Fakulta informatiky a informačných technológií
Slovenská technická univerzita

VRLab Radiant

Dokumentácia k inžinierskemu dielu

(1. kontrolný bod)

Akademický rok: 2020/2021

Pedagogický vedúci: Ing. Juraj Vincúr

Členovia tímu (č.16): Bc. Patrik Tománek
Bc. Ľubomír Kurčák
Bc. Tomáš Sabo
Bc. Erik Paľa
Bc. Viktor Beňo

OBSAH

Úvod 3
1 Globálne ciele pre zimný semester 4
2 Celkový pohľad na systém 4
3 Moduly systému 5
3.1 Breadboard riešenie 5
3.1.1 Analýza 5
3.1.2 Návrh 6
3.2 Blokové (Grid) riešenie 6
3.2.1 Analýza 6
3.2.2 Návrh 7
3.3 Prekladač modelu na obvod pre breadboard riešenie 9
3.3.1 Analýza 9
3.3.2 Návrh 9
3.4 Prekladač modelu na obvod pre blokové (grid) riešenie 10
3.4.1 Analýza 10
3.4.2 Návrh 11
3.5 Vyhodnotenie návrhov a riešení 12
4 Prototypy 12
4.1 prototyp pre ovládanie a interakciu 12
4.1.1 Analýza 12
4.1.2 Návrh 13
4.1.3 Implementácia 14
4.1.4 Testovanie 16
Záver 17

ÚVOD

Tento dokument predstavuje dokumentáciu k inžinierskemu dielu projektu v rámci predmetu Tímový projekt. Cieľom nášho projektu je vytvorenie laboratória vo virtuálnej realite pre distančné vzdelávanie, určené pre výučbu a simuláciu elektrických obvodov.

Súčasťou tohto dokumentu je opísanie globálnych cieľov nášho projektu v rámci zimného semestra, celkový pohľad na vytváraný systém, prehľad doposiaľ vytvorených modulov systému a na koniec opis prototypu, ktorý sme vytvorili. Prototyp bol vytvorený v Unity a modely, s ktorými sme pracovali sme si vytvorili v grafickom prostredí Blender.

1 GLOBÁLNE CIELE PRE ZIMNÝ SEMESTER

Autor kapitoly: Ľubomír Kurčák, Patrik Tománek

Za zimný semester by sme chceli vytvoriť jednoduchý funkčný prototyp, ktorý nám napovie ďalší smer vývoja. Cieľom je vytvoriť program, v ktorom bude možné vytvárať elektrické obvody zložené z blokových súčiastok uložených do mriežky. Správanie vytvoreného obvodu bude simulované a výstupy budú použité na ovplyvnenie virtuálneho prostredia, napríklad rozsvietená žiarovka, krútiaci sa moment kolies, a iné.

2 CELKOVÝ POHĽAD NA SYSTÉM

Autori kapitoly: Ľubomír Kurčák, Patrik Tománek

Celkový pohľad na systém sa dá opísať ako kooperatívne virtuálne prostredie, v ktorom používatelia môžu spolu vytvárať simulovaný elektrický obvod a experimentovať a testovať jeho správanie. Používatelia interagujú so skladacou mriežkou, do ktorej vkladajú elektronické súčiastky. Z mriežky so súčiastkami sa následne vygeneruje vstup pre simulátor elektrických obvodov. Tento krok je úmyselne oddelený, čo nám poskytuje nezávislosť od jednotlivých simulátorov, a takisto nám umožňuje jednoduchšiu rozšíriteľnosť pre nové súčiastky. Súčiastky v obvode sa dajú kategorizovať do troch skupín z pohľadu energie; zdroje, ako je napríklad baterka, manipulátory toku, ako rezistory a vodiče a spotrebiče energie, ako napríklad žiarovka, elektrický motor a ďalšie. Výstup zo simulátora nám vráti prúd, napätie a odpor v jednotlivých častiach obvodu, na základe ktorých vieme simulovať jeho správanie v hernej logike a zobraziť výstupy audiovizuálnou formou. Jednotlivý klienti sa podieľajú na tvorbe tohto simulovaného obvodu. Zmeny vo virtuálnom prostredí sú prenášané sieťou, tak aby všetci používatelia zostali synchronizovaní.

Mriežka simulovaného obvodu je dvojrozmerná a má rozmery m×n, ktoré sú parametrizovateľné. Je možné uvažovať o rozšírení do troch rozmerov, no túto variantu sme z hľadiska jednoduchosti použitia, tvorby a ladenia obvodu nezvolili. Je potrebné zvoliť také parametre veľkosti, ktoré umožňujú dostatočnú komplexnosť obvodovej logiky na splnenie úloh, či tvorbu zaujímavej funkcionality. Na druhú stranu, čím väčší obvod simulujeme, tým dlhšie budú výpočty trvať, čo môže zapríčiniť nízku obnovovaciu frekvenciu simulácie a zobrazenia a teda stratu interaktivity. Na každú pozíciu v mriežke vieme vložiť jednu súčiastku. Každá zo súčiastok môže byť pripojená ku každej susednej súčiastke v mriežke. Niektoré súčiastky môžu mať viac pripojovacích soketov ako iné. Napríklad súčiastka ako baterka má dva sokety, jeden ktorý predstavuje kladné a druhý záporné napatie. Ďalej napríklad rozvetvenie vodiča predstavuje trojvetvovú križovatku, a teda prislúchajúca súčiastka bude mať 3 sokety. Súčiastky je pri vkladaní možné orientovať do štyroch hlavných smerov mriežky. Jedná sa o rotácie o 0°, 90°, 180° a 270°. Pri rotáciách sa pozície soketov menia, čím je možné vytvoriť z tej istej pozície prepojenia na iné súčiastky.

Používatelia vložia súčiastky do mriežky, ktoré následne prekladáme na vstup, ktorý vie spracovať simulátor elektrických obvodov. V tomto kroku je potrebné určiť susedenosť a prepojenia jednotlivých súčiastok. Tieto informácie sú určené pozíciou a orientáciou súčiastok. Susedné súčiastky, ktoré majú na spoločnej hrane obe voľný soket považujeme za spojené. Vygenerovaný opis obvodu dáme následne na vstup simulátoru. Výstup simulátoru opäť interpretujeme a použijeme ako výsledok hernej logiky, ktorý graficky vizualizujeme.

