Blockly

Bevezetés a Blockly világába

Bácsi Sándor (2020.)
Mezei Gergely (2023.)

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

1117 Budapest, Magyar tudósok körútja 2.
Q. ép. B. szárny 207. • www.aut.bme.hu
Telefon: 463-2870 • Fax: 463-2871

Bevezetés

Célkitűzés

A Blockly kliensoldali Javascript library megismerése, amellyel vizuális blokk alapú nyelveket lehet létrehozni.

Előfeltételek

A labor elvégzéséhez szükséges eszközök:

- HTML és Javascript editor (pl. Visual Studio Code)
- Block Factory a custom blokkok létrehozásához

Átalános bevezetés a Blockly-ról

A Blockly egy kliensoldali Javascript library, amellyel vizuális blokk alapú nyelveket lehet létrehozni. Néhány népszerű projekt, ami Blockly-ban készült:

- AppInventor: Oktatási céllal fejlesztett Android app fejlesztő platform. Magasabb absztrakciós szinten lehet elkészíteni Blockly-ban az alkalmazásunk logikáját, amiből Java kód generálódik.
- Code.org: A programozás oktatására készített oldal, ahol játékos feladatokat oldhatnak meg a gyerekek különböző korosztályok szerint. A feladatokhoz a kódot Blockly-ban készítik el.
- SparqlBlocks: SPARQL lekérdezések készítésére használható felület.

Kiinduló projekt áttekintése

Töltsük le a kiinduló fájlokat és tekintsük át őket:

- blockly_compressed.js: A Blockly-hoz tartozó JS függőségek.
- blocks_compressed.js: Beépített blokkokat tartalmazza.
- **blocks.js**: Ebben a .js fájlban fogjuk tárolni a saját custom blokkok reprezentációját.
- en.js: A beépített blokkokhoz tartozó angol nyelvű feliratok.
- **generator_stub.js**: A custom blokkokhoz készült kódgenerálási logikát fogjuk implementálni ebbe a fájlba.
- **index.html**: A kezdő oldalért felelős HTML fájl. Ebbe a fájlba fogjuk berakni a toolbox-ba a saját blokkjainkat.
- javascript_compressed.js: A Blockly javascript generálásához szükséges függőségek.

A labor ideje alatt a **félkövérrel jelölt fájlokat** kell szerkeszteni.

BLOCK FACTORY MEGISMERÉSE

A saját custom blokkjaink létrehozásához a Block Factory-t fogjuk használni. A Block Factory megkönnyíti a blokkok létrehozását, hiszen grafikusan tudjuk elkészíteni őket. Tekintsük át a Block Factory felületét az alábbi szempontokat figyelembe véve:

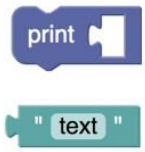
- **Szerkesztőfelület**. Nézzük át azokat a komponenseket, amelyekből a blokkokat lehet felépíteni:
  - **Input kategória**:
    - **Value input**: Horizontálisan tudunk újabb blokkokat kapcsolni a value inputba. Fontos megjegyezni, ha az inline input beállítást választjuk, akkor a blokkban fog megjelenni egy slot, ahova egy másik blokkot tudunk berakni.
    - **Statement input**: Egy olyan tároló, amibe vertikálisan tudunk egymás alá/fölé elhelyezni további blokkokat beágyazott módon.
    - **Dummy input**: Nem valódi input, nem tartalmaz kapcsolódási pontokat. Arra szolgál, hogy egy adott sorba rendezzük a field-eket.
  - **Field kategória**: A különböző input-okhoz úgynevezett field-eket tudunk rendelni, amelyekkel testre szabhatjuk a blokk konkrét szintaxisát.
  - **Type:** Type constraint jelleggel tudjuk szabályozni a blokkok kapcsolódási szabályszerűségeit a típusok alapján.
  - **Colour:** A blokkok színét tudjuk állítani.
- **Preview:** Az aktuálisan készülő blokkunk előnézetét tudjuk megtekinteni és kipróbálni.
- **Block definition:** Az aktuálisan készülő blokkunk kód alapú reprezentációja. A laboron **a JS reprezentációt** fogjuk bemásolni a **blocks.js** fájlba.
- **Generator stub:** A kódgeneráláshoz szükséges csonkot kapjuk meg, amelyet kiegészítve elkészíthetjük a blokkunk tartozó logikát. Ennek a megvalósításához a legegyszerűbb módszer, ha string összefűzést alkalmazunk, amely kevésbé elegáns megközelítés, viszont a labor feladatok megoldásához bőven elegendő.

