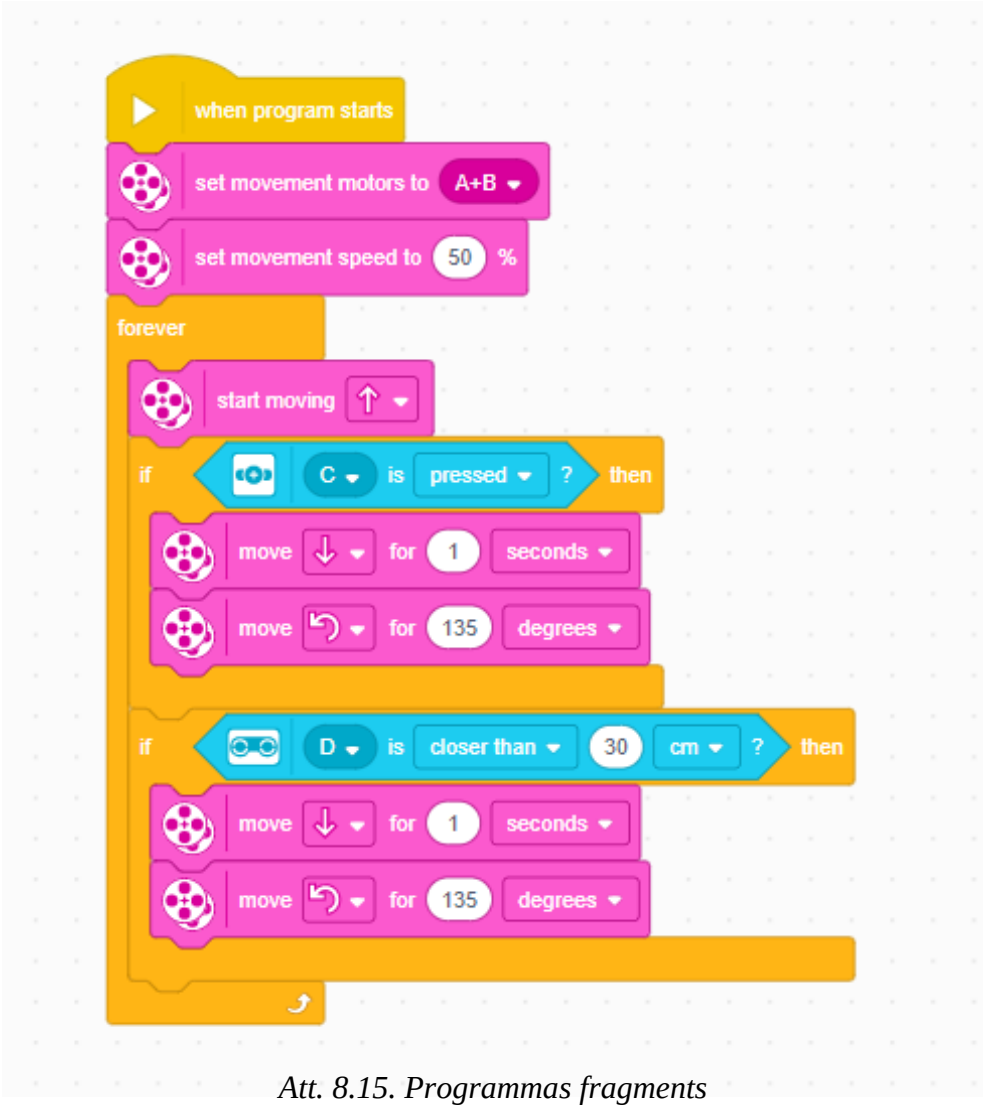
Att. 8.13.Programmas fragments

Tālāk jāpievieno programmā salīdzināšana, kas pārbaudīs, vai attāluma sensorā ir vajadzīgā distance vai arī poga ir nospiesta (skat.att.8.14.).



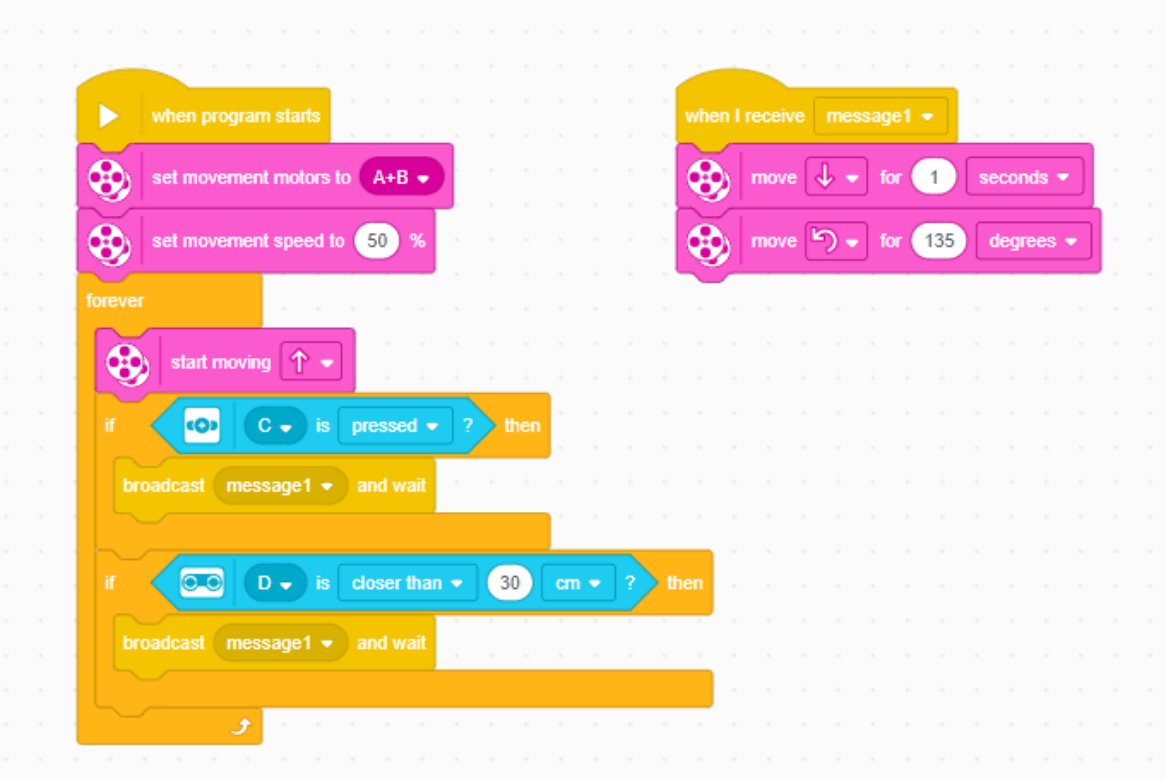
Att. 8.14.. Programmas fragments

Šeit ir pievienoti divi "Ja, tad" bloki, kas veic vajadzīgo salīdzināšanu. Robots brauc uz priekšu un veic salīdzināšanu. Šīs darbības notiek viena pēc otras un bez apstājas, jo ir ievietotas mūžīgajā ciklā. Tālāk jāpievieno robota apgriešanās. Tā kā uzdevumā netika precizēta distance, ar kādu ir jābrauc atpakaļgaitā un cik daudz ir jāpagriežas, tad to var pievienot pēc paša izvēles (skat.att.8.15.).



Att. 8.15. Programmas fragments

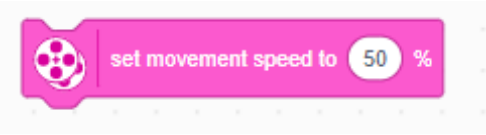
Šeit tiek piedāvāts braukt atpakaļ gaitā 1 sekundi un pagriezties uz kreiso pusi par 135 grādiem. Šeit redzams, ka divās vietās ir pilnīgi identiskas darbības. Lai gan šajā piemērā tiek ir tikai divi bloki, var gadīties piemēri, kad šīs darbības var sastāvēt no vairākiem blokiem. Piedāvātais risinājums ir ieviest ziņu pārraidi (skat.att.8.16.).



Att. 8.16. Programmas fragments

Katrā vietā, kur notikusi darbība, kas atkārtojās, tiek ieviests bloks, kad pārraida ziņu. Un otrā kolonna, saņemot šo ziņu, izpilda pabraukšanu atpakaļ un pagriešanos. Tiek lietots bloks, kas pārraida ziņu un gaida, jo tādā gadījumā tiks pārtraukta braukšana uz priekšu.

Tālāk tiks aplūkots vēl kāds piemērs, kā var lietot sensoru datus. Tas ir, aplūkojot robota kustības ātruma bloku (skat.att.8.17.).

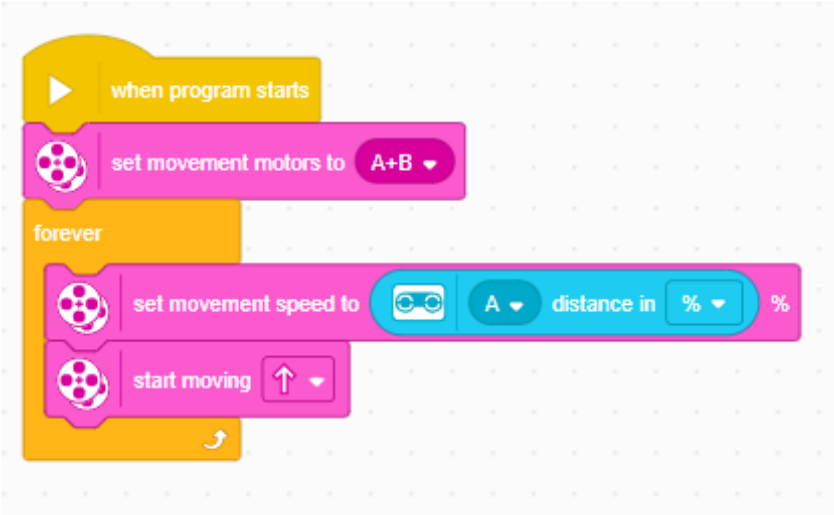


Att. 8.17. Programmas bloks

Var redzēt, ka tajā ir iespējams ievietot sensora lasījumus (skat.att.8.18.).

Att. 8.18. Programmas bloki

Svarīgi, ka šeit tiek lietoti sensora lasījumi kā procenti, jo ātruma procenti nevar pārsniegt 100, bet cm šajā gadījumā varētu pārsniegt. Tātad ievietojot šo fragmentu ciklā, iegūst (skat.att.8.19.)



Att. 8.19. Programmas fragments

Palaižot programmu vajadzētu novērot, ka robots brauc ar maksimālo ātrumu, ja sensoram nav nekā priekšā. Taču parādoties kādam objektam, tas lēnām sāks samazināt savu ātrumu.

Šādā formātā varbūt nav diez cik praktiski lietot šo attāluma sensoru. Dažkārt praktiskā uzdevumā būs ērtāk šo sensora lasījuma vērtību ievietot kādā matemātiskā funkcijā. Jo, robots apstāsies tikai tad, ja tam pilnīgi priekšā būs kaut kas. Bieži ērtāk būs vienkārši sensoru lasījumu vietā ievietot kādu mainīgo, kas satur sensoru datus, kuri ir apstrādāti ar kādu matemātisko formulu. Par mainīgajiem būs minēts nākamajā nodaļā.

8.3. Uzdevumi ar blokshēmām

Sadalīties grupās vai pāros un aplūkot jautājumus:

1. **Uzdevums-diskusija**

a. Kādas iespējas paver sazarojumi, kuras nebija pieejamas runājot par vienkārši lineāriem algoritmiem?

b. Kāda veida problēmas var risināt tagad, kad ir ieviesti sazarojumi, bet nevarēja risināt tad, kad bija tikai lineārie algoritmi?

c. Vai ikdienā biežāk parādās algoritmi, kas ir lineāri vai kas ir ar sazarojumiem? Kāpēc tā tas varētu būt?

2. **Uzdevums-diskusija**

* Vai no pieejamajiem sensoriem ir kāds, ar kuru var sastapties ikdienā?
* Kādi praktiski lietojumi ikdienā varētu būt pieejamajiem sensoriem?
* Vai ir vēl kāds sensors vai sensori, kurus būtu vērts pievienot, lai varētu atrisināt reālās dzīves problēmas?

3.Uzdevums - Diskusija

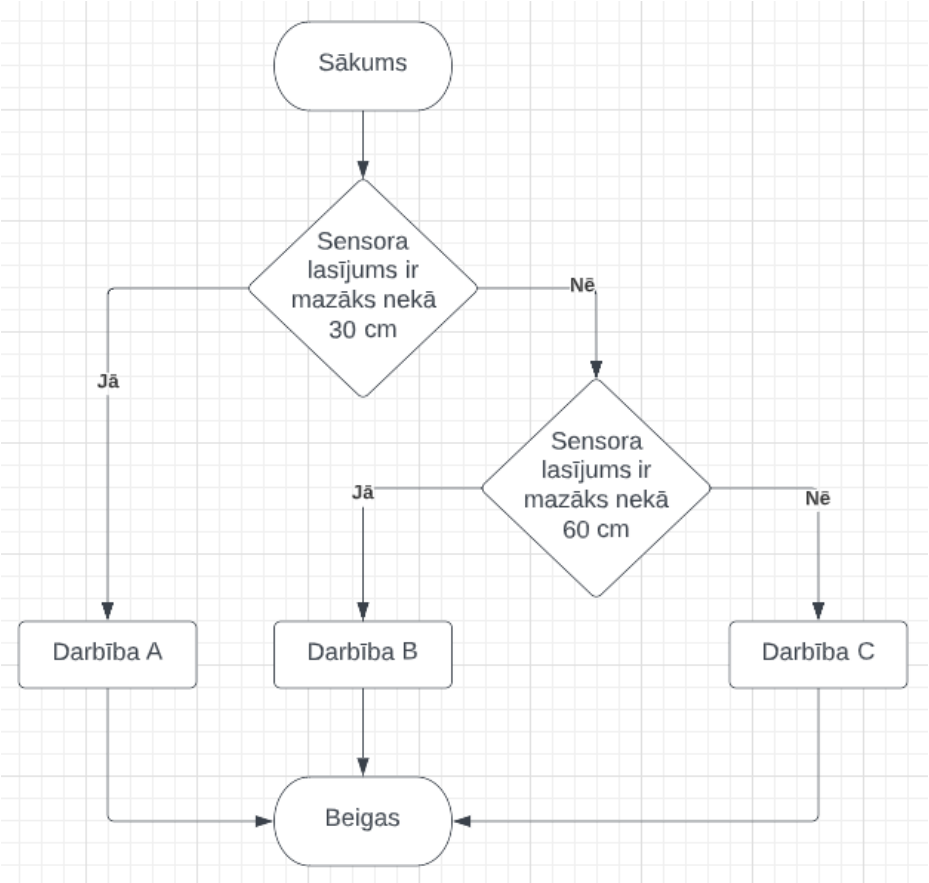
- Kādas iespējas paveras, kad ir pieejami gan cikli, gan sazarojumi?
- Kādas papildus problēmas var risināt tagad, kad ir pieejami cikli?
- Kādas varētu būt iespējamās problēmas, no kurām vajadzētu izvairīties lietojot ciklus un sazarojumus?

4. Uzdevums - Diskusija

Viena no vietām, kur parādās gan roboti, gan arī sensori, ir automašīnas, kas pašas spēj braukt. Bet kā tieši šīs automašīnas spēj pašas braukt? Kādi sensori tām ir nepieciešami? Kāds varētu būt algoritms, kas viņām ļauj pašām braukt? Uzdevums ir veidot diskusiju par šiem jautājumiem, kā arī mēģināt ieskicēt blokshēmu, kas varētu atbilst šo automašīnu darbībām.

Jautājumi par blokshēmām

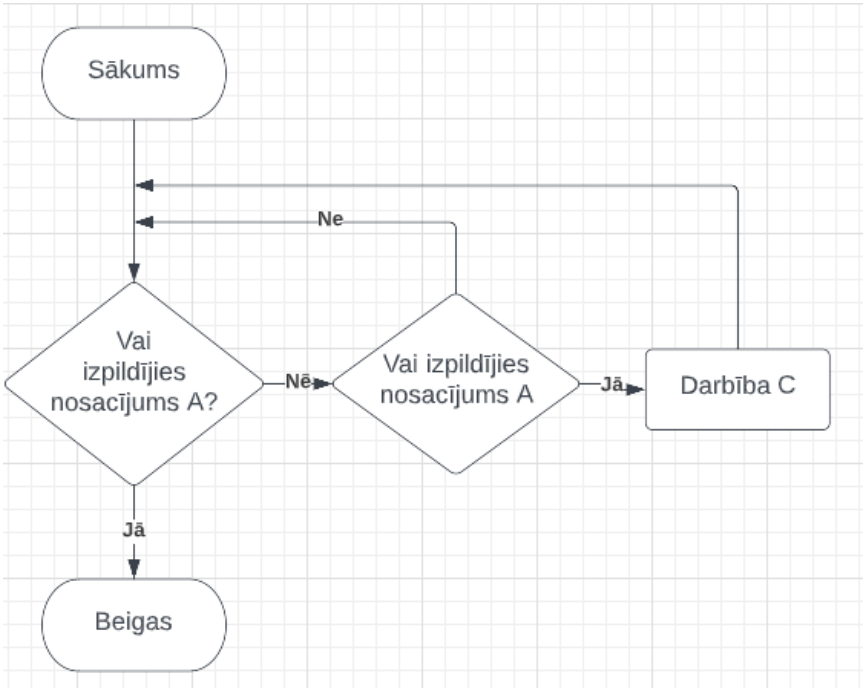
1. uzdevums:



Att. 8.20. Blokshēma ar vairākiem cikliem

- Kura darbība tiks paveikta, ja sensora lasījums būs 50 cm ?
- Kura Darbība tiks paveikta, ja sensora lasījums būs 15 cm?
- Kura Darbība tiks paveikta, ja sensora lasījums būs 100 cm ?
- Kura darbība tiks paveikta, ja sensora lasījums būs 30 cm ?

2. uzdevums:



Att. 8.21. Blokshēma ar sazarojumiem un cikliem

- Izskaidrot šī algoritma darbību!
- Vai pastāv situācija, kurā tiks paveikta darbība C ?
- Kā būtu jāpārveido šis algoritms, lai rastos situācija, kurā Darbība C tiks paveikta vismaz vienu reizi? (Nosacījums A ir kāds patvaļīgs nosacījums)

9. Mainīgie

Skaitīšana. Mūsdienās var diez gan daudz ko skaitīt. Piemēram, garām braucošā vilciena vagonu skaitu vai garām ejošo cilvēku skaitu. Bet šie vairāk ir piemēri, kuros nav dziļākas jēgas vai pielietojuma. Tas ir vairāk skaitīt, skaitīšanas pēc. Tāpēc rodas jautājums, kuru var apspriest grupās:

Kur ikdienā tiek lietota skaitīšana, kur ar to var saskarties?

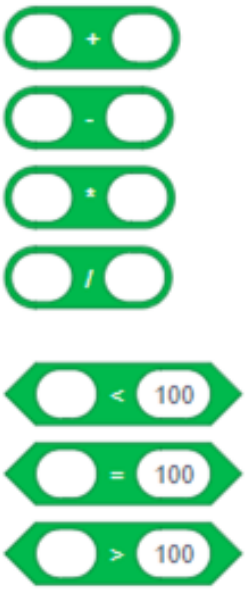
Kā piemērus var minēt sociālo tīklu skatītāju skaitu kādiem video klipiem vai nospiesto "Man patīk" pogu skaitu. Vai arī brīvo vietu skaitu kādā no auto stāvlaukumiem.

Bet tad varētu rasties jautājums, kā robots var skaitīt?

Varbūt, lai labāk saprastu, vajag padomāt, kā cilvēki prot skaitīt. Var skaitīt uz pirkstiem, taču lielākus skaitļus tā skaitīt sarežģīti . Var ar vārdiem, bet kā tieši tas notiek? Ir skaitlis un tam vienkārši pieskaita vienu. Tātad ir jāzina pašreizējais skaitlis un jāprot tam pieskaitīt vieninieku. Robots māk pieskaitīt 1 pie esoša skaitļa. Bet vai robots prot atcerēties pašreizējo skaitli?

Jautājums: Kā robots varētu varēt atcerēties skaitļus? Kā tas varētu sasaistīties ar mainīgajiem?

Tātad mainīgais ir objekts, kurā var uzglabāt vērtības. To var uzskatīt par grozu. Tajā var ievietot, piemēram, kādu skaitu ābolu. Tas būtu, ja groza vērtība ir pieci, tad grozā ir pieci āboli. Tajā uzglabātās vērtības var lietot aprēķinos, ievietot citos blokos, kuros var ievietot apaļas formas blokus. Piemēram, mainīgajā uzglabāt robota ātrumu un to lietot. Šāds piemērs nebūtu īsti praktisks, jo tādā gadījumā nemaz nevajadzētu mainīgo. Šo piemēru būtu jāpapildina, ka ātrums tiek mainīts, piemēram, atkarībā no sensoru lasījumiem. Mainīgajos var vienkārši uzglabāt vērtības. Piemēram, līdz šim tuvāko distanci, kuru robots ir nolasījis no attāluma sensora.



Att. 9.1. Programmas bloki dažādām darbībām

Tālāk tiks aplūkoti bloki, kas atrodas sadaļā *"OPERATORS"*. Kā jau iepriekš tika minēts, šajā nodaļā blokiem atbilst dažādas loģiskās darbības vai matemātiskās operācijas vai funkcijas.

Aplūkojot blokus, kuri atrodas šajā sadaļā var redzēt, ka tiem ir gan apaļas, gan kantainas formas. Tās aplūkojot, kļūst skaidrāka katra bloka forma. Aplūkojiet šos blokus (skat.att.9.1.)

Var redzēt, ka augšējiem blokiem ir apaļas formas un tie apzīmē dažādas matemātiskās operācijas. Apakšējie bloki ir ar kantainu formu un veic to salīdzināšanu.

JAUTĀJUMS: Kas kopīgs visiem 4 apaļajiem un kas kopīgs visiem 3 kantainajiem blokiem?

IETEIKUMS: Padomāt par vērtībām, kas būs šo bloku izpildes rezultātā.

ATBILDE: Vajadzētu būt saprotamam tam, ka izpildot kādas matemātiskās darbības, rezultātā tiks iegūts kāda skaitliska vērtība. Bet ko pavērsīs atpakaļ pārējie trīs bloki? Piemēram otrais, salīdzinās, vai divas vērtības ir vienādas. Tātad intuitīvi, vērtības var būt vienādas vai arī var nebūt vienādas.

No tā izriet, ka apaļie bloki var pavērst atpakaļ skaitliskās vērtības, bet kantainie bloki var pavērst atpakaļ loģisko vērtību formā patiess vai aplams. Programmēšanā bieži ērtāk šīs loģiskās vērtības ir aizstāt ar 1 un 0. Tas ir, *patiess* vietā domāt par vērtību 1 un *aplams* vietā domāt par vērtību 0. Piemēram, ja salīdzina, vai skaitlis 2 ir vienāds ar 4, tad šis apgalvojums ir aplams un tiek pavērst atpakaļ kā vērtība 0. Taču, ja salīdzina, vai skaitlis 2 ir mazāks nekā 4, tad šis apgalvojums ir patiess un tiek pavērsta atpakaļ vērtība 1.

Līdzīgi varēja novērot sadaļā pie cikliem, kur ciklu blokos bija kantains caurums, kur varēja ievietot cikla beigšanas nosacījumu. Līdzīgi ir ar sazarojumiem. Jo cikls beidzas, kad tiek izpildīts beigu nosacījums, tāpēc šim nosacījumam ir jāpieņem vai nu patiesa vai aplama vērtība.

Turpinot aplūkot šīs sadaļas blokus (skat.att.9.2.)

Att. 9.2. Programmas bloks gadījuma skaitļu ģenerēšanai

Šis bloks pavērsīs atpakaļ skaitlisko vērtību. Taču šī vērtība ir gadījuma skaitlis, kas tiek paņemts no kāda intervāla. Piemēram, šajā piemērā tiktu izvēlēts kāds skaitlis robežās starp 1 un 10. Ieskaitot galapunktus. Pavērstais atpakaļ skaitlis būs vesels skaitlis. Taču norādot kā vienu no gala punktiem decimālskaitli, arī vērtība, kas tiks pavērsta atpakaļ būs decimālskaitlis. Tātad, atkarībā no robežas veida, tiks pavērsta atpakaļ kāda vērtība starp šīm abām vērtībām.

Tālāk tiks aplūkotas vairākas loģiskās operācijas.

Sākumā tiks aplūkot loģiskais bloks “un” (skat.att.9.3.)

Att. 9.3. Programmas bloks "Un"

Lai labāk saprastu, kā šis bloks strādā, tiek parādīta tabulā:

Apgalvojums A	Apgalvojums B	A un B
aplams	aplams	aplams
patiess	aplams	aplams
aplams	patiess	aplams
patiess	patiess	patiess

Šajā blokā var ievietot divus blokus, kuri būs loģiskās operācijas. Loģiskās operācijas var būt vai nu patiesas vai arī aplamas. Kā tas ir redzams tabulā, ja šajā blokā ievieto divas vērtības, kas ir aplamas, tad arī šī bloka loģiskā vērtība būs aplama. Gadījumā, ja viens no šiem blokiem ir tikai aplams, bet otrs patiess, tik un tā, galējā vērtība šim blokam būs aplama. Abiem šiem notikumiem vajag būt patiesiem, lai gala vērtība būtu patiesa.

Piemērs: Gadījumā, ja tiek aplūkots šāds programmas fragments (skat.att.9.4.)

Att. 9.4. Programmas bloks "Un"

Kurā parādās tikko minētais bloks "un". Tajā ir ievietoti bloki, kas nosaka, vai izpildās nosacījums attiecībā uz attāluma sensoru. Kreisajā pusē bloks pārbauda, vai attālums sensorā ir mazāks, nekā 100 cm, kamēr otrs pārbauda, vai attālums ir lielāks, nekā 65 cm. Aplūkojot augstāk esošo tabulu, var saskatīt, ka šis bloks "un" būs patiess tikai tad, ja izpildīti abi nosacījumi attāluma sensoram. Tas ir, robots attāluma sensorā redz distanci, kas ir starp 65 un 100 cm.

Tālāk tiks aplūkots loģiskais bloks "vai" (skat.att.9.5.)

Att. 9.5. Programmas bloks "Vai "

Lai labāk saprastu, kā tas darbojas, tiek piedāvāta tabula, kas apraksta tā darbību:

Apgalvojums A	Apgalvojums B	A vai B
aplams	aplams	aplams
patiess	aplams	patiess
aplams	patiess	patiess
patiess	patiess	patiess

Līdzīgi kā iepriekš, ir pievienota tabula, kas raksturo, kādas vērtības bloks "vai" pavērsīs atpakaļ atkarībā no vērtībām, kādas tam tiek padotas. Atšķirībā starp "un", šis bloks var pieņemt patiesu vērtību arī tad, ja vismaz viens no tajā ievietotajiem blokiem ir patiess, nevis gadījumā, kad abi divi ir patiesi.

Piemērs: Var aplūkot šo programmas fragmentu (skat.att.9.6.)

Att. 9.6. Programmas bloks "Vai "

Šajā piemērā "un" bloks ir nomainīts ar "vai" bloku. Iepriekš, loģiskais bloks pieņēma patiesu vērtību tikai tad, ja abi no nosacījumiem attiecībā uz attālumu bija spēkā. Šajā gadījumā, bloks pieņems patiesu vērtību, ja kaut viens no nosacījumiem ir spēkā. Tas ir, ja attālums būs mazāks, nekā 100 cm vai lielāks, nekā 65, tad bloks 'vai" pieņems patiesu vērtību.

Trešo bloku var saukt par "negāciju" (skat.att.9.7.)

Att. 9.7. Programmas bloks "negācija"

Tā apgalvojumam, kas tajā atrodas piešķir pretējo vērtību:

Apgalvojums A	Negācija no A
aplams	patiess
patiess	aplams

Tas ir, ja tajā ievietos bloks, kas pieņem patiesu vērtību, tad negācija no tā būs aplama un otrādi.

Piemēram: aplūkojot fragmentu (skat.att.9.8.):

Att. 9.8. Programmas bloks "Negācija "

Gadījumā, ja sensorā būs redzama sarkanā krāsa, tad negācija no tā būs aplama un loģiskā vērtība, ko šie bloki pavērš atpakaļ, ir aplams. Un otrādi, ja sensorā nav redzama sarkanā krāsa, tad negācija no aplams būs patiess, un loģiskā vērtība šim blokam būs patiess.

Visbeidzot atliek bloks (skat.att.9.9.)

Att. 9.9. Programmas bloks "Vai skaitlis pieder dotajam intervālam"

Kas pārbauda, vai viena no vērtībām atrodas starp dotajām divām vērtībām. Piemēram, šeit tiek pārbaudīts, vai 0 atrodas starp 10 un -10. Tā kā tas ir patiess apgalvojums, tad šī bloka loģiskā vērtība ir aplama. Gadījumā, ja 0 vietā būtu 20, tad bloka vērtība būtu aplama, jo 20 neatrodas starp -10 un 10.

Piebilde: tiek aplūkots vai ievadītā vērtība atrodas starp dotajām vērtībām, ieskaitot gala punktus. Tas ir, ja pārbaudītu, vai 10 atrodas starp -10 un 10, tad šis bloks pavērstu atpakaļ vērtību patiess.

Tālāk var aplūkot blokus:

Dalīšana ar atlikumu (skat.att.9.10.)

Att. 9.10. Programmas bloks dalīšanai ar atlikumu

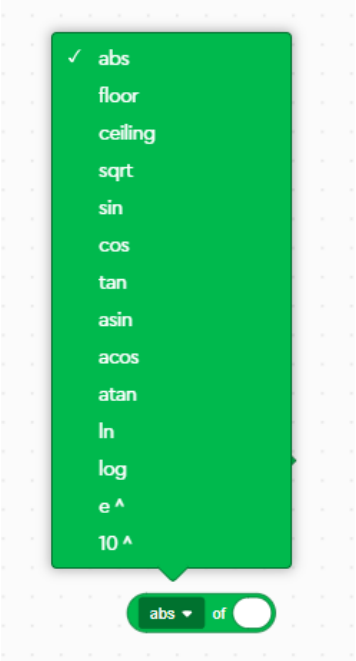
Šis bloks pavērš atpakaļ atlikumu, kurš rodas dalot pirmo skaitli ar otro, kur pirmais un otrais skaitlis ir vērtības, kuras var ierakstīt šajā blokā. Tātad, kreisajā pusē ievadot vērtību 10 un labajā 3, tiktu pavērsta atpakaļ vērtība 1, jo tāds ir atlikums dalot 10 ar 3.

Skaitļu noapaļošana (skat.att.9.11.)

Att.9.11. Programmēšanas bloks skaitļu noapaļošanai

Šis bloks vienkārši noapaļo tajā ievietoto skaitli, Tas ir, ja tiek ievadīts 3.4, tad noapaļo uz 3. Bet ievadot 3.5, tiek noapaļots uz 4.

Paliek dažādas matemātiskās funkcijas (skat.att.9.12.)