Používatelia môžu program ovládať buď pomocou nástrojov virtuálnej reality, alebo klasickou klávesnicou a myšou. Program je navrhnutý aby bol nezávislý od typu vstupu. Voľba predmetu na interakciu je vykonaná pretnutím lúču s najbližším predmetom k začiatku lúča. So zvoleným predmetom môže používateľ voľne manipulovať a umiestniť ho do mriežky. Niektoré súčiastky majú ďalšie funkcie, ktoré je možné ovládať. Používateľ môže náhľadom skontrolovať prúd napätie a odpor v jednotlivých častiach obvodu. Používateľ môže použiť multimeter na merania v ľubovoľných častiach v obvode.

Sieťová komunikácia bude riešená jedným z existujúcich modulov pre Unity engine. Synchronizované budú avatary používateľov, súčiastky v obvode a mimo obvodu, a samotný obvod a jeho vplyv na konečné spotrebiče.

3 MODULY SYSTÉMU

V tejto kapitole je rozpísaných niekoľko modulov, ktoré sú súčasťou nášho projektu. Každý modul je rozpísaný v samostatnej podkapitole spolu s autorom alebo autormi, ktorí na danom module pracovali.

3.1 BREADBOARD RIEŠENIE

Autor podkapitoly: Erik Paľa

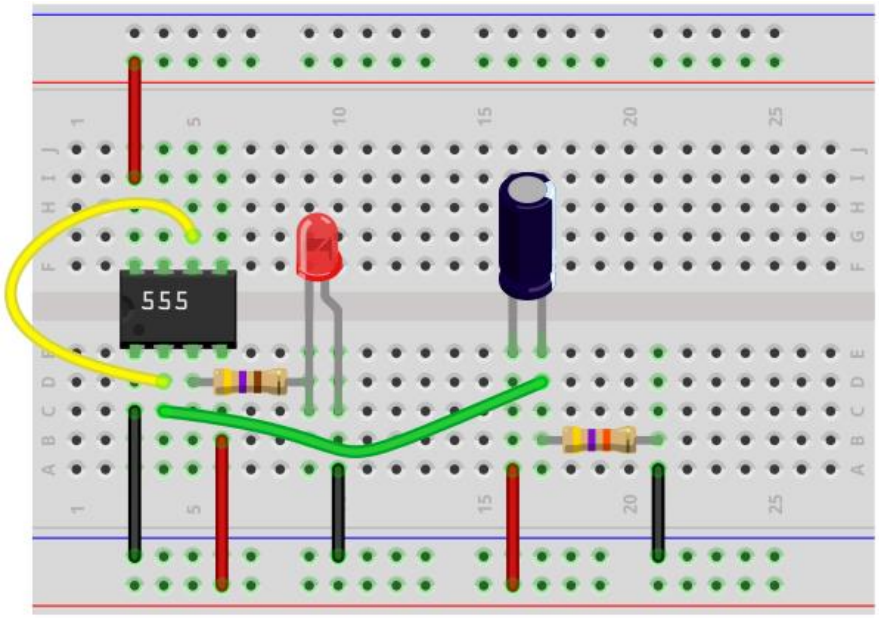
TIE PODSEKCIE BUDEME ROBIT IBA TIE CO MAME, VYMAZTE TIE, KTORE NEMAME

3.1.1 Analýza

Riešenie takéhoto typu by predstavovalo simuláciu reálneho zapojenia obvodu v realite. Toto riešenie umožňuje vytvárať komplexné zapojenia, kde používateľ nie je obmedzený počtom komponentov v uzle.

Výhodou tohto riešenia je to, že pre jednotlivé komponenty by boli použité ich reálne obrazy. Ďalšou veľkou výhodou tohto riešenia je taktiež možnosť vyskúšať si zapojenie zo simulátora v realite, kde reálny obvod by bolo možné zapojiť presne podľa zapojenia v simulátore.

Nevýhodou tohto riešenia je to, že v porovnaní s blokovým riešením je pre neskúseného používateľa oveľa zložitejšie na pochopenie. Ďalšou veľkou nevýhodou vysoká náročnosť na zhotovenie riadiacej logiky pre simuláciu takýchto zapojení. V porovnaní s blokovým riešením je nevýhodou aj to, že je potrebné riešiť pozície jednotlivých nožičiek pre každú súčiastku.



Obrázok 1 - Príklad breadboard zapojenia

3.1.2 Návrh

Základ tohto riešenia by predstavoval breadboard a pre jednotlivé komponenty by boli použité ich reálne obrazy, príklad breadboard zapojenia môžete vidieť na Obrázok 1

Samotná doska by bola rozdelená na jednotlivé uzly, kde v stredných radoch pinov jeden stĺpec predstavuje jeden uzol a pri vrchných a spodných radoch, ktoré sú oddelené od stredných, jeden riadok reprezentuje jeden uzol. Veľkosti uzlov je možné ľubovoľne zväčšiť tak, že vodičom spojíme 2 samostatné uzly.

Výhodou oproti blokovému riešeniu je, že nie je potrebné riešiť počet nožičiek, ktoré súčiastka má, nakoľko breadboard umožňuje zapojenie komponentov s ľubovoľným počtom nožičiek.

3.2 Blokové (Grid) riešenie

Autor podkapitoly: Viktor Beňo

TIE PODSEKCIE BUDEME ROBIT IBA TIE CO MAME, VYMAZTE TIE, KTORE NEMAME

3.2.1 Analýza

Riešenie takéhoto typu by sa podobalo na stavebnice zapájania elektrického obvodu, v ktorom sú el. komponenty kocky(alebo iné vhodné tvary) zapájajúce sa do pripravenej podložky, príklad môžete vidieť na Obrázok 2. Na tejto podložke sú vyznačené miesta, na ktoré sa bloky stavebnice ukladajú a väčšinou sa jedná o mriežku s rozmermi bloku.

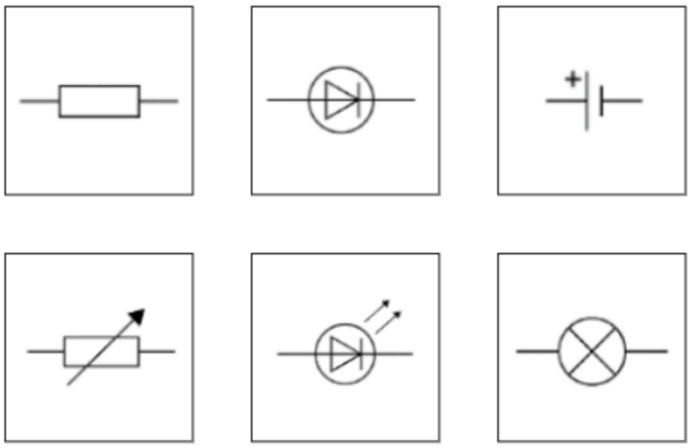
Elektrický obvod, zapojený s použitím týchto blokov pôsobí prehľadnejšie a je ľahší na pochopenie. Taktiež by bolo takéto riešenie jednoduchšie na implementáciu, zvlášť ak by boli použité na spájanie diskrétne bloky rovnakej veľkosti, ktoré sa spájajú magneticky.

