TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

INTERAKTIVNÍ SIMULÁTOR AUTOMATIZAČNÍCH PROCESŮ V 3D PROSTŘEDÍ UNITY

Diplomová práce

Studijní program: N2612 – Elektrotechnika a informatika
Studijní obor: 1802T007 – Informační technologie

Autor práce: **Bc. Tomáš Košek**
Vedoucí práce: Ing. Mojmír Volf

Liberec 2015

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: **2014/2015**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Tomáš Košek**
Osobní číslo:	**M13000194**
Studijní program:	**N2612 Elektrotechnika a informatika**
Studijní obor:	**Informační technologie**
Název tématu:	**Interaktivní simulátor automatizačních procesů v 3D prostředí Unity**
Zadávající katedra:	**Ústav nových technologií a aplikované informatiky**

Zásady pro vypracování:

1. Vyberte automatizační technologie vhodné pro implementaci do herních mechanismů
2. Ve zvoleném 3D vývojovém prostředí vytvořte algoritmy pro podchycení všech herních mechanismů
3. Vytvořte ergonomické uživatelské rozhraní aplikace
4. Vytvořte sadu edukativních scénářů včetně jednoho průvodce řešením vybraného zadání
5. Zdokumentujte vývoj, obsluhu a tvorbu uživatelských scénářů

Rozsah grafických prací: **dle potřeby**

Rozsah pracovní zprávy: **cca 40-60 stran**

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

[1] BLACKMAN, Sue. Beginning 3D game development with Unity 4: all-in-one, multi-platform game development. Second edition. Berkeley, Calif: Apress, 2013. ISBN 14-302-4899-8.
[2] THORN, Alan. Practical Game Development With Unity and Blender: all-in-one, multi-platform game development. Second edition. Berkeley, Calif: Cengage Learning, 2014. ISBN 13-050-7470-X.
[3] THORN, Alan. Pro unity game development with c#: all-in-one, multi-platform game development. Second edition. Berkeley: Apress, 2014. ISBN 14-302-6746-1.

Vedoucí diplomové práce: **Ing. Mojmír Volf**
Ústav nových technologií a aplikované informatiky

Datum zadání diplomové práce: **20. října 2014**
Termín odevzdání diplomové práce: **15. května 2015**

prof. Ing. Václav Kopecký, CSc.
děkan

Technická univerzita v Liberci - 14 -
L.S.

prof. Dr. Ing. Jiří Maryška, CSc.
vedoucí ústavu

V Liberci dne 20. října 2014

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu TUL.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé diplomové práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum: 14.5.2015

Podpis: Tomáš Kořík

Poděkování

Chtěl bych poděkovat svému vedoucímu diplomové práce ing. Mojmíru Volfovi za odborné vedení a také komunitě Unity za výborné vyučující materiály, díky kterým jsem se naučil dobře ovládat toto prostředí.

Abstrakt

Česká verze

Tématem této práce je vytvoření interaktivního simulátoru automatizačních technologií. Práce je vytvořena ve formě logické hry, která obsahuje několik zástupců zmíněných technologií. Hra využívá moderní engine Unity a je napsána v jazyce C#. Základem hry jsou jednotlivé komponenty, které představují prvky reálných zařízení. Tyto komponenty je pak možné propojit pomocí systému vstupů a výstupů. Jednou z komponent je i programovatelný elektronický obvod. Ten umožňuje hráči vytvořit obvod ve speciálním editoru podle svých potřeb. Cílem hry je pak splnit konkrétní úkoly vybraného scénáře, které zahrnují především přesun a třídění materiálu. Jednotlivé scénáře je možné vytvořit v editoru přístupném pro všechny hráče. Výsledná hra obsahuje i některé vzdělávací prvky, jako je například tvorba elektronických obvodů. Lze ji tak využít jako zábavnou pomůcku při výuce obvodové logiky.

Klíčová slova

Hra, Unity, C#, komponenty, elektronika

English version

Main focus of this thesis is to create interactive simulator of automatization technology. Thesis was made in form of logic computer game. The game itself contains several representants of automatization technology. The game is powered by Unity engine and written in C#. Foundations of the game are individual components which represents devices used in real life. These components are connected to each other via system of inputs and outputs. Programmable electronic circuit is one of these components. Players can program this circuit according to current needs in special editor. The goal of this game is to accomplish individual tasks defined by current scenario. These tasks are usually about sorting and moving material to specific area. These scenarios are created in editor which is accessible to all players. The final version of this game also contains some education elements: mainly the programming of electronic circuits. It is possible to use it as a entertaining instrument during lectures about circuit logic.

Keywords

Game, Unity, C#, components, electronics

Obsah

Seznam kapitol

1 Seznámení s použitými nástroji....10
1.1 Unity 3D....10
1.2 C#....11
1.3 Blender....11
2 Základy herního návrhu....13
2.1 Nápad....13
2.2 Nákresy....13
2.3 Rozvíjení detailů....13
2.4 Design dokument....14
3 Vybrané automatizační techniky....15
3.1 Programovatelný logický obvod (PLD)....15
3.2 Přepravní pás (pásový dopravník)....15
3.3 Laserový senzor....16
3.4 Průmyslová kamera....16
3.5 Tenzometrické váhy....16
3.6 Sýpka / zásobník....17
4 Vytvoření herního prostředí....18
4.1 Základní návrh....18
4.2 Prostředí....18
5 Uživatelské rozhraní....19
6 Vytvoření herních komponent....20
6.1 Návrh systému komponent....20
6.1.1 Vstupy a výstupy....20
6.1.2 Funkčnost komponenty....21
6.1.3 Chování komponent v režimu „běh“....23
6.2 Implementace jednotlivých technologií....24
6.2.1 Přepínač....24
6.2.2 Spínač....24
6.2.3 Elektronický obvod....25
6.2.4 Váhy....25
6.2.5 Přepravní pás....25
6.2.6 Pásová výhybka....26
6.2.7 Kamera....26
6.2.8 Sýpka / zásobník....27
6.2.9 Signální světlo....27
6.2.10 Displej....28
6.2.11 Pásové zarážky....28
6.3 Začlenění do hry....28
6.3.1 Uživatelské rozhraní obchodu....28
6.3.2 Umístění komponenty....29
6.3.3 Informace a editace komponent....29
6.3.4 Propojení komponent....31
7 Elektronické obvody....32
7.1 Struktura....32
7.2 Prvek obvodu „CircuitElement“....33

7.3 Funkčnost obvodů....34
8 Editor elektronických obvodů....35
8.1 Reprezentace prvku....35
8.2 Propojení prvků....36
8.3 Ovládání kamery editoru....37
8.4 Panel nových bloků....38
8.5 Režim informací a editace....38
8.6 Tvorba propojení....38
8.7 Editace propojení....39
8.8 Další možnosti editace propojení....39
8.9 Testování obvodu....40
9 Dokončení hry....41
9.1 Sběrač....41
9.2 Tvůrce....41
9.3 Struktura úkolů a podmínek....42
9.4 Běh hry....43
9.5 Vytvoření materiálu....44
10 Vytvoření editoru scénářů....45
10.1 Tvorba prostředí....45
10.2 Správa inventáře....46
10.3 Správa komponent....47
10.4 Sběrače a tvůrci....47
10.5 Nastavení scénáře....48
11 Ukládání a načítání dat....50
11.1 Problém 1: Vector3, Vector2 a Quaternion....50
11.2 Problém 2: MonoBehaviour....50
11.3 Problém 3: GameObject....51
11.4 Problém 4: kopírování objektu....51
12 Hlavní menu....52
13 Návod k použití....53
13.1 Hra....53
13.1.1 Pohyb po mapě....53
13.1.2 Obchod / inventář....53
13.1.3 Přesun komponent....54
13.1.4 Propojení komponent....54
13.1.5 Editace komponent....55
13.1.6 Editor obvodů....55
13.1.7 Spuštění běhu hry....56
13.2 Editor....57
13.2.1 Tvorba prostředí....57
13.2.2 Komponenty....57
13.2.3 Tvůrci a sběrače....57
13.2.4 Nastavení úrovně....58
14 Závěr....59
14.1 Cíle práce....59
14.2 Postup....59
14.3 Výsledný stav práce....59
14.4 Budoucnost....60
15 Seznam použitých zdrojů....61

Seznam obrázků

Obrázek 1: Ukázka tvaru dílků....18
Obrázek 2: Ukázka uživatelského rozhraní během hry....19
Obrázek 3: Ukázka komponenty (sýpka) a jejích vstupů/výstupů....21
Obrázek 4: Znázornění jednotlivých fází běhu komponenty (obvod s AND hradlem)....23
Obrázek 5: Přepravní pás v pohybu a jeho náklad....25
Obrázek 6: Pásová výhybka ve stavech „vypnuto“ a „zapnuto“....26
Obrázek 7: Editace přepravního pásu....30
Obrázek 8: Ukázka propojení přepínače a transportního pásu....31
Obrázek 9: Kompletní obvod vytvořený v editoru obvodů....35
Obrázek 10: Ukázka tří vytvořených prvků (vstup, AND hradlo a JK klopný obvod)....36
Obrázek 11: Obvod v testovacím režimu (červené spoje značí log. 1)....40
Obrázek 12: Ukázka stromové struktury úkolů a podmínek....43
Obrázek 13: Vytváření textury pomocí "Substance Textures"....44
Obrázek 14: Editor v režimu "Tvorba mapy"....46
Obrázek 15: Editace konkrétní sady materiálů u tvůrce....48
Obrázek 16: Editace sady úkolů....49
Obrázek 17: Výběr úrovně k editaci v hlavním menu....52
Obrázek 18: Kompletní uživatelské rozhraní u editoru elektronických obvodů....56

Úvod

Hra, které je věnována tato diplomová práce, vychází z komerční hry Minecraft. Minecraft je 3D voxelová Sandbox hra, což znamená, že herní část není tak svázaná a nabízí velkou volnost ve způsobu hraní. Samotné prostředí je složené z jednotlivých bloků, které je možné herními činnostmi ovlivnit. Součástí Minecraftu je systém „Redstone“, který je obdobou reálné elektroniky a umožňuje ve hře vytvořit elektronické systémy (hradla, klopné obvody, 8-bitový procesor). Hlavním problémem je zde ale pouze velmi malý počet stavebních prvků. Výsledné obvody pak zabírají příliš velkou plochu na to, aby byly praktické.

Téma počítačové hry jsem si vybral také proto, protože se chystám na kariérní dráhu herního vývojáře. Koncepci mojí diplomové práce jsem tedy založil na mechanické a elektronické části Minecraftu. Během školních let jsem navíc nenarazil na žádný slušný simulátor elektronických obvodů, proto jsem tuto oblast také zakomponoval do své práce. Ve výsledku tak samotná hra získala i svoji vzdělávací část.

Ideou hry je její rozdělení do jednotlivých scénářů, které jsou na sobě vzájemně nezávislé. Scénář pak určuje herní prostředí, cíle i vybavení. Pro jednoduchost jsou úkoly jednotlivých scénářů založeny na přesunu materiálu ze startovacích bodů do určených cílů (ty jsou určeny scénářem). Tyto scénáře jsou vytvořeny v editoru scénářů, který je tak součástí této práce a je přístupný i hráčům.

Prvním cílem práce bylo vybrat vhodný engine pro vytvoření hry. Byla zvolena jsem Unity, protože i její bezplatná verze je dostatečně bohatá na funkčnost a je multiplatformní. Obsahuje velmi dobrý fyzikální model, který lze využít pro simulaci pohybu materiálu. Prostředí Unity zajišťuje i renderování hry nebo detekce kolizí a usnadňuje tedy samotný vývoj. Unity podporuje několik programovacích jazyků, ale všechny mají v rámci Unity stejné vyjadřovací schopnosti. Pro tuto diplomovou práci jsem vybral jazyk C#.

Dále jsem navrhl herní prostředí, ve kterém se budou jednotlivé herní scénáře odehrávat. Zvolil jsem tradiční model 3D Voxelových her, který jsem navíc obohatil o možnost natočit jednotlivé bloky.

Jednotlivé kusy vybavení v daném scénáři jsem nazval komponentami. Komponenty jsou tedy všechny akční členy výsledného zařízení (spínače, senzory, elektronické obvody). Hráč pak může v rámci scénáře manipulovat pouze s komponentami. Každá komponenta má určitý počet vstupů a výstupů. Tyto vstupy/výstupy pak slouží k propojení jednotlivých komponent do funkčního celku.