Att. 9.12. Programmas bloks dažādām funkcijām

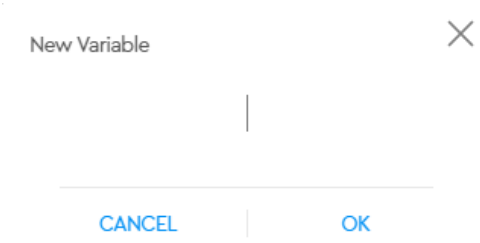
Attiecīgās funkcijas apkopotas tabulā:

Funkcija	Nosaukums	Apraksts
abs	Absolūtā vērtība	pavērš atpakaļ dotās vērtības absolūto vērtību
floor	Noapaļošana uz leju	pavērš atpakaļ doto vērtību noapaļotu uz leju līdz tuvākajam veselam skaitlim
ceiling	Noapaļošana uz augšu	pavērš atpakaļ doto vērtību noapaļotu uz augšu līdz tuvākajam veselajam skaitlim
sqrt	Kvadrātsakne	pavērš atpakaļ kvadrātsakni no dotās vērtības
sin	Sinusa funkcija	pavērš atpakaļ sinusa vērtību no dotās vērtības
cos	Cosinusa funkcija	pavērš atpakaļ cosinusa vērtību no dotās vērtības
tan	Tangensa funkcija	pavērš atpakaļ tangensa vērtību no dotās vērtības
asin	Arcsinusa funkcija	pavērš atpakaļ arcsinusa vērtību no dotās vērtības

acos	Arccosinusa funkcija	pavērš atpakaļ arccosinusa vērtību no dotās vērtības
atan	Arctangensa funkcija	pavērš atpakaļ arctangensa vērtību no dotās vērtības
ln	Naturālais logaritms	pavērš atpakaļ logaritma ar bāzi *e* vērtību no dotās vērtības
log	Decimāllogaritms	pavērš atpakaļ logaritma ar bāzi 10 vērtību no dotās vērtības
e^	Pakāpes funkcija ar bāzei e	pavērš atpakaļ *e* kāpina ievadītajā pakāpē un pavērš atpakaļ iegūto vērtību
10^	Pakāpes funkcija ar bāzi 10	Skaitli 10 kāpina ievadītajā pakāpē un pavērš atpakaļ iegūto vērtību

Tālāk var sākt aplūkot mainīgo definēšanu. Mainīgo var izveidot sadāļā *"**Variables**"* (skat.att.9.13.).

Att. 9.13. Mainīgo izveide



Att. 9.14. Mainīgā nosaukuma piešķiršana

Att. 9.15. Programmas bloki darbībām ar mainīgajiem

Klikšķinot uz *"MAKE A VARIABLE"*. Atveras šāds logs (skat.att.9.14.).

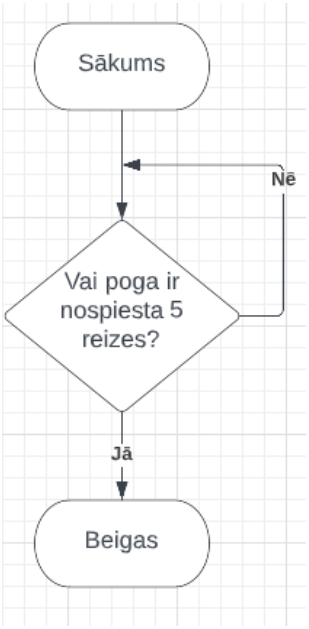
Tajā ir jāievada vārds, kurā vēlās nosaukt šo mainīgo. To ievadot, klikšķina uz "*OK*". Sadaļā *"VARIABLES"* vajadzētu parādīties jauniem blokiem (skat.att.9.15.).

Pirmais bloks ir apaļas formas, tātad tajā tiek glabāta kāda vērtība. Šajā gadījumā, tā ir pati mainīgā vērtība. Definējot mainīgo, tam pēc noklusējuma ir iestatīta vērtība - 0. Taču ar **otro bloku** var iestatīt citu mainīgā vērtību. Piemēram, ja sākumā ir 0, tad var to mainīt uz 10. Un **trešais bloks** ir veikt izmaiņas mainīgā vērtībai. Tas ir, var vai nu pieskaitīt, vai nu atņemt, pieskaitot negatīvo skaitli.

Tālāk tiks aplūkots piemērs, kurā tiek izmantots mainīgais. Tiks izveidota programma, kura savu darbu beigs, kad poga tiks nospiesta 5 reizes. Tāpēc ir nepieciešams pie robota galvenā bloka pievienot pogas sensoru. Tālāk tiks aplūkots pats algoritms:

Uzdevums: uzzīmēt blokshēmu šim piemēram.

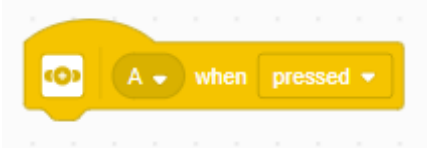
Atbilde: Tātad, ir zināms, ka robots gaida, kamēr tiks nospiesta poga 5 reizes un tikai tad savu darbību beigs. Šis skaidrojums atgādina ciklu. Tas atkārtojās 5 reizes un beidzas (skat.att.9.16.)



Att. 9.16. Blokshēma uzdevumam

Tātad cikls atkārtojas, kamēr poga nav nospiesta 5 reizes. Taču kā to realizēt programmā? Šeit var būt vairāki varianti. Viens varētu lietot ciklus, kas atkārtojās 5 reizes un tajā tiktu ievietots bloks, kas gaida, kamēr tiek nospiesta poga. Otrs varētu būt mainīgais. Tajā ievadot sākotnējo vērtību 5 un katru reizi, kad tiek nospiesta poga, no šī mainīgā atņem skaitli 1. Un tā, līdz tiek sasniegta 0.

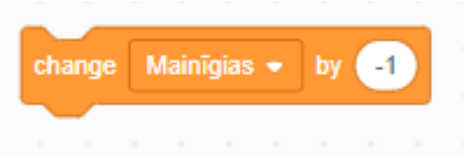
Sākumā ir nepieciešams izveidot jaunu programmu programmēšanas vidē. Tajā izveidot jaunu mainīgo. Dažkārt lielu programmu izstrādi ir ērti sadalīt pa daļām, tāpēc sākumā var izveidot daļu, kas izmaina mainīgā vērtību katru reizi, kad tiek nospiesta poga.



Att. 9.17. Programmas bloks

Atbilde: Tātad sākumā ir jāmeklē bloks, kas veic darbības, kad tiek nospiesta poga. Ērti šeit ir lietot bloku (skat.att.9.17.)

Zem šī bloka ir nepieciešams izveidot darbību, kas atņem no mainīgā skaitli 1. To var ērti izdarīt ar bloku (skat.att.9.18.)

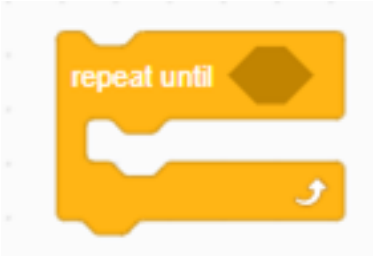


Att. 9.18. Programmas bloks

Šeit jau ir nomainīta vērtība no 1 uz -1. Atstājot 1, katru reizi, kad nospiestu pogu, šī vērtību tiktu palielināta, nevis samazināta. Tā var darīt, tad beigu nosacījums ciklam mainītos no 0 uz 5. Tas ir, sākotnējā mainīgā vērtība būtu 0 un beigu nosacījums būtu, kad šis mainīgais pieņemtu vērtību 5. Kas ir pretēji tam, kā šeit tiek realizēts algoritms. Abos gadījumos programma pildītu vienu un to pašu funkciju.

Tālāk var izveidot pašu algoritmu. Lai labāk varētu saprast, kad programma strādā un kad nestrādā, var lietot attēlus. Tas ir, iededzināt LED gaismu uz robota galvenā bloka. Tātad sākumā ir nepieciešams izvēlēties ciklu, kuram var iestatīt kā beigu nosacījumu, mainīgā vienādību ar 0.

Tiks lietots šis cikla bloks (skat.att.9.19.). Tajā var ievietot salīdzināšanas operāciju (skat.att.9.20.)



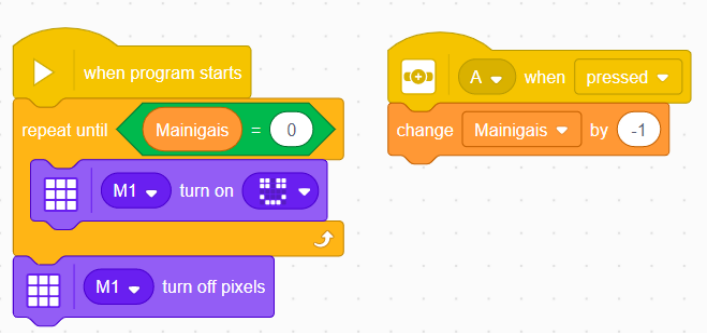
Att. 9.19. Programmas bloks

Att. 9.20. Programmas bloks

Vienā pusē ir nepieciešams ievietot Mainīgā vērtību un otrā beigu vērtību. Šajā gadījumā tā ir 0 (skat.att.9.21)

Att. 9.21. Programmas bloks

Tātad, šis ir cikls, kas darbosies līdz brīdim, kad mainīgā vērtība būs vienāda ar 0. Ciklā ievietojot LED ieslēgšanu, tiek iegūta gala programma (skat.att.9.22.)



Att. 9.22. Programma

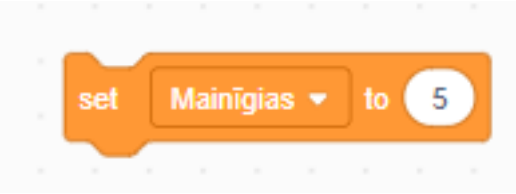
Tālāk nepieciešams palaist programmu un aplūkot, vai robots strādā pareizi.

UZDEVUMS: *Kāpēc programma nestrādā?*

Atbilde: Tika pieņemts, ka sākotnējā mainīgā vērtība ir vienāda ar 5. Tāpēc no tās piecas reizes atņemot vieninieku tiktu iegūta 0. Taču šajā programma nevienā vietā šī mainīgā vērtība netika iestatīta uz 5. Tāpēc pēc noklusējuma tā pieņēma vērtību 0. Un tāpēc programma uzreiz pēc sākšanas arī beidza savu darbību.

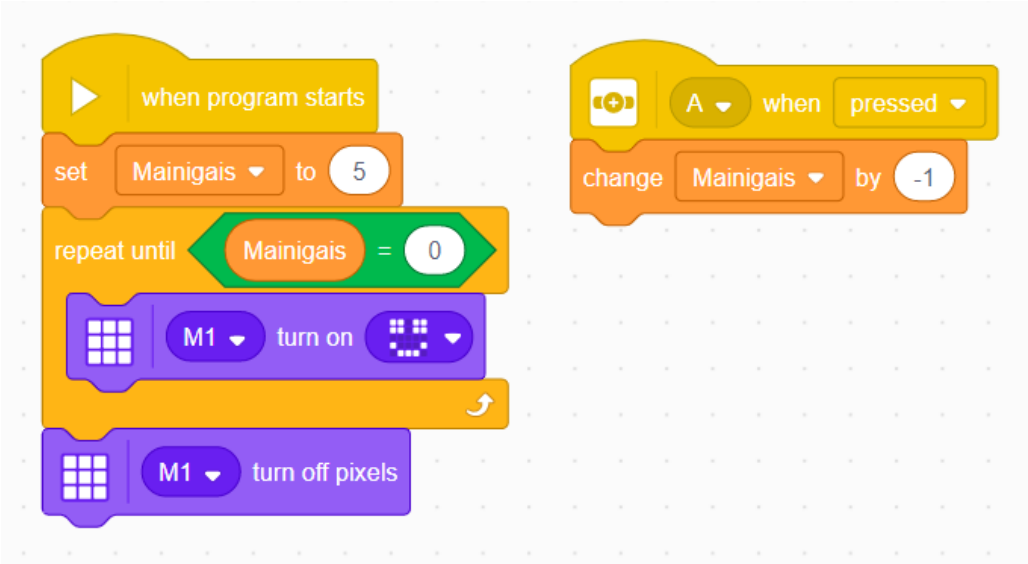
UZDEVUMS: *izlabot kļūdu.*

ATBILDE: Izlabot šo kļūdu var pavisam vienkārši, ar šo bloku (skat.att.9.23.)



Att. 9.23. Programmas bloks

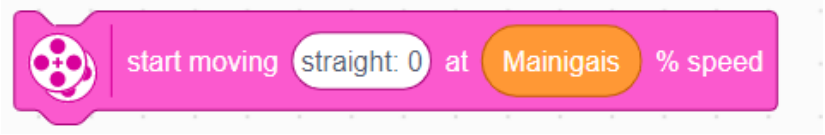
Tas iestata mainīgā vērtību uz 5. Lai programmai sākoties šī vērtība tiktu iestatīta uz 5, šis bloks ir jāievieto pirms cikla. Tas izskatītos šādi (skat.att.9.24.).



Att. 9.24. Programma

Tagad palaižot robotu, programmai vajadzētu darboties korekti.

Šis piemērs vairāk demonstrē, kā definēt mainīgo un to ievietot programmā. Tāpēc būtu jāņem citi, daudz praktiskāki lietojumi mainīgajiem. Tie ļauj uzglabāt skaitliskas vērtības, tai skaitā sensoru lasījumus. Tāpēc, ja būtu jāveic vairākas darbības ar vienu sensora lasījumu, būtu ērti to uzglabāt mainīgajā un strādāt ar šo mainīgo. Varbūt krietni konkrētāks piemērs varētu būt robota ātruma maiņa atkarībā no kādiem sensoru lasījumiem. Piemēram, robots maina ātrumu atkarībā no attāluma sensorā redzamās distances. Šeit, protams, var darīt to vienkārši blokā (skat.att.9.25.).



Att. 9.25. Programmas bloks

Mainīgā vietā ievietot kādu darbību, kas atkarīga no sensora, taču tas var padarīt šo bloku nepārskatāmu, ja šī darbība sevī iekļauj vairākas darbības. Tāpēc no programmas organizācijas viedokļa varētu būt labāk aprēķinus veikt citā kolonnā un ar mainīgā palīdzību iestatīt motora ātrumā.

10. Uzdevumi

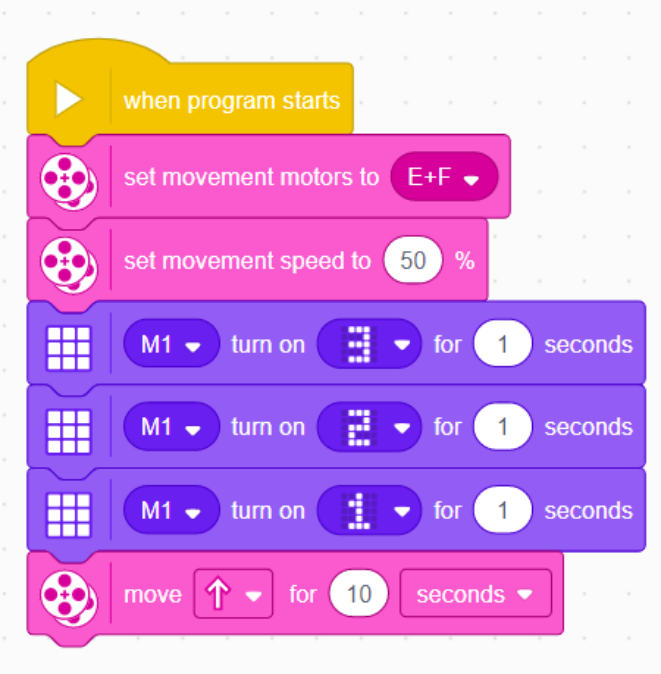
10.1. Lineārie algoritmi

10.1.1.Kopīgi veicamie uzdevumi

Šajā uzdevumā tiks aplūkots pavisam vienkāršs lineārs algoritms un patstāvīgi tiks veikta konstrukcijas salīdzināšana, pamainot vidi, uz kuras tiek palaists robots vai pašu robota konstrukciju.

Vispirms jāizveido robots. Robota būvēšanas instrukciju var atrast *BUILD* sadaļā ar nosaukumu *"HOPPER"*.

Kad robots ir izveidots, atliek izveidot programmu. Programma šajā uzdevumā ir pavisam vienkārša (skat.att.10.1.)



Kur sākumā tiek iestatīti motori un to ātrumi. Tiek veikta laika atskaite un tad seko 10 sekunžu ilga kustība. Palaižot programmu, vajadzētu novērot, ka robots veic atskaiti un sāk lekt uz priekšu 10 sekundes.

Tagad, kad robots ir izveidots, atliek izpētīt konstrukciju. Robotam pie motoriem ir pievienotas *"Kājas"*. Pirmais solis būtu pamēģināt palaist šo robotu bez izmaiņām uz virsmām, ar atšķirību segumu. Šeit ir jānovēro, vai mainot segumu, mainās attālums, kuru veic robots.

Otrs solis ir pamainīt *kāju* konstrukciju. Šeit svarīgi neizmantot riteņus. Kā piemērs - elementu vietā, kas atrodas *kājas* galā, lietot kādu no zobratiem. Kā viens no veidiem, kā pateikt, vai kāda *kājas* konstrukcija ir labāka, nekā citas, ir veidot savstarpējas sacensības.

10.1.2 Uzdevumi patstāvīgajam darbam

1. Uzdevums.

   Izveidot robotu ar nosaukumu *"Driving Base 1"*. No *Build* sadaļas. Ar šo robotu nepieciešams izpētīt bloku (skat.att.10.2.)

   Kā mainās robota pārvietošanās, cik tālu tas aizbrauc vai daudz pagriežas ja:

   *) Tiek mainīts pagrieziens (skaitliskā vērtība)

   *) Tiek mainīts tas, cik "daudz" robots pārvietojās, mainot vērtību otrajā logā.

   *) Tiek mainīts tas, pēc kā mēra kustību (trešais logs blokā)

2. uzdevums.

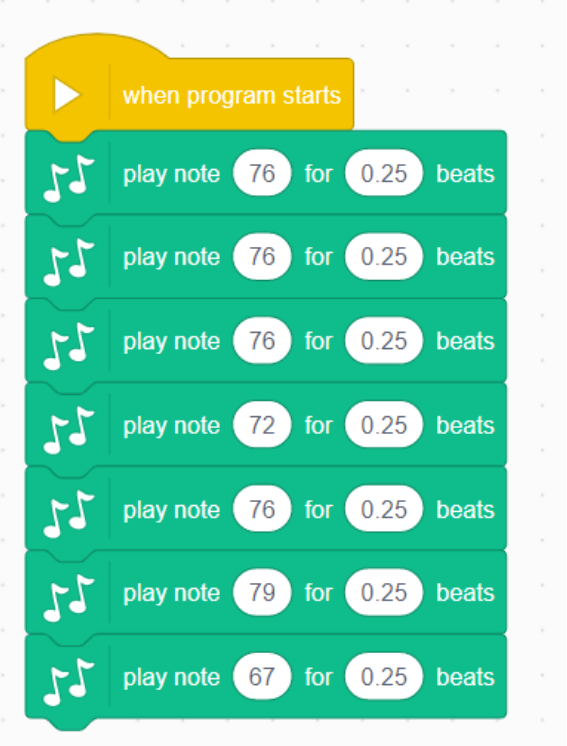
   Izveidot robotu ar nosaukumu *"DRIVING BASE 1"* no *BUILD* sadaļas. Ar šo robotu izveidot programmu, kas liktu robotam pabraukt uz priekšu 30 cm, apgriezties, atbraukt atpakaļ iepriekš nobrauktos 30 uz sākotnējo pozīciju un atgriezties atkal otrādi.

3. uzdevums.

   Uzdevums ir noklausīties kādu skaņdarbu internetā un lietojot visus iespējamos skaņu efektus robotā, mēģināt atdarināt šo skaņdarbu ar robota darbībām.

4. uzdevums.

   Šajā uzdevumā būs nepieciešams izveidot mūziku. Tātad, ir dots šī piemērs:



5. Kas atbilst pirmajām notīm no "**Overworld Theme**", kas ir mūzika no spēles *Super Mario*. Uzdevums ir atrast internetā pilnu skaņdarbu un izveidot programmu, kas to izpilda.
6. uzdevums.

   Šajā uzdevumā ir nepieciešams izveidot algoritmu un arī pēc tam izveidot programmu, kas realizē šo algoritmu.

Dažkārt ir nepieciešams izveidot robotu, kas pilda kādu funkciju reālajā dzīvē. Taču reti kad ir izdevīgi uzreiz veidot reālās dzīves modeli. Tāpēc ērti ir izveidot prototipu vai mazāku modeli, kas pilda šo funkciju un aplūkot iespējamās nepilnības un citas problēmas, kas varētu rasties veidojot pašu robotu. Tātad šajā uzdevumā ir jāizveido algoritms, kas atbilst maršrutam, kas ir dots zemāk. Doto algoritmu ir nepieciešams pārbaudīt. Lai to pārbaudītu, ir jāizveido programma un robots, kas to spētu realizēt.

Ir izveidots maršruts, kur no punkta A jānokļūst līdz punkta B. Maršruts ir redzams attēlā augstāk. Maršruts ir sadalīts vairākos posmos. Vienkāršības labad, var pieņemt, ka posms 5 un 6 patiesībā viens liels posms.

Sākumā ir nepieciešams izstrādāt algoritmu, kas varētu atbilst robota izbraukšanai pēc šāda maršruta. Pirmais solis būtu izmēru noteikšana.

Augstāk ir dots attēls ar maršrutu, no tā var nolasīt, cik liels ir katrs no pagriezieniem. Ar lielumu šeit ir domāts, cik liels ir leņķis starp abiem ceļa posmiem. Leņķus jānosaka no attēla, to noteikti var noapaļot uz pieņemamākajām vērtībām. Tālāk atliek aplūkot taisnos posmus. Vienkāršības labad, tiek veikti šādi pieņēmumi:

Posmu 8 un 9 garums ir A vienības.

Posmu 2 un 3 garums ir B vienības

Posmu 1 un 5 garums ir C vienības

Posmu 4, 6 un 7 garums ir Z vienības.

Kur attiecīgie posmi tiek pieņemti, ka ir ar vienādu garumu. Veidojot algoritmu, var atsaukties uz šiem attālumiem. Taču veidojot programmu, ir nepieciešams ievietot kādas konkrētas vērtības. Izvēlēties atbilstošus izmērus, lai labāk modelētu reālās situāciju ir svarīgi, taču vēl svarīgāk ir izvēlēties izmērus, lai būtu pieejama pietiekami liela platība, lai šo programmu varētu izmēģināt. Tāpēc izmērus ir jāizvēlas, lai būtu pietiekami daudz vietas, lai varētu notestēt algoritmu un lai tie būtu kaut cik proporcionāli reālās dzīves situācijai. Nav arī ieteicams ņemt ļoti mazus izmērus, jo tad nevarēs noteikt, vai robots algoritmu veic pareizi.

Lai veiktu šo programmas testēšanu, nav teikts, ka būtu jāizvēlas kāda konkrēta robota konstrukcija, var izveidot savu vai arī lietot piemēram, *BUILD* sadaļā "*DRIVING BASE*". Galvenais, lai robots būtu spējīgs braukt.

10.2. Sazarojumi

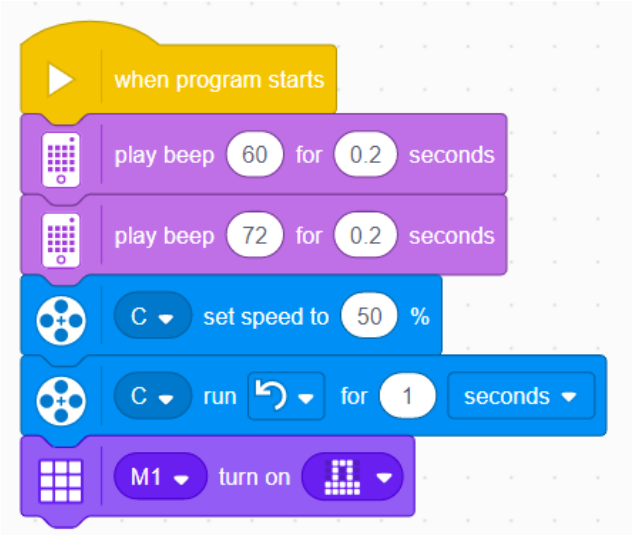
10.2.1. Kopīgi aplūkotie uzdevumi

Kopīgi aplūkotais 1. uzdevums

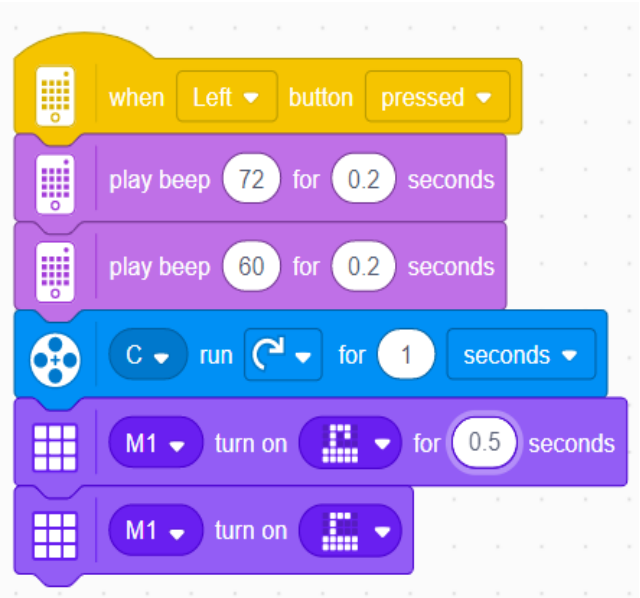
Šajā uzdevumā tiks aplūkoti sazarojumi drošības sistēmā. Šajā uzdevumā šī sistēma būs seifs, kuru var atvērt, veicot iepriekš iestatīto kombināciju. Šeit arī parādīsies sazarojumi: pārbaudot, vai kombinācija ir ievadīta korekti. Šajā uzdevumā tiks aplūkoti vienkārši sazarojumi, sazarojumi, kuri seko viens pēc otra.

Vispirms ir jāizveido pats *seifs*. Tā būvēšanas instrukciju var atrast *BUILD* sadaļā ar nosaukumu *"SAFE DEPOSIT BOX"*.

Kad robots ir uzbūvēts, to var sākt programmēt. Pirmais solis būtu šī seifa aizslēgšana. Lai to aizslēgtu, motors pie porta C ir jāpakustina, lai pie tā piestiprinātā *LEGO* detaļa ieietu "atslēgas caurumā". Motors, kas ir pievienots pie porta B tiks lietots, kā atslēga, ar kuru ievadīt nepieciešamo kombināciju. Tāpēc, tiek piedāvāta šāda programma:

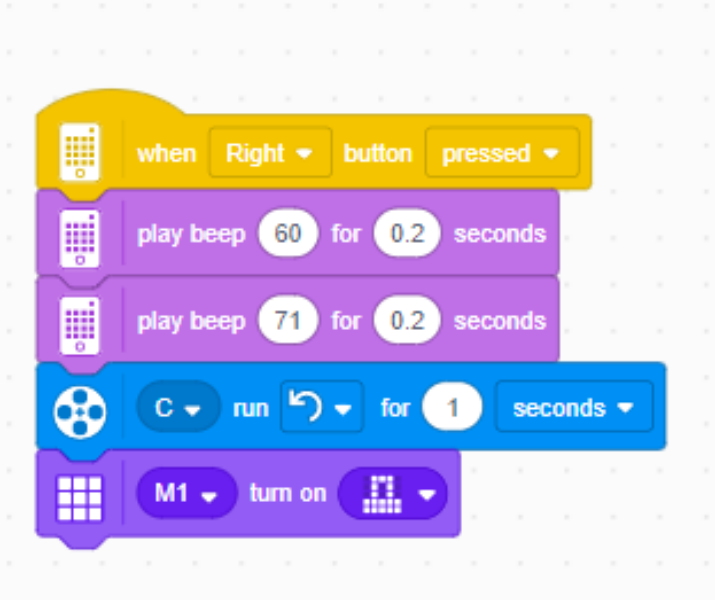


Tajā tiek iestatīs motors C un ar tā kustību vienkārši tiek aizslēgts seifs. Tātad, lai to atslēgtu, tas pats motors būtu jādarbina uz otru pusi ar tikpat lielu distanci. Šajā gadījumā, tā ir 1 sekunde. Var sākt ar pavisam vienkāršu pirmo kombināciju atslēgšanai:



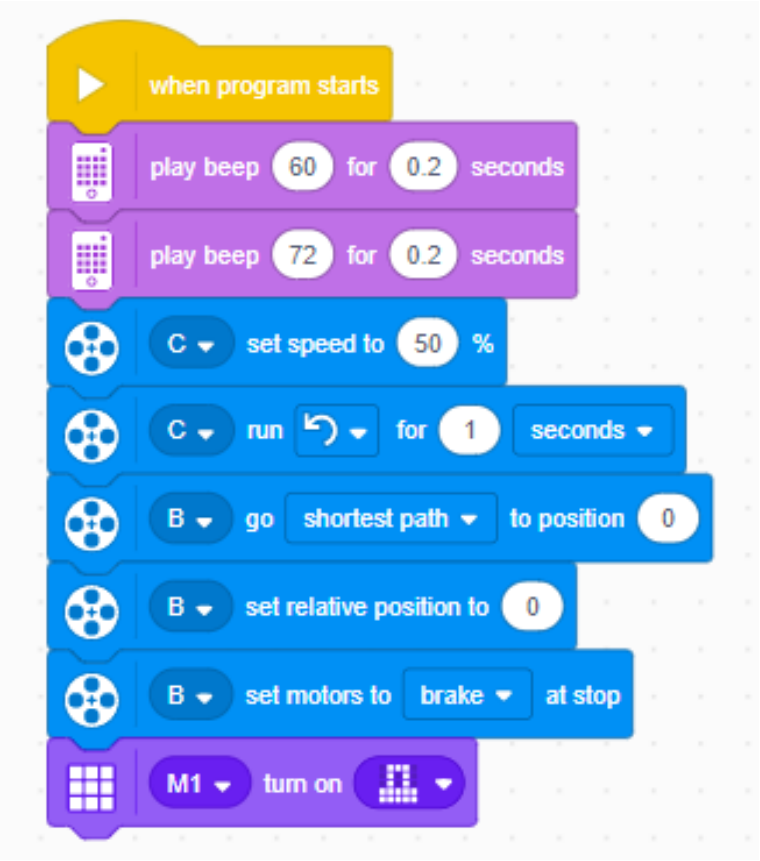
Seifs ir aizslēgts, līdz tiek nospiesta bultiņa uz galvenā bloka. Šeit parādās sazarojums, ja tiek nospiesta norādītā poga, robots atver seifu. Taču, mēģinot uzzīmēt šīs programams blokshēmu, varētu novērot, ka patiesībā zem šī sazarojuma slēpjas cikls. Praksē sazarojumi ar cikliem ļoti bieži parādās kopā, sazarojumi tiek lietoti ciklu beigšanai. Kā sazarojuma nosacījumi tiek lietoti sensoru dati.

Šajā programmā varētu novērot vienu trūkumu: Seifu var atvērt, taču to nevar aizvērt. Lai aizvērtu, nepieciešams programmu palaist no jauna. Ērtāk būtu programmā izveidot funkciju, kas to izdara. Šeit var lietot labo pogu, kas atrodas uz galvenā bloka. Tad kreisā būtu, lai robotu aizvērtu, taču labā, lai atvērtu. Seifu var aizvērt lietojot to pašu bloku, ar kuru atvērt, tikai kustības virziens ir pretējs. Programmas fragments varētu būt:



Līdz šim, programmā parādās vienkārši sazarojumi, poga tiek nospiesta un tiek izpildīta kāda darbība. Pretējā gadījumā robots vienkārši gaida.