Obrázok 2 - Produkt IQube ako príklad blokového riešenia

3.2.2 Návrh

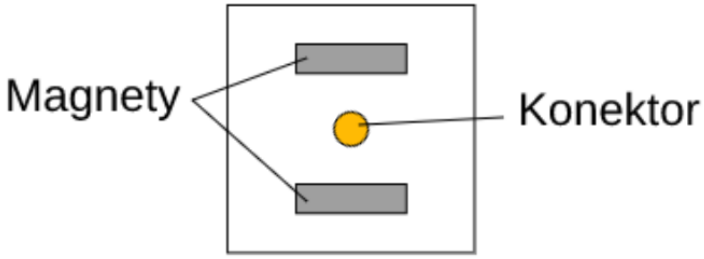
Použité by boli kocky predstavujúce elektrické komponenty ako hlavné stavebné bloky elektrického obvodu. Zapájali by sa do plochej dosky s mriežkovým vzorom pre pravidelné usporiadanie, ktoré bude pôsobiť prehľadne.

Kocky by mali na vrchnej strane znázornené schematické značky elektrických komponentov, ktoré prestavujú. Na Obrázok 3 je pohľad zhora pre šesť kociek elektrických komponentov.



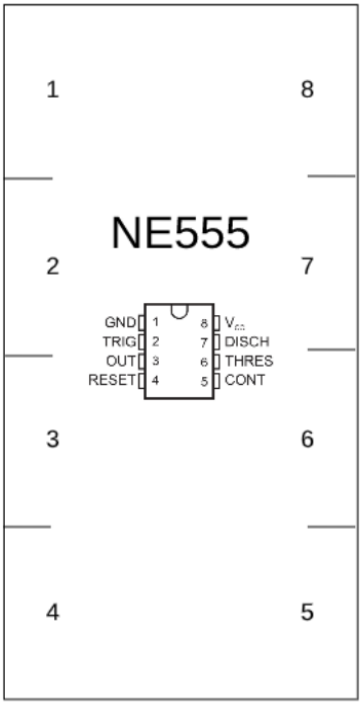
Obrázok 3 - Návrh kociek blokového riešenia pri pohľade zhora

Kocky budú spájané magneticky pre jednoduchšie skladanie obvodov. Na bočných stranách, podľa typu elektrického komponentu, sa budú nachádzať dva magnety pre správne spojenie kociek a potrebný počet konektorov, príklad môžete vidieť na Obrázok 4.



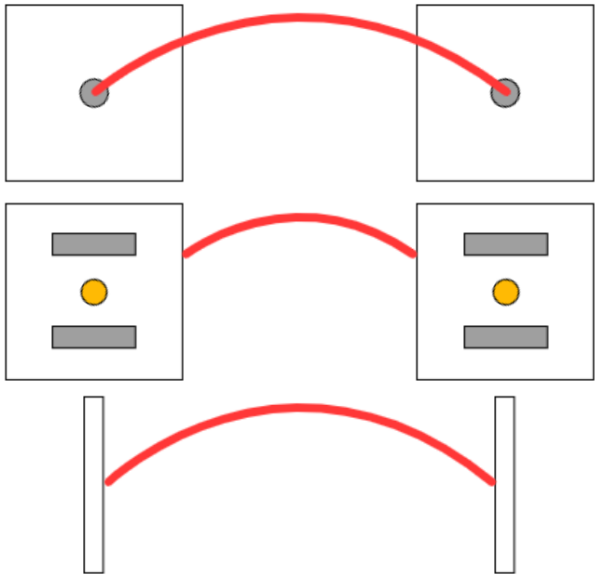
Obrázok 4 - Návrh kocky blokového riešenia pri pohľade z boku

Pri súčiastkach s väčším počtom zapojení je najľahším riešením vytvorenie väčšieho bloku, kde pre každé potrebné zapojenie vstupu alebo výstupu elektrického komponentu bude použitá jedna kocka. Takýto blok, zobrazený na Obrázok 5, sa dá aj redukovať na menší počet kociek, avšak výsledný menší blok môže pôsobiť neprehľadne.



Obrázok 5 - Návrh pre viac blokovú súčiastku

Kocky by sa mohli prepájať prípadne aj káblami, ktoré by sa tiež pripájali magneticky. Takýto príklad môžete vidieť na Obrázok 6.



Obrázok 6 - Návrh pre káblové prepojenie blokov

3.3 PREKLADAČ MODELU NA OBVOD PRE BREADBOARD RIEŠENIE

Autor podkapitoly: Erik Paľa

3.3.1 Analýza

Pri breadboard modeli bude treba riešiť zapojenie komponentov do uzlov. Pri breadboarde sú jednotlivé uzly reprezentované stĺpcami na vnútornej časti dosky a dvoma riadkami pinov na vrchu a na spode dosky.