Feladatok

1. Basic Blocks

1.1. Print block

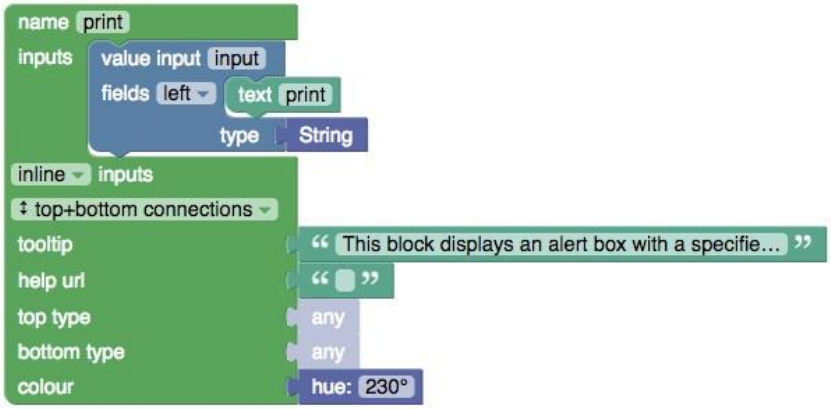
Hozz létre egy blokkot, amely segítségével egy megadott szöveget lehet kiírni a JavaScript –es Alert boxba. A megoldáshoz egy olyan blokkot készíts, amelynek egy String inline value inputja van és ehhez lehet kapcsolni egy olyan blokkot, amiben az Alert hívás String paraméterét lehet megadni.



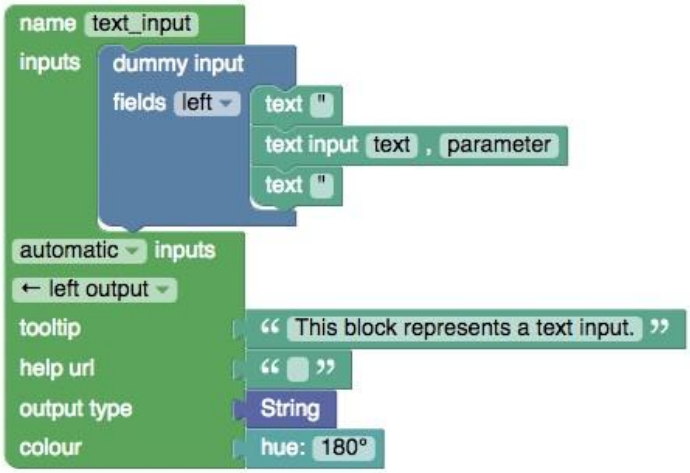
Megoldás

Block Factory-ban az alábbi blokkokat kell létrehoznunk.

Print block:



Text block:



blocks.js:

```
Blockly.Blocks['print'] = {
  init: function() {
   this.appendValueInput("input")
       .setCheck("String")
.appendField("print");     this.setInputsInline(true);
this.setPreviousStatement(true, null);
this.setNextStatement(true, null);
this.setColour(230);
 this.setTooltip("This block displays an alert box with a specified message and an OK button.");   this.setHelpUrl("");
  }
 };
```

```
 Blockly.Blocks['text_input'] = {
  init: function() {
   this.appendDummyInput()
       .appendField("\"")
       .appendField(new Blockly.FieldTextInput( "text"), "parameter")
       .appendField("\"");
   this.setInputsInline(true);
   this.setOutput(true, "String");
   this.setColour(180);
 this.setTooltip( "This block represents a text input.");
 this.setHelpUrl("");
  }
 };
```

generator_stub.js:

```
Blockly.JavaScript['print'] = function(block) {
 var value_input = Blockly.JavaScript.valueToCode(block, 'input', Blockly.JavaScript.ORDER_ATOMIC);
 var code = 'alert('+value_input +');\n';
 return code;
};
```

```
Blockly.JavaScript['text_input'] = function(block) {
 var text_parameter = block.getFieldValue('parameter');
 var code = '"' + text_parameter + '"';
 return [code, Blockly.JavaScript.ORDER_ATOMIC];
};
```