Tālāk ir jāturpina seifa drošības sistēma. Pašlaik seifu var atvērt vienkārši nospiežot pogu. Tas nav diezgan praktiski, jo tā ir salīdzinoši vienkārša kombinācija. Šeit parādās otrs motors pie porta B. Kā kombinācijas var lietot leņķi, kādā šis motors ir pagriezts. Tātad, pirmais solis būtu novietot šo motoru sākuma pozīcijā, kurā tas nav sagrozīts (tas ir pagriezts par nulle grādiem). Tas izskatās šādi:

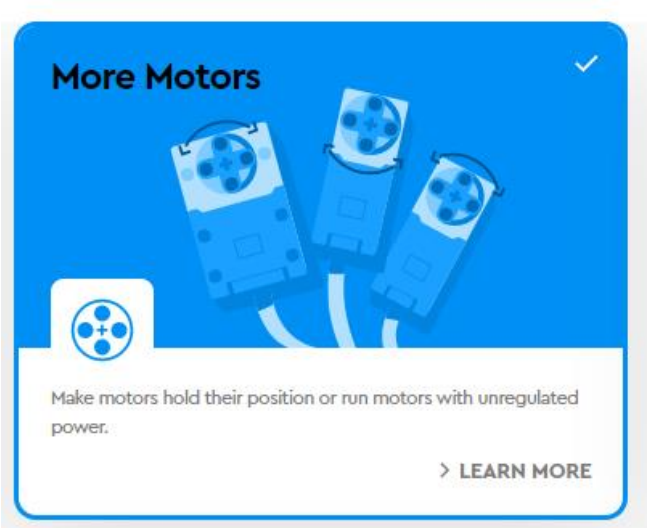


Vienkārši sākuma kolonna pievieno blokus, kas motoru B novieto nulles pozīcijā. Šajā kolonnā, pirmspēdējais bloks pasaka, ka motoram beidzot griezties, tas tiks nobremzēts. Lai vairāk izprastu, uz šī bloka var klikšķināt ar labo peles pogu un nospiest *Help* pogu.

Lai izveidotu sarežģītāku kombināciju var lietot blokus:

Bloku, kas atrodas iekšā salīdzināšanai var atrast apakšā, kreisajā stūrī spiežot uz:

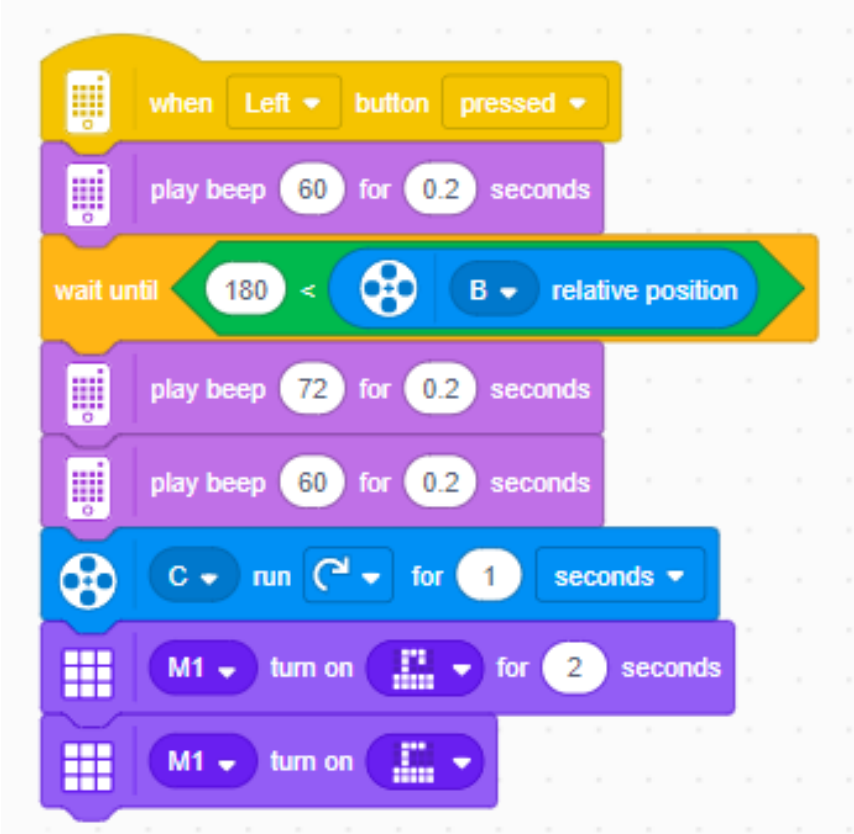
Un tur ieklikšķinot:



Atšķirībā no bloka: 

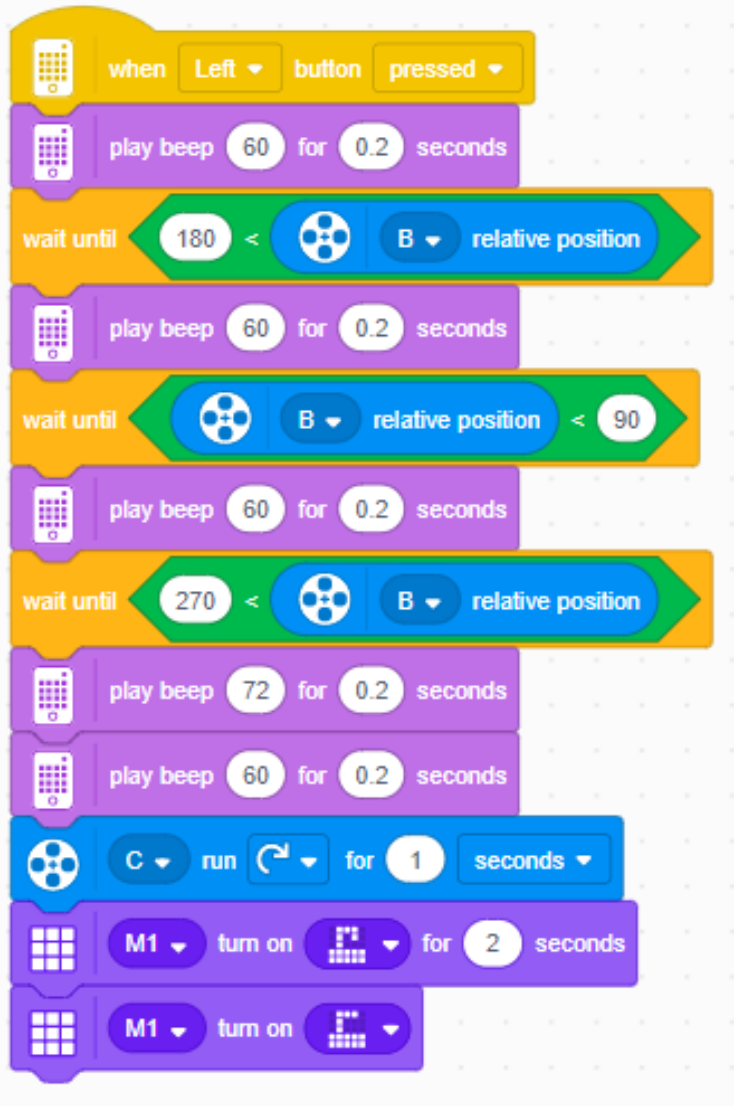
Šeit lietotais bloks nolasa motora relatīvo pozīciju, par kādu leņķi tas ir pagriezts kopš programmas sākuma brīža vai kopš pēdējās reizes, kad tā pozīcija programma tika iestatīta, kā nulle. Motora pozīcija ir cik daudz motors izkustējies no nulles pozīcijas.

Piemērs modificētai kolonnai ar kādu atslēgšanas kombināciju:



Tagad, nospiežot pogu atskanēs signāls un robots gaidīs, līdz motors pie porta B tiks pagriezts leņķī, kas ir lielāks nekā 180 grādi.

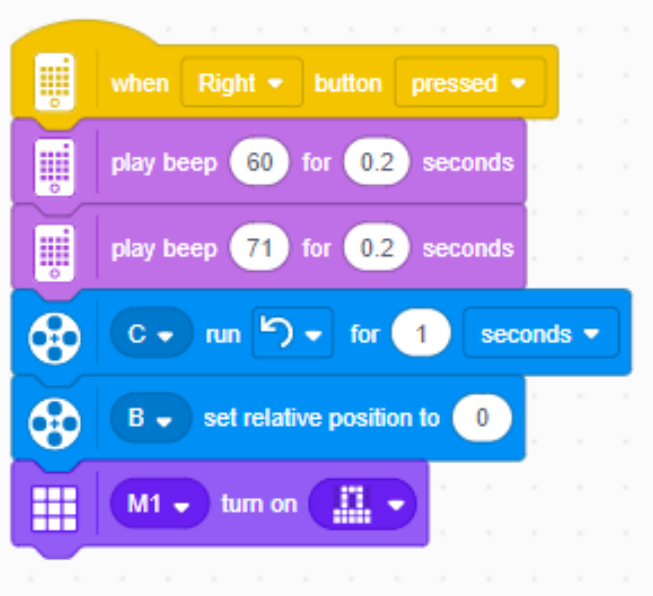
Lai izveidotu vēl sarežģītāku šifru, var pievienot vairākus šifrus:



Var vienkārši pievienot vairākus blokus. Šeit pievienoti skaņas elementi, ja tiek izpildīta kāda kombinācija. Tas var atvieglot robota testēšanas daļu, jo būs saprotams, ja tiek veikta kombinācija. Praktiski būtu jānoņem skaņas efekti katru reizi, kad tiek izpildīta šifra kombinācija.

Šajā programmas kolonnā var redzēt vairākus sazarojumus. Katrs no tiem gaida, kad notiks attiecīgā darbība. Tie ir secīgi, viens pēc otra, kamēr nav izpildīts pirmais, otrais nevar uzsākt darbību.

Lai pabeigtu uzdevumu, atliek veikt pēdējās modifikācijas pie kolonnas, kas robotu atkārtoti aizver, nospiežot labo pogu uz robota galvenā bloka:



Šeit tiek pievienots bloks, kas norāda, ka pašreiz motors B ir relatīvā pozīcijā un nulle grādu. Tādā gadījumā neradīsies problēmas atkārtoti mēģinot atvērt seifu. Piemēram, veicot seifa atvēršanu, pēdējais solis ir pavirzīt motoru tā, lai tā relatīvā pozīcija ir virs 270 grādiem. Atkārtoti mēģinot atvērt seifu, ja šo motoru neizkustina, tas automātiski izpildīs pirmo soli, kas nozīmē pavirzīt motoru virs 180 grādiem. No drošības viedokļa tas nav labs risinājums.

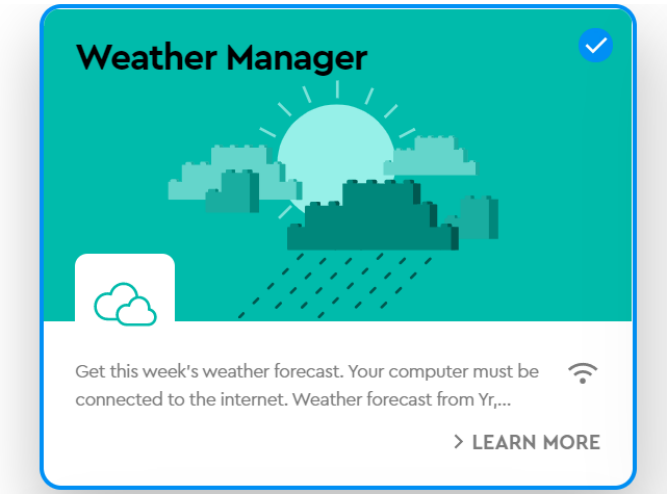
Kopīgi aplūkotais 2. uzdevums

Šajā uzdevumā tiks aplūkots viens no vissastopamākajiem gadījumiem, kad reālajā dzīvē ir nepieciešams saskarties ar sazarojumiem. Tas ir saistībā ar laika apstākļiem. Tiks izveidots robots, kas spēj paredzēt laika apstākļus. Tiks lietoti sazarojumi, lai varētu atšķirt situācijas, kad tiek noteikti atšķirīgi laika apstākļi.

Šajā uzdevumā tiks lietots robots, kura būvēšanas instrukciju var atrast sadaļā *Build* ar nosaukumu *" Weathercaster"*.

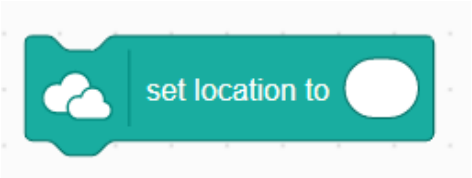
Ieteikums: Pirms sāk programmēt šo robotu, nepieciešams atvērt programmēšanas vidi un tur, kreisajā apakšējā stūrī klikšķināt uz

Tad vajadzētu atvērties logam, kurā nepieciešams atzīmēt ar ķeksīti šajā blokā, ja tas nav izdarīts:

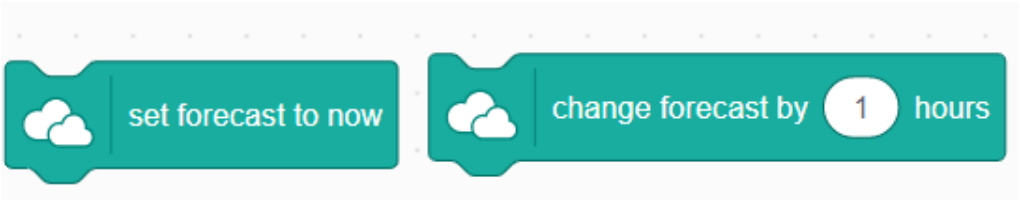


Pretējā gadījumā programmēšanas vidē neparādīsies bloki, kas paredzēti laika apstākļu noteikšanai.

Tagad var sākt laika apstākļu noteikšanu. Sākumā var izpētīt pieejamos blokus:

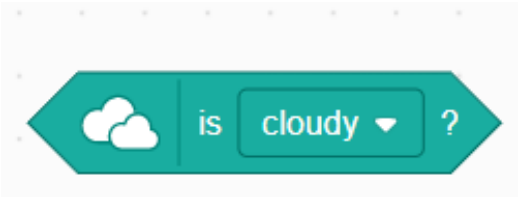


Ar šo bloku var norādīt lokāciju, kurā būs nepieciešams noteikt laika pstākļus. Tukšajā logā ir jāklikšķina un jāieraksta vēlamā lokācija. Piemēram, tur var ierakstīt “Riga” un tālāk tiks aplūkoti laika apstākļi Rīgā.



Ar šiem blokiem var noteikt, kad tiks aplūkoti laika apstākļi. Lietojot kreiso bloku, tālāk aplūkojot laika apstākļus, tie tiks skatīti tieši šajā brīdī. Ar otru bloku var norādīt, ka vajag paredzēt laika apstākļus vēlāk. To var norādīt, nomainot stundas, cik jāskatās uz priekšu.

Ar šo bloku:

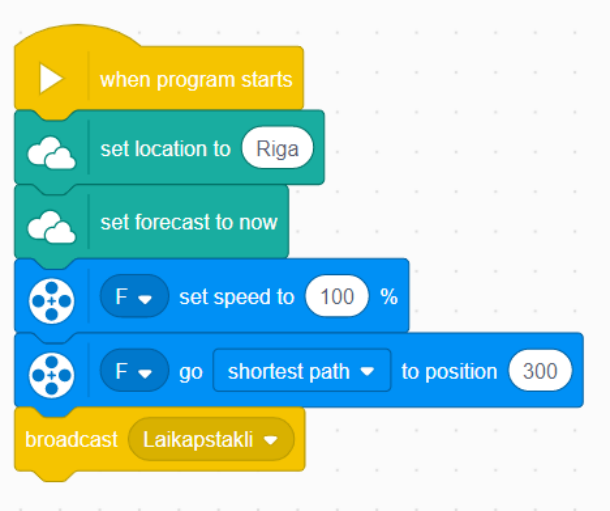


Var iepriekš norādītajā laikā pārliecināties par laika apstākļu sakritību. Tas strādā līdzīgi, kā sensoru lasījumi. Tie var sakrist ar norādīto vai nesakrist.

Tagad var sākt domāt par laika apstākļu noteikšanu. Pirmais solis varētu būt izdomāt, kā robots varētu parādīt nolasītos laika apstākļus. Pareizi būtu sākt ar kaut ko nelielu, piemēram,

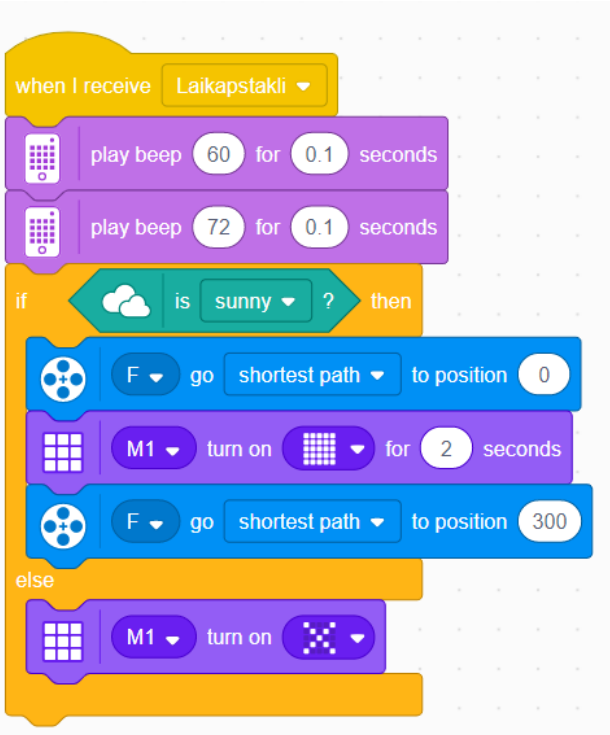
gadījumu, kad ārā spīd saule. Šeit ērti būtu lietot robotam pievienotās brilles. Tātad, saulaina laika gadījumā robots uzvilks saules brilles. Tas nozīmē, ka programmas sākumā ir jābūt darbībai, kas sagatavo robotu un nolaiž brilles lejā. Tad saulainu laika apstākļu gadījumā tās varēs pacelt un redzēt, ka robots darbojas.

Tālāk tiek piedāvāta programma:



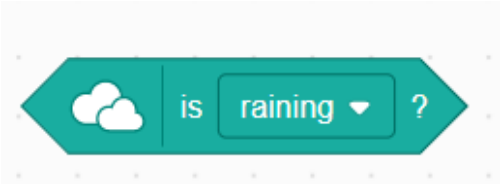
Kas pasaka, ka laika apstākļi tiks aplūkoti Rīgā un tieši tagad. Tālāk tiek norādīts motors, pie kura ir pievienotas brilles un tā sākuma pozīcija. Šajā pozīcijā robots ir nolaidis savas brilles. Tālāk seko ziņas nosūtīšana. Tas tiek lietots, lai programmu varētu sadalīt vairākās kolonnās, kas padarītu to pārskatāmāku. Kā arī, ja vajadzētu vairākas reizes noteikt laika apstākļus, piemēram, pēc vienas, divām un trīs stundām, tad vieglāk ir atkārtoti izsaukt šo funkciju, nekā vienu zem otra likt vairākus sazarojumus, kas izskatā būs pilnīgi identiski.

Tagad atliek tikai aplūkot laika apstākļus:



Šis programmas fragments uzsākt darbību tiklīdz tiek pārraidīta ziņa no iepriekšējā fragmenta. Šajā fragmentā pavisam vienkārši, tiek pārbaudīts, vai ārā ir saulains. Gadījumā, ja pašlaik ir saulains, robots paceļ savas brilles un uz galvenā bloka uzzīmē sauli un nolaiž brilles. Pretējā gadījumā uz galvenā bloka tiek parādīts X. Tas ir, gadījumā, ja ārā nav saulains.

Šo programmu tagad var instalēt robotā un palaist. Gadījumā, ja norādītajā pilsētā nav saulains un nevar notestēt to gadījumu, var mēģināt internetā atrast pilsētu, kurā šobrīd spīd saule. Vai arī var nomainīt, ja, piemēram, ārā līst lietus, tad sazarojumā nomainīt to nosacījumu:

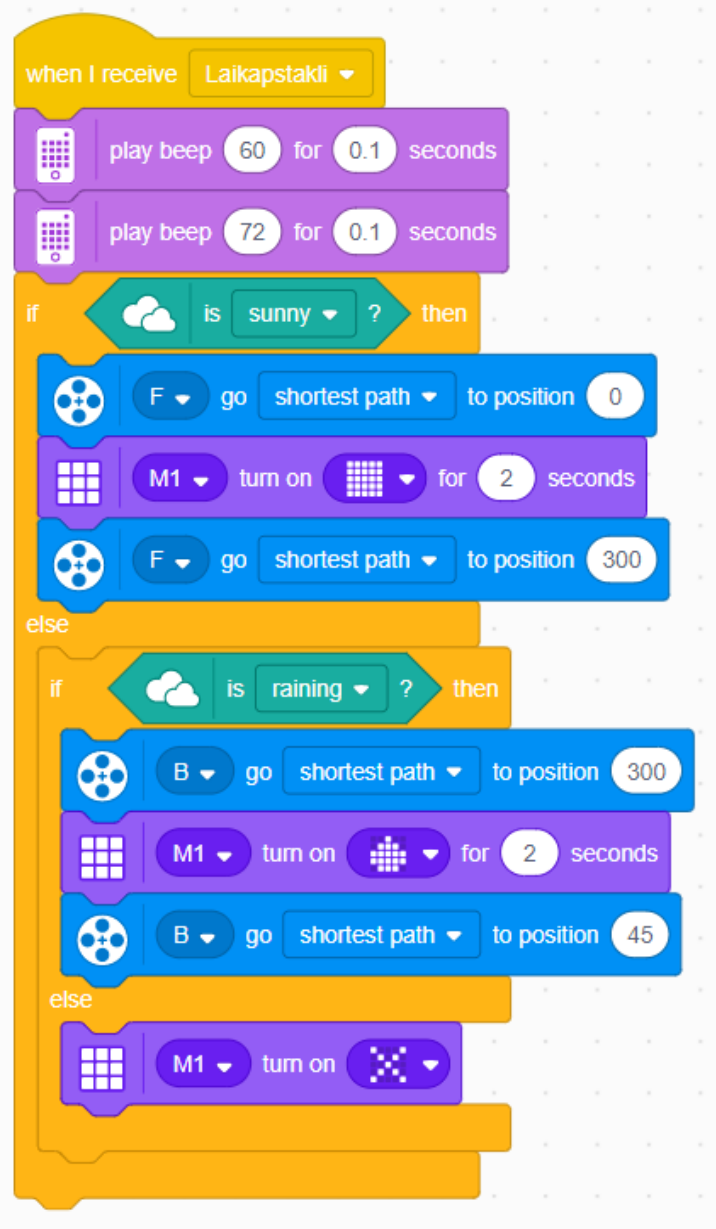


Lai varētu notestēt, vai robots darbojas pareizi.

Ieteikums: *Palaižot robotu ir svarīgi pārliecināties, vai ir pieejams interneta pieslēgums ierīcei. Jo šis bloks meklē datus internetā un bez pieslēguma tas nevar strādāt.*

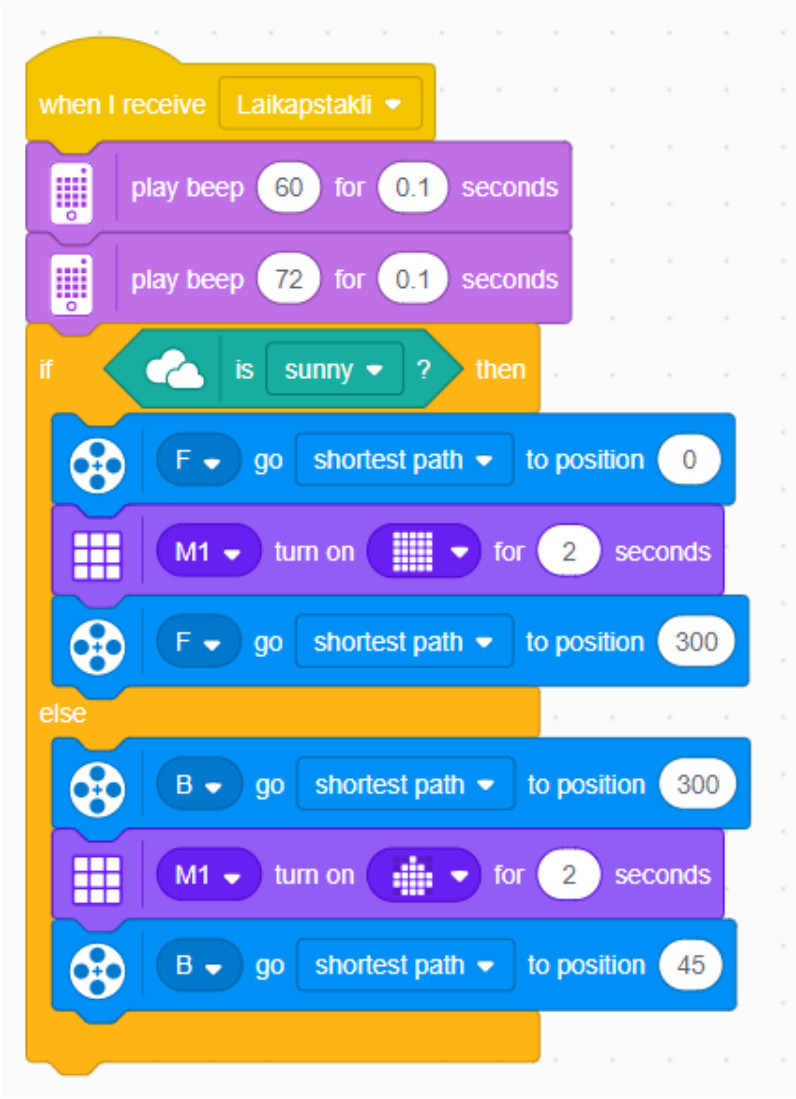
Tagad, kad robots ir spējīgs noteikt, vai ārā ir saulains laiks, var pievienot kādu citu gadījumu. Ērti ir aplūkot lietus gadījumu, jo robotam rokās ir lietussargs. Tātad, ja ārā līst lietus, robots var pacelt lietussargu un tādā veidā parādīt, ka līst lietus. Tātad, pirmais solis ir papildināt pirmo kolonnu, kas sākoties programmai nolaiž arī *roku*, kurā ir lietussargs:

Lietussargs tiek nolaists, ja motors pie porta B tiek pagriezts līdz pozīcijai 45. Sākumā tiek arī iestatīts šī motora ātrums. Pārējais paliek tāpat. Tagad atliek izmainīt otru programmas kolonnu. Šeit tiek piedāvāts šāds risinājums:



Šeit, atšķirībā no iepriekšējā attēla, vietā, kur bija vienkārši uzzīmēts X uz bloka, ir ievietots vēl viens sazarojums. Tajā, līdzīgi kā gadījumā, kad tika pārbaudīts tikai saulainais laiks, tiek pārbaudīts, vai ārā līst lietus. Un ja līst, robots uz galvenā bloka atspoguļo ūdens pilei līdzīgu simbolu un paceļ lietussargu. Pretējā gadījumā atspoguļo lielu X.

Ieteikums: šeit var rasties problēma, ja nepareizi izveido sazarojumus. Nepareizs ir šāds gadījums:



Kur papildus sazarojuma vietā vienkārši tiek ievietot darbība, kas atbilst lietainam laikam tur, kur iepriekš parādīja lielu X, ja nebija saulains. Tas ir kļūdaini, jo tagad gadījumos, kad ārā nebūs saulains ,robots izpildīs tās darbības, kas atbilst lietainam laikam. Tas ir, ja ārā būs vienkārši apmācies, robots veiks darbību, kas atbilst lietainam laikam. Dažkārt, ja nav skaidrs, vai sazarojums ir izveidots pareizi, ir vērts uzzīmēt tam atbilstošo blokshēmu un padomāt, vai pie visiem iespējamajiem gadījumiem programma strādās pareizi.

Lai pārliecinātos, vai robots darbojas pareizi, programmā var nomainīt sazarojumu nosacījumus tā, lai tiktu izpildīts vēlamais gadījums. Piemēram, ja ir nepieciešams pārbaudīt gadījumu, kad līst lietus, bet ārā spīd saule, var apmainīt vietām nosacījumus, līdzīgi, kā iepriekš.

10.2.2. Patstāvīgi veicamie uzdevumi

1. uzdevums.

   Šajā uzdevumā būs nepieciešams uzzīmēt blokshēmu atbilstoši situācijas aprakstam: Pusdienlaikā, ieejot ēdnīcā, katram noteikti ir kāds princips, pēc kura izvēlēties, ko katrs

labprāt ēstu pusdienās. Taču dažkārt var novērot, ka ēdnīcā ne vienmēr ir tas, ko katrs varētu vēlēties. Šādos gadījumos, ja nav visiekārojamākās sastāvdaļas, tās vietā katrs izvēlās ko citu un tā tālāk. Piemēram, ja cilvēks labprāt uz šķīvja redzētu kartupeļus, bet viņu nebūtu, tad šis cilvēks izvēlētos kartupeļu vietā rīsus un līdzīgi ar pārējām sastāvdaļām piedāvājumā. Uzdevums ir katram uzzīmēt savu blokshēmu, kas atbilstu algoritmam, pēc kura tiek izvēlēts pusdienu saturs.

2. uzdevums.

   Rudens vakaros bieži rodas situācija, kad ārā ir salīdzinoši gaišs, taču apsēžoties savā istabā pie datora ekrāna, pēkšņi ārā kļūst ļoti tumšs un ir nepieciešams ieslēgt gaismu. Tas dažkārt nozīmē, ka ir nepieciešams nopauzēt savas darbības un ieslēgt gaismu. Līdzīgi var būt brīžos, kad tomēr istabā paliek pārāk gaišs un ir grūti ko padarīt, ja gaisma spīd acīs vai atspīd pret datora ekrānu un tajā neko nevar redzēt. Šeit kā risinājumu var minēt aizkaru/žalūziju aizvēršanu.
   Būtu diezgan ērti, ja būtu sistēma, kas spētu gan aizvērt/atvērt žalūzijas/aizkarus, gan arī izslēgt vai ieslēgt gaismu istabā. Tāpēc uzdevums: Izveidot blokshēmu šādai sistēmai.

   Ieteikums: šeit gan jāatceras, ka pa nakti aizkari/žalūzijas tiek aizvērtas un pa dienu gaisma tiek izslēgta un gaišumu var regulēt ar aizkaru atvēršanu.

3. uzdevums.

   Nākamais uzdevums būs aplūkot, kas veido skaņu. Programma būs pavisam vienkārša, robotam ir jārāda uz ekrāna - *DISPLAY:*

4. skaļums, kādu tas uztver.

   Mēģiniet noteikt, kas rada skaļu skaņu un kas nē. Šajā uzdevumā sazarojumi neparādīsies tieši programmā. Salīdzināšana notiks starp objektiem ,kurus lieto, lai radītu skaņu.

   Kad tas ir darīts, var padomāt, kas īsti rada skaņu? Kādu objektu saskarsme rada lielāku skaņu, kāda mazāku?

5. uzdevums.

   Šajā uzdevumā vienkārši nepieciešams izveidot robotu, kas spēj atpazīt krāsas. Nav noteikta kāda specifiska konstrukcija, galvenais, lai robots varētu atpazīt krāsas. Uzdevumā būs nepieciešams lietot bloku:

   Kuru var atrast sadaļā *"DISPLAY"*. Uzdevums ir izveidot programmu, kas spēj uzrakstīt krāsu, kura sensorā ir redzama, lietojot šo *DISPLAY* bloku. Tas ir, pieliekot sensoru pie sarkana objekta, uz ekrāna tiek uzrakstīts "Sarkans!".

   Ieskicēt šim uzdevumam blokshēmu.

6. uzdevums.

   Izveidot robotu ar nosaukumu *"DRIVING BASE 3"* no *BUILD* sadaļas. Izveidot šim robotam programmu, kas liek robotam braukt taisni. Ja robots krāsu sensorā ierauga sarkanu krāsu, tas apstājas un gaida, līdz sensorā parādās zaļā krāsa un tikai tad atsāk kustību taisni.