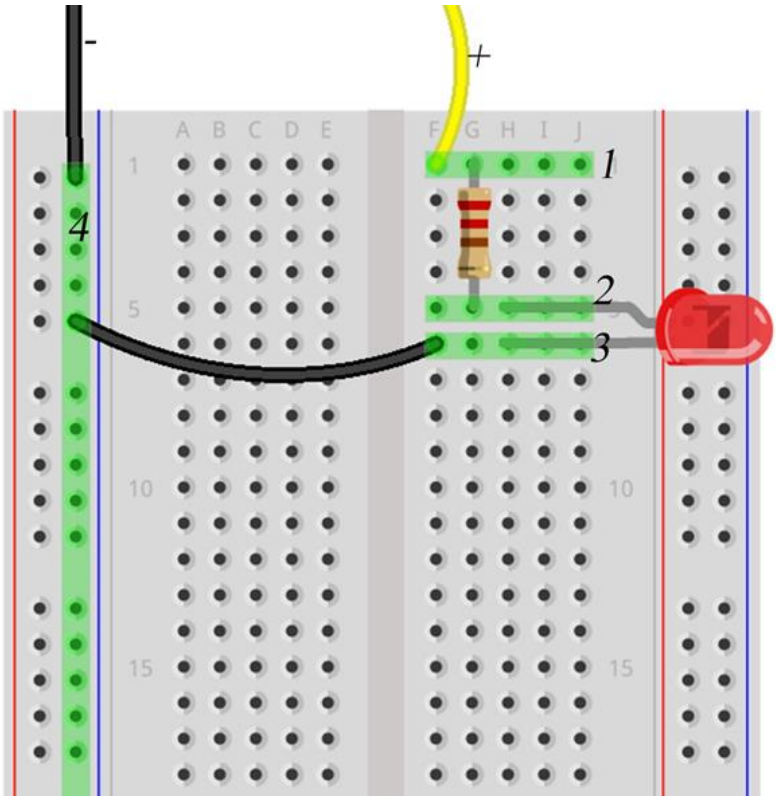
Pre tvorbu samotného obvodu poskytuje *SpiceSharp* triedu *Circuit*, do ktorej sa následne môžu pridávať jednotlivé súčiastky z knižnice. Pri pridávaní komponentov je potrebné nastaviť ich názov, hodnoty, uzly, v ktorých je súčiastka zapojená a prípadne iné parametre.

SpiceSharp má implementovanú funkcionalitu pre vytváranie uzlov (*nodes*), kde ak súčiastky majú uzol so spoločným názvom, považujú sa za spojené.

3.3.2 Návrh

V *SpiceSharp*-e sa súčiastky pripájajú k uzlom označeným ako “nodes”, ktoré sú označené premennou typu *String*. Vzhľadom na to, že pri breadboarde predstavuje pri vnútorných riadkoch pinov každý stĺpec 1 uzol a pri vrchných a spodných dvoch radoch pinov je každý riadok 1 uzol, je potrebné nastaviť jednotlivé piny na doske tak, aby patrili uzlu, ktorý reprezentuje daný stĺpec alebo riadok.

Jednotlivé piny sú reprezentované dvoma hodnotami, kde písmeno reprezentuje riadok a číslo stĺpec v ktorom sa pin nachádza. Uzly je možné reprezentovať samostatnou sadou súradníc, kde sa pre vnútornú sadu už neriešia jednotlivé riadky ale táto časť dosky bude rozdelená len na vrchnú a spodnú polovicu. Čísla stĺpcov zostávajú zachované, nakoľko každý stĺpec predstavuje 1 uzol. Pre správnu funkcionalitu uzlov je teda potrebné nastaviť, ktoré riadky na doske sú v ktorej polovici a teda do ktorého uzla spadajú.



Obrázok 7 - Jednoduchý elektrický obvod pri breadboard riešení

Na Obrázok 7 môžeme vidieť jednotlivé uzly vyznačené zelenou farbou a označené číslami. Pri zostrojení jednoduchého obvodu, ktorý pozostáva zo zdroja, rezistora a diódy je pri pridávaní jednotlivých komponentov potrebné nastaviť pozíciu ich nožičiek korektne, aby obvod mohol fungovať.

Pri pridávaní rezistora je potrebné nastaviť mu hodnotu odporu, ktorú má v obvode predstavovať a polohu nožičiek, ktoré v tomto konkrétnom prípade sú pre prvú nožičku G1 a pre druhú nožičku G5, kde G1 spadá do prvého uzla a G5 do druhého uzla. Následne pri pridávaní diódy je potrebné dbať na jej otočenie. Pri jej pridaní do obvodu sa nastaví poloha nožičiek, kde prvá nožička (anóda) má súradnice H5 a druhá nožička (katóda) má súradnice H6. Nakoľko rezistor aj dióda majú nožičku v uzle 2 (stĺpec 5), sú považované za prepojené. Zdroj tohto obvodu je pripojený do uzla 1 a uzla 4. Môžeme si všimnúť, že ako bolo skôr v dokumente uvedené, celý spodný riadok predstavuje jeden uzol. Celý obvod sa uzavrie pridaním prepojenia medzi uzlom 3 a uzlom 4.

Pri vytváraní elektrického obvodu na breadboard-e je potrebné ošetriť, aby sa pri nastavovaní súradníc pre nožičky súčiastok nevyskytla situácia, že oba nožičky jednej súčiastky by sa nachádzali v rovnakom uzle.

3.4 Prekladač modelu na obvod pre blokové (grid) riešenie

Autor podkapitoly: Viktor Beňo

3.4.1 Analýza

Pri blokovom modeli bude treba riešiť spájanie blokov ako elektrických komponentov a ich pridanie do výsledného elektrického obvodu, ktorý bude simulovaný v *SpiceSharp*-e. Bloky sa budú dať rotovať takže sa bude musieť riešiť správne zapojenie súčiastok v obvode.

SpiceSharp poskytuje triedu *Circuit* pre vytvorenie elektrického obvodu, do ktorej sa následne môžu vkladať jednotlivé elektrické komponenty, ktoré knižnica poskytuje, príklad obvodu môžete vidieť na Obrázok 8. Pri vytváraní elektrických komponentov sa nastavuje ich názov, hodnoty elektrických veličín príslušné komponentu, uzly napojenia komponentu a prípadne iné parametre.