Dalším krokem bylo vybrat reálné technologie, které bude možné přenést do samotné hry. Vybíral jsem takové, které není obtížné implementovat a které obohacují nějakým způsobem hratelnost. Některé komponenty fungují prakticky stejně jako jejich reálné protějšky, některé byly více pozměněné (například sýpka).

Protože jsem chtěl, aby bylo možné vytvořit jakýkoliv elektronický obvod, vytvořil jsem komponentu elektronický blok a také speciální editor, kde je možné obvod vytvořit. Hráči je tak k dispozici široká paleta funkčních bločků (například hradla a klopné obvody), které může umístit do elektronického obvodu a navzájem pospojovat. Součástí tohoto editoru je i možnost testování vytvořeného obvodu.

Posledním krokem bylo vytvoření vstupních a výstupních míst pro materiál a také cíle scénáře. Tyto vstupní a výstupní místa jsem vytvořil tak, aby byly snadno nastavitelné a bylo možné vytvořit scénáře přesně podle potřeb. Po úkolech scénáře jsem chtěl, aby byly dostatečně komplexní, takže není problém vytvořit daný scénář přesně podle potřeb tvůrce.

Celý proces tvorby této hry pak zahrnoval neustálé testování hratelnosti, kódu a odstraňování chyb. Samostatnou kapitolou bylo vytvoření ukládání a načítání herních dat ze souboru.

1 Seznámení s použitými nástroji

1.1 Unity 3D

Unity 3D je multiplatformní herní engine použitelný pro tvorbu dvourozměrných i trojrozměrných her. Jednotlivé projekty je možné exportovat pro 21 různých platforem včetně Linuxu, mobilních platforem a konzolí. Aktuální verze 5 (rok 2015) vychází ve dvou základních verzích: Unity Pro a Unity Personal. Unity Personal je volně dostupná a výsledné aplikace je možné prodávat. Pokud jednotlivec nebo celá skupina vydělává ročně na těchto hrách 100 000$ nebo více, musí si koupit Unity Pro. Obě verze obsahují všechny funkce editoru. Unity Pro pak obsahuje několik pomůcek pro usnadnění práce, týmové licence a možnost Cloudu.

Hlavními funkcemi Unity 3D jsou:

- multiplatformnost (včetně nových platforem jako WebGL a Oculus Rift),
- podpora více programovacích jazyků (JavaScript, C# a Boo),
- možnosti skriptování (podpora .NET, práce s internetovými prohlížeči, práce se skripty atd.),
- fyzikální systémy NVIDIA PhysX 3 a Box2D,
- podpora pro 3D i 2D aplikace,
- optimalizace,
- systém animací (stavové stroje, blend trees, vrstvení atd.),
- grafické funkce (částicové systémy, fyzikálně založené stínování, post-processing efekty atd.),
- systém uživatelského rozhraní,
- audio funkce (možnost svázání s animací, nativní pluginy, ovládání přes skripty atd.).

Unity také obsahuje přístup k tzv. Asset Store, což je kolekce různých stavebních prvků pro jednotlivé projekty: textury, modely, zvuky, skripty, animace a další. Do Asset Store může přispívat každý uživatel Unity a za svůj příspěvek si může účtovat cenu. Existuje ovšem veliké množství příspěvků, které jsou kvalitní a zdarma.

1.2 C#

C# je vysokoúrovňový objektově orientovaný programovací jazyk od společnosti Microsoft. C# je založen na programovacích jazycích Java a C++. Jazyk samotný poskytuje podporu pro principy softwarového inženýrství: hlídání hranic polí, neinicializovaných proměnných a garbage collector. C# je také dobře přenositelný a je tak vhodný pro vývoj softwarových komponent v různých prostředích. I když se programy napsané v C# chovají velmi optimalizovaně, programy v C nebo v jazyce symbolických adres pracují rychleji.

Důležitými vlastnostmi jazyka C# jsou:

- Neexistuje zde vícenásobná dědičnost. Třída ovšem může implementovat neomezený počet rozhraní.
- C# nepodporuje globální proměnné nebo metody (vše musí být definováno uvnitř tříd). Částečně lze konceptu globálních proměnných či metod dosáhnout pomocí statických metod a proměnných.
- Z důvodu zapouzdření se často používají tzv. Property, které definují Get a Set funkce pro přístup k datovému atributu.
- Oproti C++ je C# typově bezpečnější. Převody mezi datovými typy jsou možné jen směrem z odvozeného typu na rodičovský (například Short na Integer).
- Oproti jazyku C není potřeba provádět dopřednou deklaraci a nezáleží na pořadí metod.
- C# je case sensitive (rozlišuje velká a malá písmena).

1.3 Blender

Blender je multiplatformní open-source program pro práci s 3D. Podporuje kompletní škálu nástrojů pro práci s 3D: tvorba modelů, animace, tvorbu kostry (rigging), simulace, renderování, editace videa i možnost tvorby her. Blender je vytvořen v jazyce Python s pomocí OpenGL a podporuje tak kromě systémů s Windows i Linux a Macintosh. Pomocí dalších Python skriptů je možné vytvořit pluginy, které funkcionalitu Blenderu rozšiřují. Pokud jsou pluginy dobré, mohou se objevit v další oficiální verzi Blenderu.

Blender je zcela zdarma a je možné ho využívat k jakémukoliv účelu (definováno v licenci Blenderu GNU General Public Licence). Je tak vhodný jak pro jednotlivce, tak i pro malá studia.

Blender samotný je vyvíjen svojí komunitou, každý může pomoci s vývojem (open source). Je tak aktivně vyvíjen několika sty vývojářů z celého světa.

2 Základy herního návrhu

Oblast herních návrhů není nijak standardizovaná, proto existuje mnoho různých přístupů. Někteří tvůrci věnují v této fázi velké množství času, někteří zase naopak téměř žádný. Záleží také na tom, zda je tvůrce návrhu zaměřen spíše více na samotný příběh a vyprávění, nebo je více zaměřen na systém a hratelnost. Obecně ale existuje několik kroků, které jsou tvůrci her ve většině případů dodržovány.

2.1 Nápad

Prvním krokem je vždy nápad. Je potřeba nejdříve načerpat inspiraci pro novou hru. Tvůrce hry se může inspirovat událostmi z každodenního života, filmů, knih, jiných her nebo dokonce i snů. Často se tak dělají setkání týmu, kde probíhá shromažďování nápadů (brainstorming). Z jednotlivých nápadů se následně vybere ten nejzajímavější (ale proveditelný). Pro tvorbu her je důležité znát cílové publikum a přizpůsobit tomu výběr nápadu. Také je dobré podívat se po konkurenčních hrách a přijít na to, čím bude tato nová hra lepší nebo odlišná. Dalším důležitým hlediskem je také časová a finanční náročnost.

2.2 Nákresy

Pro správnou realizaci nápadu je vhodné vytvořit doprovodné ilustrace nebo skici, které mohou zobrazovat herní prvky, úrovně nebo postavy. Ty pak umožní tvůrci si představit konkrétní herní situace a podle toho pak upravit následující kroky. Skici se často doprovázejí pár řádky nebo odstavci doprovodného textu, který umožní práci se skicou i s větším časovým odstupem. Skici nemusejí být krásné nebo úpravné, slouží především k rozvinutí nápadu. Také je možné si vytvořit první nástin případných animací jednotlivých postav nebo předmětů hry.

2.3 Rozvíjení detailů

Tento krok představuje nejtěžší část, protože je nutné vytvořit konkrétní herní svět. V tomto kroku je potřeba vymyslet všechny aspekty hry: příběh, postavy, předměty, herní mechaniky, události, zvukový doprovod a další. Cílem je důkladně popsat všechny prvky a pravidla, které budou tvořit herní universum. Jednotlivé události a chování hry by pak neměly porušovat pravidla tohoto vytvořeného světa.

Jednotlivé detaily je nutné vymyslet v této fázi, protože je tak mnohem snadnější udržet celistvost celé hry. Každý detail hry, který se nenaplánuje dopředu, se musí vkládat do hry během

samotné tvorby. To pak může vést k rozbití souvislostí herního světa a hráči se tak hůře ponoří do herního zážitku.

Základem práce vývojáře je definování vztahu hráče a hry. Jakým způsobem hru ovládá, jak ji může ovlivnit, jaké jsou jeho herní možnosti, kolik hráčů ji může současně hrát a další. Pokud hráč ovládá nějakou postavu, je potřeba vymyslet jak se s ní může ztotožnit. Vymyslet tak minulost postavy, její motivace a povahu. Dalším prvkem je vymyšlení všeho vybavení nebo schopností, které může mít během hry k dispozici.

Kromě těch nejjednodušších hříček má každá hra nějaký příběh. Detailní rozepsání a propracování příběhu je základním kamenem pro další práci – definování cílů hry a možností, jak jich dosáhnout, a definování role hráče. Součástí příběhu jsou také cíle hry, které se musí promyslet. Jaké jsou cíle hry, jak je možné jich dosáhnout, proč je potřeba jich dosáhnout, co brání hráči v dosáhnutí cíle a podobně.

2.4 Design dokument

Design dokument je souhrn všech informací o vyvíjené hře a připomíná tak scénář k filmu. I v případě jednoduchých her se tento dokument snadno rozroste do desítek stran. V dokumentu se roztřídí a uloží všechny detaily vzniklé v předchozích krocích. Někteří proto vytvářejí design dokument přímo už při tvorbě prvních detailů hry. Při vytvoření dobrého design dokumentu lze mnohem lépe odhalit případnou necelistvost herního světa a udělat úpravy.

Čím více péče se vloží do tvorby design dokumentu, tím snadnější je pak další práce na hře. Čím méně změn se dělá během vývoje hry, tím lépe drží herní svět pohromadě. Je samozřejmě možné provádět změny i v průběhu vývoje, ale v tom případě je potřeba důkladně změnu promyslet tak, aby hladce zapadla do vytvořeného světa. Například ve fantasy hře není problém v polovině hry přidat nové kouzlo s unikátními účinky či použitím, ale není dobré zavézt magii do hry, která magii neobsahuje, v pozdějších fázích vývoje – je pak potřeba vše zpětně předělávat a může tak nastat celá řada problémů.

3 Vybrané automatizační techniky

V této kapitole jsou shrnuty automatizační technologie, které byly vybrány a aplikovány do samotné hry. Jejich herní pojetí se může lišit od skutečnosti (pro zachování dobré hratelnosti), ale základní princip a účel zůstal zachován. Jednoduché prvky typu spínač a přepínač zde uvedeny nejsou.

3.1 Programovatelný logický obvod (PLD)

Jedná se o elektronickou součástku, která nemá na rozdíl od např. hradel definovanou funkčnost během jeho výroby. Před použitím součástky je nutné obvod nejdříve naprogramovat. Dva základní používané typy jsou CPLD (komplexní programovatelné logické obvody) a FPGA (programovatelná hradlová pole).

CPLD je spojení několika obvodů typu PAL (programovatelné logické pole) nebo GAL (programovatelné hradlové pole) pomocí programovatelných propojovacích polí. V dnešní době obvody CPLD dokáží zastat práci několika set tisíc logických hradel. Většinou se tyto obvody programují pomocí speciálního rozhraní (například s pomocí JTAG) po zapájení do desky plošných spojů.

FPGA se od CPLD liší v tom, že po zapnutí napájení se musí nejdříve nahrát jejich konfigurace z paměti (EEPROM nebo FLASH). Pokud je paměť součástí obvodu, programuje se podobně jako CPLD po zapájení. V případě, že tato paměť není dostupná (větší FPGA) a je proto nutné je po každém obnovení napájení znovu nakonfigurovat. V případě těchto velkých obvodů také bývá možnost zapojit i procesory.

3.2 Přepravní pás (pásový dopravník)

Jedná se o poddruh dopravníku (zařízení pro nepřetržitý pohyb daného materiálu). Tento stroj se skládá obvykle z nosné konstrukce s otočnými válci, které podpírají a otáčejí většinou pryžový pás. Hlavní účel tohoto dopravníku je transport sypkého materiálu (zemina, hornina) nebo menších předmětů (balíky, výrobky).

Důležitá vlastnost konkrétní konstrukce je směr přepravy: vodorovný nebo částečně vertikální (šikmý) – ty jsou často doplněny o speciální kapsy nebo zarážky pro zabránění sklouznutí materiálu. Samotný pohon pásu je obvykle řešen asynchronním elektromotorem skrze poháněcí buben.