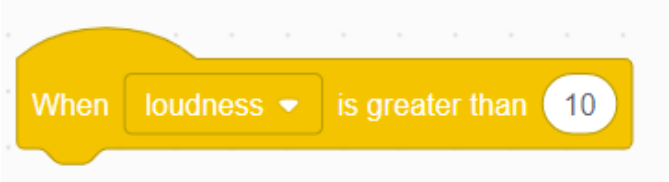
7. uzdevums.

   Lai veiktu šo uzdevumu, ir nepieciešams pie robota pievienot attāluma sensoru. Izveidojiet programmu, kas atkarībā no attāluma, kādā ievilkts klāt objekts attāluma sensoram, nospēlē kādu skaņu. Piemēram, ja attālums ir starp 10 un 15 cm, tad tiek nospēlēta nots A, bet ja šis attālums ir starp 15 un 20, tad nospēlē citu noti/skaņas signālu.

   Papilduzdevums: Izveidot šim robotam konstrukciju, lai tas atgādinātu ģitāru un to varētu arī spēlēt.

8. uzdevums.

   Atrodoties bibliotēkas lasītavā, ir svarīgi ievērot klusumu, lai netraucētu apkārtējiem. Taču dažreiz gadās situācijas, kad darot dažādus darbus grupās, sanāk neievērot klusumu. Tāpēc būtu labi, ja varētu izveidot asistentu, kas palīdzētu ievērot klusumu. Lai šo realizētu, būs nepieciešams lietot bloku:



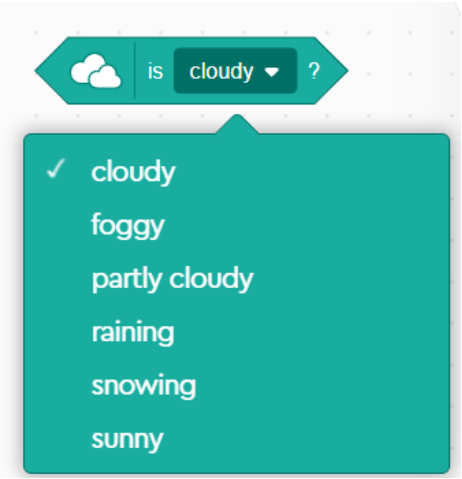
   Uzdevums ir nevis parādīt, ka telpā ir pārāk skaļš, bet izveidot robotu, kas pakāpeniski brīdina, ka telpā paliek skaļš, tas ir, sākumā tas var būt uzraksts, taču tiklīdz paliek

skaļāk, robots var pielietot pats savus skaņas efektus. Šajā uzdevumā nav dota konkrēta konstrukcija, kuru būtu nepieciešams izveidot.

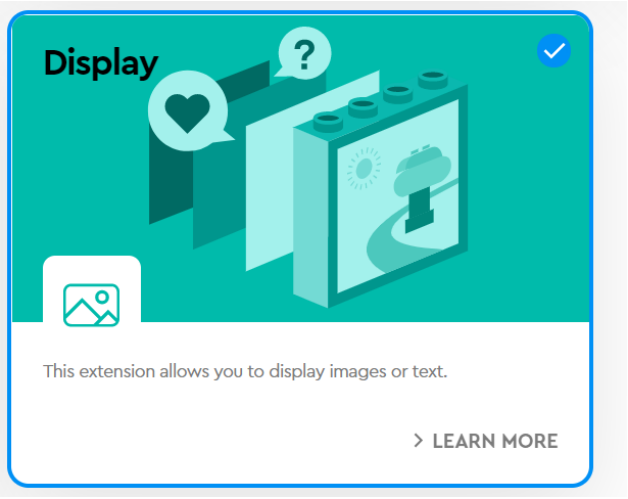
Nepieciešams: patstāvīgi izveidot programmu, kas atbilst augstāk aprakstītajiem nosacījumiem. Katrs var izstrādāt pats savu metodi, kā brīdināt par pieaugošo skaļumu. Šim uzdevumam nav nepieciešams nekāds īpašs dizains, taču būtu patīkamāk, ja uzdevums ar tik svarīgu uzdevumu nebūtu vienkārši bloks, tāpēc ir vēlams arī izveidot patstāvīgi pašu robotu.

9. uzdevums.

Iepriekš tika aplūkots uzdevums, kurā tika izveidots robots, kas spēj pateikt, vai uz šobrīd ārā līst lietus vai spīd saule. Uzdevums ir papildināt šo programmu tā, lai tas spētu noteikt visus iespējams laika apstākļus, kurus piedāvā bloks:



Lai to izdarītu, ieteicams ir izmantot blokus no sadaļas:



kas ļaus rezultātus uzrakstīt uz uznirstošā loga, programmēšanas vidē.

10. uzdevums.

Ir radusies situācija, kurā ir vairākas pilsētas, kur doties pārgājienā, taču nav skaidrs uz kuru pilsētu doties. Viens no veidiem, kā šo jautājumu risināt, būtu aplūkot laika apstākļus katrā no pilsētām. Iepriekš tika aplūkots robots, kurš spēj pildīt šīs darbības.

Šajā uzdevumā var lietot to pašu robotu. Tāpēc uzdevums ir izvēlēties vairākus galamērķus (pilsētas), kur varētu doties pārgājienā. Aplūkot laiku, kad uz šīm pilsētām varētu doties pārgājienā un noteikt laika apstākļus un temperatūru. Un no šiem datiem atrast pilsētu, kurā būs saulains un vissiltākais laiks.

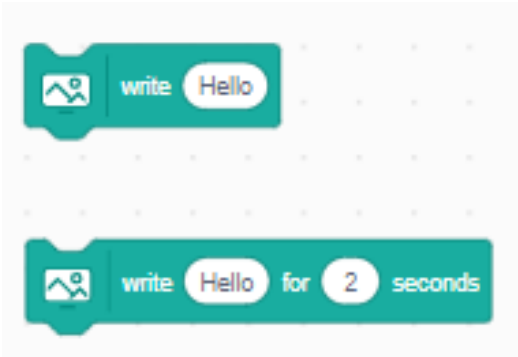
Tātad, uzdevums ir atrast pilsētu, kurā pilsētā ir visaugstākā temperatūra un spīd saule. Tā kā doties pārgājienā nav patīkami lietus laikā, ja aplūkotajā laika momentā nav saulains, tad ekskursija būs jāpārceļ uz citu laiku.

Ieteikums: Nosacījumu, ka jābūt saulainam var nomainīt uz nosacījumu, ka nelīst vai nesnieg.

11. uzdevums.

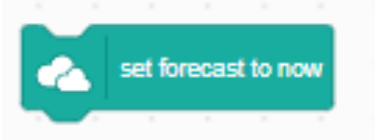
Ir saņemta ziņa, ka kādā meteoroloģijas stacijā ir nepieciešams veikt vēja ātruma lasījumus, lai varētu izveidot prognozi nākamajām stundām. Tātad, uzdevums ir izveidot robotu, kas spētu nolasīt vēja ātrumu un spētu parādīt nolasītos datus.

Šī uzdevuma veikšanai var lietot robotu ar nosaukumu *" Wind Indicator"*. Vēja ātrumu ir nepieciešams noteikt divējādi, lietojot bloku *DISPLAY*:



lai parādītu precīzus rezultātus un izmantojot robotu. Robots var noliekties uz leju atmuguriski, jo stiprāks vējš, jo zemāk noliecas robots. Piemēram, ja ārā vēja ātrums ir starp 0m/s un 2 m/s, tad robots noliecas mazliet, ja starp 2m/s un 5 m/s, tad robots noliecas zemāk un tā tālāk.

Lai nolasītu rezultātus, jārīkojas līdzīgi, kā ar laika apstākļu noteikšanu. Vispirms norāda lokāciju. Lai nolasītu datus:



Kad dati nolasīti, vēja ātrumu var iegūt lietojot:

10.3. Cikli

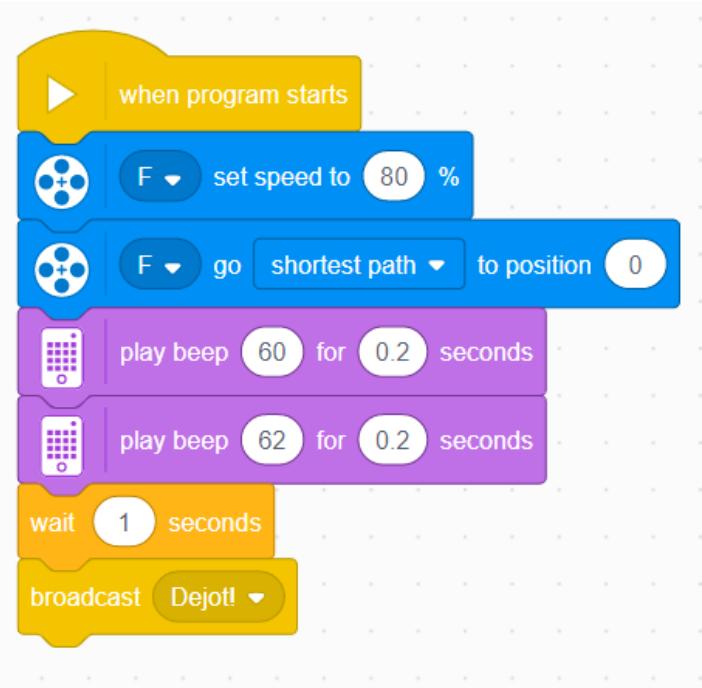
10.3.1. Kopīgi aplūkotie uzdevumi

Kopīgi aplūkotais 1. uzdevums

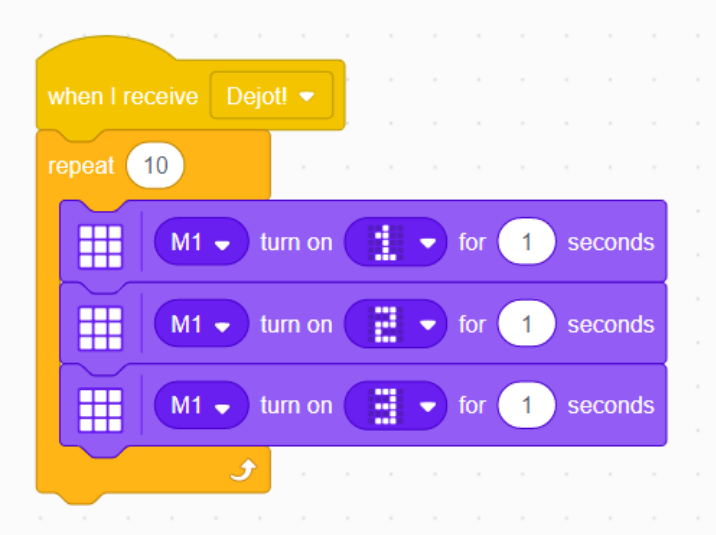
Šajā uzdevumā tiks aplūkota reālās dzīves situācija, kurā parādās cikli. Tā ir dejošana. Šī uzdevuma mērķis būs sinhronizēt cikliskas kustības. Piemēram, izveidot savu mūziku un to savienot ar robota kustībām.

Pirmais solis ir izveidot pašu robotu. Robota būvēšanas instrukciju var atrast *BUILD* sadaļā ar nosaukumu "*BREAK DANCER"*.

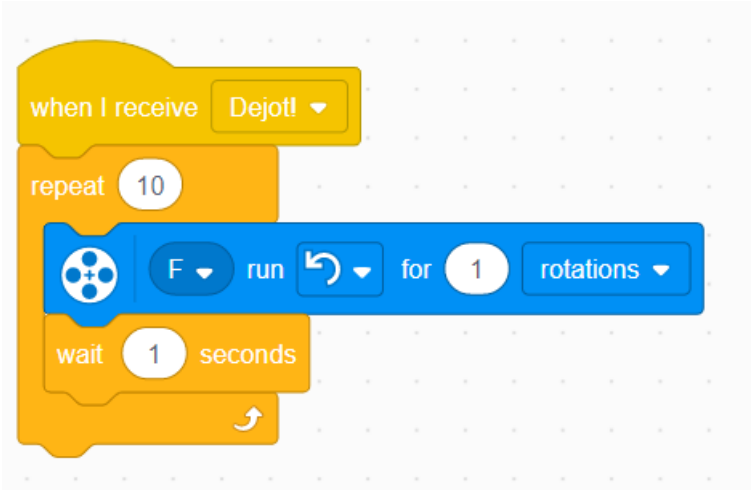
Kad robots ir uzbūvēts, var sākt tā programmēšanu. Pirmais programmēšanas uzdevums ir likt robotam kustināt kājas un uz galvenā bloka skaitīt līdz trīs sekundēm. Tāpēc nākamais solis ir izveidot programmas fragmentu, kas sagatavo robotu dejošanai:



Šis fragments novieto robotu sākuma pozīcijā. Iestata motora kustību ātrumu un dod tālāk ziņu, ka var uzsākt dejošanu. Šeit ērti ir lietot ziņas nosūtīšanu, jo tas katrai darbībai - mūzikai, kustībām, gaismām, ļauj izveidot savu kolonnu. Tālāk atliek pievienot kolonnu, kas parāda ciparus uz galvenā bloka:



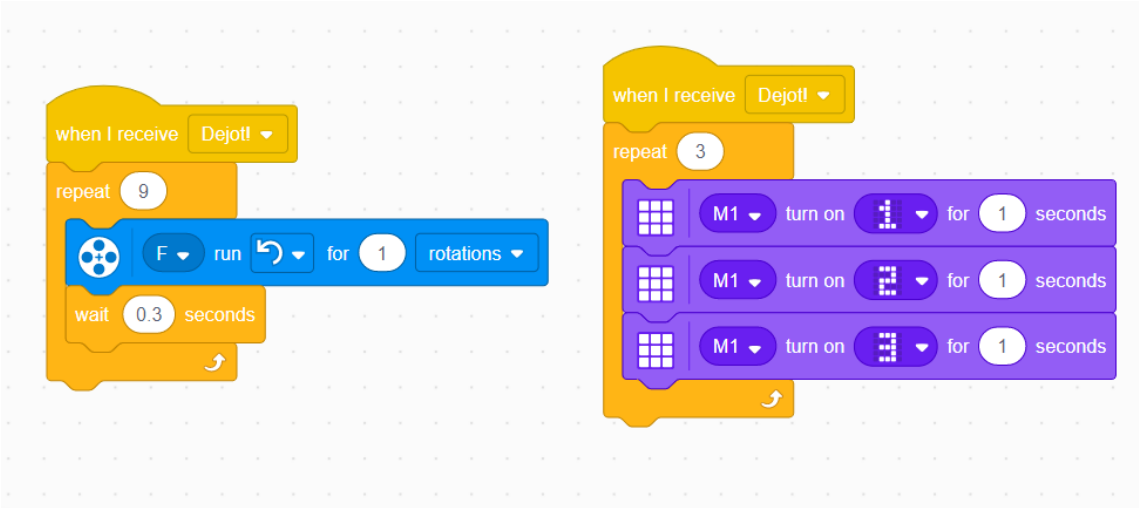
Šeit pavisam vienkārši, desmit reizes tiek atkārtota ciparu parādīšana uz galvenā bloka. Tālāk atliek pievienot kāju kustības.



Arī šeit darbības ir pavisam vienkāršas, motoram pie porta F veicot vienu apgriezienu, robots vienu reizi turp un atpakaļ pakustina gurnus. Un pēc tam pagaida 1 sekundi. Šo 1 sekundi var pamainīt gadījumā, ja šīs kustības nav vienā ritmā ar skaitļiem. Tagad atliek palaist robotu un skatīties, vai kustības ir sinhronizētas ar skaitļu maiņu. Vajadzētu novērot, ka šīs kustības nav sinhronizētas. Tās nav ritmā. Atliek veikt korekcijas ar laiku pie kāju kustībām. Otrs, ko varētu ievērot , ka skaitļi turpina mainīties, lai gan robots ir beidzis dejot. Tas nozīmē ir nepieciešams veikt korekcijas.

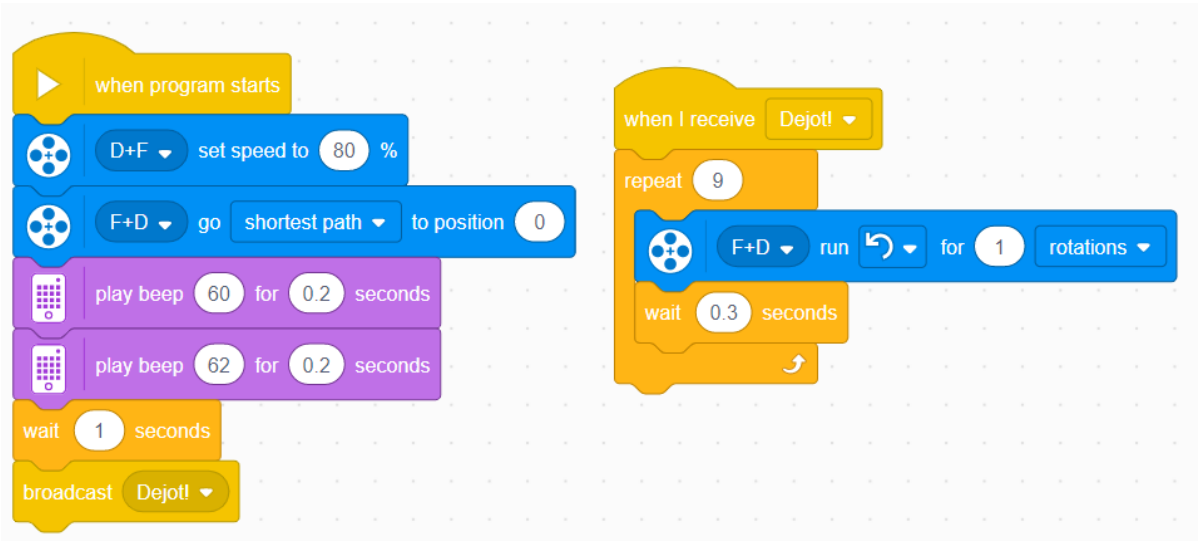
IETEIKUMS: katram atrast laika vienību, kas liktu robotam kustināt kājas ritmā ar skaitļiem un atkārtotos.

Šeit piedāvātais risinājums ir nomainīt laika vienību uz:



laika gaidīšanu uz 0.3 sekundēm un izmainīt darbību atkārtošanās skaitu. Katrs var piemeklēt sev tuvāko atkārtojumu skaitu. Šis noteikti nav vienīgais variants, kā likt dejot tik pat ilgi, cik rādīt ciparus uz galvenā bloka.

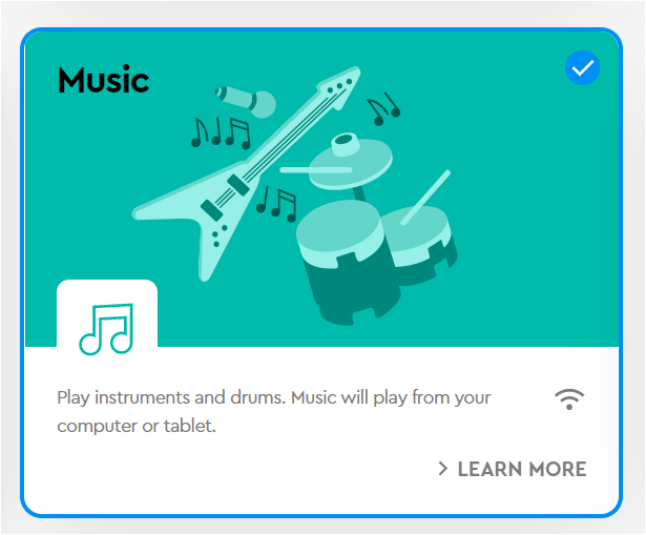
Tagad, kad ir izveidotas kāju kustības, tālāk ir nepieciešams pievienot roku kustības. Kā risinājums varētu būt vienkārši nokopēt visu to, kas bija nepieciešams kāju kustībām. Taču vienkāršākais risinājums ir:



Tur, kur programmā parādās motors, kas kustina kājas, klāt vienkārši pievienot motoru, kas kustina rokas. Šādi neviens jauns bloks nav jāpievieno programmai.

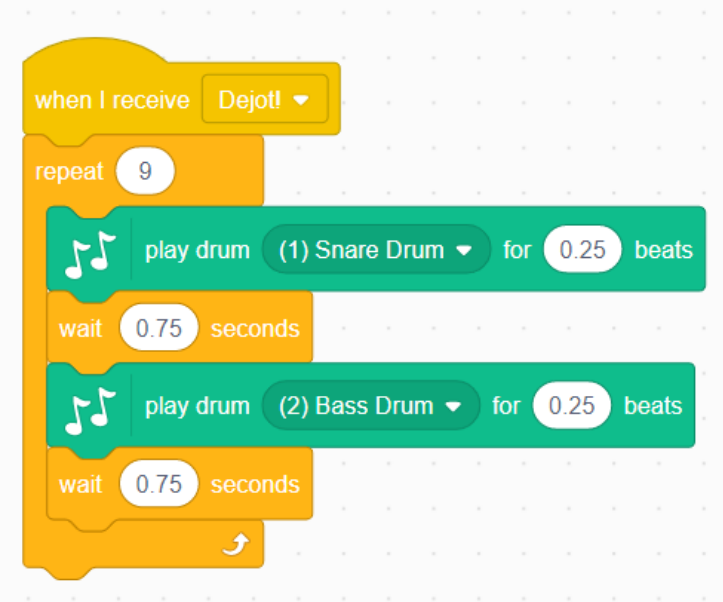
Tagad atliek tikai pievienot mūziku. Lai veidotu mūziku, programmēšanas vidē, kreisajā, apakšējā stūrī ir jāspiež uz ikonas:

un logā, kas atveras, ir jāieklikšķina mūzikas sadaļa:



Tagad vajadzētu parādīties papildus mūzikas blokiem. Tālāk, līdzīgi kā ar pārējām darbībām, pēc ziņas "Dejot!" saņemšanas sākas cikls:

IETEIKUMS: Arī šeit ikviens ir aicināts patstāvīgi izveidot savu mūziku, kas ir vienā ritmā ar jau iepriekš izveidotajām kustībām.



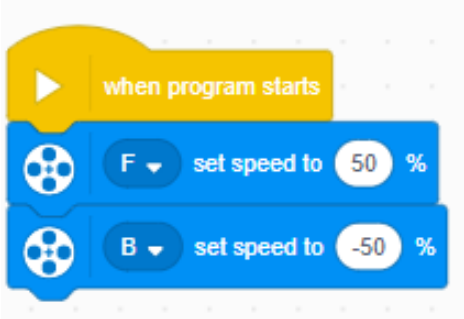
Piemērs, kur programmā izveidotā mūzika ir vienā ritmā ar dotajām kustībām. Šeit mūzika ir pavisam vienkārša.

Kopīgi aplūkotais 2. uzdevums

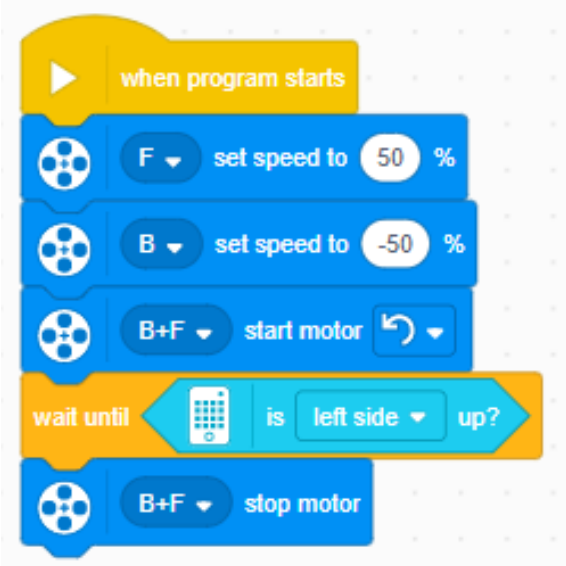
Šajā uzdevumā tiks aplūkots kāds ikdienā sastopams process, kurā var novērot ciklus. Tā ir sportošana. Veicot fiziskās aktivitātes diezgan bieži kāda konkrēta darbība tiek atkārtota vairākas reizes. Piemēram, lekšana ar lecamauklu, skriešana pa stadionu. Konkrēti šajā uzdevumā tiks aplūkota cikla izveide un veikto ciklu skaitīšana un tiks izveidots robots, kurš var asistēt fizisko aktivitāšu laikā. Šis robots var kalpot kā personīgais treneris.

Vispirms jāizveido pats robots. Robotu var atrast sadaļā *Build* ar nosaukumu: *"Leo The Trainer"*. Šis robots ir spējīgs veikt ķermeņa augšdaļas pacelšanu jeb presi.

Sākumā pašam robotam ir jāiemāca veikt presi. Sākumā svarīgi ir novērot, ka motori ir pretēji novietoti. Tas nozīmē, ka abi motori sāks griezties vienlaicīgi , viens motors galveno bloku celtu augšā, kamēr otrs laistu lejā. Lai to paveiktu, vajag iestatīt ātrumus, ar pretējām zīmēm:



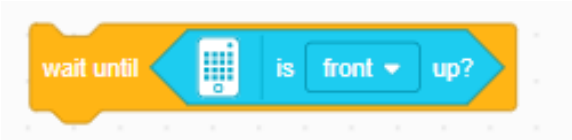
Tagad liekot abiem motoriem griezties, tie abi grieztos uz vienu un to pašu pusi. Lai sāktu veikt presi, robotam ir jāapguļas. Lai to izdarītu, tas ir jānolaiž uz leju. To var izdarīt liekot motoriem griezties uz priekšu.



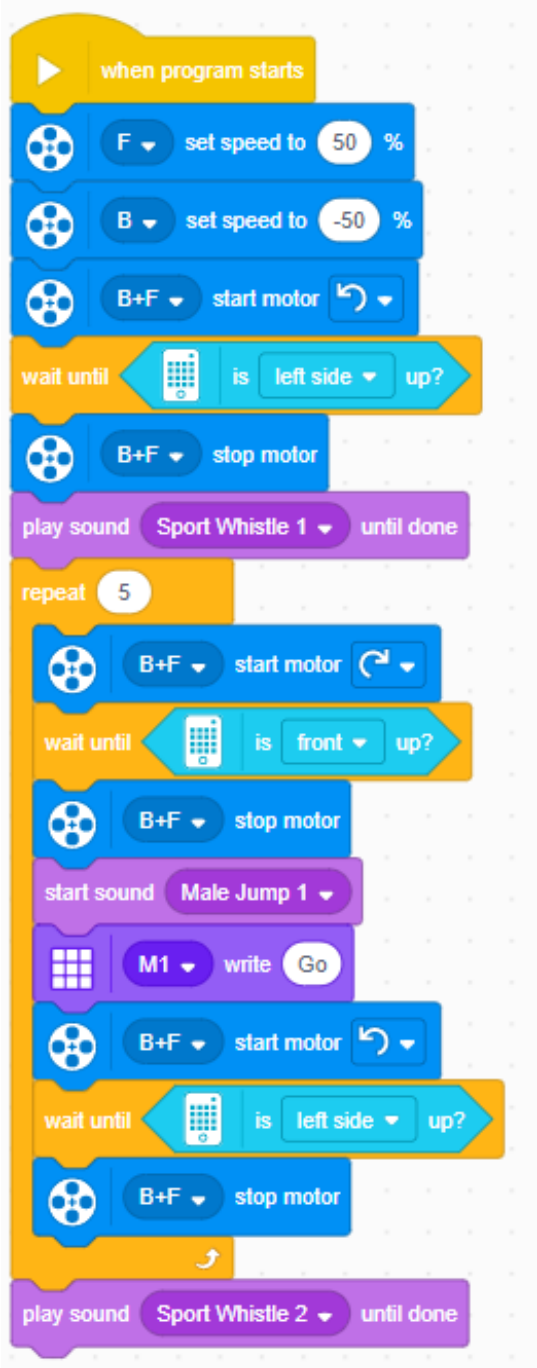
Tas varētu izskatīties šādi. Šeit varēja likt robotam griezties noteiktu rotāciju skaitu, kas to apguldītu, taču ērtāk ir izmantot sensoru, kas ir iebūvēts galvenajā blokā: žiroskopu. Robots griezīs motorus, līdz galvenā bloka kreisā puse būs vērsta uz augšu, tad motori apstāsies. Šeit

var novērot ciklu, kuram beidzas sazarojums, kas pārbauda, vai kreisā puse ir pacelta. Lai labāk izprastu šo programmas fragmentu, to ir ieteicams palaist uz robota.

Kad robots ir noguldīts, var uzsākt preses veikšanu. Aplūkojot, kā tika veikta robota apguldīšana, varētu būt skaidrs, kā robots varētu veikt presi. Var lietot to pašu sazarojumu, tikai pamainīt to uz:



kas liktu robotam pacelties un pēc tam atkārtot jau iepriekš lietoto sazarojumu. Kopējais fragments varētu izskatīties šādi:

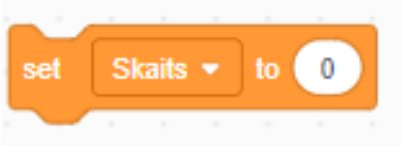


Kopā robots veic 5 preses. To var pateikt no tā, ka cikls, kurā atrodas preses veikšana veic 5 atkārtošanas. Tālāk vienkārši seko iepriekš parādītais algoritms preses veikšanai: Lai paceltos, tiek lietots sazarojums, kas pārbauda, vai robota galvenais bloks ir augšā un otrādi.

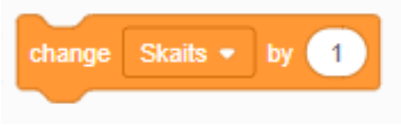
Šis fragments ir interesants, jo pirmajā acu uzmetienā izskatās, ka tajā ir tikai viens cikls, taču patiesībā tā nav. Tajā ir vairāki cikli. Katru reizi, kad programmā parādās:

robots veic ciklu. Robots veic komandu, kas tam ir dota pirms šī bloka atkārtoti, līdz stājās spēkā nosacījums, kas šajā blokā ir norādīts un izbeidz darbību.