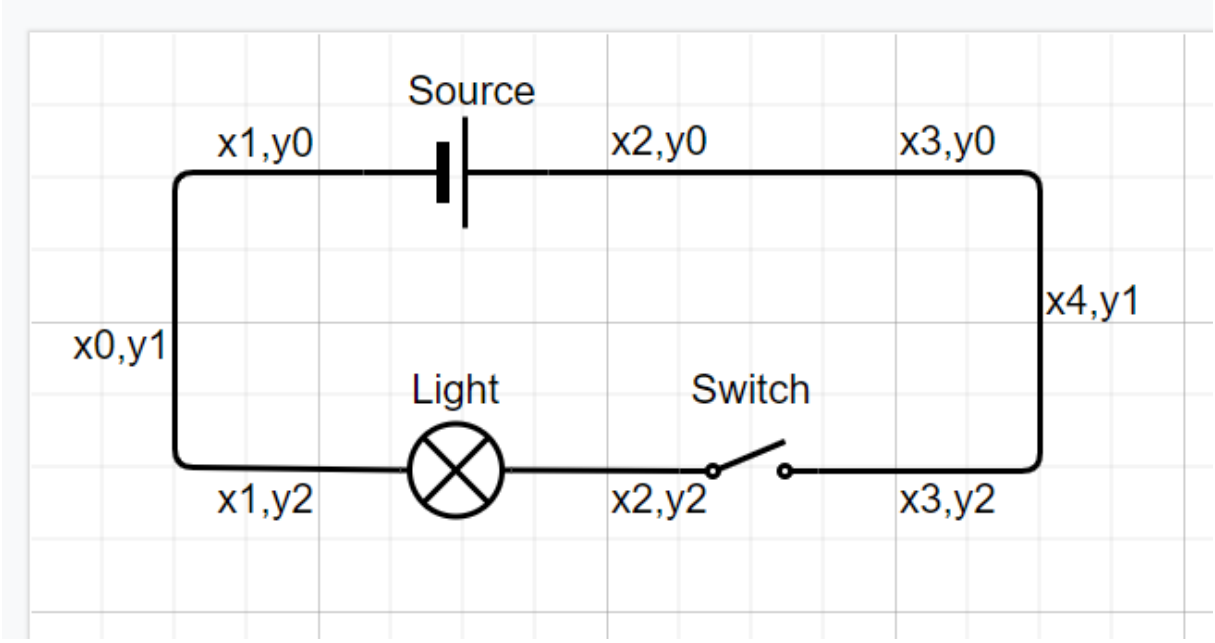
```
// Build the circuit
var ckt = new Circuit(
    new VoltageSource("V1", "in", "0", 1.0),
    new Resistor("R1", "in", "out", 1.0e4),
    new Resistor("R2", "out", "0", 2.0e4)
    );
```

Obrázok 8 - Vytvorenie obvodu s tromi el. komponentami v prostredí SpiceSharp

SpiceSharp rieši spájanie súčiastok v obvode cez uzly (nodes), ktoré má každá súčiastka a predstavujú názov uzla, s ktorým je táto súčiastka spojená. Ak majú napríklad komponenty nastavený uzol s rovnakým názvom, tak to znamená, že sú prepojené.

3.4.2 Návrh

Keďže v *SpiceSharp*-e sa súčiastky pripájajú jednoducho k uzlom označeným ako “nodes”, ktoré sa označujú premennou typu *String*, pre mriežku, s ktorou budeme pracovať, si môžeme vytvoriť maticu alebo dvojrozmerné pole *String*-ov. V tejto matici budú uložené názvy pre jednotlivé uzly v obvode, kde sa dotýkajú dva bloky na mriežke. Na Obrázok 9 sú tieto názvy uzlov príkladovo označené (napríklad “x1,y0”).



Obrázok 9 - Jednoduchý elektrický obvod pri blokovom riešení

Na začiatku sa vytvorí v *SpiceSharp*-e nový objekt obvodu, do ktorého budú postupne pri akcii vloženia alebo odobratia bloku vkladané alebo odobraté komponenty elektrického obvodu, ktoré dané bloky predstavujú.

Pri vložení nového bloku na určené miesto v mriežke sa podľa typu el. komponentu, umiestnenia a rotácie bloku zistia uzly pre daný komponent. Ďalej sa už len tento komponent vytvorí s napojením

na správne uzly a poskytnutými parametrami komponentu uložených v objekte bloku. Po vytvorení sa komponent pridá do obvodu.

Bloky predstavujúce križovatku (iba spájajú všetky susedné bloky) vytvoria nový uzol, s ktorým sa pomocou rezistorov pripoja všetky 4 uzly bloku. Podobne sa postupuje pri kríženiach typu "T".

V objekte bloku sa udržuje typ el. komponentu, jeho parametre(alebo pred vytvorený komponent) a napájacie uzly, ktoré môžu byť kódované *integer* polom, pričom budú použité hodnoty 0 a 1 pre určenie tých strán bloku, na ktorých je napájací uzol. Napríklad pre rezistor horizontálnej polohe by bolo takéto pole (0,1,0,1), pričom prvá hodnota by predstavovala spodnú stranu bloku a nasledujúce hodnoty ďalšie strany v smere hodinových ručičiek. Takéto kódovanie sa môže podľa potreby a implementácie upraviť.

Pri odobratí bloku z mriežky a teda z elektrického obvodu by sa podľa konkrétneho názvu komponentu tento komponent vyhľadal v obvode a z neho následne vymazal.

3.5 VYHODNOTENIE NÁVRHOV A RIEŠENÍ

Vzhľadom na vysokú zložitosť riešenia tohto projektu pri použití breadboard-u sme sa rozhodli uprednostniť blokové riešenie, ktoré je prehľadnejšie a jednoduchšie na implementáciu.

Cieľom tohto blokového riešenia bude vytvoriť interaktívne prostredie, kde jednotlivé komponenty budú reprezentované 3D blokmi, ktoré bude možné uložiť na mriežku a tým zostrojiť elektrický obvod.

Jednotlivé komponenty budú reprezentované rôznymi typmi blokov. Napríklad štandardné súčiastky ako rezistor budú reprezentované ako kocka, na ktorej bude symbol rezistora. Aktívne súčiastky ako napríklad spínač alebo dióda budú reprezentované ich 3D modelmi.

Výhodou tohto riešenia je jeho prehľadnosť pre používateľa, ktorý nemusí pracne pripájať jednotlivé nožičky súčiastok do breadboard-u, ale stačí mu otočiť súčiastku ako potrebuje a umiestniť ju do mriežky, ktorá automaticky vytvorí uzly na miestach kde sa bloky spájajú.

4 PROTOTYPY

Do konca 3. šprintu sme mali za úlohu vytvoriť prvý prototyp nášho projektu, funkcionalitu tohto prototypu uvádzame nižšie.

4.1 PROTOTYP PRE OVLÁDANIE A INTERAKCIU

Autori podkapitoly: Ľubomír Kurčák, Patrik Tománek

Cieľom prvého prototypu bolo umožniť hráčovi pohybovať sa v 3D priestore, ako aj dovoliť hráčovi interagovať s elektrickými súčiastkami a grid boardom v tomto prostredí.

4.1.1 Analýza

Pohybovanie sa v 3D priestore ako aj interakcia s elektrickými súčiastkami a grid boardom sú kľúčové prípady použitia pre náš projekt. Počas našej analýzy bolo potrebné určiť si akým prístupom túto funkcionalitu implementujeme do nášho riešenia. Rozhodli sme sa tento prvý prototyp implementovať ako desktopové riešenie pre podporu klávesnice a myši.