Tipp: A kódgenerálási logikában a string összefűzésnél használhatunk [template literálokat](#) is:

```
`string text ${expression} string text`
```

Fontos, hogy **a text_input** block-nál a **Blockly.JavaScript.ORDER_ATOMIC** –al térjünk vissza, hogy jól működjön az operátorok kiértékelése.

index.html: A toolbox-on belülre vegyük fel:

```
<category name="Basic blocks" colour= "180">
    <block type="print"></block>
    <block type="text_input"></block> </category>
```

Próbáljuk is ki az elkészült blokkokat, ellenőrizzük a végeredményt az alábbi módon:

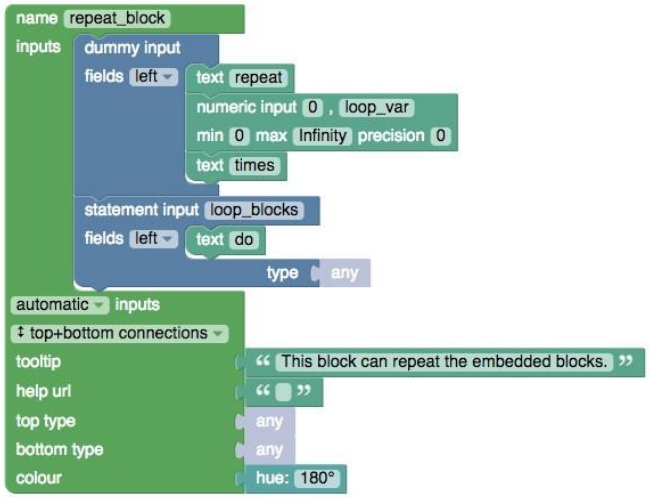
A generált kód megjelenítéséhez használjuk a „**Show code**” gombot, a futtatáshoz „**Run**” gombot.

1.2. REPEAT BLOCK

Készíts egy olyan repeat blokkot, ami a beágyazott blokkokat annyiszor ismételi meg, amennyi a megadott érték. Ügyeljünk arra, hogy az ismétlés mennyiségének megadásakor csak szám értéket fogadjunk el. A blokk előnézete a következő legyen:

Megoldás

Block Factory:



blocks.js:

```
Blockly.Blocks['repeat_block'] = {
  init: function() {
    this.appendDummyInput()
        .appendField("repeat")
        .appendField(new Blockly.FieldNumber(0, 0), "loop_var")
        .appendField("times");
    this.appendStatementInput("loop_blocks")
        .setCheck(null)
.appendField("do");
this.setPreviousStatement(true, null);
this.setNextStatement(true, null);
this.setColour(180);
 this.setTooltip("This block can repeat the embedded blocks.");    this.setHelpUrl("");
  }
};
```

generator_stub.js:

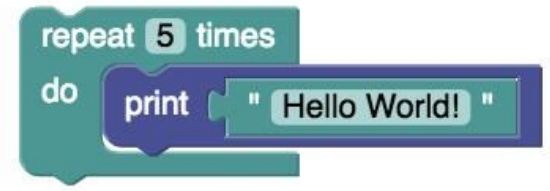
Ennél a résznél nyilvánvalóan több jó megoldás is lehetséges.

```
Blockly.JavaScript['repeat_block'] = function(block) {  var
loop_var = block.getFieldValue('loop_var');
 var statements_loop_blocks = Blockly.JavaScript.statementToCode(block, 'loop_blocks');  var code
= 'var repeats = 0;\n' +   'while (repeats <' + loop_var +') {\n'+
 statements_loop_blocks +
 'repeats++;\n' +
 '}\n';
 return code;
};
```

index.html: A Basic blocks kategórián belülre vegyük fel:

```
<block type="repeat_block"></block>
```

Próbáljuk is ki az elkészült blokkot, például írjunk ki 5-ször egy tetszőleges szöveget.