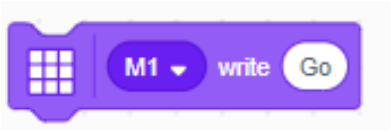
Tagad, kad robots spēj veikt presi, atliek aplūkot iespējas, kā skaitīt veikto presi. Tātad, uzdevums būtu uz robota galvenā bloka parādīt, cik preses ir veiktas. Tas nozīmē, ka būs nepieciešams ieviest kādu mainīgo, kas uzglabā šīs reizes. Vispirms ir nepieciešams to definēt. Sadaļā *VARIABLES* jāspiež uz *MAKE A VARIABLE* un jāizveido mainīgais. Pirmais solis būtu programmas sākumā mainīgajam piešķirt nulles vērtību, lai no iepriekšējām programmas darbības reizēm tam neuzkrātos kāda vērtība un netiktu parādītas nepareizs skaits. Tam lieto bloku:



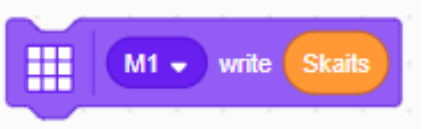
kur *Skaits* ir mainīgā nosaukums. Tālāk ir jāpieskaita viena reize katru reizi, kad robots ir piecēlies. To var veikt lietojot:



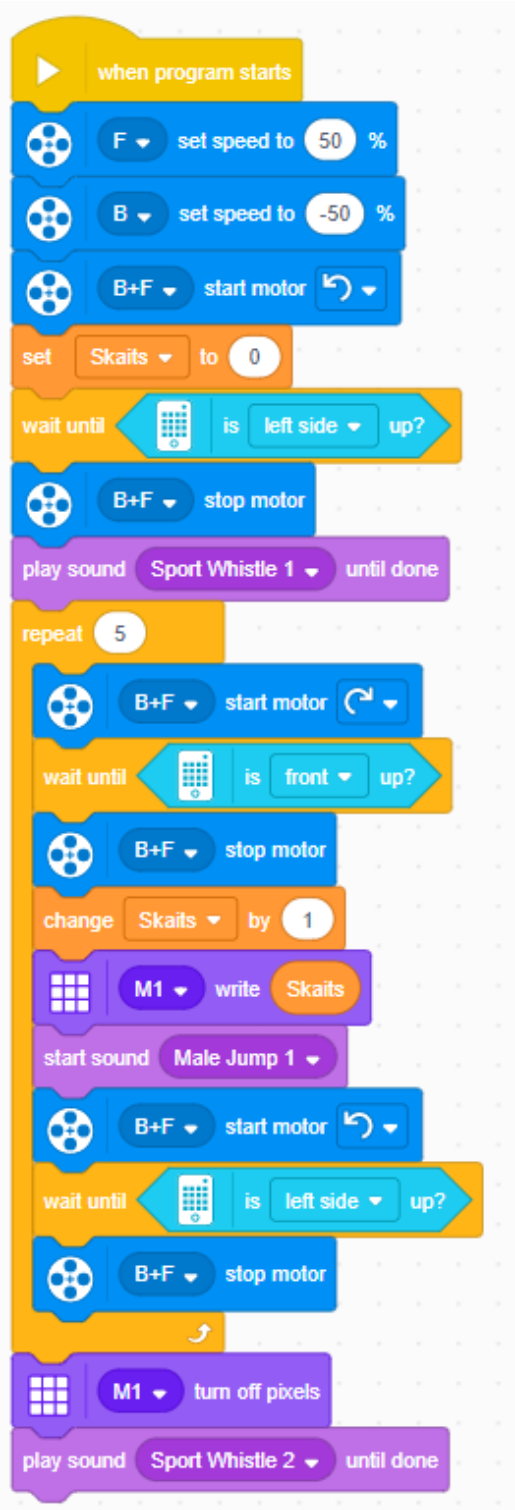
Un lai parādītu šo uz ekrāna, bloku:



Var aizstāt ar :



Kur vienkārši mainīgā vērtība ir ievietota vietā, kur iepriekš ir rakstīts *Go*. Kopā tas varētu izskatīties:

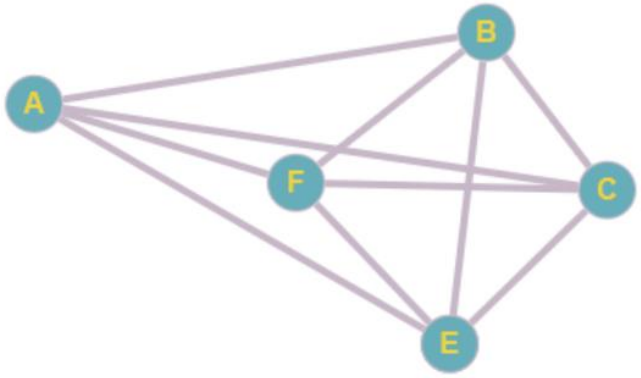


Šeit vēl beigās tiek pievienots bloks, kas nodzēš visas gaismas, kas tika ieslēgtas uz galvenā bloka. Tādējādi uz robota nespīdēs skaits, kādu robots veica ceļot ķermeņa augšdaļu. Šeit var palielināt atkārtojumu skaitu no 5 uz kādu patvaļīgu vērtību un pārliecināties, vai viss tik un tā strādā.

10.3.2. Patstāvīgie uzdevumi

1. **uzdevums**.

   Ir izveidota kāda jauna loģistikas firma. Kā viens no pirmajiem uzdevumiem ir izveidot maršrutu starp pilsētām. Tātad, ir dotas piecas pilsētas, A,B,C,F,E. Katra pilsēta ir savienota ar viena ar otru, taču šie attāluma nav vienādi. Šie izmēri ir zināmi. Jautājums, kāds varētu būt visīsākais maršruts, lai tiktu apbrauktas visas pilsētas.



   Padomājiet, kāds varētu būt veids, lai noteiktu visīsāko maršrutu? Vai tas strādātu, ja būtu vairāk pilsētu? Vai būtu iespējams izdomāt labāku algoritmu, lai atrastu īsāko maršrutu?

   Attālumi starp pilsētām doti tabulā zemāk.

	A	B	C	E	F
A	0	6	10	7	3
B	6	0	1	2	1
C	10	1	0	1	2
E	7	2	1	0	1
F	3	1	2	1	0

   Ieteicams arī padomāt, vai šis algoritms būtu optimāls patvaļīgā gadījumā.

2. **uzdevums**.

   Šajā uzdevumā būs nepieciešams lietot galvenajā blokā iebūvētos sensorus, lai veiktu precīzus pagriezienus. Šajā uzdevumā var lietot robotu ar nosaukumu *"Driving Base 1"*. Uzdevums ir izveidot programmas, kas ļautu robotam veikt attiecīgi 90, 180, 270 un 360 grādu lielus pagriezienus.

3. **uzdevums**

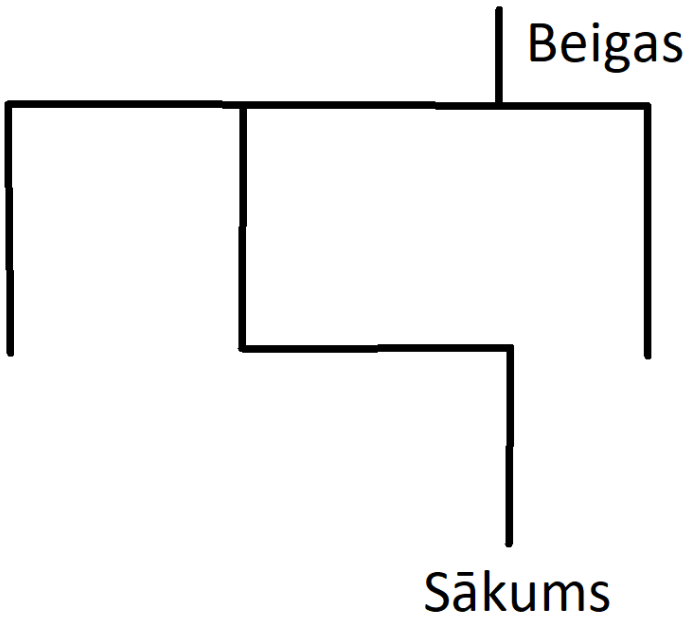
   Izveidot programmu, kad robots iedomājas 7 nejaušos skaitļus diapazonā no 1 līdz 20, parāda tos uz LED ekrāna un sakasītā šo skaitļu summu. Kad summa ir saskaitīta izvadīt to uz ekrāna.

4. **uzdevums**

   Izveidot robotu, kas veic teritorijas patrulēšanu. Patrulēšanas teritoriju noteikt patstāvīgi. Patrulēšanas laika, ja tas satiek kādu objektu ir nepieciešams apstāties un izvadīt skaņas signālu.

5. **uzdevums**

   Šajā uzdevumā būs nepieciešams izveidot robotu, kas ir spējīgs izbraukt līnījlabirintu. Var lietot robotu ar nosaukumu *"Driving Base 3"*. Šajā uzdevumā kā piemērs ir dots šāds labirints:



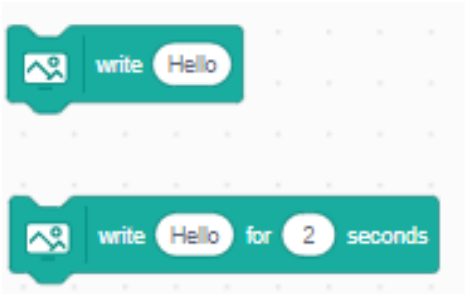
   kur ir iezīmēts sākums un beigas. Uzdevums ir izveidot programmu, kas spētu izbraukt šo labirintu. Ar melno krāsu ir iezīmēta līnija, kas atrodas uz zemes un pa kuru robotam būtu jāizbrauc trasi. Starp pagriezieniem ir taisns leņķis. Katra taisnā posma garumu var ņemt vienādu - 20 cm.

   Uzdevumu var dažādot un līnijas vietā ap trasi izvietot kastes. Tādā gadījumā robotam būtu jāizbrauc labirints, sekojot gar sienām ap to, nevis pa līniju. Uzdevumā var lietot arī savādāka izskata labirintu.

6. **uzdevums**.

   Vienā no iepriekšējiem uzdevumiem tika izveidota programma, kas dotajā brīdī spēj noteikt laika apstākļus. (Otrs no kopīgi aplūkotajiem uzdevumiem par sazarojumiem). Ir nepieciešams atjaunot tajā uzdevumā izveidoto programmu un robotu. Tālāk ir

nepieciešams papildināt programmu tā, lai tā spētu noteikt laika apstākļus nākamajām 5 stundām. Tas ir , robots spēj noteikt laika apstākļus, kas būt pēc stundas, divām, trīs un tā līdz piecām. Tā kā robotam ir limitēta iespēja tos parādīt, tad būtu ieteicams šos rezultātus parādīt lietojot vienu no blokiem:



7. **uzdevums**.

   * Šajā uzdevumā tiks aplūkots piemērs, kurā robotam ir jāizbrauc kvadrāts. Tas ir, robotam ir jāpabrauc uz priekšu un jāveic 90 grādu pagrieziens un jāatkārto kustības, kamēr tiek apbraukts kvadrāts.

   * Lietojot ciklus, izdomāt, kā izmainīt iepriekšējo punktu, lai katru reizi, kad robots brauc taisni, tas katru reizi veiktu arvien lielāku distanci. Tas ir, sākumā robots piemēram, veic 10cm, pagriežas par 90 grādiem un tad nobrauc 20 cm un tā katru reizi palielinot pa 10 cm. Vietas trūkuma dēļ, var lietot mazāku distanci un tās izmaiņas.

   * Atkārtot pirmajā punktā doto programmu, vienīgi, izbraucot vienu kvadrātu, robotam ir nepieciešams izbraukt vēl vienu kvadrātu, kurš ir lielāks. Realizēt šo ar vairākiem cikliem un pamēģināt arī ar vienu ciklu.

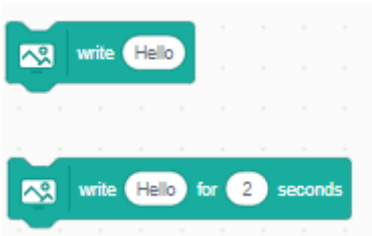
8. **uzdevums**.

   Intuitīvi mēs noprotam, ka metot monētu, ir vienāda iespēja uzmest gan ģerboni, gan ciparu. Taču vai tā tas patiešām ir? To var pārbaudīt metot monētu vairākas reizes, taču tas noteikti būs pārāk laikietilpīgi. Tāpēc var lietot robotu. Uzdevums būs simulēt monētas mešanu.

   Ieteikums:

   Ar šo bloku var simulēt monētas mešanu. Piemēram, ja tiek izvēlēts skaitlis 1, tad uzkrīt ģerbonis un 2 cipars vai otrādi.

   Monētas mešanu nepieciešams atkārtot vairākas reizes. Lai saglabātu vērtības, jāizvēlas pie vienas no vērtībām, kad tā uzkrīt, pieskaitīt mainīgajam +1. Un beigās parādīt rezultātu lietojot kādu no blokiem:



Monētas mešanu izmēģināt uzmest 10, 100 un 1000 reizes. Vai var novērot, ka rezultāts arvien tuvāk tiecās tieši ½ no metieniem?

Kā arī, robotam ir vēlams izveidot kaut kādu dizainu, lai tas nebūtu vienkārši kā viens pats bloks.

9. **uzdevums**.

Iepriekšējā uzdevumā tika simulēta monētas mešana. Monēta tika mesta N reizes un aplūkots, cik, piemēram, reizes uzkrita ģerbonis. Taču no fizikas kursa iespējams ir palicis atmiņā, ka dažkārt ir nepieciešams veikt vairākus mērījumus. Lai to izdarītu, būtu jāpalaiž programma vairākas reizes, taču tas būtu diez gan neparocīgi. Tāpēc šajā uzdevumā ir nepieciešams papildināt iepriekšējo uzdevumu tā, lai tas atkārtotu simulāciju vairākas reizes.

Ieteikums*:* Visus uzkritušos ģerboņus var saskaitīt un atrast vidējo vērtību no tiem.

Šeit tika lietots bloks:

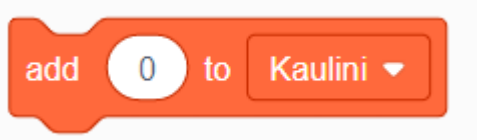
Kas raksturotu monētu, kurai ir vienāda iespēja uzkrist gan ģerbonim, gan ciparam. Taču kas notiktu, ja šīm darbībām nebūtu vienāda varbūtība izpildīties? Piemēram, ģerbonim būtu lielāka varbūtība uzkrist? To var simulēt nomainot apgabalu, no kura izvēlas ciparus:

Var piemeklēt savas vērtības, aplūkot kā mainās uzkrišanas skaits pie citām vērtībām.

10. **uzdevums**.

Vairākās galda spēlēs tiek lietoti metamie kauliņi. Dažās viens, dažādi divi, dažādas vairāk. It kā metot metamos kauliņus, katram no cipariem ir vienāda varbūtība uzkrist, taču, ja tiek lietot vairāki metamie kauliņi, noteikti varētu novērot, ka dažas vērtības parādās biežāk, nekā citas. Vai tā tas patiešām ir? Līdzīgi, kā iepriekš, ar roku mest kauliņus būs krietni sarežģītāk, nekā izveidot programmu, kas ļautu šo simulēt krietni ātrāk. Tāpēc šajā uzdevumā tiks simulēta 2 metamo kauliņu mešana. Lai šo realizētu, ir jāaplūko dažas lietas: Masīvi. Sadaļā pie mainīgajiem, var izveidot

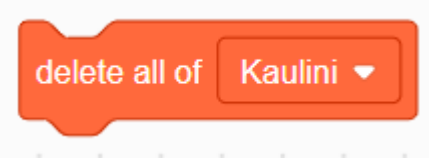
arī masīvus zem pogas *"MAKE A LIST"*. Masīvi ir kaut kas līdzīgs mainīgajiem, tikai tajos var uzglabāt nevis tikai vienu vērtību, bet gan vairākas. Taču, lai tajos ko varētu uzglabāt, vajag izveidot vietu masīvos, kur kaut ko var uzglabāt. Tāpēc vajag lietot bloku:



11. **uzdevums**.

Kā izveidot jaunu elementu masīvā, kurā var uzglabāt vērtības.?

Tā kā tiks simulēta 2 kauliņu mešana, tad maksimālā vērtība, kuru var sasniegt ir 12. Tāpēc vajag izveidot 12 elementus šajā masīvā. To var izdarīt atkārtojot šo darbību 12 reizes ar vienkāršu ciklu. Pirms tam, līdzīgi, kā ar mainīgajiem, masīvus vajag iztīrīt, tāpēc pirms pievienošanas, visu vajag izdzēst:



Masīvs ir vajadzīgs, lai tajā varētu uzglabāt to, cik daudz reizes uzkrīt katra kombinācija. Tas nozīmē, ja uzkrīt skaitlis 7, tad jāpieskaita 1 vienība 7. Elementam masīvā:

Tālāk ir atkārtoti jālieto šo:

kas simulētu 2 metamo kauliņu mešanu. Šo vērtību saglabāt kaut kādā mainīgajā. Un ar šī mainīgā palīdzību piekļūt attiecīgajam masīva elementam un pieskaitīt vienu vienību. Šo darbību jāatkārto vairākas reizes un jāparāda rezultātus.

Rezultātu parādīšanai ērti būtu uzzīmēt grafiku. Diemžēl stabiņu diagramma nepiedāvā uzzīmēt tik daudz atsevišķu elementu. Tāpēc masīvu ir jāparāda logā ar *DISPLAY* bloku. Šos rezultātus var norakstīt un iekopēt *Ms Excel*, kur uzzīmēt stabiņu diagrammu šīm.

Kura vērtība parādījās visbiežāk? Kāpēc, kā varētu pamatot šādus rezultātus? Kas notiek, ja pamaina skaitu, cik reizes tiek mests metamais kauliņš?

10.4.Uzdevumi ar sensoriem

10.4.1. Kopīgi veicamie uzdevumi ar sensoriem

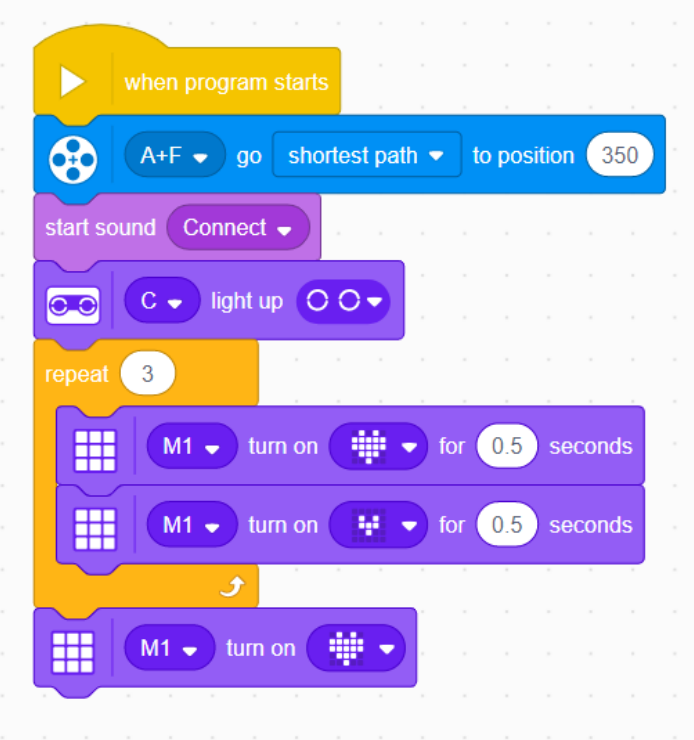
Kopīgi aplūkotais 1. uzdevums

Šajā uzdevumā tiks parādīts, kā sensori var tikt lietoti reālajā dzīvē. Uzdevumā tiks aplūkota situācija, kad ražotnē ir nepieciešams izveidot veidu, kā noteikt, kuras no precēm ir kvalitatīvas un kuras ir bojātas.

Šajā uzdevumā tiks izmantots robots, kura instrukciju var atrast *BUILD* sadaļā ar nosaukumu: *"QUALITY CHECK ROBOT"*.

Situācijas apraksts: Jūs strādājat rūpnīcā, kas ražo produktus. Šie produkti ir kvalitatīvā stāvoklī, ja tie ir violetā krāsā. Taču, ja kāds no produktiem ir zaļā krāsā, produkts ir bojāts. Mērķis ir lietot tikko izveidoto robotu, lai noteiktu produktu kvalitāti.

Pirmais solis varētu būt paša robota iedarbināšana un sagatavošana darbībai. Lai varētu lietot robota "galvu" rezultātu vizualizēšanai, to ir nepieciešams sagatavot. Sagatavot šajā kontekstā būtu novietot to tādā pozīcijā, lai to varētu brīvi kustināt un tā kustību netraucētu vadi.

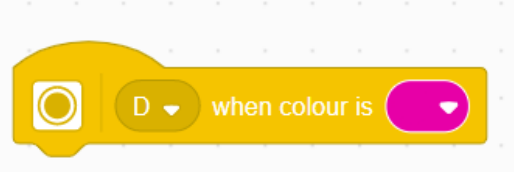


Šeit tiek piedāvāts abus motorus novietot pozīcijā 350. Papildus tam, tiek pievienoti dažādi vizuāli un skaņas efekti. Tas var ļaut noteikt, vai robots ir darba kārtība vai arī tas ir izslēgts. Brīdī, kad beidz mirgot un vienkārši deg apakšējā blokā parādītā ikona, robots ir pilnībā iedarbināts.

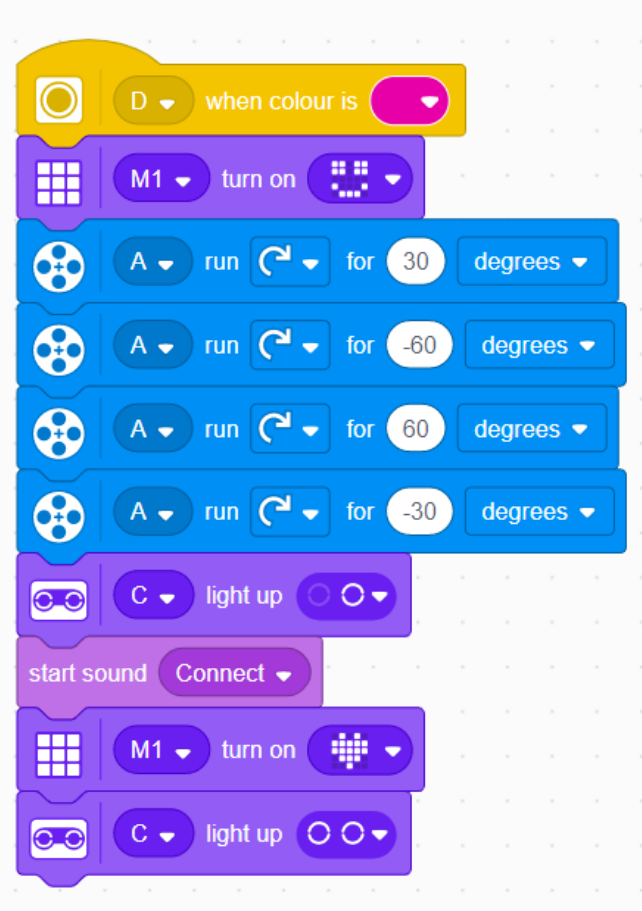
Ieteikums: Robots skaņas efektu atskaņos caur pievienoto ierīci. Tāpēc, lai to dzirdētu, ir jāpārliecinās, vai ierīcē ir ieslēgta skaņa un vai tā nav pārāk klusa.

Tagad ir izveidota programma, kas iedarbina robotu. To ir nepieciešams papildināt ar funkcijām, kas veic ražošanas produktu kvalitātes pārbaudi.

Pirmais solis varētu būt kvalitatīvo produktu pārbaude. Ir zināms, ka šie produkti ir violetā krāsā. Tas nozīmē, ka būs jāizmanto bloks:

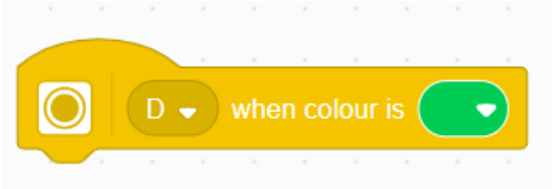


Tālāk atliek izdomāt, kā vizualizēt to, ka pārbaudītais produkts ir kvalitatīvs.

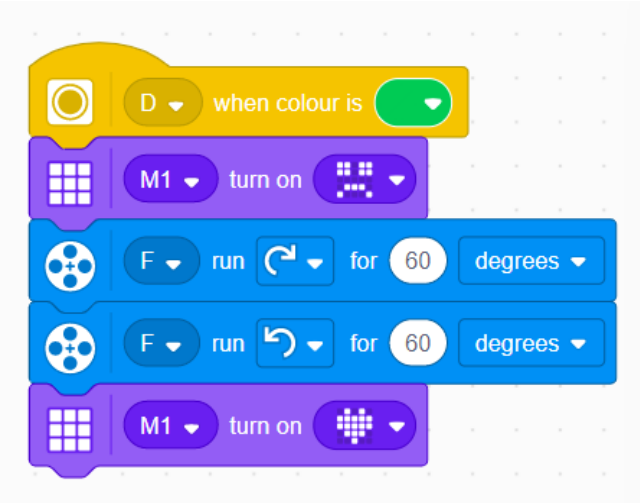


Šeit tiek piedāvāts šāds risinājums. Uz galvenā bloka tiek parādīta smaidīga seja un tikmēr robota "galva" pamāj, simbolizējot: "Jā!". Beigās uz galvenā bloka tiek atpakaļ nomainīts sākotnējais logo.

Tālāk atliek izveidot pretējo gadījumu, kad tiek pārbaudīts nekvalitatīvs produkts. Līdzīgi arī šeit tiks lietots bloks:



Un šeit analoģiski var lietot pretējo, kā tas bija pie kvalitatīviem produktiem:



Sākumā tiek parādīta bēdīga seja un robota "galva" pamāj, simbolizējot atbildi "Nē!". Un beigās tiek parādīts sākotnējais logo uz robota.

Iemesls, kāpēc zīmējums uz galvenā bloka tiek rādīts, kamēr robots nepārbauda produkciju ir tāpēc, ka pretējā gadījumā tur būtu pēdējā lasījuma zīmējums. Tas ir, parādot robotam bojātu preci, uz robota galvenā bloka paliktu uzzīmēta bēdīga seja. Tāpēc tiek uzzīmēta sirds forma. Šo formu var aizstāt ar vienkāršu zīmējumu, kurā visas gaismas ir nodzēstas.

IETEIKUMS: Pārbaudīt, vai kolonnās, kas atbilst kvalitātes pārbaudei, motoru kustībai ir uzlikti grādi ("degrees"), nevis rotācijas ("rotations"). Šī kļūda var būtiski ietekmēt robota darbību.

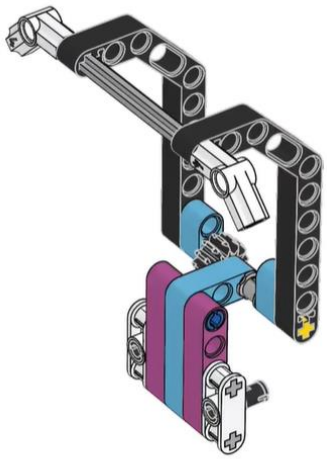
Tagad, atliek tikai iedarbināt robotu un pārbaudīt, vai robots darbojas pareizi. Ja robots darbojas pareizi, tas ir, krāsu sensorā parādot Zaļo krāsu, robots kļuva neapmierināts, parādīja bēdīgu seju un pamāja "nē". Taču, ja tika parādīta violetā krāsa, robots kļuva priecīgs un pasmaidīja, pamāja "jā" un atskaņoja skaņu. Ja robots darbojas pareizi, tad uzdevums ir pabeigts!

Kopīgi aplūkotais 2. uzdevums

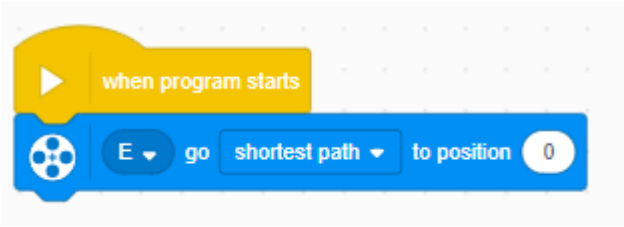
Šī uzdevuma mērķis ir parādīt sensoru nozīmīgumu un kā tie palīdz kontrolēt robotu. Piemēram, taisna gabala veikšana un apstāšanās noteiktā attālumā. Tiks aplūkots mēģinājums šo ieceri realizēt bez sensoru palīdzības un ar sensoru palīdzību.

Šajā uzdevumā tiks izmantots robots, kuru var atrast *BUILD* sadaļā ar nosaukumu: "*DRIVING BASE 2*".

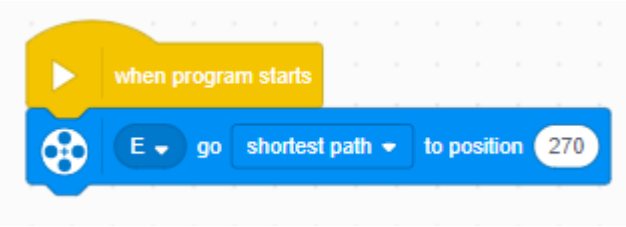
Ieteikums robota veidošanas laikā: Pirms tiem pievienota "*roka*":



Ieteicams ir izveidot šādu programmu:



kas liks motoram E aizbraukt uz sākuma pozīciju un tikai tad pievienot šo detaļu. Tas pēc tam ļaus vieglāk kontrolēt šo detaļu, lai to nolaistu, varēs lietot grādus, nevis likt griezties motoram noteiktu laiku, kas var izrādīties neprecīzi. Piemēram, tagad liekot motoram doties uz pozīciju 270:

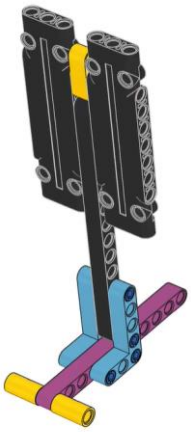


šī *roka* tiks nolaista.

IETEIKUMS: svarīgi nemēģināt *roku* kustināt un pieskarties, kamēr programma nedarbojas. Tas varētu nevajadzīgi izkustināt zobratus liekot motoram kustēties uz nulles pozīciju, un *roka* nestāvētu pacelta augšā. Gadījumā, ja šāda situācija rodas, var atvienot *roku* un bez *rokas* motoram likt aiziet uz nulles pozīciju un atkārtoti pievienot paceltu *roku*.

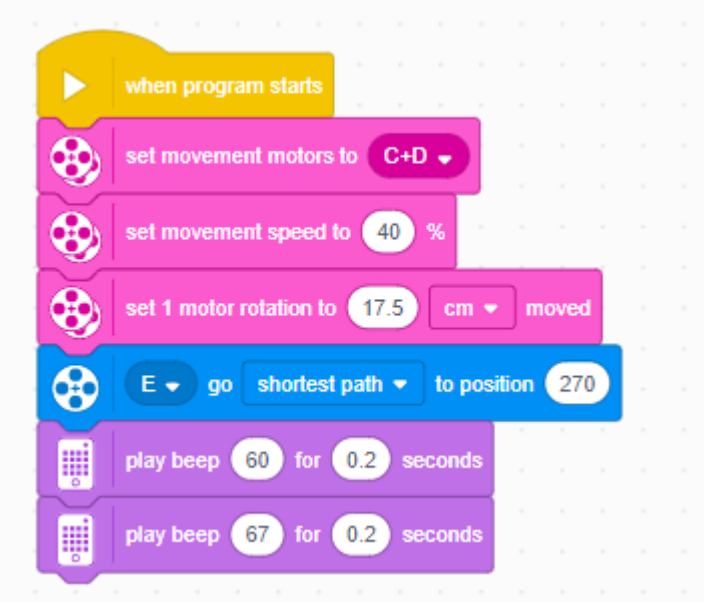
IETEIKUMS: Ja *roku* cenšas pacelt, motoram liekot braukt noteiktu rotāciju skaitu vai sekundes, tad brīdī, kad *roka* būs jau augšā, bet kustība nebūs beigusies, motors tiks bojāts. Tāpēc šāds variants *rokas* kustībai ir ieteicamāks.

Pirmais uzdevums būs aplūkot noteiktas distances veikšanu ar un bez sensoru palīdzības. Sākumā ir nepieciešams novietot robotu 40 cm no šī statīva tā, lai statīvs:



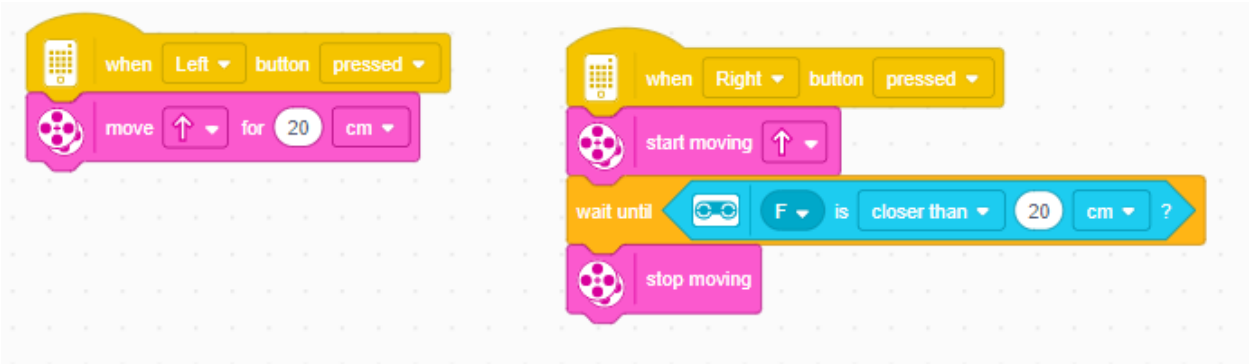
būtu tieši priekšā robotam.