3.3 Laserový senzor

Laserové senzory se nejčastěji používají pro měření vzdálenosti nebo jako světelná závora. V obou případech pracují na principu vyslání tenkého laserového paprsku a jeho následnou detekcí (většinou odraženého).

Světelná závora pracuje se dvěma částmi: přijímačem a vysílačem. Vysílač vysílá laserový paprsek a přijímač pak tento paprsek detekuje a reaguje na aktuální stav. Přijímač tak dokáže detekovat částečné nebo úplné zakrytí tohoto paprsku. Některé typy závor mají přijímač i vysílač dohromady v jedné součástce a ve druhé součástce je umístěn pouze reflexní prvek, který se postará o odraz světla.

Měřič vzdálenosti pracuje na principu vyslání laserového paprsku a následným měřením času do přijmutí odrazu. Protože rychlost světla je konstantní, není problém tuto vzdálenost změřit (ale záleží zde hodně na přesnosti časovače).

3.4 Průmyslová kamera

Jedná se o specializovaný druh běžné digitální kamery. Většinou se vyznačují vysokou citlivostí, rychlým snímkováním, konstrukcí pro připevnění a často i možností snímání v infračerveném spektru. V případě umístění kamery jako kontrolního prvku v rámci automatizace, zachycený obraz je okamžitě posílán do počítače, kde je zpracován pomocí předem daného postupu. Zpracování může zahrnovat například prahování, detekci vzorů nebo defektů. Výsledek zpracovaného obrazu pak ovlivní další průběh daného automatizačního procesu (například vyřazení vadného výrobku).

3.5 Tenzometrické váhy

Stejně jako ostatní druhy vah slouží ke změření hmotnosti objektů umístěných na horní části. Měření je dosaženo změřením deformace způsobené tíhou objektů pomocí piezoelektrickému jevu. Tento typ vah se vyznačuje vysokou přesností. Rozsah hmotností je možné dále rozšířit pomocí mechanických převodů, které snižují deformační síly. Rozsah se tak může pohybovat od mikrogramů po desítky tun. Protože se jedná o elektronické váhy, je snadné jejich propojení s počítačem, který se postará o zpracování naměřených hodnot a může automatizační proces upravit podle svých potřeb (může vyřadit z produkce výrobek s hmotností mimo limit).

3.6 Sýpka / zásobník

Je to velký kontejner ve tvaru obrácené pyramidy (obvykle z oceli), který se používá v průmyslu především k uskladnění hmoty získané ze vzduchu. Obvykle se používají ve velkých skupinách pro lepší sběrnou schopnost. Často se tak používají jako sběrače prachu a filtrační zařízení. Hmota do zásobníku obvykle vstupuje pomocí speciálního zařízení podle typu sbírané hmoty. Jakmile je zásobník plný (nebo pokud je potřeba vydat hmotu dříve), otevře se výstup zásobníku na dně. Stěny mají většinou sklon kolem 60°, který umožňuje hladký výdej materiálu. Zásobník je často kombinován s dopravníkovými pásy.

4 Vytvoření herního prostředí

4.1 Základní návrh

Celý koncept hry se je postaven na předem připravených scénářích. Proto jsem celou hru i editor rozvrhl do 3 základních vrstev: prostředí, komponenty a úkolová část. Při samotné hře je pak hráč schopen přímo manipulovat pouze s komponentovou vrstvou a nepřímo i s úkolovou.

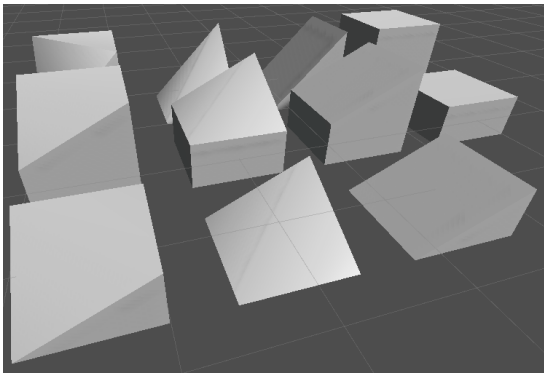
4.2 Prostředí

Pro vytvoření prostředí jsem zvolil voxelový políčkový systém (jaký má například známý Minecraft). Celé prostředí je složeno z jednotlivých předem připravených tvarů umístěných do pravidelné prostorové mřížky. Na každém bodě mřížky pak může, ale nemusí, být právě 1 dílek. Vzdálenosti mezi jednotlivými body jsem zvolil stejně pro všechny tři osy: 0,5 metru. Každý dílek se tedy vejde do krychle o hraně 0,5 metru a tím na sebe jednotlivé dílky přesně navazují. Velikost celé mapy se pak určuje automaticky podle umístěných dílků.

Aby bylo možné vytvořit dostatečně komplexní tvary, je potřeba mít k dispozici dostatek různých dílků. První možností je vytvořit všechny možné tvary v jednotlivých rotacích a ty umísťovat. To by bylo velmi neefektivní a limitovalo by to tvůrce scénářů. Druhá možnost je vytvořit pouze základní tvary a umožnit pak jejich rotaci. Tímto směrem jsem se vydal a jednotlivé tvary vytvořil v Blenderu editací vertexů základní kostky a následně exportoval a zvětšil na potřebnou velikost.

Při tvorbě prostředí si tak tvůrce může vybrat z 11 různých tvarů. Každý jednotlivý dílek může být natočen ve všech třech osách o násobky 90 stupňů – tím vzniká až 24 možných různých rotací na dílek. Další vlastností dílku je jeho materiál, který určuje, jakou bude mít samotný tvar texturu.

Skripty vytvářející toto prostředí jsou pak obsaženy ve skriptech „CreatorControl.cs“ (editor) a „GameControl.cs“ (hra). V první fázi vývoje se prostředí načítalo přímo z jednotlivých příkazů ve skriptu. Po vytvoření editoru a propojení s hrou se prostředí načítá z uloženého souboru.

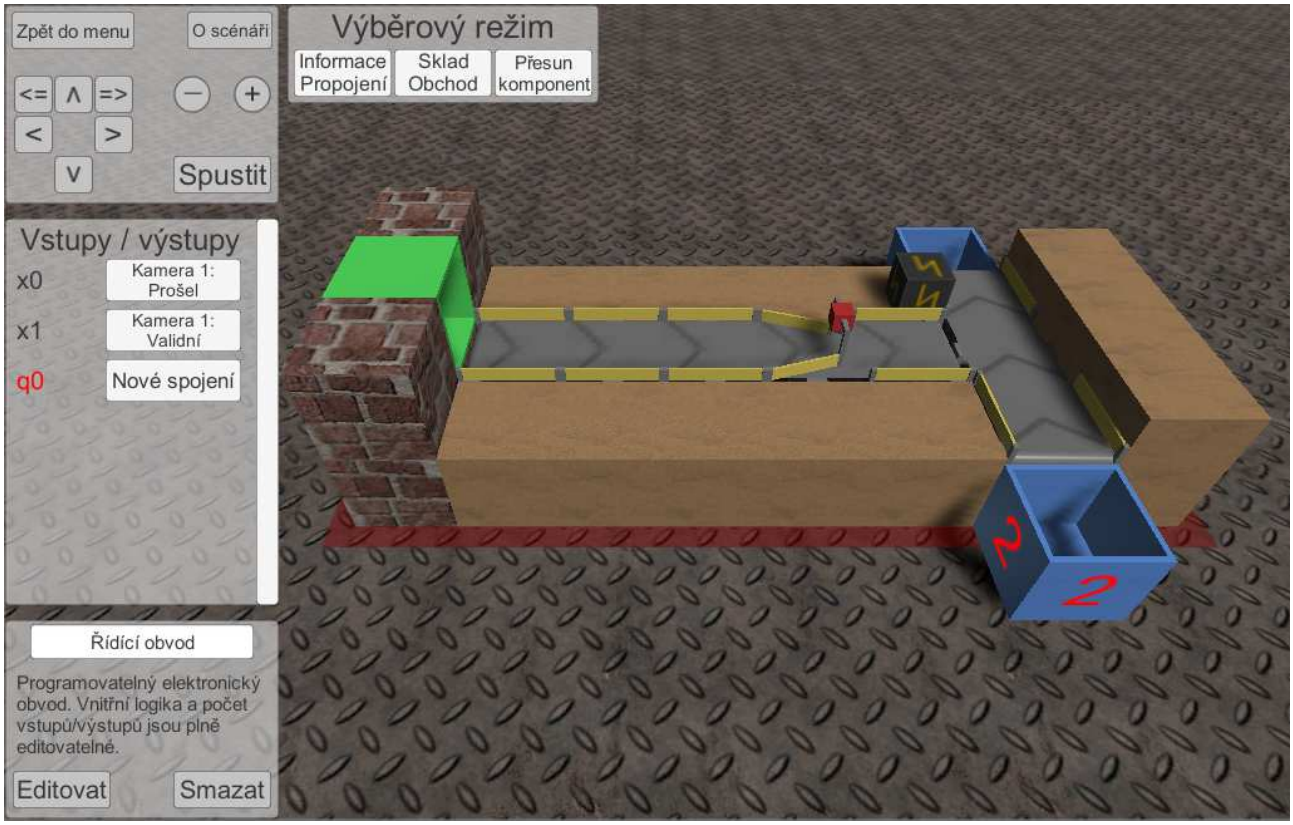


Obrázek 1: Ukázka tvaru dílků

5 Uživatelské rozhraní

Pro tvorbu uživatelského rozhraní jsem zvolil nativní systém v Unity (nový od verze 4.6). Výhodou je dobrá dokumentace a velká flexibilita. Podle potřeby je možné vytvářet jednotlivé prvky s pevnými rozměry nebo s rozměry poměrovými k velikosti rodičovského prvku či celé obrazovky. Tím pádem je výsledné uživatelské rozhraní nezávislé na použitém rozlišení uživatele (pokud si nezvolí velmi nízké rozlišení). Systém už obsahuje všechny prvky, které budou potřebné pro tuto hru (tlačítka, zaškrtávací políčka apod.).

Samotné uživatelské rozhraní jsem rozdělil do jednotlivých tématických panelů. Každý má na starost jednu konkrétní oblast ovládání (například pohyb kamery po mapě). Protože na celé ovládání hry je potřeba mnoho prvků, je hra rozdělena do jednotlivých módů, mezi kterými lze libovolně přepínat skrze jeden z hlavních panelů. Podle aktuálního módu se pak jednotlivé panely znevidetelňují, aby na ploše zůstaly pouze ty relevantní k aktuální situaci.



Obrázek 2: Ukázka uživatelského rozhraní během hry

6 Vytvoření herních komponent

6.1 Návrh systému komponent

Komponenty představují všechny akční členy v úrovni, které jsou nastavitelné hráčem. Pro snadnější implementaci i ovládání jsem se snažil o co nejvíce univerzální návrh. Všechny komponenty pak dědí od společné třídy („GameComponent“), která definuje celou řadu vlastností a metod. Díky tomu mohou být všechny komponenty na scéně uloženy do společného seznamu a lze snadno volat potřebné společné metody bez znalosti toho, o jaký objekt se jedná.

Základními vlastnosti komponenty jsou:

- pozice a rotace,
- odkaz na samotný objekt na scéně a jeho řídící skript,
- identifikační číslo,
- jméno,
- list vstupů a výstupů.

6.1.1 Vstupy a výstupy

Vstupy a výstupy jsou určeny k propojení jednotlivých komponent. Jejich hodnoty pak ovlivňují chování samotné komponenty. Vstupy i výstupy jsou představovány společnou třídou - „InputOutput“.

Každý vstup či výstup je definován následujícími vlastnostmi:

- zda se jedná o vstup nebo výstup,
- počet bitů (aktuálně existují 1bitové a 8bitové prvky),
- výchozí hodnota (pokud není napojen na jiný prvek),
- jméno,
- list propojení.

Každý typ komponenty pak definuje, kolik vstupů nebo výstupů má a jaké jsou jejich vlastnosti. Elektronický obvod je jedinou výjimkou, protože u něj si počet vstupů a výstupů definuje

hráč. Během běhu hry má konektor 2 hodnoty: aktuální se kterou pracuje („value“) a budoucí hodnotu („nextValue“), která se nastavuje během metody „Run“ u komponenty. Pokud nedojde k nastavení hodnoty, bere se následující hodnota jako 0. Hráč pak může propojit jednotlivé vstupy a výstupy podle svého uvážení, ale zároveň musí dodržet několik pravidel:

- oba konektory musí být různého typu (vstup a výstup),
- oba konektory musí mít stejný počet bitů,
- výstup může vést na libovolný počet vstupů,
- každý vstup může být napojen pouze na jediný výstup.