1.3. Variables kategória hozzáadása

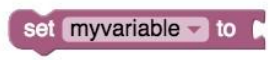
A Blockly-nak van egy beépített JavaScript-es változókezelő kategóriája. Az ehhez tartozó blokkok a **Blockly.JavaScript.variableDB** –t használják. Ezen blokkok segítségével könnyedén tudunk új változót létrehozni és módosítani annak értékét. A következő feladatban egy **for ciklust reprezentáló blokkot** fogunk készíteni, ahol jó lenne, ha tudnánk a ciklusváltozó értékét lekérdezni, illetve módosítani. Adjuk hozzá a toolbox-hoz a beépített változókezelő kategóriát.

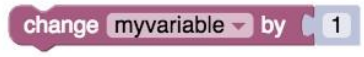
Megoldás

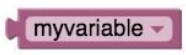
Index.html: A toolbox-on belülre vegyük fel az új kategóriánkat az alábbi módon:

```
<category name="Variables" colour="%{BKY_VARIABLES_HUE}" custom="VARIABLE"></category>
```

Próbáljuk ki az újonnan felvett kategóriákat. Nézzük meg, hogy pontosan milyen blokkok tartoznak ide. Vegyünk fel egy új változót és próbáljuk ki a blokkokat.







1.4. Number block

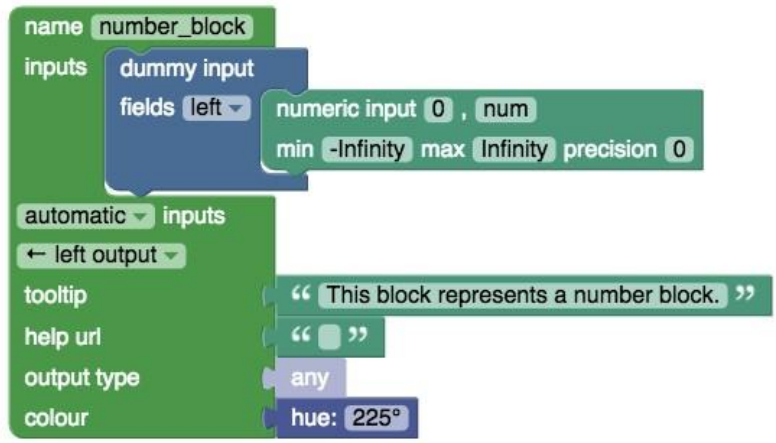
Ahhoz, hogy a későbbiekben Number típusú paraméterrel is jól tudjuk használni a blokkjainkat (value input-hoz is tudjunk számot rendelni), hozzunk létre egy olyan blokkot, ami left output-ként egy number típusú értéket tud visszaadni. Erre majd a különböző A-Frame-es testek létrehozásánál lesz szükségünk. A blokk előnézete a következő legyen:

Megoldás

Index.html: A toolbox-on belülre vegyük fel az új kategóriát:

```
<category name="VR blocks" colour= "225">
```

Block Factory:



blocks.js:

```
 Blockly.Blocks['number_block'] = {    init:
function() {
    this.appendDummyInput()
        .appendField(new Blockly.FieldNumber(0), "num");
this.setOutput(true, "Number");      this.setColour(225);
  this.setTooltip("This block represents a number block.");    this.setHelpUrl("");
   }
 };
```

generator_stub.js:

```
Blockly.JavaScript['number_block'] = function(block) {  var
number_num = block.getFieldValue('num');  var code =
number_num;
 return [code, Blockly.JavaScript.ORDER_ATOMIC];
};
```

Fontos, hogy **Blockly.JavaScript.ORDER_ATOMIC** –al térjünk vissza, hogy jól működjön az operátorok kiértékelése.