Tātad, pirmais solis ir robotam parādīt, kuri motori ir jākustina un novietot robotam pievienoto *roku* vērstu uz leju. Šai daļai kods varētu izskatīties šādi:



Sākumā tiek parādīts, pie kuriem portiem ir pievienoti motori un ar kādu ātrumu tiks kustināts robots. Tālāk tiek iestatīts attālums, kādu veic robots, ja motors veic vienu pilnu apgriezienu. Šis attālums ir saistīts ar robotam pievienoto riteņu izmēru. Pievienojot mazākus riteņus, veicot vienu motora apgriezienu, nobrauktā distance būs mazāka. Tālāk pievienotā *roka* tiek nolaista uz leju, jo pretējā gadījumā tā atradīsies priekšā attāluma sensoram un to nevarēs lietot, lai noteiktu attālumu līdz statīvam. Kad tas viss ir pabeigts, robots atskaņo skaņu, lai paziņotu, ka var turpināt darbību.

Lai nebūtu jāveido divas programmas, lai salīdzinātu kustību ar un bez sensora, tiek piedāvāts izveidot atsevišķas kolonnas katram no gadījumiem, kuras var uzsākt ar pogas nospiešanu, kas atrodas uz robota galvenā bloka:

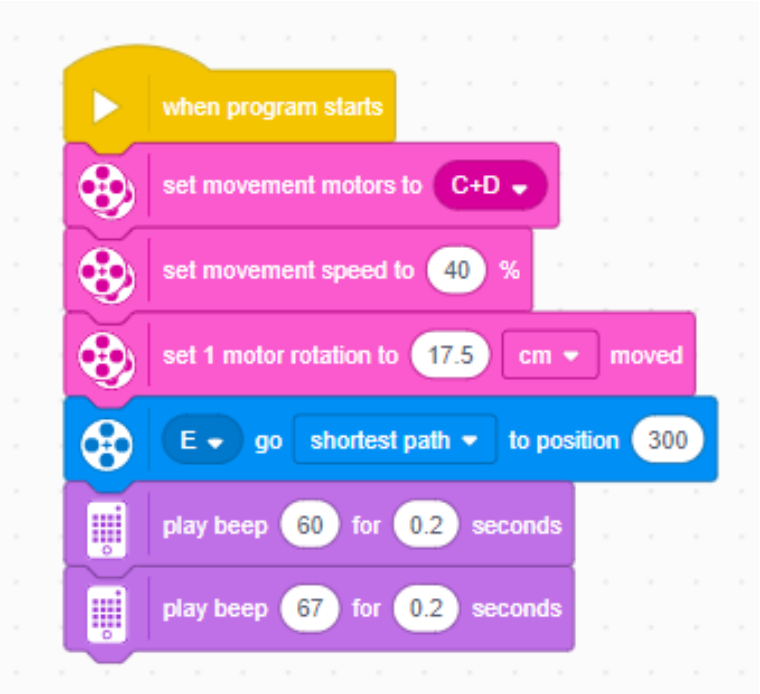


Tātad, nospiežot kreiso bultu, robots pārvietosies uz 20 cm. Nospiežot labo bultu, robots pārvietosies uz priekšu, līdz sensorā parādīsies distance, kas ir mazāka, nekā 20 cm.

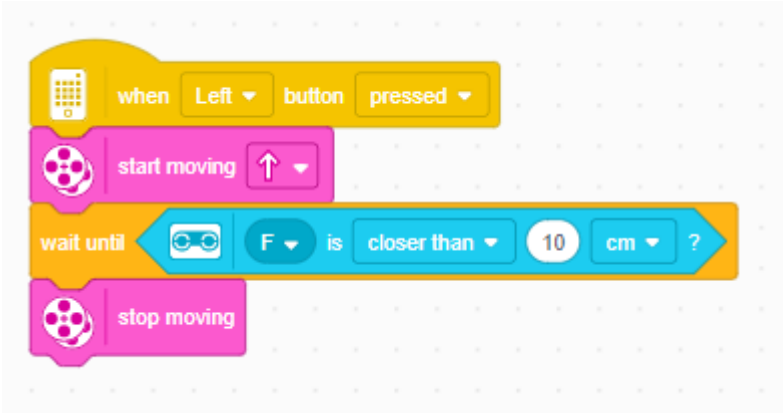
Tagad nepieciešams palaist šo programmu uz robota un aplūkot, kas notiek katrā no gadījumiem. Kurš no šiem varētu būt precīzāks? Atkarībā no tā, kā tika mērīti šie 40 cm, varētu rasties situācija, ka bez sensora, 20 cm tika veikti precīzāki. Taču šeit ir jāsaprot, ka šie 20 cm tika mērīti no sensora, nevis no pašas robota priekšas.

Nodaļā par lineāriem algoritmiem bija teikts, ka viena no lineāro algoritmu lielākajām problēmām bija robota noslīdēšana. Tas nozīmē, ka robotam veicot šos 20 cm, ja tas noslīdēs, robots nebūs spējīgs tos veikt precīzi. Turpretim, ja tiek braukts līdz sensoram, tad lasījumi ir zem 20cm, un nav svarīgi, vai robots noslīd vai nē. Galvenais, lai šis objekts, pēc kura vadās robots , tam ir priekšā.

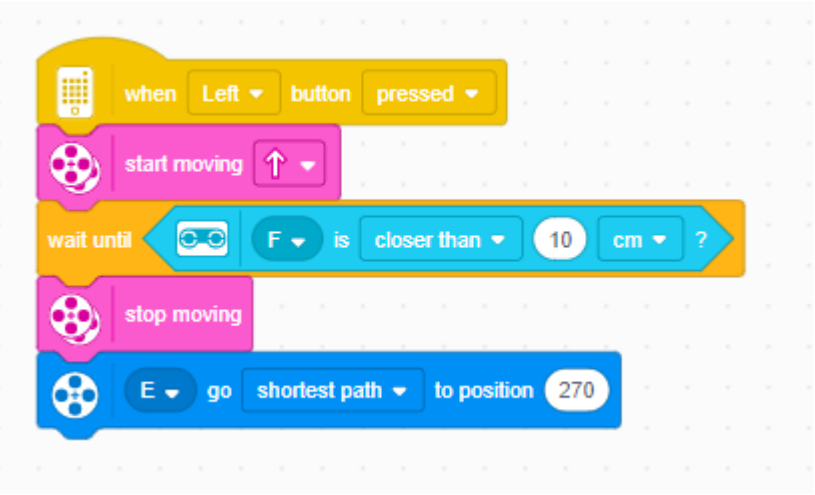
Nākamais uzdevums ir robotam likt pabraukt uz priekšu, nolaist roku tā, lai tā saturētu izveidoto kluci un ar visu kluci atkāptos atpakaļ. Sākumā nepieciešams novietot kluci aptuveni 10 cm no statīva un robotu novietot tam tieši priekšā. Svarīgs novērojums ir, ka robotam pārvietojoties ar nolaistu roku, robots nevarēs satvert šo kluci, jo roka būs priekšā, tāpēc ir nepieciešams veikt izmaiņas programmas kolonnā, kas atbilst motoru iestatīšanai:



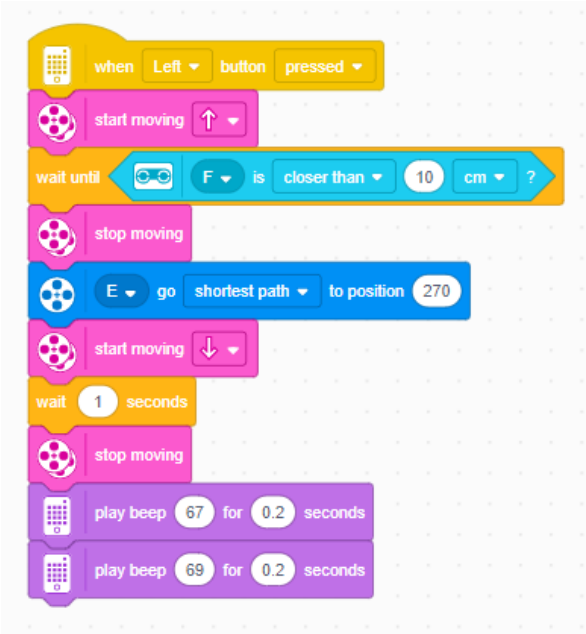
sākotnējā motora E pozīcija tiek iestatīta uz 300. Tādējādi *roka* netraucēs satvert kluci un nebūs priekšā sensoram. Nākamais solis varētu būt robotam aizbraukt līdz šim klucim. Tur var izmantot iepriekšējās kopas piemēru, vienīgi nepieciešams nomainīt distanci no 20 uz 10 cm:



Tagad robots ir apstājies ar kluci tā, lai nolaižot *roku*, klucis tiktu satverts. Tāpēc nākamais solis ir vienkāršs nolaist *roku*:



Šeit var lietot motora pozīciju 270, taču var nolaist drusku zemāk, līdz 260. Ja vēl vairāk samazina šo vērtību, tad jāskatās, lai pats klucis vai statīvs netraucē *roku* nolaist vēl zemāk. Un tālāk atliek vienkārši uzsākt kustību atpakaļ.



Šeit vienkāršs tiek piedāvāts kustēties vienu sekundi atpakaļ un apstāties. Svarīgi ir atcerēties pievienot apstāšanās bloku, jo savādāk robots vienmēr brauks atpakaļ.

Veicot šo uzdevumu, varēja novērot, ka nebija svarīgi, cik lielā attālumā robots tiek novietots no kluča, jo robots brauks uz priekšu, līdz sensora lasījumi būs mazāki par 10cm līdz statīvam. Ja šo mēģinātu realizēt nelietojot sensoru, bet gan konstruējot lineāru algoritmu, būtu ļoti svarīgi, cik lielā attālumā robots atrodas no statīva. Tātad, ja uzdevums būtu bijis salīdzināt šo kluča pabīdīšanu no dažādiem sākuma attālumiem līdz tam, šī programma nebūtu jāmaina, taču programmā, kas nelieto sensoru, būtu jāmaina braukšanas distance. Taču, jo lielāku distanci robots veiks, jo lielāka iespēja tam ir noslīdēt.

10.4.2. Patstāvīgi veicamie uzdevumi

1. **uzdevums**

   Viena no problēmām, ar kuru var saskarties mājdzīvnieku īpašnieki ir došanās vairākas dienas ārpus mājas, jo nav kam pabarot palikušos mājās mājdzīvniekus. Viena no opcijām, kā šo varētu risināt, būtu automatizēta barotava.

   Izdomāt, kādas funkcijas būtu jāveic šim robotam. Kādi sensori tam būtu nepieciešami un kāds varētu būt šī robota darbība princips? Ieskicēt šī robota blokshēmu. Kādi vēl roboti būtu nepieciešami, lai mājdzīvnieki varētu justies komfortabli, kamēr cilvēki ir prom ?

2. **uzdevums**

   Vairāku pilsētu centros ir ierīkotas lielas atkritumu tvertnes, kas paredzētas ir gan plastmasas atkritumiem, gan arī stiklam. Šeit problēma ir tāda, ka šīs tvertnes tiek tīrītas pēc kāda noteikta laika, piemēram, reizi mēnesī. Kāpēc tā ir problēma? Jo pastāv iespēja, ka šie konteineri varētu piepildīties krietni ātrāk un būt pilni kādu laiku vai arī tieši pretēji, atbraucot tos izvest, tie varētu būt gandrīz tukši un izvešana nemaz nebūtu vajadzīga.

   Uzdevums ir izdomāt robotu, kas varētu risināt šo problēmu un ieskicēt, kāda varētu būt blokshēma, pēc kuras šis robots darbotos.

3. **uzdevums** ir aplūkot tālāko nodaļu, kurā ir par sacensību noteikumiem. Atrast apakšnodaļu, kur ir minēts par *Lego Sumo* robotiem un vadoties pēc tiem, izveidot robotu, kas spētu piedalīties šajā disciplīnā.

4. **uzdevums**

   Kādā ražotnē ir nepieciešams izveidot robotu, kas spētu apsekot ražošanas iekārtas. Robotam piebraucot pie kādas no iekārtām ir nepieciešams apstāties, lai tajā ievietotu kravu. Lai robots spētu pārvietoties un orientēties, uz zemes tika novilkta melna līnija, kurai robots var sekot. Lai robots spētu identificēt, kad nepieciešams apstāties, uz melnās līnijas ir posmi, kad līnija ir nokrāsota sarkanā krāsā. Robotam uzbraucot uz sarkanās krāsas, ir nepieciešams apstāties. Pašlaik nav iespējams paredzēt iekraušanas laiku, tāpēc robotu pēc kravas iekraušanas ir manuāli jāpalaiž.

   Uzdevums ir ieskicēt blokshēmu, kas varētu atbilst tikko sniegtajam aprakstam. Kā manuālo robota palaišanu, kas minēta beigās, var iegūt ar pogas nospiešanu uz robota.

   Ieteikums: Tā kā uzdevumā būs nepieciešam realizēt šo opciju, tad nederēs blokshēmā vienkārši ievietot bloku: “Sekot līnijai”. Programmējot robotu, nav pieejama tāda bloka.

5. **uzdevums**

   Šajā uzdevumā ir nepieciešams izveidot robotu, kas spēj braukt taisni. Tam jāiestata

braukšanas ātrums vienāds ar 100%. Tālāk, ir nepieciešams modificēt robotu ar sensoriem tā, lai varētu noteikt šī robota braukšanas ātrumu. Uzdevums ir vienkāršs ar sensoru palīdzību noteikt, kāds ir robota braukšanas ātrums, ja tas brauc ar 100% ātrumu. Veicot mērījumus jāatceras, ka sensoru lasījumi nav precīzi, tāpēc būtu vēlams veikt vairākus mērījumus.

6. **uzdevums**

   Ceturtajā uzdevumā tika aprakstīta situācija un tai tika uzzīmēta blokshēma. Šajā uzdevumā ir nepieciešams realizēt aprakstīto blokshēmu. Tas nozīmē, ka, pirmkārt, ir nepieciešams izveidot robotu, kas būtu spējīgs sekot līnijai, kas novilta uz zemes. Šeit nav nepieciešams izveidot robotu tā, lai tam būtu kāds kravas nodalījums. Kad robots ir izveidots, atliek realizēt blokshēmu programmā.

   Ieteikums: Ar vienu krāsu sensoru līnijai nav iespējams izsekot, jo robots nespēj noteikt, uz kuru pusi tam jānobrauc nost. Tāpēc ieteikums ir nevis braukt pa līniju, bet tikai pa vienu līnijas pusi.

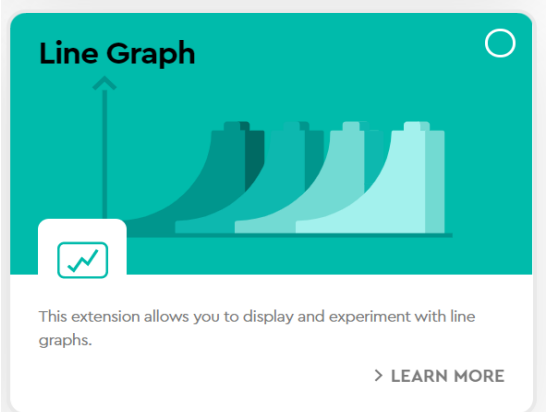
7. **uzdevums**

   Uzdevums ir aplūkot nākamajā nodaļā aprakstītos robotikas sacensību noteikumus disciplīnā *"FOLKRACE"* un izveidot robotu, kas atbilstu noteikumiem un spētu piedalīties sacensībās šajā distancē.

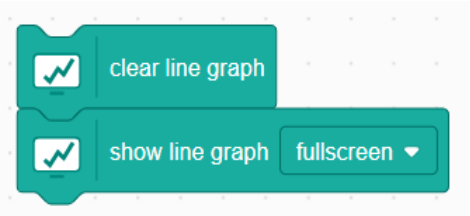
8. **uzdevums**

   Šajā uzdevumā tiks izveidots robots, kas uzkrās datus un tos parādīs grafikā. Uzdevumā ir paredzēts lietot robotu, kura konstrukciju var atrast *BUILD* sadaļā ar nosaukumu *"SMART KETTLEBELL "*. Uzdevumā, pēc pogas nospiešanas, kamēr poga tiek turēta, jālasa attāluma sensora dati un tie jāparāda grafikā. Pogu atlaižot, ir jāpārtrauc datu ieguve. Kamēr poga ir nospiesta, var mēģināt pietupties un palēkties pēc iespējas augstāk un aplūkot sensoru lasījumus grafikā.

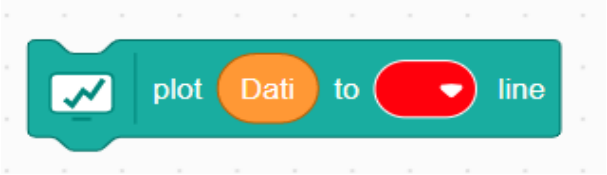
IETEIKUMS: Darbības ar grafikiem. Tās var atrast zem sadaļas:



Pirmais solis ir notīrīt grafiku, lai netraucētu iepriekš nolasītie dati un parādīt pašu grafiku.



Lai parādītu datus, var lietot:



kur šis tiek ievietots ciklā. Kamēr cikls darbosies, dati tiks parādīti grafikā. Lai varētu strādāt ar datiem, noteikt izmaiņas lēciena laikā vai pietupjoties, vajag iestatīt sākuma vērtību. Tas ir, nolasīt vērtību, kāda ir stāvot kājās un šo vērtību atņemt no datiem, kas tiek parādīti grafikā. Tādējādi, stāvot kājās būs nulles vērtība, pietupjoties būs negatīva un palecoties pozitīva.

Kad grafiks ir uzzīmēts, padomāt, kādus aprēķinus var veikt no šiem datiem. Kādus lielumus var izrēķināt? Vai var izrēķināt kāda potenciālā enerģija piemīt atrodoties visaugstākajā punktā?

9. uzdevums.

   Atrast vai izdomāt kādu problēmu, kas ir pašlaik ir aktuāla un izveidot robotu, kas varētu risināt šo problēmu. Pirmais solis būtu atrast problēmu un tad aplūkot visus iespējamos risinājumus. Pēc tam, aplūkot izdomātos risinājumus, izvēlēties vienu, kas vislabāk varētu risināt šo problēmu. Uz lapas ieskicēt blokshēmu, kas varētu atbilst šim risinājumam. Pēc tam atliek izveidot pašu robotu, kas būtu spējīgs izpildīt izveidoto algoritmu un programmu ,kas to pilda.

11. Elektronikas un rasēšanas pamati

Elektronika programmēšanai ir viena no svarīgam disciplīnām. Ar tās palīdzību ir iespējams uzzināt kādā veida darbojas robotas, ka tiek padotas komandas saprast kā darbojas datorshēmas un pašiem iemācīties šīs shēmas veidot. Katram robotam ir sava specifiskā konstrukcija.

Viens no robotu piemēriem, kuru var izmantot, lai iemācīties elektronisku ir SumoBoy. Šis robots tiek ražots Latvijā. Šī robota pamatā ir platforma Arduino. Papildus robotam ir prototipēšanas komplekts, ar kura palīdzību var mēģināt iepazīties ar elektronikas pamatiem un veidot sava shēmas.

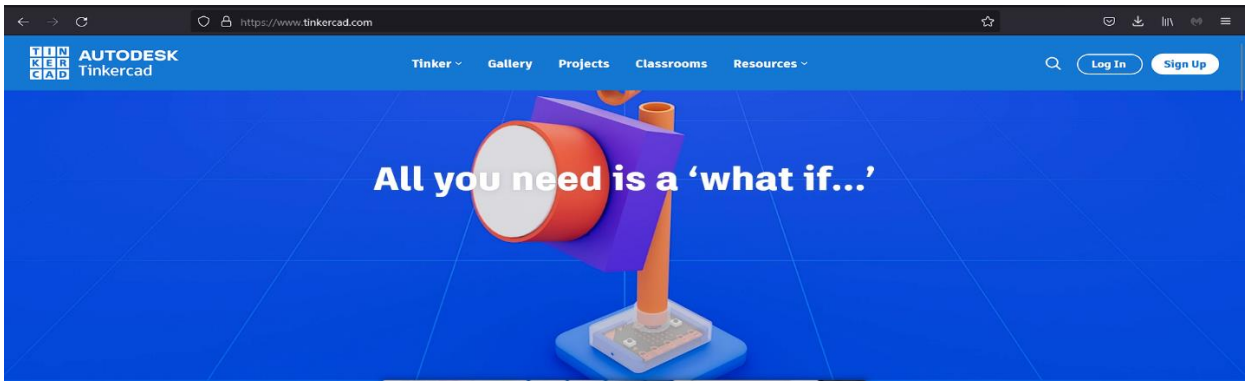
Vairāk par šo elektrotehnikas pamatiem, SumoBoy robotu, tā darbības principu un uzbūvi Jūs varat iepazīties speciāli izgatavota mācību materiālā, kas ir pieejams SumoBoy ražotāja mājas lapā, sadaļā "Lejupielādes": http://robot-nest.com/lv#lejupielades

11.1. Reģistrācija 3D modelēšanas rīkā Tinkercad

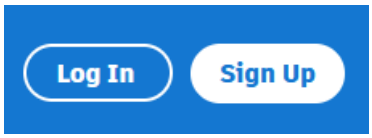
TinkerCad ir tiešsaistes 3D modelēšanas rīks, ko ir iespējams izmantot uz jebkura datora vai planšetdatora. *Tinkercad* rīkam nav nepieciešama instalācija, līdz ar to ir vienkārši izmantot. Lietotnes pamatā tiek izmantots *"Drag and drop"* (paņemt un novietot) princips, 3D modeļi tiek veidoti jau no gatavām ģeometriskām figūrām, kuras tiek pārveidotas un apvienotas. Rīks ir vienkāršs un saprotams arī bez 3D modelēšanas priekšzināšanām, tāpēc ir lieliski piemērots bērniem, jo pamatā rīks arī veidots bērniem.

Reģistrācijas soļi

Reģistrāciju iespējams veikt vietnē: www.tinkercad.com, vietnē nepieciešams izmantot lietotāju *skolotājs*, jo skolēniem ir iespēja veidot projektus izmantojot lietotāju *skolotājs*. Lietotājs *skolotājs* dos iespēju apskatīt skolēnu projektus vienuviet, kā arī dos iespēju skolotājam tos labot kopā ar skolēnu.

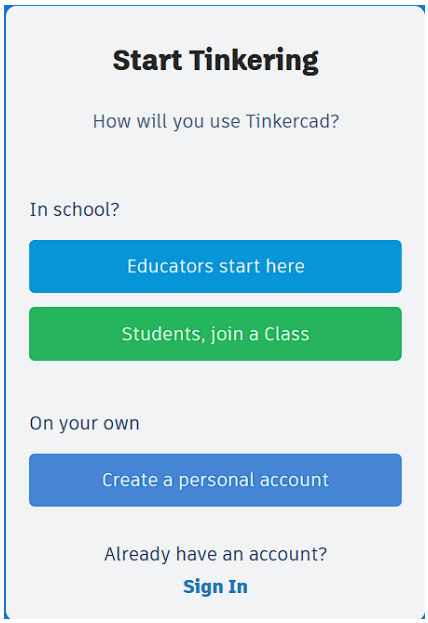


1. Solis.



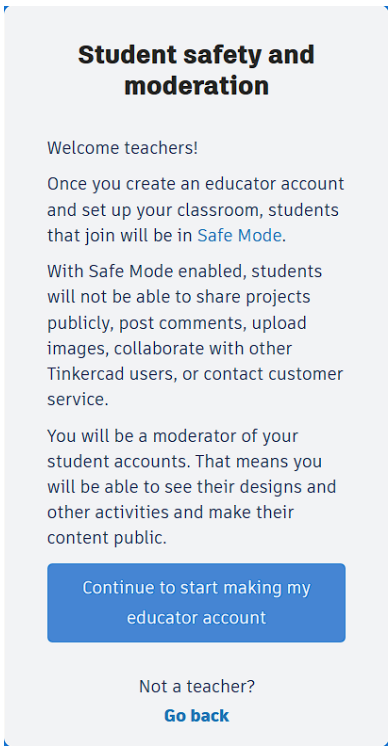
Skolotāja lietotāja izveidei vietnē jāizvēlas ***Sign Up*** poga

2. Solis.



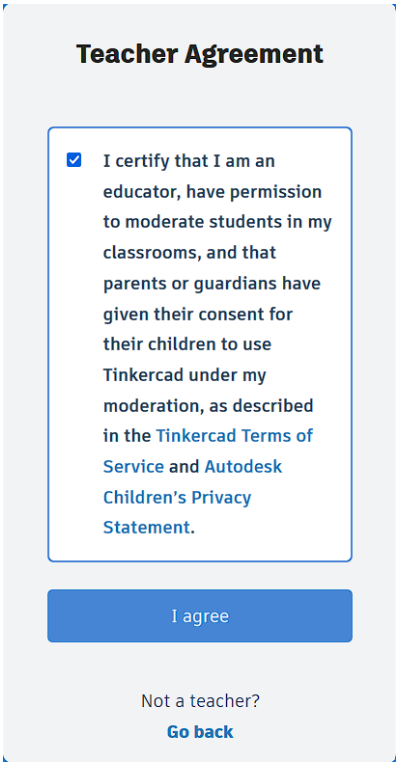
Lai tiktu izveidots lietotājs *skolotājs*, izvēlamies *"**Educators start here**"*

3. Solis.



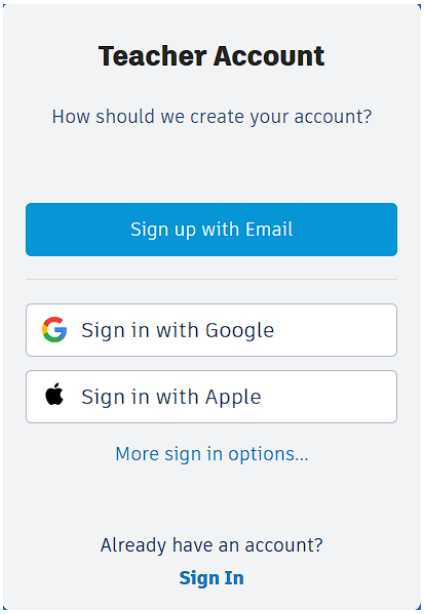
Šajā logā, ir informācija, ka izmantojot *Safe Mode* (Drošo režīmu), skolēni nevarēs publicēt un koplietot savus projektus, šīs funkcijas varēs veikt skolotājs. Lai turpinātu izveidot skolotāja lietotāju izvēlamies *"**Continue to start making my educator account**"*.

4. Solis.



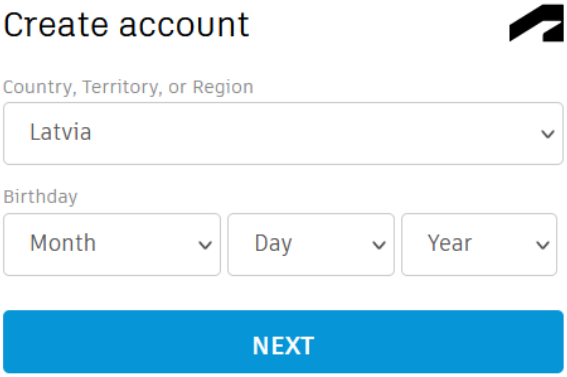
Šajā logā nepieciešams apstiprināt, ka esat skolotājs un Jums ir tiesības apmācīt bērnus. Lai šo apstiprinātu, ieliekam "ķeksīti" un spiežam pogu *"**I agree**"*.

5. Solis



Reģistrācijas turpināšanai izvēlamies pogu ***"Sign up with Email"***

6. Solis

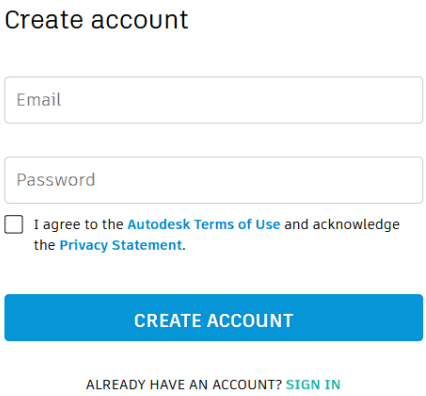


Izvēlamies valsti;

Ievadam dzimšanas datus (Mēnesis, diena, gads);

Spiežam pogu *"Next"*, lai nokļūtu reģistrācijas nākamajā solī.

7. Solis



Ievadam reģistrācijai izvēlēto e-pasta adresi.

Izveidojam paroli (Vismaz 1 burts, 1 cipars, paroles garums vismaz 8 simboli).

Atzīmējam, ka piekrītam noteikumiem.

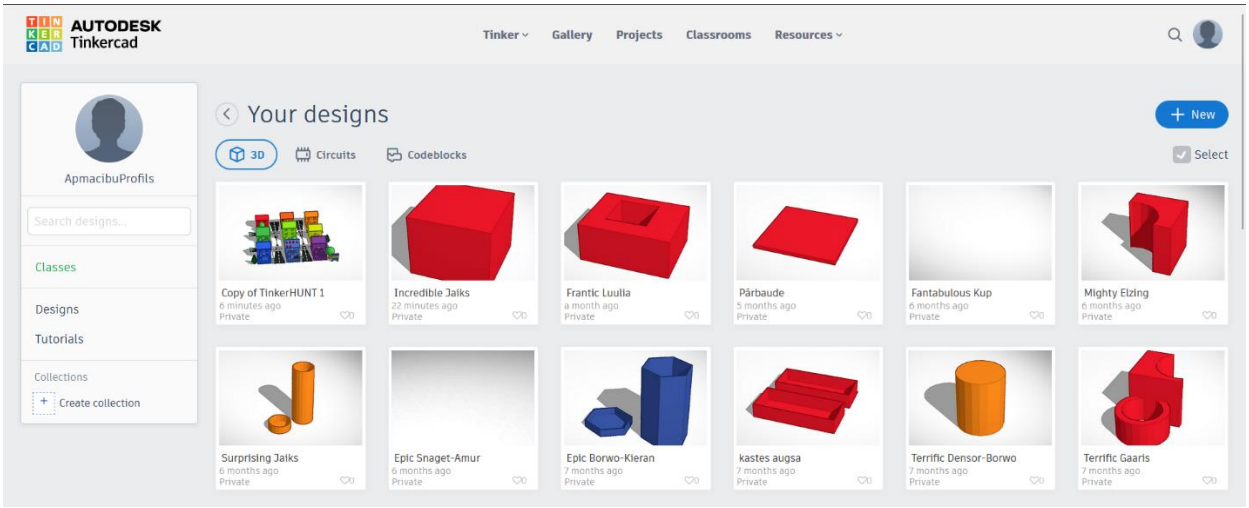
Spiežam pogu *"Create account"*, lai izveidotu lietotāju.