Samotné propojení (třída „Connection“) je pak uloženo na obou kontaktech a obsahuje potřebné identifikátory (ID komponenty a ID vstupu/výstupu). V případě elektronických obvodů pak navíc obsahuje informace o grafické reprezentaci v editoru obvodů (více informací dále v kapitole 7 Elektronické obvody).

Obrázek 3: Ukázka komponenty (sýpka) a jejích vstupů/výstupů

6.1.2 Funkčnost komponenty

Každá komponenta je potomkem třídy „GameComponent“. Tato třída obsahuje celou řadu společných metod, které jsou pak využívány všemi komponentami. Obsahuje normální metody, abstraktní metody, virtuální metody i statické metody. Zde je několik těch důležitějších:

- „Initialization“: inicializace základních proměnných jako u konstruktoru.

- „ComponentParameter“: vrací parametr komponenty. Konkrétní komponenta si pak určí, o jaký údaj se jedná (barva světla, délka pásu, zpoždění tlačítka...).
- „ActualizeInputs“ a „ActualizeOutputs“: aktualizuje hodnoty na svých vstupech a výstupech.
- „ConnectInputOutputs“: vytvoření propojení z vybrané komponenty s vybraným cílem. Metoda vytvoří propojení na obou konektorech.
- „RemoveConnection“: zrušení propojení u vybrané komponenty. Dojde ke smazání propojení u obou konenktorů.
- „RemoveInputOutput“: smazání vstupu nebo výstupu. Před samotným smazáním se nejdříve projdou všechna napojení na daný konektor a provede se u nich metoda „RemoveConnection“ popsaná výše. Následně se projdou všechna zbývající propojení do vybrané komponenty a sníží se identifikátory kontaktů vyšších, než je právě mazaný. Nakonec je se samotný kontakt smaže.
- „ResetComponent“: v základu nedělá nic, jedná se o akce prováděné při startu nebo restartu běhu hry. Konkrétní akce jsou definovány jednotlivými druhy komponent.
- „ResetInputOutputs“: nastavení vstupů a výstupů do výchozích hodnot.
- „Interaction“: akce prováděné při stisku tlačítka myši na danou komponentu během běhu hry. Akce si definují jednotlivé komponenty.
- „Run“: tato metoda určuje, co dělá konkrétní komponenta za běhu hry. Veškerý kód si určuje konkrétní komponenta sama. Jedná se o změny na fyzickém objektu na scéně, úpravy hodnot na výstupech a podobně.
- „RemoveComponent“: statická metoda starající se o úplné odstranění komponenty ze hry. Musí odstranit všechna napojení na vybranou komponentu s použitím metody „RemoveConnection“ a následně snížit všechny vyšší indexy komponent i propojení. Nakonec se odstraní fyzický objekt ze scény a skript komponenty se odstraní z listu komponent.

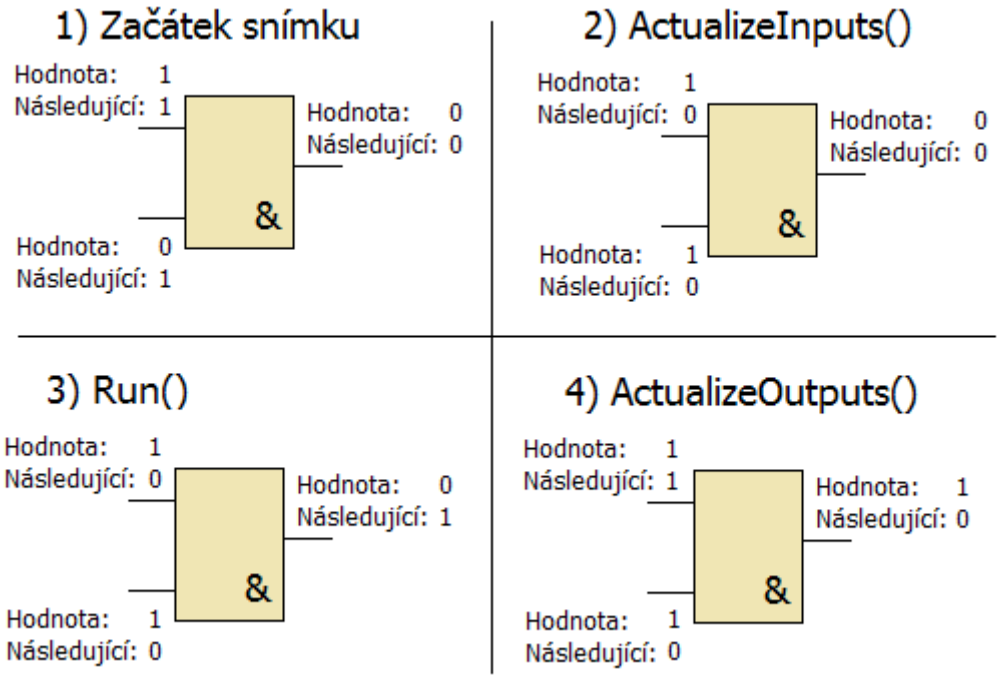
Samotný fyzický objekt na scéně obsahuje skript „ComponentAvatar.cs“ nebo nějaký z jeho potomků. Tyto skripty slouží k provázání s řídícím systémem komponenty, proto obsahují identifikátory ke komponentě a odkaz na hlavní řídící algoritmus hry či editoru („GameControl.cs“ nebo

„CreatorComponents.cs“). Základní činností skriptu je registrace kliknutí myši na objekt a následné předání události skriptu komponenty nebo řídícímu systému.

6.1.3 Chování komponent v režimu „běh“

O správu komponent a některé údaje o komponentách se stará skript „ComponentSystem.cs“. Pokud je hra v režimu „běh“, tak skript provádí každý snímek (obvykle 60 krát za vteřinu) následující akce:

1. Každá komponenta si zavolá metodu „ActualizeInputs“. Tím se do aktuální hodnoty u vstupů zkopíruje následující hodnota nastavená v předchozím snímku. Nová následující hodnota se nastaví na 0.
2. Každá komponenta zavolá metodu „Run“. Zde proběhne zpracování vstupů a případně se nastaví následující hodnoty pro výstupy. Aktuální hodnoty se nemění, aby nedošlo k tomu, že by některé prvky pracovaly s různou hodnotou, přestože pochází od stejného prvku.
3. Každá komponenta provede metodu „ActualizeOutputs“. Nejdříve se aktualizuje aktuální hodnota u výstupů na následující (nastavenou v tomto snímku metodou „Run“) a následně se projde list propojení a nastaví se následující hodnota napojených konektorů na aktuální hodnotu.



Obrázek 4: Znázornění jednotlivých fází běhu komponenty (obvod s AND hradlem)

Tento systém zaručí, že se hodnoty nebudou měnit během činnosti komponent, ale pouze na začátku a konci snímku. Také to znamená, že se signál nešíří okamžitě, ale je u každé komponenty o snímek zpožděn. Pokud by hráč například zřetězil 6 komponent za sebou, tak by trvalo 6 snímků (desetina vteřiny), než by poslední komponenta zaregistrovala změnu. Vzhledem k rychlosti snímkování a alternativnímu řešení skrze elektronické obvody to ale nepředstavuje problém.

6.2 Implementace jednotlivých technologií

Každou komponentu jsem vybral z existujících technologií. Některé jsem pak lehce poupravil, aby byly hratelně zajímavé a použitelné. Jednotlivé komponenty jsou pak rozděleny do jednotlivých tématických skriptů:

- „ControlDevices.cs“: přepínač a spínač.
- „Circuit.cs“: elektronický obvod.
- „RotorDevices.cs“: přepravní pás, pásová výhybka, pásové zarážky a váhy.
- „OtherDevices.cs“: světlo, displej, sýpka a kamera.

6.2.1 Přepínač

Jednoduchý kontrolní blok s jedním 1-bitovým výstupem. Během běhu hry lze kliknutím na přepínač změnit výstupní hodnotu. Aktuální hodnota na výstupu se pak přenáší na „SwitchScript.cs“ umístěný na fyzickém objektu. Ten se oproti základnímu skriptu navíc stará o správné natočení páčky v závislosti na aktuální hodnotě. Samotný model jsem vytvořil kombinací modelu z Blenderu (základna) a kapsle přímo v Unity (páčka).

6.2.2 Spínač

Jedná se o tlačítko s jedním 1-bitovým výstupem. Je možné u tlačítka nastavit zpoždění po puštění tlačítka. Při stisknutí (kliknutí) tlačítka se nastaví výstup na 1. Po puštění myši a uplynutí nastavené doby se hodnota vrátí na 0. Jednotlivé fáze tlačítka jsou zobrazeny změnou barvy a polohy tlačítka. O to se stará skript „ButtonScript.cs“. Model je vytvořený z primitivních tvarů v Unity.

6.2.3 Elektronický obvod

Elektronický obvod je inspirován programovatelnými logickými obvody. Počet vstupů a výstupů si určí tvůrce daného obvodu. Vnitřní struktura obvodu je taktéž určena hráčem. Protože jsou obvody velmi komplikované a je pro ně vytvořený zvláštní editor, vyhradil jsem jim celou kapitolu (kapitola 7 Elektronický obvod).

6.2.4 Váhy

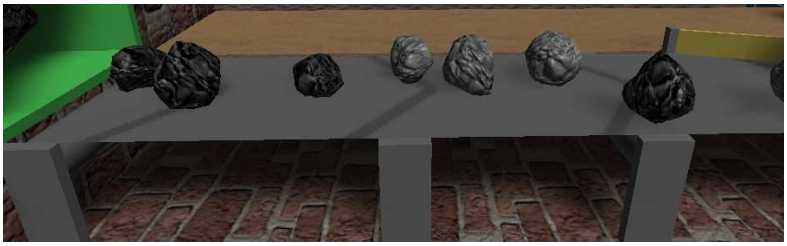
Původní návrh pro váhy byl takový, že by pouze změřil hmotnost předmětů ležící na ní a výsledek uložil do dvou bajtových hodnot. Protože jsem ale nechtěl příliš komplikovat používání váhy společně s písty a podobnými stroji, rozhodl jsem se, že budou váhy schopny náklonu horní plošiny. Snížením citlivosti z gramů na desítky gramů jsem se pak mohl zbavit druhého bajtového výstupu. Ve výsledku tedy mají váhy 1 bajtový výstup pro naměřenou hmotnost a 1-bitový vstup pro náklon. Váha také během náklonu změní svůj fyzikální materiál na hladký pro snadnější posun materiálu. Pohyb váhy je pak řešen skriptem „ScalesScript.cs“ umístěným na objektu. Stejně jako u většiny komponent jsou i váhy tvořeny pouze primitivními geometrickými tvary.

6.2.5 Přepravní pás

Pro implementaci do hry jsem zvolil jednoduché nečlánkované pásy s nastavitelnou délkou skrze editaci. Pro povrch pásu jsem vytvořil jednoduchou texturu pro lepší představu o natočení pásu. Protože se jedná o hladké pásy, je povolený pouze horizontální směr (výška se nemění). Pás je ovládán přes 2 vstupy. První určuje, zda má být pás v pohybu. Druhý pak obrací směr pohybu pásu.

Při vytvoření pásu jsem ale musel vyřešit problém jejich funkčnosti. Ve skutečnosti se materiál pohybuje společně s pásem. V Unity by to ale znamenalo složit pás z velkého množství pohyblivých objektů. Tím by se zachovala realističnost, ale bylo by to za cenu výkonu počítače. Moje řešení proto spočívá v tom, že při běhu posunuji pouze texturu pásu podle aktuální rychlosti.

Následně pak musím řešit, co se stane s materiálem, který se pásu dotýká. Prvním řešením je jednoduše měnit polohu materiálu společně s rychlostí. Tím by se zachoval realistický pohyb, ale obcházel by se fyzikální model, o který se stará Unity – mohlo by tak docházet k neošetřeným kolizím. Proto jsem se uchýlil k druhému řešení: na objekt, který je v kontaktu s pásem, je aplikována síla ve směru pohybu pásu. Toto řešení funguje, má ale za následek zdánlivě neopodstatněné kutálení materiálu po pásu. Pro omezení tohoto efektu jsem aplikoval fyzický materiál s vysokou třecí sílou na povrch pásu.