Súčasťou finálneho prototypu sú taktiež modely pre jednotlivé elektrické súčiastky, ktoré sme si spoločne vybrali pre náš projekt. Tieto modely vytvoril Ľubomír Kurčák v grafickom prostredí Blender. Hlavný element rozhodovania pri výbere súčiastok, ktoré budú súčasťou nášho riešenia bol ten, že všetky súčiastky museli byť podporované knižnicou *SpiceSharp*, ktorú budeme používať na simuláciu správania elektrických obvodov v našom projekte. Po dokončení tvorby zoznamu podporovaných súčiastok sa mohlo prejsť na ich tvorbu.

Čo sa týka tvorby prototypu v Unity, rozhodli sme sa pre vytvorenie ukážkovej scény pre prvý prototyp, ktorá obsahovala prostredie, v ktorom sa hráč pohybuje. Takisto sa v tomto prostredí nachádzal grid board ako aj jednotlivé elektrické súčiastky, s ktorými mohol hráč interagovať. Funkcionalita jak pohybovania tak aj interakcie bola tvorená pre klávesnicu aj myš takým spôsobom, aby sa dané riešenie mohlo aplikovať aj na následnú integráciu VR do nášho projektu.

4.1.2 Návrh

Pri tvorbe prototypu v Unity, sme sa rozhodovali, akým smerom sa vyberieme. Nakoniec sme sa dohodli na finálnom návrhu.

Hráč bude reprezentovaný ako *GameObject* spolu s kamerou ako aj ďalším *GameObjectom*, ktorý bude reprezentovať jeho ruky. Hráč sa bude môcť pohybovať do všetkých smerom pomocou klávesnice a myšou bude ovládať smer akým sa hráč pozerá. Ruky hráča budú slúžiť ako kontajner pre všetky elektrické súčiastky, ktoré si vezme do rúk. Spôsob detekcie, akým hráč bude zdvíhať jednotlivé súčiastky, ako aj ich ukladanie do grid boardu bude zabezpečený pomocou *Raycast* funkcionality. Táto funkcia slúži na interakciu hráča s objektom, na ktorý sa hráč momentálne pozerá, pretože tento *Raycast* bude vysielaný zo stredu obrazovky hráča, a teda stredu danej kamery, ktorá je hráčovou súčasťou. *GameObject* hráč bude teda vo finále obsahovať 3 hlavné komponenty. Jeden komponent bude slúžiť ako kontrolér pre ovládanie, ďalší pre interakciu a posledný komponent bude kontrolérom pre hráčove ruky.

Elektrické súčiastky budú taktiež reprezentované ako *GameObject*. V tejto fáze prototypu sme neimplementovali logiku jednotlivých súčiastok, takže tieto súčiastky zatiaľ zohrávajú úlohu modelu, s ktorým hráč môže interagovať a vkladať ho do grid boardu. Tieto súčiastky budú súčasťou rodičovského kontajneru pokiaľ sa ich hráč nerozhodne zdvihnúť. Keďže hrá ma pôsobiť reálne, je potrebné aby na tieto súčiastky bola naviazaná fyzika, ktorá bude zabezpečená *Rigidbody* komponentom, ktorý je súčasťou Unity engine. Ďalej každá súčiastka bude obsahovať komponent, ktorý hlavne bude mať za úlohu zisťovania stavu, v akom sa súčiastka nachádza. Stavy môžu byť rôzne, ako napríklad súčiastka je v rukách hráča, súčiastka je na zemi alebo súčiastka je uložená v grid boarde.

Grid board bude reprezentovaný ako priestor, do ktorého môže hráč vkladať elektrické súčiastky. Grid board bude obsahovať niekoľko priestorov určených pre spomínané súčiastky. Na každom z týchto miest bude naviazaný komponent, ktorý bude slúžiť ako kontrolér pre vloženie súčiastky do tohto priestoru. Okrem iného tento komponent bude takisto obsahovať informáciu, či je tento priestor voľný alebo doňho už nejaká súčiastka bola vložená. Opäť súčasťou tohto prototypu nie je implementovaná logika reálneho grid boardu, takže jeden komponent pre každý priestor bol zatiaľ dostačujúci pre pointu tohto prototypu. Takisto každý úložný priestor musí obsahovať vlastný *collider*, ktorý bude kontrolovať, či sa nejaká elektrická súčiastka nachádza v jeho prostredí a ak áno, túto súčiastku si automaticky pripne do svojho priestoru. Tento prístup bude obzvlášť potrebný pri integrácií VR prototypu, aby hráč mohol elektrickú súčiastku jednoducho priložiť pri ľubovoľný voľný priestor a súčiastka sa doňho automaticky vloží.

Interakcia hráča s elektrickými súčiastkami ako aj grid boardom má niekoľko rôznych funkcií. Čo sa týka interakcie so súčiastkami, hráč bude môcť ľubovoľnú súčiastku zdvihnúť zo zeme a zobrať ju do rúk, následne ju bude môcť v rukách otáčať okolo jednej osi, tento smer bude definovať ako bude súčiastka vložená do gridu (vertikálne/horizontálne), hráč takisto môže súčiastku z rúk pustiť naspäť na zem. Aby hráč vedel, s ktorou súčiastkou práve interaguje, a teda, ktorú súčiastku po stlačení príslušného tlačidla zodvihne do rúk, daná súčiastka bude hráčovi zvýraznená. Pri interakcií s grid boardom je potrebné, aby hráč vedel, do ktorého priestoru bude súčiastka uložená po stlačení príslušného tlačidla. Rozhodli sme sa do implementovať prístup simulácie. Táto simulácia bude simulovať súčiastku, ktorú hráč bude držať v ruke do priestoru, na ktorý sa hráč díva. Simulácia súčiastky bude presnou kópiou stavu súčiastky, ktorú hráč drží v ruke. To znamená, že pokiaľ bude hráč túto súčiastku v ruke rotovať, bude sa rotovať aj daná simulácia v priestore na grid boarde.

4.1.3 Implementácia

Hierarchia scény predstavovala *Canvas*, *Room (priestor, v ktorom sa hráč nachádza), Player (hráč).*

Canvas v prvom prototype slúžil iba ako ukazovateľ kam sa hráč pozerá. Tento ukazovateľ predstavoval zelenú guličku v strede obrazovky.