8. Solis

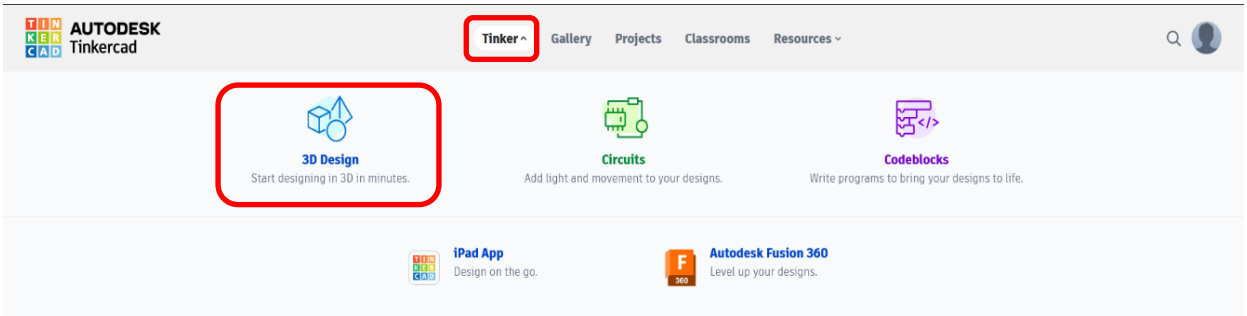
Pēc lietotāja izveides, spiediet pogu ***"Log In"***, lai pieslēgtos, ar izveidoto lietotāju, izmantojot iepriekš norādīto e-pastu un paroli.

11.2. Tinkercad vide

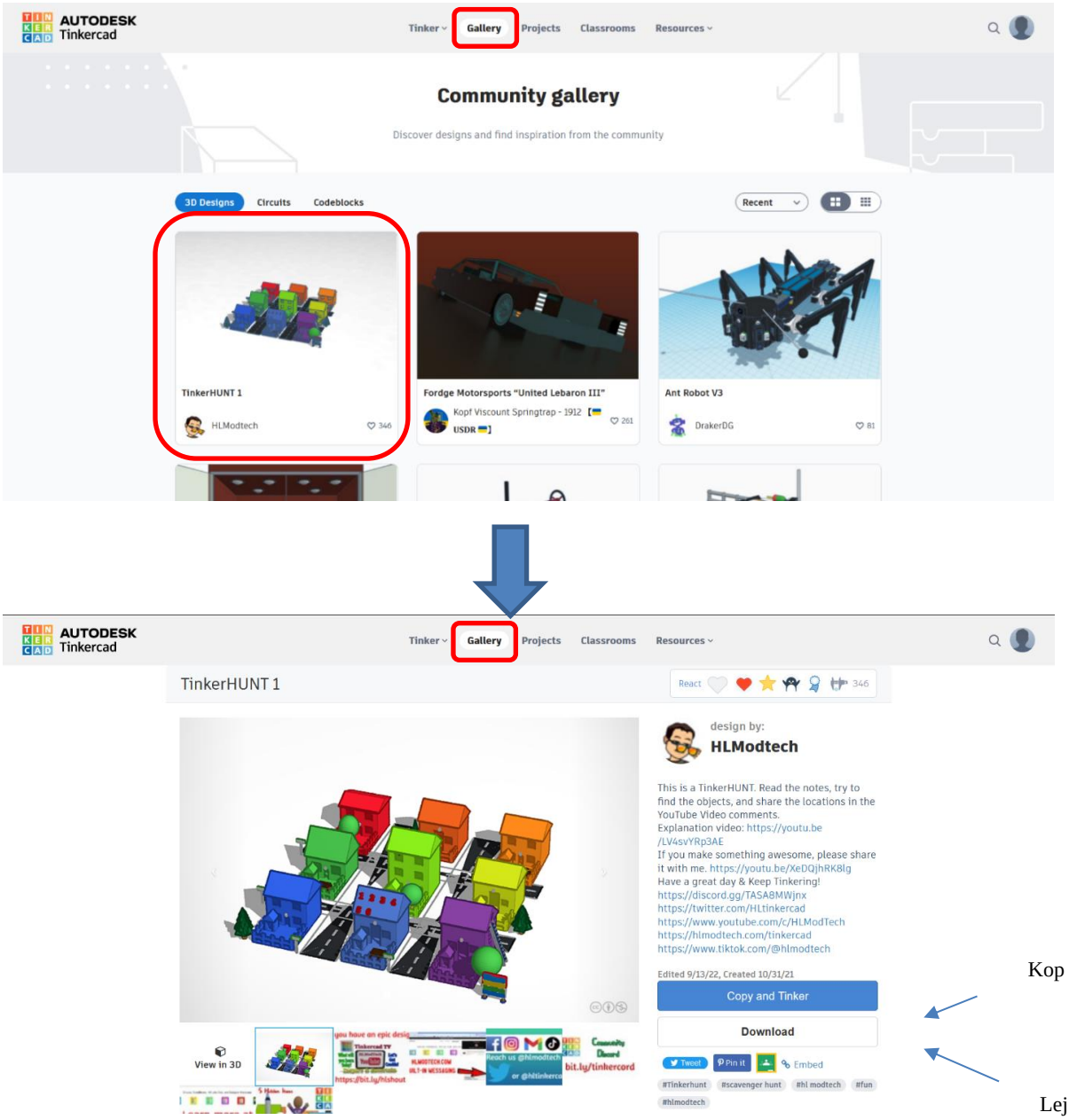
1. *Tinkercad* sākumā ekrānā redzams Jūsu lietotājvārds, galvenā izvēlne, klašu pārvaldība, projekti



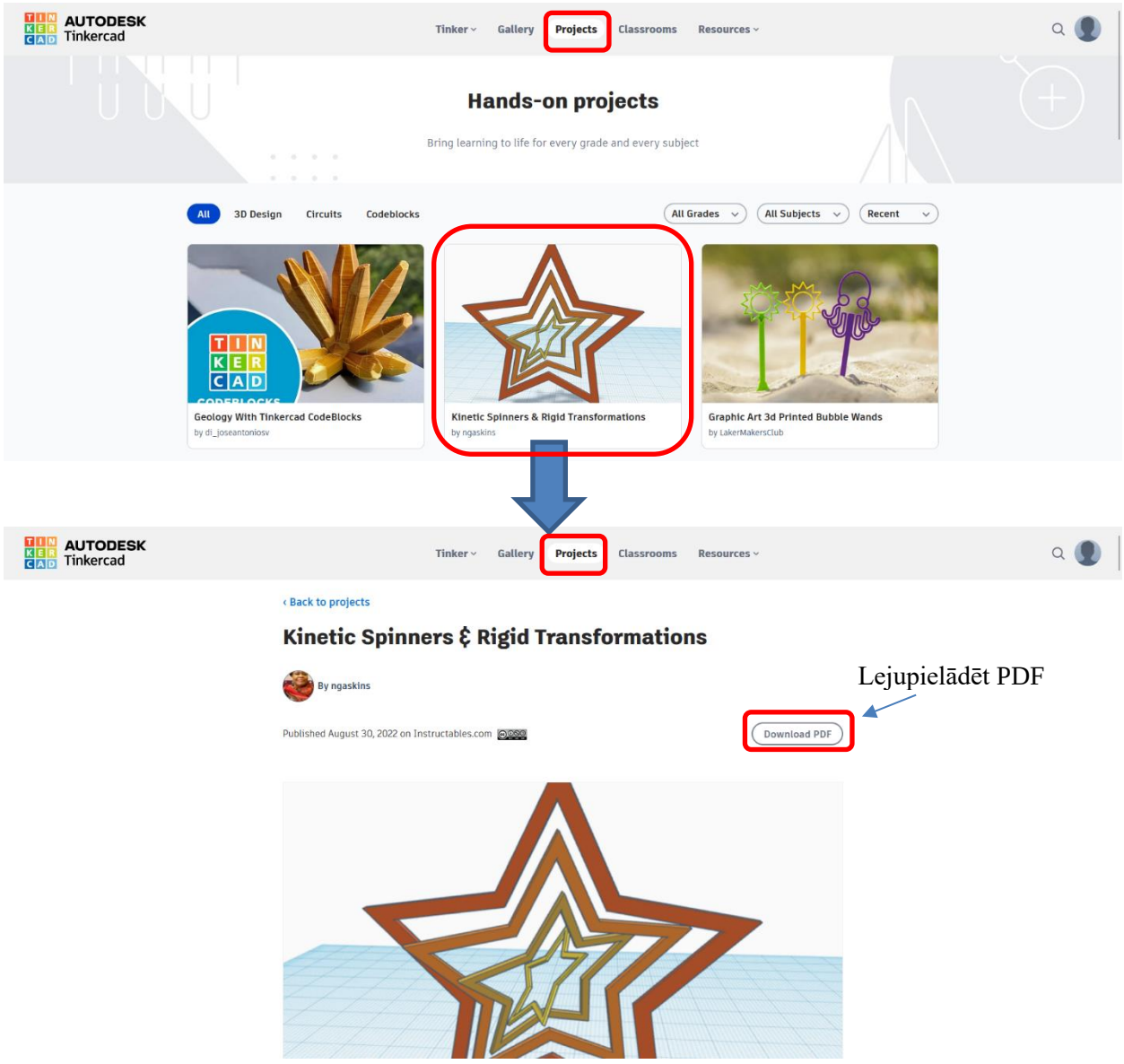
2. Sadaļā *Tinker*, Jums ir iespēja izvēlēties kādu projektu vēlaties veidot. Pieejamie projektu veidi: 3D modelis, elektro slēgumu shēmu veidošana, Bloku programmēšana, iPad aplikācija, Autodesk Fusion 360. 3D modeļu izveidei izvēlamies 3D Design



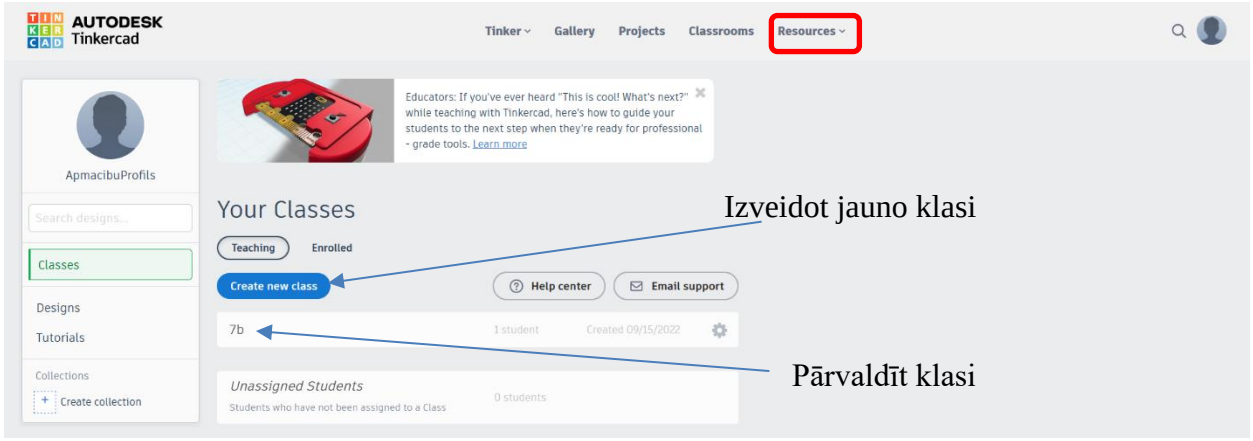
3. Sadaļā *Gallery*, jūs varat atrast citu lietotāju veidotus 3D modeļus, modeļus ir iespējams lejupielādēt 3D drukai, kā arī daļu no modeļiem var nokopēt un rediģēt. Noklikšķinot uz izvēlēto modeli galerijā Jums parādīsies iespēja to lejupielādēt un/vai nokopēt un labot.



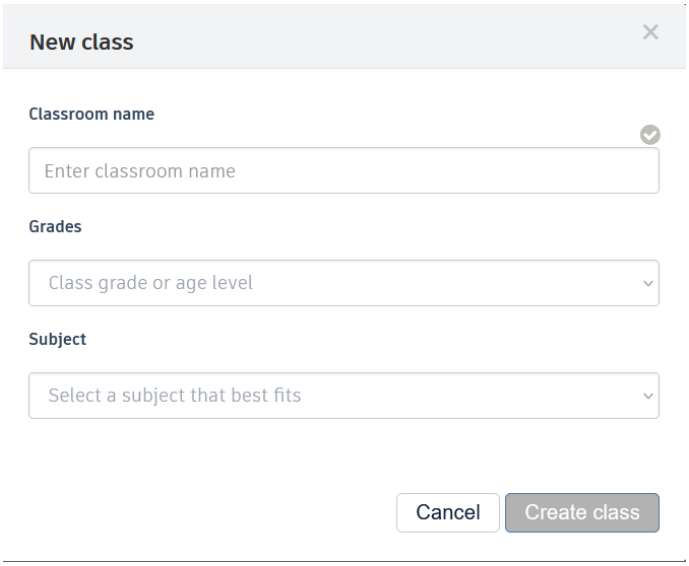
4. Sadaļā *Projects* Jums ir iespēja izvēlēties dažādus lietotāju projektus, ar aprakstu kā šo projektu izveidot. Projekta aprakstu iespējams arī lejupielādēt PDF formātā. Šo sadaļu var izmantot, lai smeltos idejas skolēnu uzdevumiem.



5. Sadaļā *Classroom* Jums ir iespēja veidot, pārvaldīt klases, pievienot skolēnus. Redzēt visus skolēnu darbus, kuri ir pieslēgti izmantojot Jūsu lietotāju.



Jaunas klases izveides logs



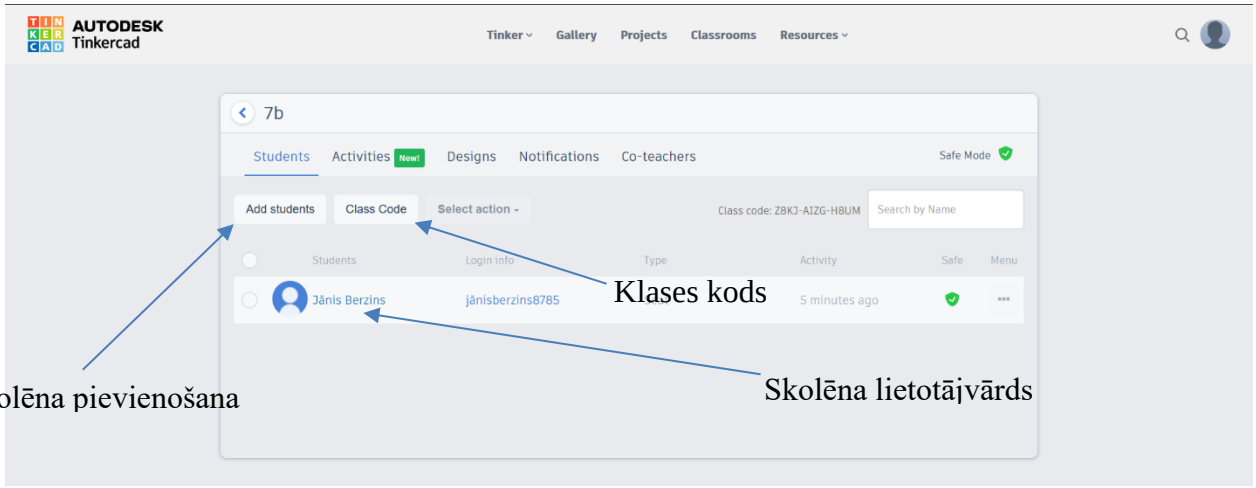
1)Klases nosaukums

2)Vecuma grupa

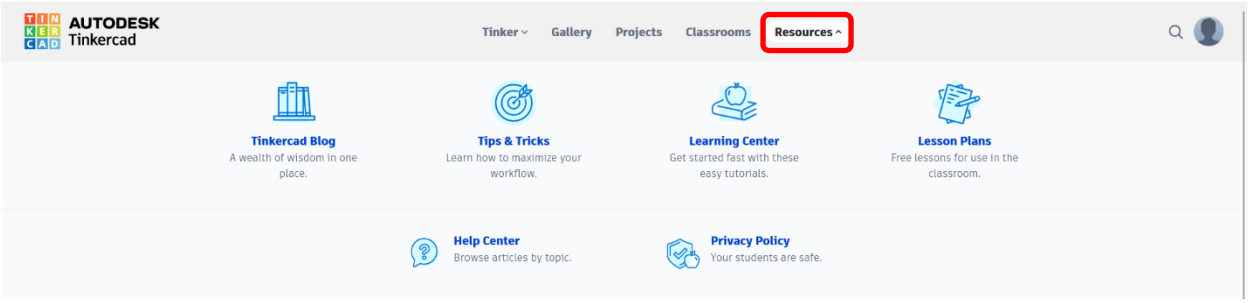
3)Mācību priekšmets

4)Nospiest pogu *Create Class*, lai izveidotu klasi

Skolēnu pievienošana. Nosūtot skolēnam klases kodu, tas varēs pievienoties konkrētajai klasei ar skolotāja izveidotu lietotāja vārdu, kurš redzams sadaļā *Login info*

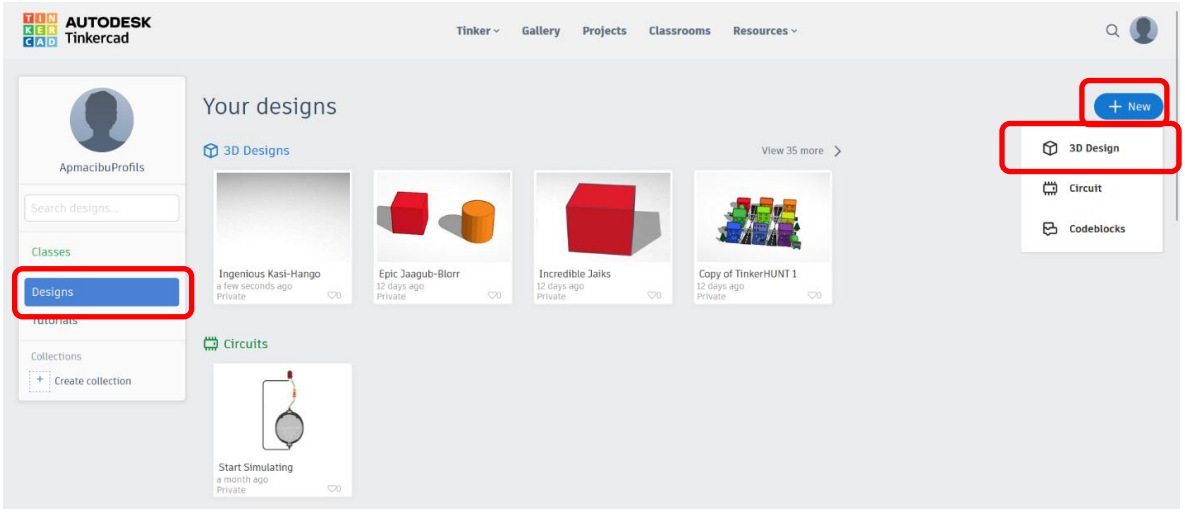


6. Sadaļā *Resources* ir pieejams, Tinkercad blogs, ieteikumi, Apmācību centrs, nodarbību plāni, palīdzības centrs, Privātuma politika.

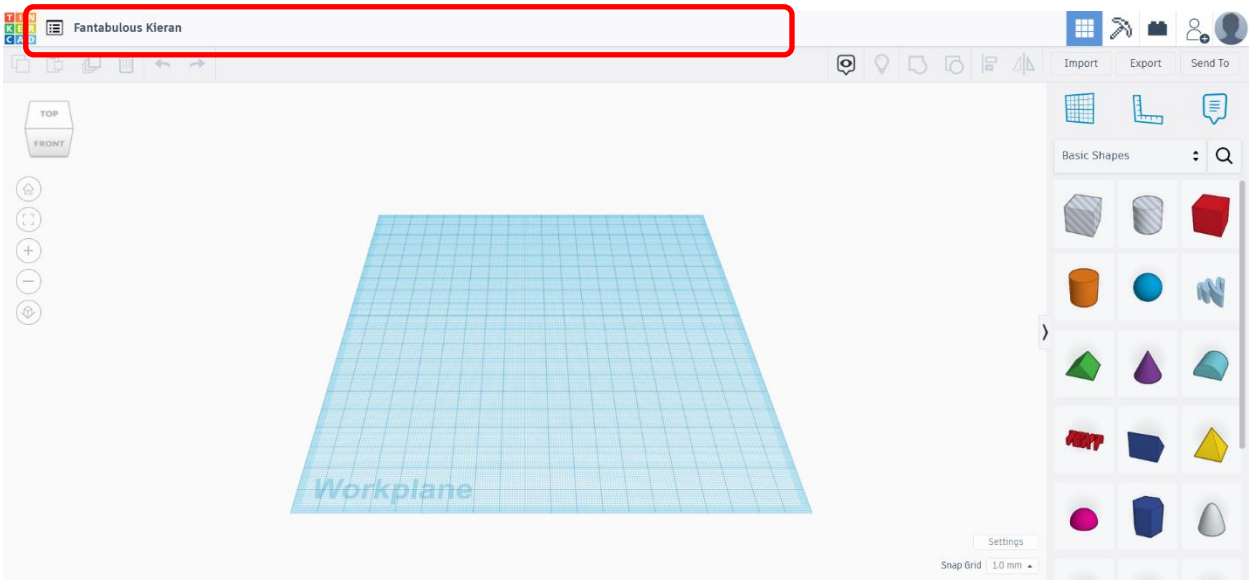


11.3. Tinkercad 3D modelēšanas vide

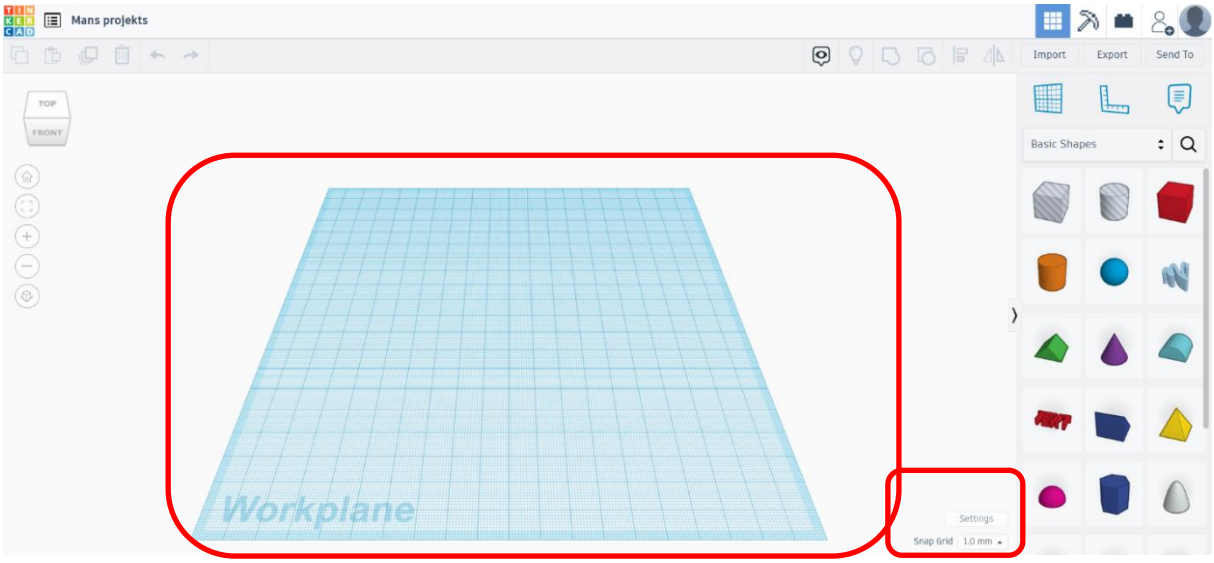
Lai sāktu veidot 3D modeli, savā lietotāja profilā izvēlamies sadaļu *Designs – New – 3D models*



Tincercad rīks, Jūsu izveidotajam modelim automātiski izveidos nosaukumu. Nosaukums redzams rīka augšējā kreisajā stūrī, lai nomainītu nosaukumu nepieciešams noklikšķināt redzamajā logā ar peles kreiso taustiņu. Lai vieglāk atpazītu skolēnu darbus, vēlams skolēnam nosaukumā norādīt savu vārdu un uzvārdu.



Laukuma vidū redzama zila plakne, kuras izmērs ir 20x20cm, plaknes laukumā ir režģa līnijas, kuras ir 1 mm attālumā viena no otras, lai vieglāk būtu redzēt izveidotā objekta izmērus. ***Snap Grid*** sadaļā iespējams mainīt režģa soļa izmēru, ja būs izvēlēts 1 mm, tad, piemēram, pārvietojot objektu tas pārvietosies par 1 mm. Sadaļā ***Settings*** jūs varat mainīt plaknes parametrus un izmērus.



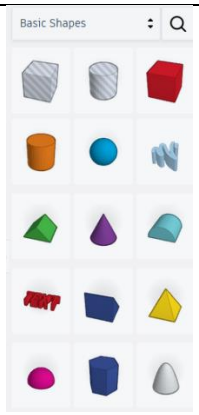
Darbības pogas Tinkercad vidē

Kopēt	Ielīmēt	Dublēt	Dzēst	Atcelt iepriekšējo darbību

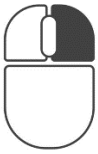
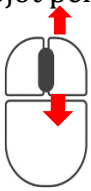
Parādīt/slēpt piezīmes	Parādīt visu	Pielīdzināt	Spoguļattēls	Grupēt/Atgrupēt objektus

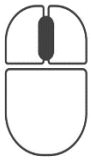
Import	Export	Send To
Importēt objektu	Eksportēt objektu	Nosūtīt objektu

Plaknes rīks	Lineāls	Piezīmju rīks

	Gatavo formu izvēlne

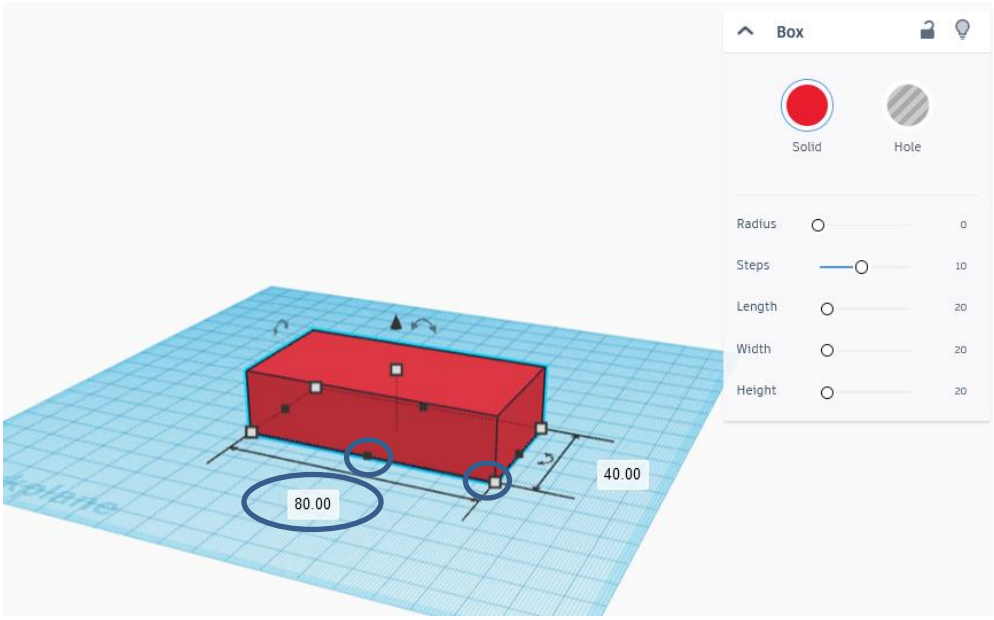
Plaknes navigācija, ērtākai rīka lietošanai, skolēnus nepieciešams nodrošināt ar datorpeli, kas ļaus precīzāk darboties *Tinkercad* rīka vidē.

Plaknes skata maiņa un rotācija	Izmantojot kuba ikonu ekrāna kreisajā augšējā stūrī  Vai Turot nospiestu peles labo taustiņu un kustinot peli nepieciešamajā virzienā 
Pietuvināšana un attālināšana	Izmantojot “+” un “-” ikonas rīka vidē  Vai Izmantojot peles rullīti 

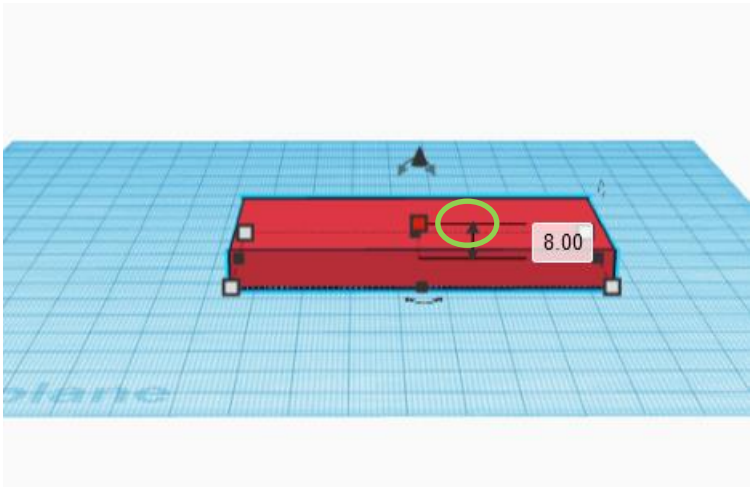
Plaknes pārvietošana	Turot peles rullīti 
Objekta/Objektu iezīmēšana	Piespiežot un turot peles kreiso taustiņu 

1.uzdevums. Personalizēts piekariņš.

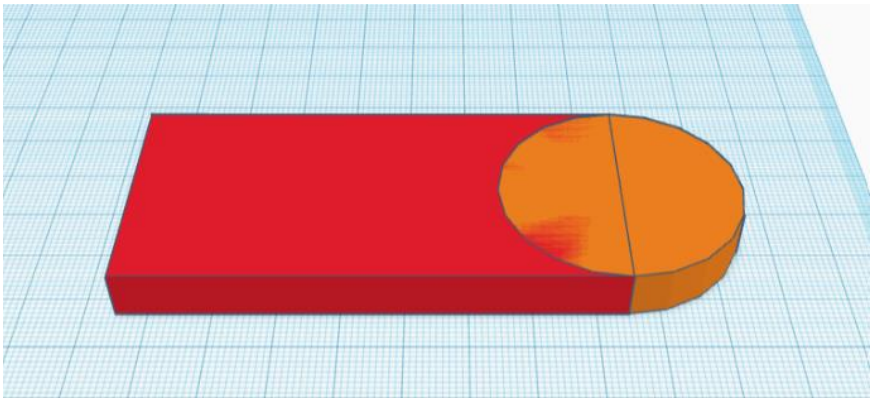
Lai izveidotu piekariņa pamatni, tiek izmantota kuba forma, kuba formu mēs varam novietot plaknē turot peles kreiso peles taustiņu un velkot to uz plakni, tā jānovieto uz plaknes un jānomaina izmērs, lai to izdarītu, spiediet uz formas, lai tā kļūst aktīva un attiecīgi formas katrā malā ir pieejami balti lodziņi, ar kuru palīdzību iespējams mainīt formas izmēru platumu un garumu, vēl lai mainītu platumu un garumu var izmantot arī mērvienības lodziņu, ierakstot nepieciešamo izmēru. Lieciet skolēniem nomainīt garumu uz 80 mm un platumu uz 40 mm. Izmēri norādīti mm. Izvēloties balto kvadrāta atzīmi stūros jūs varat mainīt garumu un platumu, izvēloties katras malas vidū novietoto melno kvadrātu, tiks mainīti tikai konkrētās malas izmēri.