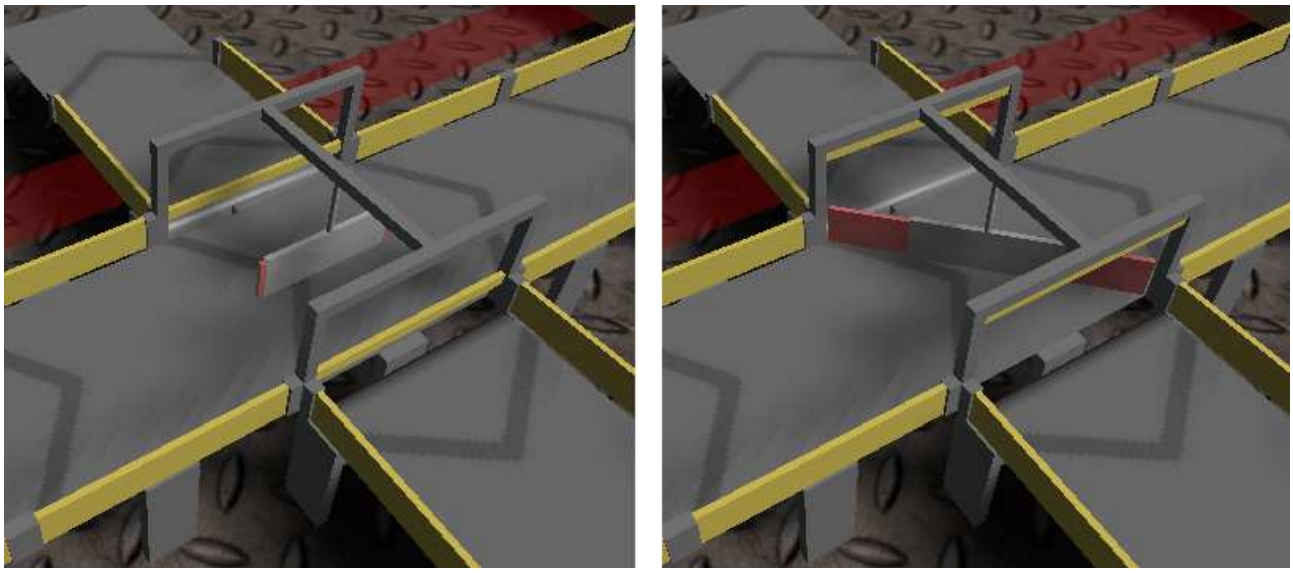
Obrázek 5: Přepravní pás v pohybu a jeho náklad

Pás se je tvořen jednotlivými objekty pospojovanými do celistvého pásu. Dílky začátku a konce pásu se vytvoří vždy. V editaci je pak možnost pás několikrát prodloužit. O to se stará skript na páse „ConveyorScript.cs“. Ten nejdřív vytvoří neviditelný objekt o velikosti dílku pásu na místo, kam by se měl posunout koncový dílek při expanzi. Pokud u tohoto testovacího dílku nedojde ke kolizi s jiným objektem, je možné pás bezpečně rozšířit. Posune se koncový dílek a vytvoří se středový dílek. Smrsknutí pásu už je potom snadné, kontroluje se pouze aktuální délka.

6.2.6 Pásová výhybka

Existuje celá řada technologií, jak přesměrovat přepravované věci z pásu. Většina těchto technologií je ale vytvořena přímo na míru problému nebo je pro hru příliš komplikovaná. Proto jsem vytvořil vlastní návrh použitelný pro malé objekty. Celý je vytvořen z primitivních tvarů v Unity.

Výhybka má dva 1-bitové vstupy. První aktivuje výhybku. Pokud je v logické hodnotě „1“, dochází k přesměrování materiálu na jednu stranu. Druhý vstup určuje, zda dojde k přesměrování doprava nebo doleva. Na samotném objektu je přesměrování řešeno natočením středového dílku a zvednutím postranních ochran. Pro snadnější pohyb jsem aplikoval velmi hladký fyzický materiál na jednotlivé části. O tento pohyb se stará skript na objektu „TurnoutScript.cs“.



Obrázek 6: Pásová výhybka ve stavech „vypnuto“ a „zapnuto“

6.2.7 Kamera

Hlavním způsobem přepravy materiálu jsou pásy, proto jsem vytvořil rovnou variantu kamery pro pásové přepravníky. Kamera je složená ze základních tvarů a speciálního kolizního boxu (v

režimu „trigger“), který detekuje jednotlivé kusy materiálu. Kamera obsahuje nastavitelný parametr, který určuje, jaké předměty jsou přijatelné (validní).

Kamera má dva 1-bitové výstupy. První se nastaví na „1“ vždy, když je detekován vstupující materiál. Další snímek (frame) se výstup nastaví zpět na „0“. Druhý výstup se aktualizuje vždy, když je detekován materiál. Pokud je typ materiálu shodný s tím nastaveným v kameře, výstup je nastaven na „1“. V opačném případě je nastaven na „0“. Tato hodnota se do příchodu dalšího materiálu nemění (na rozdíl od prvního výstupu) .

6.2.8 Sýpka / zásobník

Implementace herní sýpky je více inspirovaná zásobníkem z titulu Minecraft než realitou. Pro herní účely jsem ji navrhl tak, že přijímá postupně rudu v horní části a ukládá si ji do své zásoby. Podle aktuálních vstupů ji pak předává spodním koncem dále. Má tak 2 účely: rozvrstvit posílaný materiál do pravidelné dávky nebo zadržet materiál, dokud je potřeba.

Sýpka má 2 vstupy a 2 výstupy:

- 1-bitový vstup „zadržovat“: pokud je v 1 a čítač je na 0, zadržuje nashromážděný materiál.
- 8-bitový vstup „poslat“: při změně se zvedne čítač o poslanou hodnotu. Dokud je čítač > 0, dochází k posílání materiálu bez ohledu na první vstup.
- 1-bitový výstup „prošel“: ve snímku, kdy došlo k propuštění materiálu, je nastaven na 1. Kdykoliv jindy je v 0.
- 8-bitový výstup „obsah“: počet kusů materiálu v zásobníku.

O správu skladovaného materiálu a reakce na vstupy se stará hlavní skript sýpky. Samotné vypouštění materiálu je obstaráváno skriptem „HopperScript.cs“ umístěném na objektu. Model sýpky jsem vytvořil v Blenderu.

6.2.9 Signální světlo

Signální světlo nemá na systém žádný vliv. Pouze zobrazuje jednu 1-bitovou hodnotu – má tedy jeden 1-bitový vstup. Hráč si může nastavit, jakou barvu má mít světlo. Na objektu je umístěn skript „LightScript.cs“, který má za účel pouze držet odkaz na povrch válce představující světlo.

6.2.10 Displej

Je to obdoba signálního světla. Rozdíl je pouze v tom, že slouží k zobrazení hodnoty 8-bitového vstupu. Skript na objektu „DisplayScript.cs“ pak uchovává odkaz na Text objekt umístěného na displeji a stará se o aktualizaci této hodnoty.

6.2.11 Pásové zarážky

Tyto objekty nemají žádné vstupy nebo výstupy a slouží pouze jako pomůcka při usměrňování materiálu.

6.3 Začlenění do hry

Jednotlivé komponenty může hráč přidávat skrze mód „Sklad, obchod“. V tomto módu se zobrazí postranní panel obsahující všechny komponenty, které může hráč v tomto konkrétním scénáři přidat do hry. Dostupnost je rozdělená do dvou hodnot: inventář a obchod. Obě hodnoty jsou nastavitelné v editoru scénáře.

Pokud se hodnota inventáře nerovná 0, může si hráč komponentu rovnou vzít. Tím se hodnota inventáře o 1 sníží – pokud je tedy hodnota menší než 0, má hráč k dispozici neomezené množství této konkrétní komponenty. Obchod pracuje na stejném principu, jen musí mít hráč navíc určené množství herní měny, která se při nákupu sníží o danou částku. Pokud pak komponentu stejného typu smaže, peníze se vrátí na účet.

6.3.1 Uživatelské rozhraní obchodu

Panel obsahující jednotlivé komponenty se generuje dynamicky a je možné posouvat jeho obsah. Toho jsem dosáhl tak, že jsem vytvořil panel sahající od vybraného bodu pod hlavním panelem až do spodní části obrazovky. Na tu jsem aplikoval maskovací skript a skript pro posuvnou oblast poskytnutou Unity. Tento panel jsem pak propojil s posuvníkem a vnořeným panelem představující samotný obsah.

Obsah vnitřního panelu se při načtení smaže a vygeneruje znovu. Vytvořil jsem si vzorový panel obsahující textové komponenty a tlačítko. Na ten jsem aplikoval pomocný skript „UIInventoryPiece.cs“, který drží odkazy na jednotlivé komponenty pro snadnou manipulaci při tvorbě nabídky. Pro každou komponentu se nejdříve vyhodnotí, jestli se vůbec dostane do nabídky. Pokud je stav inventáře i obchodu na 0, komponenta se přeskočí. V opačném případě se vytvoří nový panel podle vzorového a přes pomocný skript se nastaví jednotlivé komponenty.

V případě, že je možné si danou komponentu vzít nebo má hráč dostatek měny na nákup, se přiřadí tlačítku odkaz na funkci „CreateNewComponent“, která vytvoří vybranou komponentu a přepne mód do režimu přesunu.

6.3.2 Umístění komponenty

Při vytvoření nové komponenty nebo zahájení přesunu už umístěné komponenty se nejdříve vybraná komponenta připraví. Podle vybraného typu komponenty se vytvoří fyzický objekt, který obsahuje všechny potřebné skripty a vlastnosti. Objektu se aktivuje (pokud už není v aktivován) jeho „Box Collider“ (kvádr používaný pro výpočet kolizí) a zmenší se o 1 cm. Tím se zabrání detekci kolizí sousedních objektů. Dále se objektu přidá Rigidbody (díky Rigidbody působí na objekt fyzikální engine Unity) nebo se aktivuje už existující. Nakonec se objektu přidá skript „MovingComponent.cs“, který se stará o detekci překážek při umístění komponenty.

Když je komponenta v režimu přesunu, automaticky se přichytává na vypočítanou pozici na herní ploše. Tato pozice se získá simulací paprsku z hlavní kamery na pozici kurzoru myši. Do výpočtu kolize je zahrnuta pouze neviditelná plošina, která se aktivuje pouze při přesunu komponenty. Používám ji k určení výšky komponenty. Původní řešení spočívalo v tom, že jsem detekoval kolizi paprsku se samotnou úrovní a jako pozici jsem bral bod těsně před koncem paprsku. Tohle řešení fungovalo dobře na malých komponentách a nebylo díky tomu možné umísťovat objekty ve vzduchu. U větších komponent bylo ale umístění problémové, proto jsem zvolil výše zmíněné řešení pomocí pomocné plošiny. Výsledná pozice se nakonec zaokrouhluje na jednotlivé dílky.

Pokud hráč stiskne tlačítko myši během přesunu, prověří se poloha myši. Pokud není zrovna nad rozhraním, chce hráč umístit komponentu. V tom případě se ověří aktuální hodnota ve skriptu „MovingComponent.cs“. Ten obsahuje hodnotu typu float, kterou zmenšuje, když je větší než 0. V případě, že detekuje kolizi komponenty s překážkou, nastaví tuto hodnotu na hodnotu 0,5. Pokud chce hráč komponentu umístit a tato hodnota je 0 nebo méně, komponenta může být umístěna.

Při umístění se komponenta navrátí do původního stavu. Smaže se pomocný skript a také Rigidbody (pokud ho neměl už v základu). Pokud normálně nemá aktivovaný „Box Collider“, tak je deaktivován. Nakonec se aktualizují hodnoty polohy a rotace v parametrech komponenty a herní mód se přepne do původního (mód obchodu nebo mód přesunu).