Room objekt bol zložený z niekoľkých častí. Jednou bola podlaha, po ktorej sa hráč mohol pohybovať, druhá časť predstavovala grid board a posledná časť bol spomínaný kontajner pre všetky elektrické súčiastky. V tomto kontajneri sa nachádzali všetky súčiastky, ktoré neboli vložené do grid boardu, alebo ich hráč nedržal v ruke, to znamená, že pokiaľ nejakú zo súčiastok hráč pustil na zem, automaticky sa zaradilo do tohto kontajneru.

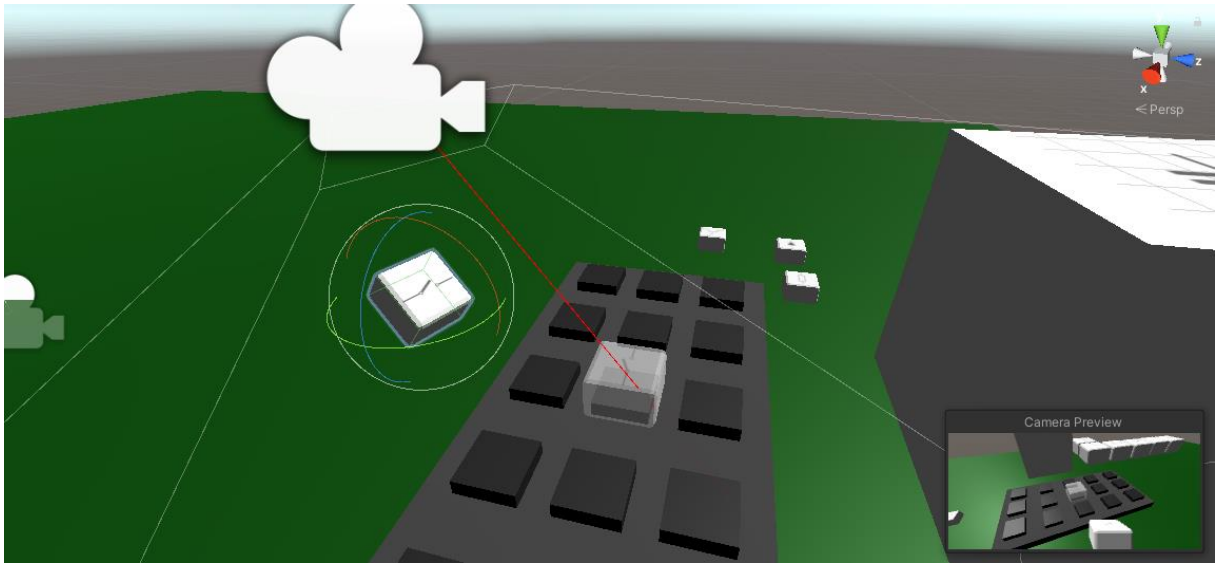
Player bol objekt, ktorý predstavoval daného hráča. Na tomto objekte bola naviazaná kamera, ktorá hráčovi renderovala obraz a takisto súčasťou tohto objektu boli ruky hráča ako aj model samotného hráča.

Následne vzniklo niekoľko komponentov, pričom každý komponent bol súčasťou príslušného *GameObject* objektu.

MovementController je komponent *Player* objektu, tento komponent je zodpovedný za pohybovanie sa hráča. Na základe hráčovho inputu tento kontrolér udáva objektu smer a rýchlosť, akou sa bude pohybovať. Takisto je tu sledované, či hráč stlačil príslušne tlačidlo pre skok, v tom prípade *Player* objekt vyskočí v hre. Tento komponent takisto pracuje s Unity komponentom *CharacterController*, ktorý je takisto súčasťou *Player* objektu. *CharacterController* slúži na simuláciu pohybovania pomocou *Move(Vector3 motion)* funkcie. *Motion* je tvorený všetkými hráčovými inputmi. V prípade, že hráč skočil a teda je vo vzduchu, do tohto *motion* sa takisto počíta aj nami zadefinovaná gravitácia.

HandsController je ďalší komponent *Player* objektu, tento komponent slúži ako kontrolér hráčových rúk. Súčasťou tohto komponentu sú funkcie ako *pickupItem(GameObject item)*, ktorý zdvihne objekt, na ktorý sa hráč pozerá a vloží mu ho do rúk. Ďalej obsahuje funkciu *dropItem()*, ktorá pustí objekt, ktorý hráč práve drží v ruke na zem. Funkcie ako *GameObject getGrabbedItem()* a *removeItem()* primárne slúžia pre logiku naviazanú na grid boarde, kde tento grid board potrebuje zistiť aký objekt hráč drží v ruke a prípadne mu tento objekt z ruky odstrániť, čím sa priamo vloží do jedného z priestorov na grid boarde. *HandsController* takisto slúži na rotáciu príslušného objektu, ktorý hráč drží. Hráč môže objekt rotovať 2 smermi, ale vždy iba na 1 osi. Iné osi nie sú potrebné pretože táto rotácia hovorí o tom, akou stranou bude daná elektrická súčiastka vložená do grid boardu.

InteractionController je zatiaľ posledný komponent *Player* objektu. Tento komponent je zodpovedný za interakciu hráča s prostredím, v ktorom sa nachádza. Zaznamenáva hráčov input ako aj jeho zorné pole a reaguje naň. Hlavnou súčasťou komponentu je *UnityEngine.Physics.Raycast* funkcionalita, ktorá zabezpečuje zaznamenávanie akýchkoľvek kolízii hráča s objektami. Je to akýsi laser pointer, ktorý kontroluje, či tento laser preťal nejaký objekt. Ak je objekt preťatý, vráti nám tento objekt, aby sme s ním mohli ďalej pracovať. Na Obrázok 10 - Ukážka Raycast funkcionalityObrázok 10 môžeme vidieť spomínaný *Raycast*, ktorý je vysielaný zo stredu kamery, a teda zorného poľa hráča. Keďže hráč drží v ruke elektrickú súčiastku a *Raycast* preťal jednu z voľných pozícií na grid boarde, táto súčiastka sa simuluje na danú pozíciu. V pravom dolnom rohu obrázku môžeme vidieť ako tento táto situácia vyzerá z pohľadu hráča.