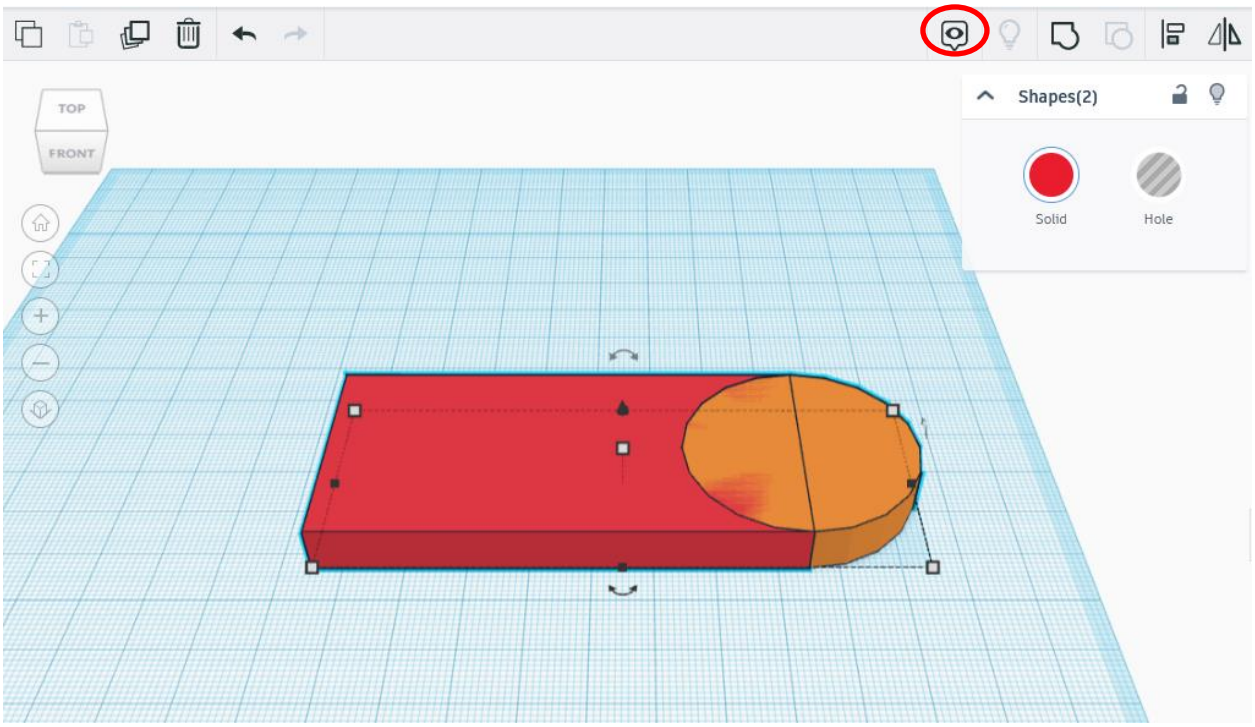
Lai mainītu modeļa augstumu, Jums to nepieciešams pagriezt paralēli skatam, lai redzama Z ass un ir iespējams nomainīt augstumu uz 8 mm.



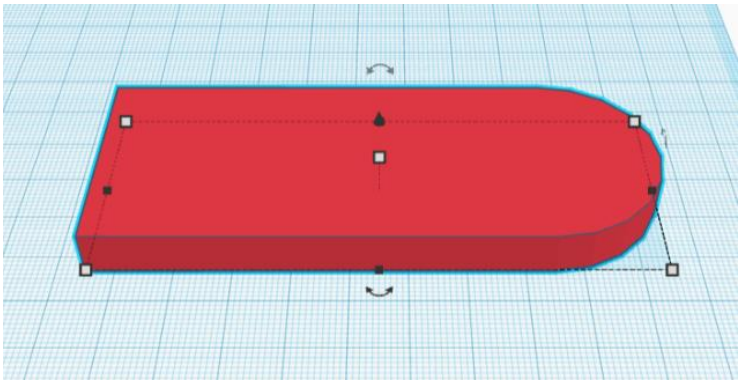
Lai apgūtu grupēšanas iespēju un novietošanas precizitāti, skolēnam nepieciešams pievienot cilindra formas objektu ar diametru 40 mm un augstumu 8 mm, kā tas attēlots attēlā.



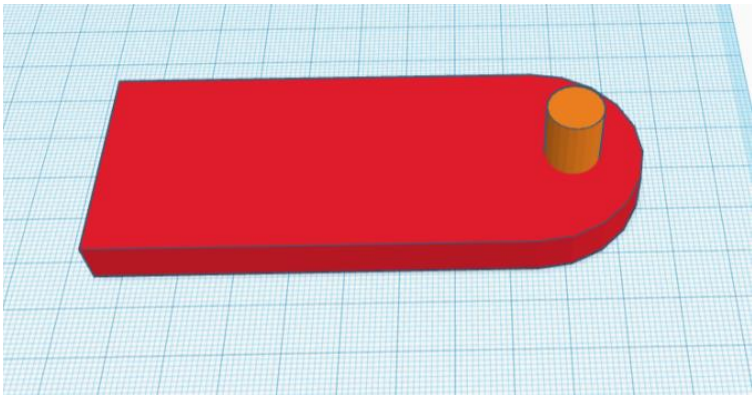
Lai mēs šos abus objektus pārveidotu vienā objektā mums tas ir jāsagrupē. Sagrupēt mēs varam iezīmējot abus objektus turot nospiestu kreiso peles taustiņu. Kad esam iezīmējuši, izvēlamies grupēšanas darbības pogu.



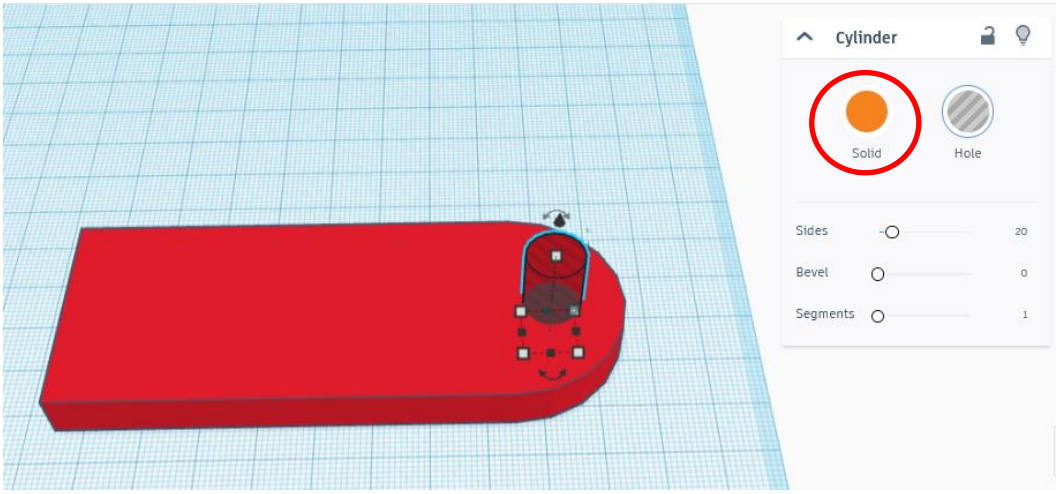
Kad tas ir paveikt mēs iegūstam divu objektu vietā vienu



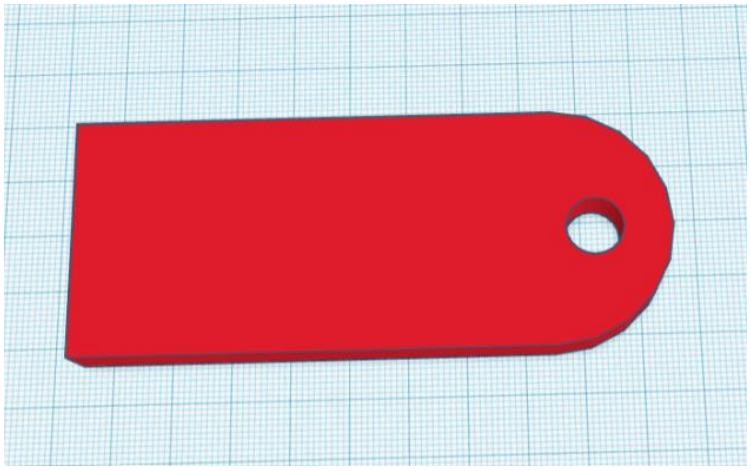
Lai mēs iegūtu piekariņam caurumu mums ir jāizvēlas cilindra forma, cilindra forma jānovieto vietā, kur paredzēts caurums. Cilindra diametru izveidojam 10 mm augstumam šoreiz nav nozīmes.



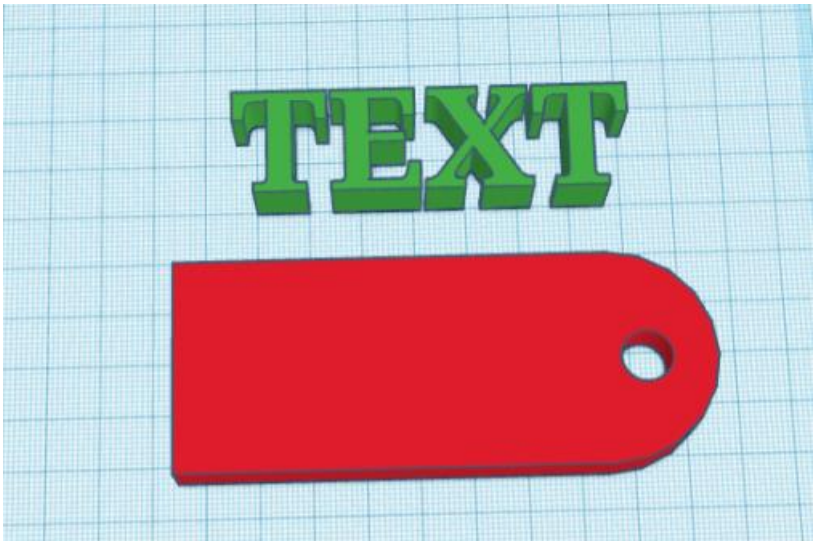
Izvēlamies cilindru un izveidojam to kā caurspīdīgu objektu nomainot krāsu uz ***Hole***



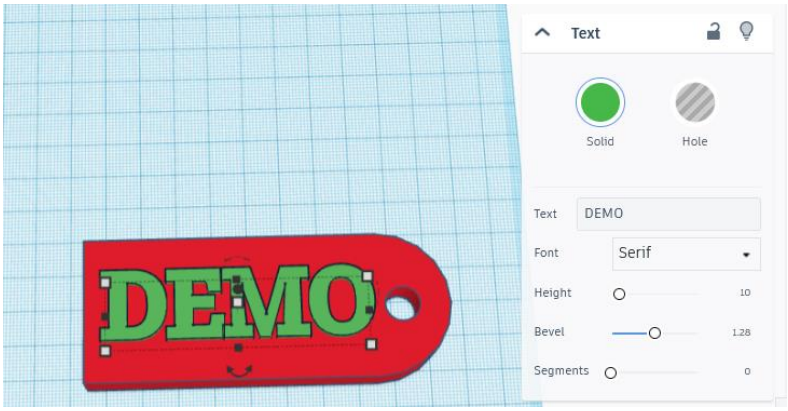
Kad esam izveidojuši cilindru kā caurspīdīgu objektu, iezīmējam visus objektus un tos sagrupējam un mēs iegūsim atveri atslēgu piekariņam.



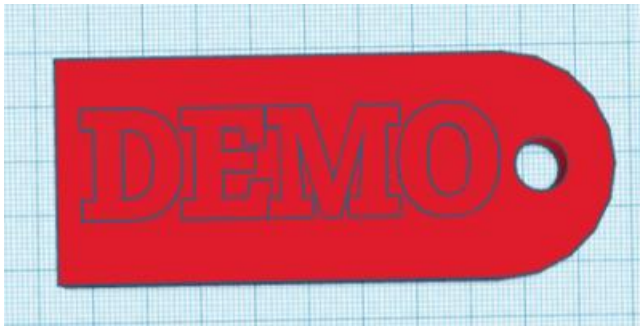
Lai ievietotu tekstu, gatavo formu izvēles joslā izvēlamies ***Text*** formu. ***Text*** formu tāpat, kā jebkuru citu formu darba plaknē varam dabūt, turot peles kreiso taustiņu un ievilkt to darba plaknē.



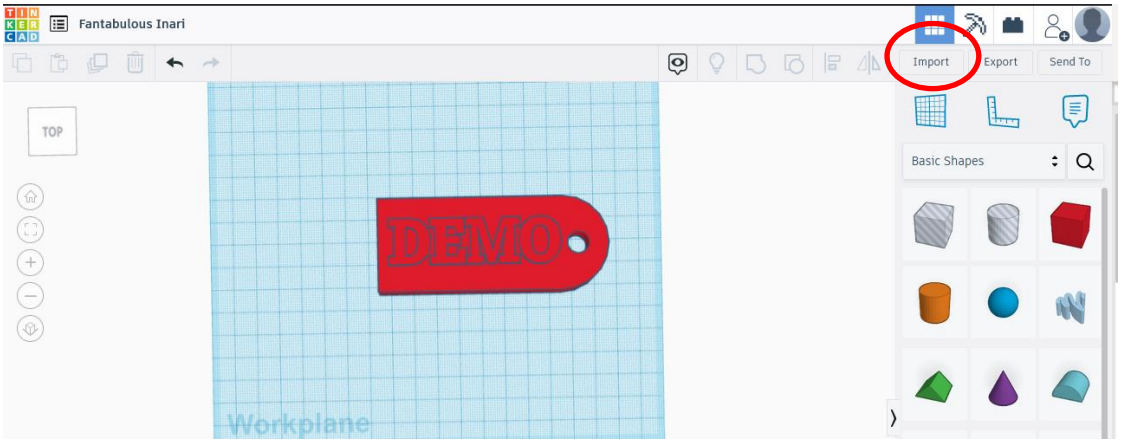
Lai labotu un formatētu tekstu nepieciešams uzklikšķināt uz novietotās teksta formas ar kreiso peles taustiņu. Formatēšanas logā, ***Text*** laukā iespējams ievadīt savu tekstu, ***Font*** laukā iespējams mainīt teksta stilu, ***Height*** laukā teksta augstumu, ***Bevel*** laukā teksta savienošanu, ***Segment*** laukā iespējams veidot tekstam reljefu.



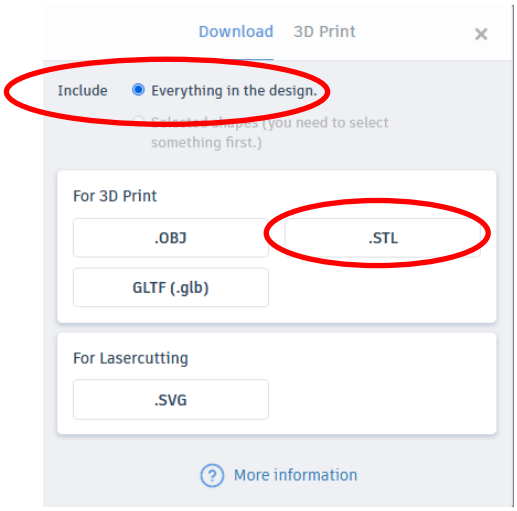
Kad esam ievadījuši un noformatējuši savu teksta formu, novietojam to jau uz esošās piekariņa formas sev vēlamā izmērā un vietā. Pēc novietošanas, iezīmējam visus objektus un tos sagrupējam. Kad esam sagrupējuši mēs iegūstam gatavu modeli, piekariņu ar izvēlēto vārdu.



Jūsu modelis automātiski tiks saglabāts Jūsu profilā un būs redzams sākuma ekrānā. Lai 3D modeli Jūs varētu izdrukāt uz 3D printera, tas Jums ir jālejupielādē. Lai to izdarītu ejat uz sadaļu ***Export.***



Izvēlaties ***Everything*** in ***the design*** un tad izvēlaties faila formātu ***stl.*** Pēc izvēles Jūsu 3D modelis tiks lejupielādēts Jūsu datorā.



2. uzdevums.

2.1. Prast strādāt ar ģeometriskām formām un to parametriem. Uzdevumi: izveidot trīs dažādus objektus no kuba formas, izveidot sfēru, kas 1 ir sarkana un 1 zila, sadalīt formu četrās vienādās daļās, katru citā krāsā.

2.2. Attēlot un interpretēt datus. Uzdevums: dot skolēniem reālu priekšmetu, veikt tā mērījumus un pēc tam izveidot tā 3D prototipu *Tinkercad* programmā.

2.3. Izstrādāt 3D prototipus. Uzdevums: izstrādāt 3D prototipu skolas suvenīram, interjera dizaina elementu vai 3D prototipu konkrētas problēmas risināšanai.

2.4. Ģeometrisku formu mērīšana. Uzdevums: izprast leņķa un leņķa mērīšanas jēdzienus, izmantot rotāciju un noteikt, piemēram, konkrēto objektu var pagriezt pa x grādiem.

2.5. Leņķis un rotācija. Uzdevums: izveidot leņķu 3D vizualizāciju, kuru skolotājs var arī izdrukāt ar 3D printeri, lai izmantotu kā reprezentatīvu materiālu.

2.6. Mērvienības. Uzdevums: izveidot savas mērierīces.

3. uzdevums

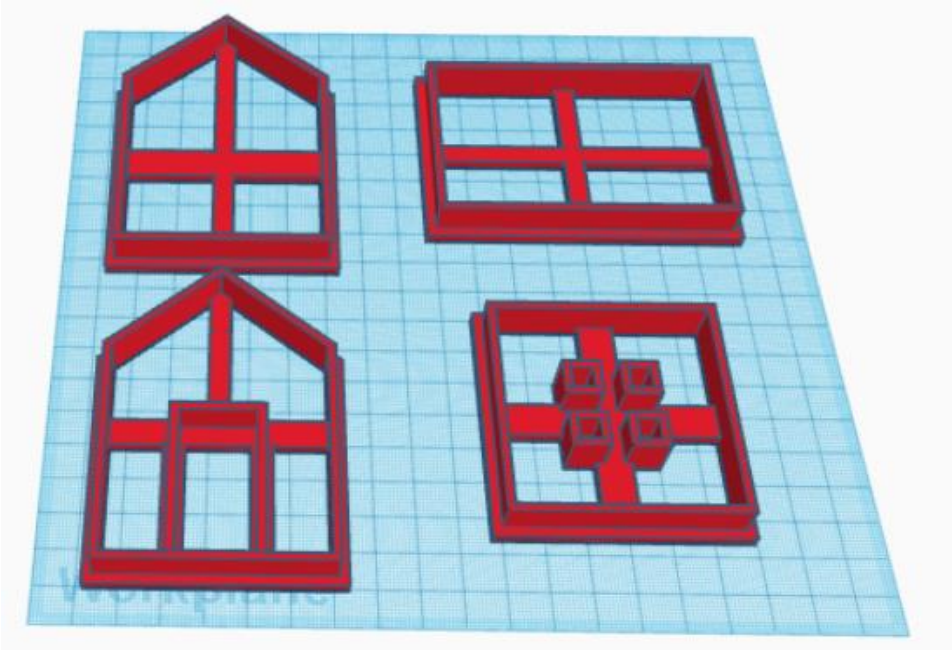
Izveidot piparkūku formas maketu, piparkūku namiņa pagatavošanai.

3.1. Formas maketam jāsastāv vismaz no 3 daļām. (Jumta daļa, gala siena, sānu sienas),

3.2. Formas maketā ir jābūt iekļautām vismaz 1 durvīm un 2 logiem,

3.3. Formas augstums nedrīkst pārsniegt 1 cm,

3.4. Visām formas daļām jāietilpst *Tinkercad* laukumā, kura izmērs ir 20x20 cm.



12. Lego Robotikas Sacensību disciplīnas un to noteikumi

Šajā sadaļā tiks aplūkoti robotikas sacensību noteikumi *Lego* disciplīnām. Šī nodaļa tiks balstīta uz robotuskola.lv pieejamajiem noteikumiem. Taču tik un tā jāpiebilst, ka pirms dalības kādās no robotikas sacensībām, ir svarīgi izskatīt noteikumus, jo dažkārt var gadīties, ka ir kādi sīkumi, kas ir pamainīti. Kā arī, šeit tiks aplūkotas populārākās disciplīnas. Tas nozīmē, ka var būt sacensības, kurās tiek piedāvātas vēl citas disciplīnas.

Svarīgi piebilst, ka šeit tiek runāts tikai par *Lego* disciplīnām. Lielākoties sacensībās bez *Lego* ir arī disciplīnas Brīvās konstrukcijas robotiem. Pēc nosaukuma, *Lego* disciplīnās ir stingri noteikts, ka robotam ir jābūt tikai un vienīgi no licenzētām *LEGO® original* vai *HiTechnic®* daļām, kamēr brīvās konstrukcijas robotiem nav noteikumu, kas ierobežotu, kādas detaļas drīkst un kādas nedrīkst lietot. Tagad tiks aplūkotas disciplīnas, kuras sacensībās ir vispopulārākās: *Folkrace, Līnījsekošana un Sumo.*

12.1. Līnījsekošana

Jau pēc disciplīnas nosaukuma var saprast, kas šajā disciplīnā robotam ir jādara. Jāseko līnijai un uzvar tas, kura robots trasi spēj uzvarēt pēc iespējas ātrāk. Runājot par noteikumiem, viss ir īsi un korekti. Lielākā uzmanība ir jāpievērš divām lietām: kādi trases elementi var sagaidīt spēlētāju un cik mēģinājumi ir pieejami katram dalībniekam.

Sākumā var aplūkot pirmo, kādi trases elementi var sagaidīt. Pirmais un pats svarīgākais ir tas, ka trases segums ir baltā krāsā un pati līnija ir melnā krāsā. Lielumi, kā līnijas biezums (15mm) un attālums starp līnijām (25 cm) parasti nemainās starp sacensībām. Tas, kas var mainīties ir kādas formas var veidot līnija, kurai ir jāseko. Tā var būt vienkārši apaļa forma vai arī krietni sarežģītāka, līkumi un krustojumi. Standarta trasē var būt līdz vienam 90 grādu krustojumam. Un trase var ietvert 90 grādu pagriezienu. Šīs būtu svarīgākās lietas, kuras jāpārbauda, vai robots spēj pārvarēt tās. Šeit var pieminēt robota izmērus, taču lielākoties sacensībās atļauj piedalīties robotiem, kuru masa nepārsniedz 1 kg un robota garums, platums un augstums nepārsniedz 25cm. Taču šos izmērus nepieciešams pārbaudīt pirms dalības sacensībās, jo tie var atšķirties kādās no sacensībām.

Otra lieta, kam nepieciešams pievērst lielu uzmanību ir atļauto braucienu skaits. Ir sacensības, kurās ir ierobežots skaits, cik braucienus ir iespējams veikt un ir sacensības, kur nav noteikts maksimālais mēģinājumu skaits.

Atziņas

Varētu rasties jautājums par pašu robotu. Kas būtu svarīgi robotam, kā to konstruēt un cik sensorus būtu nepieciešams lietot. Vislabākais ieteikums ir mēģināt dažādas kombinācijas

un eksperimentēt. Cits ieteikums varētu būt skatīties uz sacensību dalībnieku robotiem, kuri tiešām brauc ļoti labi. Taču tā ir tikai viena daļa. Otra daļa ir robota programma un algoritms, tikai tāpēc, ka robots spēj sekot līnijai, nenozīmē, ka robots spēj ātri sekot līnijai. Apvienojot šo, vislabākais ieteikums robota izstrādē ir eksperimentēt un nebaidīties kļūdīties, ja ir kādas idejas, kas varētu palīdzēt robotam braukt ātrāk, aplūkot tās un izmēģināt.
Iespējams var gadīties situācija, kad nav pieejama trase uz kuras testēt robotus. Šeit var nākt talkā melnā izolācijas lenta. Ar to izveidot visu trasi vai vismaz dažādus trases posmus un tādā veidā testēt robotu. Ideālāk, ja līniju var izveidot uz balta seguma, taču testēšanai var izmantot citas krāsas segumu, ja tas netraucē sensoru darbībai. Pirms izveidot melnu līniju uz virsmas, kas nav balta, noteikti ir ieteicams aplūkot, vai sensors joprojām spēj uztvert līniju.

12.2. Sumo

Iespējams vispopulārākā disciplīna tieši *Lego* kategorijā ir *Lego Sumo*. Disciplīna, kurā uz laukuma atrodas divi roboti un zaudē robots, kurš pirmais tiek nogrūsts no laukuma. Runājot par noteikumiem, šeit nav daudz ko minēt.

Noteikumi

Sacensību laukums jeb rings, ir apaļa forma, kura ir melnā krāsa, izņemt ap malām, kur ir balta līnija. Īpaša uzmanība ir jāpievērš tieši robota palaišanai sacensībās. Dalībnieki, kuriem ir jācīnās, apstājās katrs savā pusē ringam un pēc tiesneša komandas roboti ir jānovieto uz laukuma. Pārsvarā prasība robota novietošanai ir, lai robots pieskartos šai baltajai līnijai, kas atrodas ringa apkārtnē. Šī prasība var mainīties dažādās sacensībās. Pēc tam, kad roboti ir novietoti, tiesnesis dod komandu, lai robotus palaistu. Pēc noteikumiem, robotu palaižot, robotā ir jābūt ieprogrammētām 5 sekundēm, kuras gaida un tikai tad uzsāk kustību. Šī noteikuma pamatojums varētu būt tāds, lai paši dalībnieki paspētu pārvietoties prom no ringa un robotu sensori tos neuztvertu. Kā vēl viena svarīga piebilde, pēc tam, kad robots ir novietots uz lauku, to ir aizliegts kustināt. Atkarībā no sacensību noteikumiem, tiesnesis var likt pacelt robotus vai automātiski piešķirt zaudējumu.

Mēdz būt situācijas, kad kādam no robotiem noplīst kāda no detaļām un izkrīt ārpus laukuma. Lai gan šeit mēdz atšķirties noteiktumi attiecībā uz šo, atkarībā no nokritušās detaļas izmēra tiek pieņemts lēmums. Ja detaļa ir pietiekami liela, tiesnesis to var ieskatīt kā zaudējumu. Kā arī, ja šī nokritusī detaļa kaut kādā veidā ietekmēja pretinieka robotu un lika tam izbraukt laukā, arī šādā gadījumā tiesnesis var piešķirt zaudējumu.

Var pieminēt arī sacensību formu. Tā noteikti var mainīties starp dažādām sacensībām, lai gan pārsvarā, visus dalībniekus sadala vairākās grupās, kur katrs cīnās ar katru. Un pēc tam, labākie no grupas tiek tālāk uz izslēgšanas cīņām. Jāpiebilst, ka katrā cīņā, dalībnieki viens ar

otru cīnās, līdz pirmais uzvar otru 2 reizes. Var arī būt situācijas, kad roboti saķeras un nespēj viens otru izgrūst. Tad tiesnesis var likt izslēgt robotus un pārspēlēt attiecīgo cīņu.

Šeit ļoti svarīgi būtu pievērst uzmanību robota izmēriem. Pārsvarā robotam ir atļauts būt 20 cm platumā un 20 cm garumā un 1 kg svarā, taču mēdz būt sacensības, kur 20cm vietā ir tik 15. Tam jāpievērš uzmanība sacensību noteikumos.

Atziņas

Runājot par robota uzbūvi, šeit var mēģināt izveidot sarežģītu programmu, kas ievieš kaut kādas stratēģijas, taču pietiek ar programmu, kas liek robotam griezties, līdz tas pamana objektu un brauc uz priekšu. Šeit ļoti lietu lomu spēlē tieši uzbūve. Līdzīgi, kā iepriekš, vislabāk ir eksperimentēt ar to. Un vislabākais veids, kā eksperimentēt ir strādāt kopā ar vairākiem robotiem un cīnīties ar tiem.

Aplūkojot robotus, kas piedalās sacensībās, var novērot , ka izteikta popularitāte ir konstrukcijai, kur robotam priekšā ir "šķūre" ar domu, ka pretinieka robotam pabraucot uz šīs šķūres, to būs diez gan elementāri izgrūst. Svarīgi gan ir veidot robotu tā, lai tā konstrukcija būtu "stingra" un tam krītot ārā no ringa, tas nesabruktu.

12.3. Folkrace

Iespējams viena no visinteresantākajām disciplīnām ir tieši Folkrace. Tajā robota uzdevums ir braukt pa apļveida trasi, kurā var būt vairāk līkumi un šķēršļi. Papildus šķēršļiem var būt arī citu dalībnieku roboti. Robota mērķis ir izbraukt pēc iespējas vairāk apļus pareizajā virzienā. Virzienu nosaka sacensību tiesnesis.

Noteikumi

Kā jau iepriekš tika minēts, robotam ir jābrauc pa trasi pareizajā virzienā. Par katru apli, kas nobraukts pareizajā virzienā, robots saņem +1 punktu. Par katru apli, kuru robots nobrauc pretējā virzienā, robots saņem -1 punktu. Sacensības sastāv no 3 raundiem, katrs 3 minūtes garš. Roboti uzsāk tiesneša norādītajā vietā, novietoti viens otram blakus. Pēc starta signāla, robots drīkst uzsākt kustību tikai pēc 5 sekundēm.

Gadījumā, ja robots traucē citu robotu kustību, tiesnesim pēc 15 sekundēm ir tiesības šo robotu pacelt un novietot atpakaļ trasē pēc 10 sekundēm. Gadījumā, ja robots iestrēgst, bet netraucē citiem robotiem veikt distanci, dalībnieks var izvēlēties, vai atstāt robotu stāvam vai novietot atpakaļ uz starta līnijas. Novietojot atpakaļ uz sākuma līnijas, ir sacensības, kur robotam par nolikšanu uz sākuma līnijas tiek piešķirts -1 punkts vai arī iegūtie punkti tiek pilnībā dzēsti un robots turpina trasi ar 0 punktiem.

Pati trase ir 100 +- 10 cm plata un tai abās pusēs ir 10 +- 1 cm augsta mala. Trases standarta "grīda" ir melnā krāsā un sieniņas baltā. Trasē var būt šķēršļi, piemēram, kalns, kuras stāvums nepārsniedz 40 grādus. Padziļinājumi vai paaugstinājumi līdz 20 cm augstumā/dziļumā. Nelīdzenumi vai barjeras, trases sašaurinājumi.

Robota izmēri ir 15x20 cm platumā un garumā un robota svars var būt līdz 1 kg. Robotam nav noteikts maksimālais augstums, taču jāpatur prātā, ka var gadīties, ka trasē robotam būtu jābrauc zem iepriekš minētā tilta. Jāuzsver, ka robots brauciena laikā nevar mainīt savu konfigurāciju, tas ir, piemēram, robots brauciena laikā nevar nolaist "rokas", kas varētu traucēt citu dalībnieku robotiem orientēties trasē.

Atziņas

Šajā disciplīnā vērā ņemama ir gan robota uzbūve, gan arī tā programma. Runājot par uzbūvi, ļoti ieteicams ir veidot robotu, kurš spēj labi manevrēt, labi veikt pagriezienus. Svarīgi paturēt prātā, trasē būs vairāki roboti un tie var satriekties viens ar otru. Tāpēc svarīgi ir veidot robotu, kurš spēj izturēt šādus triecienus, nesabrukt pēc saskarsmes ar citiem robotiem. Taču stingrs robots var būt smags robots, kas varētu radīt problēmas ar uzbraukšanu kalnā, tāpēc vajag piedomāt, vai robots būs spējīgs uzbraukt kalnā.

Domājot par programmu, šeit var veidot kaut ko līdzīgu līnījsekotājam. Vienīgi līnijas vietā ir trases sieniņas. Diemžēl šīs disciplīnas lielākā problēma ir tāda, ka bez pašas trases ir grūti pārbaudīt robota darbību, vai tas spēj darboties pareizi.

Kā piebilde par katru no disciplīnām, katrā no tām robotam ir jādarbojas autonomi, tas ir, pats dalībnieks sacensību laikā nevar kontrolēt robotu ar tālvadības pulti. Taču starp sumo cīņām vai braucieniem katrā no disciplīnām, ir atļauts modificēt robota programmu.

Lai iepazītos ar pilniem noteikumiem katrā no disciplīnām, var aplūkot noteikumus, kuri bija spēkā Latvijas Robotikas Čempionātā. Ar tiem var iepazīties mājaslapā: robotuskola.lv