6.3.3 Informace a editace komponent

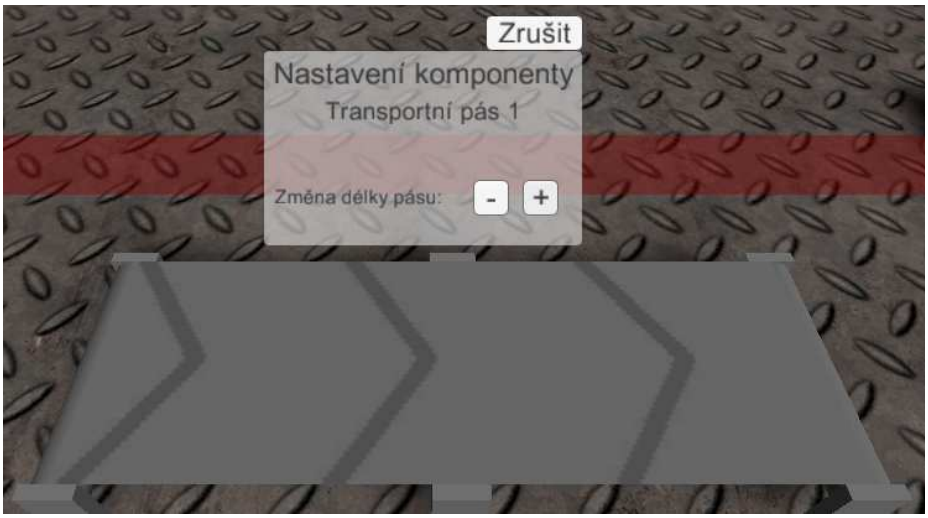
Kliknutí na komponentu mimo běh hry zavolá funkci „SelectComponent“. Následně se provede akce podle aktuálního módu hry. Možné akce jsou: editace, propojování a přesun.

Ve výchozím módu „Informace, propojení“ se zobrazí panely vstupů a výstupů a informační panel. Do informačního panelu se nahrají relevantní informace díky dědičné struktuře komponent. Pokud tvůrce scénáře nezakázal editaci, je možné změnit název komponenty. Při změně každé změně se nový název ukládá do komponenty. Komponenta nemusí mít unikátní název, takže nevadí duplicity.

Pokud tvůrce nezabránil editaci vybrané komponenty, je možné komponentu smazat. Tím se odstraní ze systému pomocí statické metody „RemoveComponent“ zmíněné dříve. V případě, že během hry nakoupil nějaký objekt tohoto druhu, dojde k vrácení peněz a zvýšení počtu kusů v obchodě. Není nutné, aby se jednalo přímo o koupený kus, stačí když je stejného druhu (není tak potřeba prohledávat jednotlivé komponenty).

Další možností je editace komponenty. Pro to musí být splněny 2 podmínky: komponenta musí mít povolenu editaci a také musí být editovatelná. První podmínku určuje tvůrce scénáře, druhá je daná. Například přepínače nebo displeje nelze editovat, ale světla nebo pásy editovat lze. Pokud hráč stiskne editační tlačítko a není zamknuté, vyhodnotí se druh komponenty. Pokud se jedná o elektronický obvod, spustí se metoda „SwitchCircuitEditor“, která nastartuje editor obvodů popsaný v kapitole 8 Editor elektronických obvodů. V opačném případě se objeví editační tabulka.

Z informací komponenty se do tabulky nahraje název a podle druhu komponenty se vybere panel, který se zobrazí. Ostatní panely se skryjí. Pro všechny aktuálně dostupné komponenty je postačující jediná hodnota typu float. Díky tomu lze všechny vyřídit stejnou funkcí. Vstupní pole i editační tlačítka jsou připojené k funkcím, které se pokusí získat aktuální hodnotu (v případě neúspěchu je výsledek 0). Tu pak pošlou do společné funkce, která parametr nastaví do komponenty. V případě přepravního pásu jsou zde tlačítka volající funkce pásu pro rozšíření nebo zmenšení pásu.



Obrázek 7: Editace přepravního pásu

6.3.4 Propojení komponent

Při vybrání komponenty v prvním režimu se kromě informačního/editačního panelu zobrazí také panel propojení. Ten je vytvořen podobně jako panel s obchodem. Postupně se prochází seznam vstupů a výstupů komponenty a pro každý se vytvoří textový prvek (nastavený předem v editoru), u kterého se změní text na název kontaktu. Poloha v ose X zůstává konstantní, poloha v ose Y je násobek proměnné, ve které se uchovává aktuální počet řádků.

Pro každý vstup či výstup procházím vnořeným cyklem všemi jeho spojeními. Za každé spojení se vytvoří tlačítko, u kterého změním jeho textovou komponentu na název druhého prvku spojení. Na tlačítko se pak zaregistruje funkce pro ukončení spojení. Ta se pak provádí skrze moji metodu vytvořenou v základní třídě komponent. Pokud je počet spojení 0, nebo se jedná o výstup, vytvoří se další tlačítko pro vytvoření propojení. Počet vytvořených tlačítek pak odpovídá aktuálnímu počtu řádku.

Pokud hráč klikne na nové propojení, hra změní režim na „hledání propojení“. Při kliknutí na jakoukoliv komponentu se mód přepne do režimu „výběr propojení“. Podobně jako u předchozích případů se i zde tvoří obsah dynamicky. Na rozdíl od panelu všech propojení se zde tvoří jeden řádek pro každý vstup nebo výstup. Pro každý kontakt se vyhodnotí, jestli je propojitelný s kontaktem vybraným dříve.

Aby byl kontakt přijatelný, musí být opačného typu a mít stejný počet bitů. Pokud je tento kandidát vstupem, nesmí být k ničemu připojen. Pokud je toto všechno splněno, vytvoří se tlačítko registrované na funkci, která propojení vytvoří. V opačném případě se vytvoří pouze odmítavá textová komponenta.

Obrázek 8: Ukázka propojení přepínače a transportního pásu

7 Elektronické obvody

Elektronické obvody jsou jedním z možných druhů komponent. Jejich účel je nejširší ze všech komponent. Lze je použít všude, kde je potřeba tvořit jakoukoliv logiku nebo zpracování vstupů. Například pokud chceme, aby se přepnula výhybka až po splnění dvou podmínek, je nutné vytvořit obvod s AND hradlem (nebo jiným řešením). Počet vstupů a výstupů není omezený, záleží tak čistě na hráči, kolik jich potřebuje. Elektronické obvody představují nejvíce vzdělávací část hry.

7.1 Struktura

Ovšem oproti ostatním komponentám je mnohem komplexnější. Základní struktura obvodu se podobá struktuře třídě „GameComponent“. Hlavní třída „Circuit“ tak obsahuje celou řadu metod pro manipulaci s vlastními prvky a především seznam jednotlivých prvků typu „CircuitElement“.

Vybrané metody třídy „Circuit“:

- „ResetComponent“: provede funkci „Reset“ pro všechny elementy. Tuto funkci si každý prvek definuje zvlášť.
- „TransmitInputs“: pro všechny elementy typu vstup (vnitřní element) se do jejich následující hodnoty nahraje aktuální hodnota propojeného vstupu samotného obvodu (venkovní kontakt).
- „TransmitOutputs“: pro všechny výstupy obvodu (venkovní kontakt) se do jejich následující hodnoty nahraje aktuální hodnota propojeného elementu typu výstup (vnitřní element).
- „ExecuteCircuitLogic“: hlavní funkce vykonávající celou logiku obvodu. Postupně volá metodu „Run“ u všech svých elementů.
- „AddInputOutput“: přidá nový vstup nebo výstup do obvodu. Současně vytvoří také nový prvek obvodu stejného typu a oba prvky propojí. Venkovní kontakt je tak spojen s vnitřním prvkem.
- „ActualizeCircuitInputs“: pro všechny prvky obvodu se projde vnořeným cyklem jejich seznam vstupů. Pokud má vstup nějaké spojení, nahraje se do aktuální hodnoty následující. Následující hodnota se nastaví na 0.
- „ResetCircuit“: provede se metoda „Reset“ u všech komponent a u všech nepřipojených vstupů se hodnota nastaví na výchozí hodnotu.

- „ActualizeCircuitOutputs“: pro všechny prvky obvodu se projde vnořeným cyklem jejich seznam výstupů. Jejich aktuální hodnota se nahradí následující a následující je nahrazena 0. Po té se projde seznam jejich propojení a pro každé se nastaví následující hodnota připojeného kontaktu na aktuální hodnotu výstupu.
- „ConnectCircuitInputOutputs“ a „RemoveCircuitConnection“: přidání propojení mezi prvky a odstranění propojení mezi prvky. Princip a funkčnost jsou stejné jako u základní třídy komponenty. Pouze se provádí na prvcích obvodu.
- „RemoveInputOutput“: kromě normálního smazání kontaktu se musí také smazat propojený vnitřní prvek.
- „RemoveCircuitElement“: slouží ke smazání prvku z obvodu. Funguje velmi podobně jako odstranění komponenty ze hry.

7.2 Prvek obvodu „CircuitElement“

Je to základní třída, od které dědí všechny prvky obvodu. Definuje několik základních vlastností a metod, které musí jeho potomci doplnit. Stejně jako základní třída komponent má i „CircuitElement“ svůj seznam vstupů a výstupů. Tento seznam nemůže hráč nijak určit, každý prvek si ho definuje sám. Třída také obsahuje definici funkce „Actualize“, která aktualizuje všechny hodnoty kontaktů na následující.

Každý prvek, který dědí od téhle třídy, pak může implementovat dvě metody: „Run“ a „Reset“. První metoda obsahuje logiku daného bloku. To znamená zpracování vstupních hodnot a nahrání následující hodnoty do svých výstupů (kromě prvku „výstup“ má každý prvek alespoň 1 výstup). Druhá metoda slouží pouze k resetování prvku. To se týká pouze bloků, kteří mají vnitřní paměť.

Jednotlivé prvky jsou pak kompletně definovány v následujících skriptech:

- „Circuit.cs“: vstup, výstup
- „LogicGates.cs“: obsahuje logická hradla AND, NOT, OR, XOR
- „FlipFlops.cs“: klopné obvody RS, D, T, JK
- „MathBlocks.cs“: čítač, porovnávač, multiplexor
- „OtherCircuitElements.cs“: převodníky mezi bitem a bajtem, hodiny

Aktuálně vytvořená struktura pak umožňuje velice snadné rozšíření v podobě dalších prvků. Nový prvek lze přidat během 5 minut. Pro přidání nového prvku stačí pouze: zkopírovat existující prvek a upravit vstupy/výstupy, texty a funkci „Run“. Následně stačí jen přidat nový prvek do seznamu ve skriptu „CircuitEdit.cs“ a přidat tlačítko s novým prvkem do rozhraní editoru.

7.3 Funkčnost obvodů

O funkčnost se stará prvek obvodu, který obsahuje jednotlivé prvky. Ten ve své metodě Run postupně zavolá následující metody:

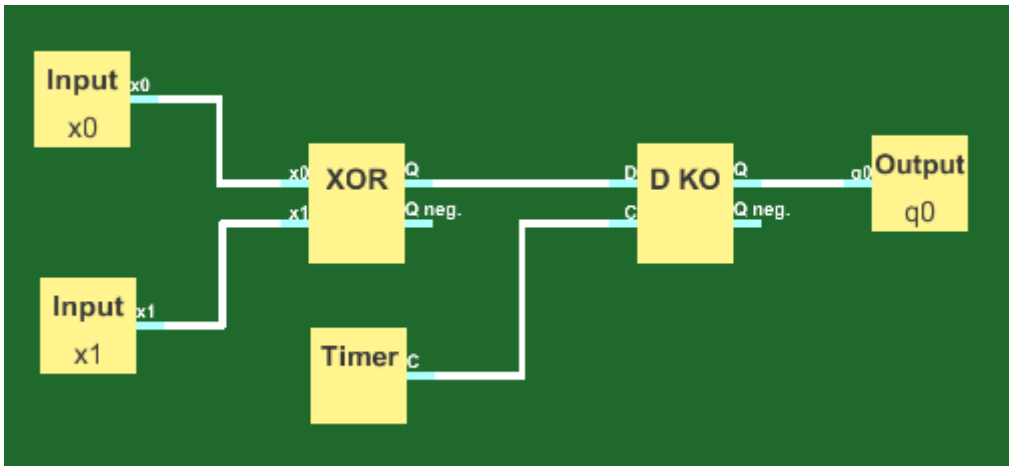
1. „TransmitInputs“: venkovní hodnoty na vstupech se převedou do propojených prvků v obvodu.
2. „ActualizeCircuitInputs“: všechny vstupy všech prvků se aktualizují na nové hodnoty z jejich následujících.
3. „ExecuteCircuitLogic“: všechny prvky provedou svojí vnitřní logiku. Tím se nastaví nové hodnoty do následujících výstupních hodnot.
4. „ActualizeCircuitOutputs“: všechny výstupy se aktualizují na vypočítanou hodnotu a tu pak předají všem vstupům, se kterými jsou propojení.
5. „TransmitOutputs“: na venkovní výstupy se do následujících hodnot nahrají hodnoty propojených prvků výstupu.