Obrázok 10 - Ukážka Raycast funkcionality (červená čiara)

Raycast využívame na sledovanie hneď niekoľkých udalostí:

- Pokiaľ hráč stlačil ľavé tlačidlo na myši skontroluj či:
    - sa hráč pozerá na elektrickú súčiastku, v tom prípade ju hráč zodvihne, pokiaľ už nedrží v ruke inú súčiastku
    - sa hráč pozerá na voľné miesto na grid boarde, v tom prípade sa súčiastka uloží do daného miesta.
- Pokiaľ sa hráč pozerá na interaktívny objekt skontroluj či:
    - sa hráč pozerá na elektrickú súčiastku, pričom v ruke nedrží inú, v takom prípade zvýrazni túto súčiastku
    - sa hráč pozerá na voľné miesto na grid boarde, pričom v ruke drží elektrickú súčiastku, v takom prípade simuluj súčiastku na dané miesto

ItemController je komponent naviazaný na elektrické súčiastky. Komponent obsahuje informácie o danej súčiastke, ako napríklad či bola súčiastka vložená do grid boardu a ak áno, tento komponent si uloží *InsertController* komponent toho miesta, do ktorého bola vložená. Tento komponent takisto disponuje *private void OnTriggerEnter(Collider other)* funkcionalitou, ktorá je zodpovedná za sledovanie kolízií tohto objektu z ostatnými. Konkrétne sa sleduje, či je tento objekt v kolízii s voľným miestom na grid boarde a ak áno, tak sa do tohto miesta uloží. Súčasťou tohto komponentu je aj informácia o rodičovskom kontajnery, pretože každá súčiastka sa vždy nachádza v tomto kontajnery, výnimkou sú iba situácie, že táto súčiastka je uložená na grid boarde alebo ju hráč drží v ruke.

InsertController je posledný komponent 3. šprintu, ktorý má na starosti vkladanie elektrických súčiastok do voľných miest na grid boarde. Sú 2 typy, ako sa súčiastka môže dostať do grid boardu:

- Súčiastka bola do grid boardu vložená z rúk hráča po stlačení pravého tlačidla na myši
- Súčiastka zachytila kolíziu z voľným miestom na grid boarde a tým pádom sa do tohto miesta uložila automaticky

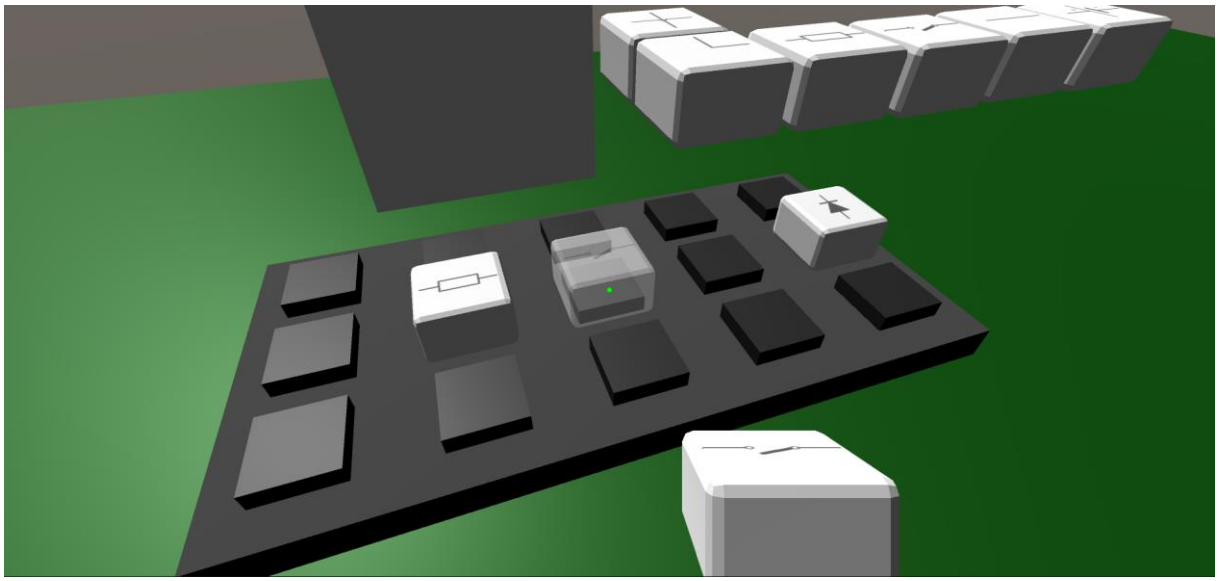
Následne pre oba prípady je použitá ďalšia metóda, ktorá zabezpečuje úspešne vloženie súčiastky do grid boardu. Súčasťou tejto metódy je takisto aj zaokrúhlenie *Z-osi* rotácie súčiastky k najbližšiemu násobku 90 stupňov, aby sa zachovalo správne usadenie súčiastky do grid boardu.

```
float itemCurrentRotationZ = Mathf.Round((insertedItem.transform.rotation.eulerAngles.z) / 90f) * 90f;
```

Poslednou časťou tohto komponentu je metóda, zodpovedná za vytiahnutie súčiastky z grid boardu, pokiaľ sa tak hráč rozhodol spraviť stlačením príslušného tlačidla.

4.1.4 Testovanie

Celé testovanie prvého prototypu prebiehalo v prostredí Unity, v ktorom naši testeri testovali všetky implementované funkcionality prototypu. Výsledok testovania nepredstavoval žiadne nájdené chyby v riešení a tým pádom testovanie prebehlo úspešne. Na Obrázok 11Obrázok 11 môžeme vidieť finálnu podobu prvého prototypu.



Obrázok 11 - Ukážka prototypu pre ovládanie a interakciu

ZÁVER

Po prvých troch šprintoch sa nám podarilo vypracovať dôležitú analýzu pre náš projekt. Na základe tejto analýzy sme si presne určili akú simuláciu budeme implementovať do nášho projektu. Konkrétne sa jednalo o implementáciu elektrických obvodov v blokovom (grid) riešení. Takisto sme sa ešte okrajovo venovali breadboard riešeniu, s ktorým vývojom nepokračujeme z viacerých dôvodov uvedených v dokumente, ale ako záložný plán sa nám do budúcnosti môže hodiť.

Výsledkom všetkých analýz bol prvý funkčný prototyp nášho projektu, kde sme sa primárne venovali implementácií pohybu a interakcií s objektami, konkrétne interakcia elektrických súčiastok s mriežkou (gridom).

V ďalších šprintoch budeme pokračovať na tvorbe nových prototypov, konkrétne integrácia VR do nášho projektu. Takisto na naše elektrické súčiastky začneme implementovať funkcionalitu poskytnutú *SpiceSharp* knižnicou. Vďaka tomu splníme naše globálne ciele pre zimný semester.