Od normálního běhu komponent se tento postup liší především krajními body starající se o přenos hodnot mezi konektory na povrchu komponenty a konektory uvnitř komponenty.)

8 Editor elektronických obvodů

Tento editor slouží k vytvoření nebo úpravě elektronického obvodu. Pokud je hráč v režimu editace komponent a obvod je editovatelný (tvůrce nezakázal editaci), vypne se hlavní kamera a aktivuje se kamera pro obvody. Ta je na rozdíl od hlavní kamery ortogonální. Současně se také vypne normální uživatelské rozhraní a aktivuje se rozhraní pro obvody. Hlavní ovladač hry „GameController.cs“ se přepne do pasivního režimu (nepřijímá žádné vstupy) a naopak do aktivního režimu se přepne „CircuitEdit.cs“. Při ukončení editace se provedou přesně opačné akce.

Při inicializaci se nastaví některé proměnné a editor se přepne do základního režimu. Dále dojde k načtení obvodu. To spočívá ve vytvoření jednotlivých bloků dle uložené informace (poloha, typ, propojení) a jejich propojení. Tyhle části jsou podrobněji rozepsány níže. Kromě pohybu kamery také často hra pracuje s polohou kurzoru na samotném obvodu (ne na obrazovce). Toho jsem dosáhnul stejně jako u umísťování komponent: simulací paprsku mezi kamerou směrem na kurzor. Následná kolize s podkladem pak udává výslednou polohu.



Obrázek 9: Kompletní obvod vytvořený v editoru obvodů

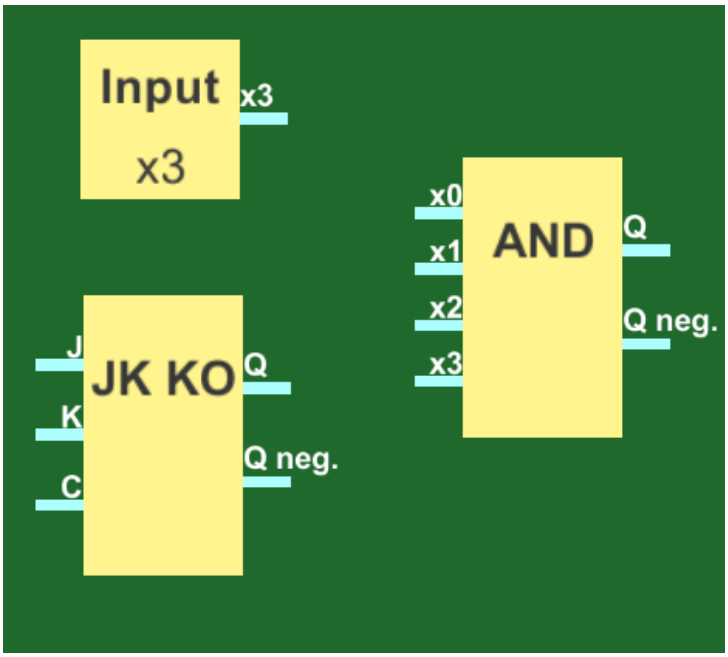
8.1 Reprezentace prvku

Základním blokem representace je pouhý kvádr, na který je přidán prvek „Canvas“, který umožňuje používat systém uživatelského rozhraní. Na rozdíl od rozhraní pro editor nebo pro hru je tento lokální (je umístěn přímo na prvku), kdežto předchozí dva se zobrazují přímo na kameru bez ohledu na pozici prvku. Toto rozhraní pak obsahuje textové komponenty pro snadnou identifikaci prvku. Representace také obsahuje skript „CircuitElementRepresentation.cs“, který obsahuje odkazy na tyto prvky a také na kontrolní skript a kameru pro editor.

Při vytvoření nového bloku se tak zkopíruje tento vzorový representant a nastaví si všechny potřebné reference. Umístění je za normálních okolností přesně pod kameru. Pokud už existuje prvek se stejnou polohou, postupně se testují násobky vybrané šířky a výška a tím se nové bloky ukládají do matice. Tento efekt je patrný pouze když hráč nehýbe kamerou a pouze přidává prvky.

Kontakty jsou vyřešeny podobně jako representant. Základem je opět jednoduchý kvádr s prvkem „Canvas“, který obsahuje popisek s názvem kontaktu. Kolizní kvádr kontaktu je schválně zvětšen a posunut, aby se zjednodušila případná manipulace s kontaktem. Kontakt také obsahuje skript „ContactID.cs“, který identifikuje daný kontakt a je napojený na rodičovský skript representanta.

Při vytvoření prvku se pak spočítá, kolik vstupů a kolik výstupů by měl prvek mít. Podle tohoto počtu může dojít ke zvětšení nebo zmenšení representanta tak, aby nebyly kontakty příliš na husto. Na základě toho se postupně vytváří jednotlivé kontakty jako kopie vzorového. Díky celkovému počtu vstupů/výstupů je snadné je rozmístit rovnoměrně. Vstupy se umísťují na levou stranu a výstupy na pravou.



Obrázek 10: Ukázka tří vytvořených prvků (vstup, AND hradlo a JK klopný obvod)

8.2 Propojení prvků

Oproti propojení komponent, které je neviditelné, propojení mezi prvky obvodů je řešeno viditelně pomocí horizontálních a vertikálních čar. Základní třída propojení „Connection“, který vyu-

žívají i komponenty, obsahuje speciálně pro obvody seznamy pozic a reprezentace spojů. Seznam pozic je posloupnost poloh jednotlivých rohů spojení prvků v editoru. Reprezentace spojů jsou pak odkazy na jednotlivé použité objekty spojení.

První řešení jak vyobrazit spojení, které jsem zkoušel, bylo použití prvku Unity „Line Renderer“. Ten slouží k renderování rovných čar, které mohou být i lomené. Lze u nich nastavit dostatečně velké množství parametrů, aby byly použitelné v editoru obvodů. První problém, na který jsem narazil, bylo špatné ovládání tohoto prvku ze skriptu, i když to jsem nakonec obešel. Mnohem závažnější problém ale bylo nativní chování této vytvářené čáry v zalomeních. Při použití ostrých rohů (90°) se generovaná čára zkroutila o 90° tak, aby se ohýbala svojí širší stranou (čára je tvořena obdélníkem). Výsledkem pak byla čára měnící tloušťku směrem k rohu až do ztracena. Z tohoto důvodu jsem toto řešení zavrhl.

Druhé řešení spočívalo v použití vykreslení čáry za pomocí série objektů typu „Quad“ (obdélník v prostoru). Tento obdélník obsahuje skript „WireLogic.cs“, který obsahuje všechny potřebné údaje k identifikaci tohoto segmentu a také odkazy na hlavní skript editoru a jeho „Mesh Renderer“ sloužící k vykreslování objektu. Tyto objekty jsou vkládány jako kopie vzorového a rozšířeny do požadovaného směru podle údajů o poloze rohů.

8.3 Ovládání kamery editoru

Ovládání je opět řešeno ze dvou míst: kontrolního panelu editoru a pomocí myši. Přibližování a oddalování kamery se provádí pro oba případy stejně. Pouze se mění velikost ortogonálního rozměru u kamery v rámci předem určeného rozsahu. V případě pohybu kamery po ploše se způsob provedení liší od zdroje příkazu.

V případě kontrolního panelu se při stisknutí tlačítka nastaví do skriptu editoru jeho směr. Při puštění tlačítka se tento směr zruší. Pokud je pak editor aktivní a je nastaven nějaký směr, pohybuje se kamera i podklad obvodu opačným směrem, aby bylo dosaženo žádaného účinku. Díky provázanosti kamery a podkladu není možné dostat se s kamerou mimo editor.

Pokud hráč posouvá kameru pomocí pravého tlačítka myši, tak se nejdříve uloží poloha myši na obrazovce. Dokud pak hráč drží pravé tlačítko myši tak se kamera s podkladem nastaví na polohu vypočítanou rovnicí založenou na rozlišení obrazovky, rozdílu původní polohy myši a té nové, velikostí kamery a pomocné konstanty (určující rychlost posunu). Výsledkem je přirozený pohyb při posunování kamery. Pokud tedy nepustí myš a vrátí se s kurzorem na původní místo na obrazovce, bude kamera opět ve své výchozí pozici.

8.4 Panel nových bloků

Panel nových bloků je vytvořený předem a jednotlivá tlačítka volají stejnou sadu funkcí. Funkcí jsem musel vytvořit více, protože normálně je možné předávat pouze jeden parametr skrze tlačítko. První parametr je tak identifikátor prvku. Jako druhý parametr pak potřebuji velikost nebo druh hrany, proto jsem vytvořil funkcí pojmenovaných „AddNewBlockS[X]“ (místo [X] patří číslice od 1 do 8) a ty volají hlavní funkci „AddNewBlock“ už s přidaným druhým parametrem.

Podle identifikátoru pak přidá do seznamu prvků obvodu nový konkrétní prvek. V případě vstupu nebo výstupu se zároveň vytváří i vstup a výstup na samotném obvodu. Ty se pak propojí, aby si mohly navzájem předávat hodnoty. Nakonec se zavolá funkce „AssignBlock“, která vytvoří reprezentanta.

8.5 Režim informací a editace

V prvním režimu je možné zobrazovat si informace o jednotlivých blocích. Po kliknutí na prvek se zobrazí nový panel s informacemi o prvku. Tyto informace pochází z vlastností vybraného prvku (jméno a popis). Současně je zde také tlačítko pro mazání komponenty, které volá funkci pro smazání prvku popsanou na začátku kapitoly.

Pokud hráč klikne na samotný kontakt, zobrazí se informace o daném kontaktu (jméno, počet bitů, rozsah hodnot, počet napojení a výchozí hodnota). Tlačítko pro smazání se zneviditelní a pokud se jedná o vstup, zobrazí se editovatelné pole s výchozí hodnotou. Při změně této hodnoty se zavolá funkce, která se tuto hodnotu pokusí získat. Pokud neuspěje, bere se výsledek jako „0“. Funkce také omezí hodnoty větší, než je rozsah.

8.6 Tvorba propojení

Ve druhém módu editoru nazvaném „editace spoje“ je možné vytvářet nové spojení nebo upravovat existující. Pokud program detekuje kliknutí na kontakt, zavolá se rozhodovací funkce „SelectContact“, která pak zavolá další funkci podle aktuálního režimu. V případě editačního režimu se zavolá „ContactInfo“, která byla popsána výše. V případě režimu pro editaci propojení se zavolá metoda „ConnectContacts“.

Činnost metody závisí na tom, jestli už byl vybrán počáteční bod propojení nebo je tohle první vybraný kontakt. V případě, že je to první vybraný kontakt, program zkontroluje, jestli se nejedná o už obsazený vstup. Pokud ne, aktivuje se „Line Renderer“, který začne vykreslovat čáru od vybraného kontaktu ke kurzoru. Kontakt se uloží jako počáteční bod.

Pokud uživatel klikne na jiný kontakt a může dojít k propojení (podle pravidel zmíněných dříve), vypne se „Line Renderer“ a editor se přepne zpátky do režimu tvorby propojení. Pro vstupní kontakt tohoto spojení se vygeneruje posloupnost souřadnic představující jednotlivé body propojení.

Pokud je vstupní kontakt napravo o výstupního, vytvoří se 4 body včetně počátku a konce. Výsledné spojení tak vede do poloviční vzdálenosti mezi kontakty, kde se zalomí a pokračuje svisle. Znova se zalomí, když se dostane do výšky druhého kontaktu. Pokud je výstupní kontakt napravo, vytvoří se 6 bodů, které pak tvoří spoj do tvaru svorky.

Po přidání bodů se zavolá funkce „InsertWireObjects“, která vytvoří jednotlivé segmenty. Vytvoří jich o 1 méně než je počet bodů. Každému se nastaví jeho identifikátory tak, aby byl po kliknutí snadno určitelný kontakt, ke kterému patří. Odkazy na jednotlivé segmenty se uloží společně s body do vstupního kontaktu. U segmentů na začátku a na konci se vypne možnost editace.

Jakmile jsou přidány a nastaveny jednotlivé segmenty, zavolá se metoda „UpdateWirePosition“. Ta má na starosti podle uložené série bodů správně umístit a roztáhnout jednotlivé obdélníky. Podle indexu segmentu lze snadno poznat, jestli se jedná o horizontální nebo vertikální část (pravidelně se střídají). Tato funkce se volá vždy, když dojde ke změně těchto bodů.

8.7 Editace propojení

Stále ve stejném režimu je možné provádět také editaci propojení. Jedná se pouze o vizuální editaci, funkčnost obvodu to neovlivní. Rozvržení se ale ukládá, takže se znovu načte i při další editaci.

Pokud nějaký segment propojení detekuje stisknutí levého tlačítka myši a je označen jako editovatelný (nejedná se o krajní segment), aktivuje se přesunování segmentu. Dokud hráč drží tlačítko myši, jsou přepisovány krajové body vybraného segmentu podle kurzoru myši. Přepisována je vždy jedna souřadnice. Která to je záleží na typu segmentu. U horizontálního segmentu se přepisuje osa Y, u vertikálního to je osa X. Dokud je tlačítko myši stlačené, volá se také funkce „UpdateWirePosition“ popsaná výše. Jakmile hráč myš pustí, zavolá se tato metoda naposledy.

8.8 Další možnosti editace propojení

Pokud je hráč ve třetím režimu („mazání propojení“), může mazat jednotlivé propojení. Pokud některý ze segmentů (bez ohledu na editovatelnost segmentu) zachytí kliknutí v tomto reži-

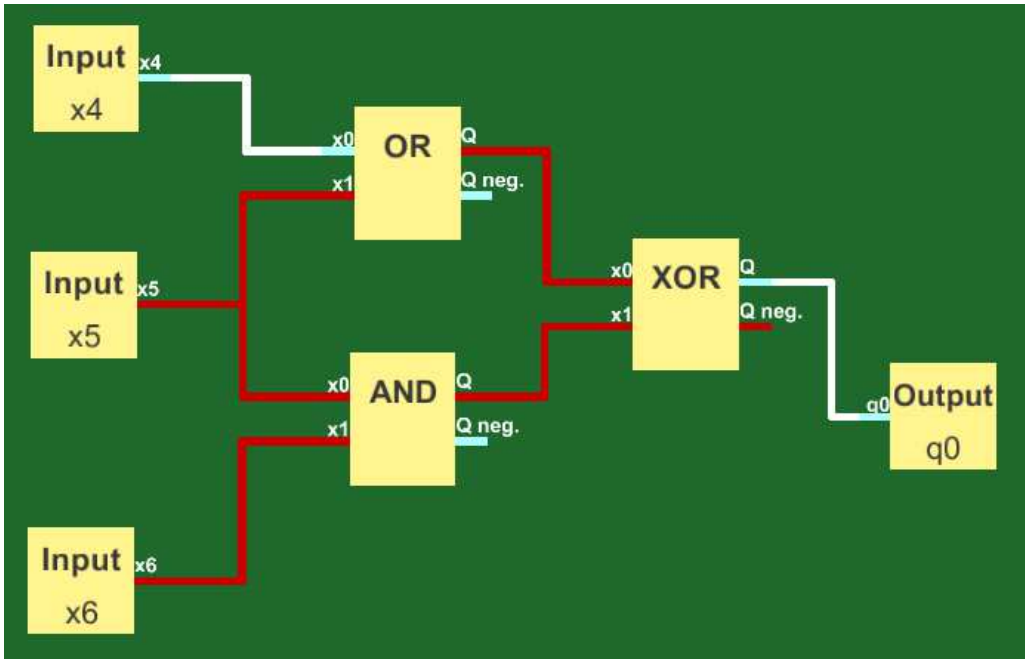
mu, zavolá se metoda obvodu na smazání propojení. Ten kromě smazání propojení u obou kontaktů také projde postupně všechny odkazy objekty spojení a smaže je. Seznamy bodů a odkazů se pak vyčistí.

Ve čtvrtém režimu („přidání ohybu“) může hráč udělat spojení složitější – přidáním dalších 2 segmentů. Zavolá se funkce „WireChangeSize“, která podle zvoleného segmentu vloží dva nové body do seznamu co nejblíže vybranému segmentu (tak aby se nenarušila funkčnost). Nové body pak vytvoří nový roh tak, aby byl snadno editovatelný. Původní segmenty se pak smaží a vytvoří se znovu.

V posledním režimu „Ubrání ohybu“ je pak možné provést opak předchozího. Pokud vybrané spojení není na svoji minimální velikosti (3 segmenty), odeberou se co nejblíže vybranému segmentu dva body. Objekty segmentů se smažou a znovu vygenerují.

8.9 Testování obvodu

Na hlavním panelu se nalézá také tlačítko pro testování obvodu. Při stisknutí se zneviditelní všechny panely a změní se režim editoru. Dokud probíhá testování, provádí se metoda „Run“ celého obvodu. Každý snímek se také projdou všechny kontakty všech prvků. Podle hodnoty se pak změní materiál kontaktu (zpřístupněn přes jeho skript) na původní nebo červený, který značí nenulové hodnoty. U vstupů se pak projde seznam objektů tvořících propojení a přes jejich skript „WireLogic.cs“ může editor měnit i jejich materiál. Pokud editor zaregistruje kliknutí na prvek 1-bitového vstupu, prohodí se jeho hodnota.



Obrázek 11: Obvod v testovacím režimu (červené spoje značí log. 1)

9 Dokončení hry

Základním cílem hry je splnit zadané úkoly. Ty jsou založeny na přepravě a zpracování materiálu. Aktuálně jsou ve hře přítomny 3 (všechno horniny), na demonstraci tento počet stačí. Součástí scénáře tak je kromě samotných úkolů také objekty sběračů a tzv. tvůrců. Všechny tyto objekty jsou statické do spuštění běhu hry (přes tlačítko spustit).

Tyto objekty umísťuje tvůrce mapy a pro hráče jsou needitovatelné. Nelze s nimi pohybovat nebo je smazat. Tvůrci během režimu hry postupně vytváří nadefinovaný materiál. Sběrače ho zase likvidují a zaznamenávají si o něm informace. Při každém zapsání se prověřují podmínky pro vítězství. O běh hry a kontrolu vítězství se pak stará řídící skript „GameMovables.cs“.

9.1 Sběrač

Jedná se o objekt ve tvaru krychle s chybějící stěnou. Obsahuje dva kolizní objekty. První je na povrchu geometrie a slouží jako překážka. Druhý je uvnitř krabice a slouží ke sbírání spadeného materiálu. Každý sběrač obsahuje také skript „Collector.cs“, který se stará o samotné odstraňování materiálu ze hry. Při odstranění si přidá nový záznam do svého seznamu materiálu obsahující typ sebraného materiálu a jeho čas. Také se při každém sebrání volá kontrola vítězství metodou „CheckGoalAccomplisment“ na řídícím skriptu. Při každém spuštění běhu hry se uložený seznam vyčistí.

9.2 Tvůrce

Geometrií je stejný jako sběrač, liší se pouze v barvě. Oproti sběrači ale obsahuje skript „Spawner.cs“, který se stará o vytváření materiálu za běhu hry. Skript definuje několik důležitých věcí: seznam tvořeného materiálu, jak často vytvářet a počáteční rychlost materiálu. Seznam tvořeného materiálu je uložen jako seznam typu „ItemSet“.

Třída „ItemSet“ nemá vlastní metody kromě konstruktoru, slouží pouze k uchování vlastností. Tyto vlastnosti jsou pak používány při generování materiálu. Jedná se o:

- Druh materiálu: určuje texturu a základní váhu materiálu.
- Množství materiálu: pokud je zadána záporná hodnota, bere program počet jako nekonečno.
- Priorita materiálu: čím je vyšší, tím je větší šance že bude vybrán při tvorbě nového materiálu.

- Hmota minimum a maximum: určují velikostní rozsah tvořeného materiálu.

Na začátku běhu hry se nastaví počty jednotlivých věcí do svých výchozích hodnot. Poté se spustí časovač. Vždy, když časovač dosáhne 0 a jeho inventář není prázdný, spustí se procedura pro tvorbu materiálu. Projde se seznam věcí a spočítá se suma priorit všech relevantních položek. Následně se vygeneruje náhodně číslo v rozsahu od 1 do vypočítané sumy a toto číslo pak určí vybraný set věcí. S vybranou sadou se pak zavolá metoda „SpawnObject“, která se stará o samotné vytvoření materiálu.

Metoda vytvoří nový kus materiálu ze seznamu předvytvořených tvarů. Dále nastaví texturu geometrie a základní hmotnost podle druhu tvořeného předmětu. Objektu se přidá skript „MovableBasics.cs“, který pomáhá s pohybem po některých objektech a uchovává identifikaci předmětu. Objektu se také přidá „Rigidbody“, aby se o jeho pohyb staral fyzikální engine Unity. Podle nastavení tvůrce se pak objektu nastaví jeho počáteční rychlost a nakonec se tento objekt přidá do seznamu materiálu v úrovni.

9.3 Struktura úkolů a podmínek

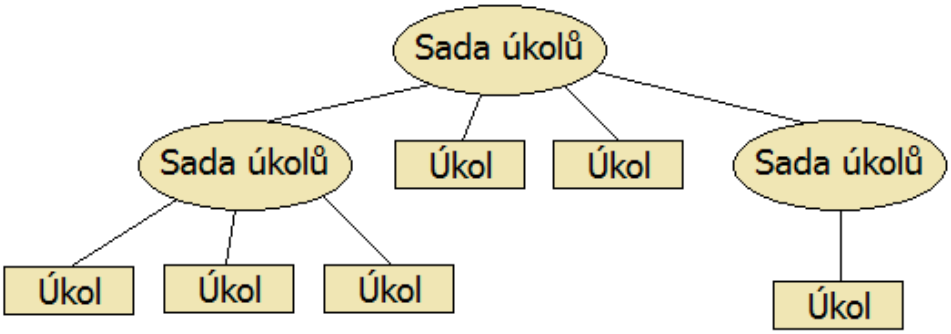
Protože jsem chtěl co nejméně omezovat tvůrce scénářů a poskytnout tak co největší flexibilitu, založil jsem strukturu úkolů na stromovém grafu. Každý uzel grafu představuje sadu úkolů, u které lze nastavit typ splnění (všechny nebo alespoň jeden). Každý list grafu pak představuje konkrétní úkol, který je potřeba splnit. Scénář vždycky obsahuje minimálně kořenový uzel. To i v případě, že chce tvůrce pouze jeden jediný úkol. Všechny třídy úkolů jsou definovány ve skriptu „GameGoals.cs“. Kromě úkolového stromu je zde také totožně vytvořený strom představující podmínky prohry.

Sada úkolů i konkrétní úkol dědí od společné třídy „GameGoalBaseClass“, která pouze definuje funkce k implementování a obsahuje boolean proměnnou ke snadnému rozlišení obou druhů. Metody k implementování jsou pak „Accomplished“, která vrací boolean hodnotu o splnění úkolů a metoda „PrintGoal“, která vrací textovou reprezentaci úkolu (používám ji v editoru scénářů).

Sada úkolů (třída „GameGoals“) obsahuje pouze seznam úkolů (rodičovského typu) a typ podmínky. Typ podmínky určuje, zda se jedná o typ OR nebo AND. Seznam úkolů je pak seznam tvořen buď sadami úkolů, nebo konkrétními úlohami. Na pořadí nezáleží.

Konkrétní úkol (třída „Goal“) pak obsahuje konkrétní definici úkolu. Ta se skládá z následujících položek:

- Seznam sběračů: pro tyto sběrače se podmínka bude ověřovat.
- Druh podmínky: určuje, co se bude ověřovat (počet, zastoupenost v %, ks/min, čas). V případě času se ignorují sběrače a povolené typy.
- Povolené typy: seznam povolených druhů materiálu, které se započítají do ověřování.
- Znaménko: určuje, jaké se použije znaménko v ověřování (=, >, >=, <, <=, <>).
- Parametr: oproti téhle hodnotě se podmínka ověřuje.



Obrázek 12: Ukázka stromové struktury úkolů a podmínek

9.4 Běh hry

Jakmile hráč stiskne tlačítko spustit, přepne se hra do režimu běhu. Provede se reset všech komponent a zneviditelní se všechny panely kromě ovládání kamery. Zobrazí se a vynuluje se panel obsahující počítadlo času. Tvůrci se nastaví do výchozího nastavení (obnoví se seznamy předmětů) a začnou s odpočtem do tvorby prvního materiálu. Během běhu se taky umožní interakce s některými komponentami (spínače a přepínače).

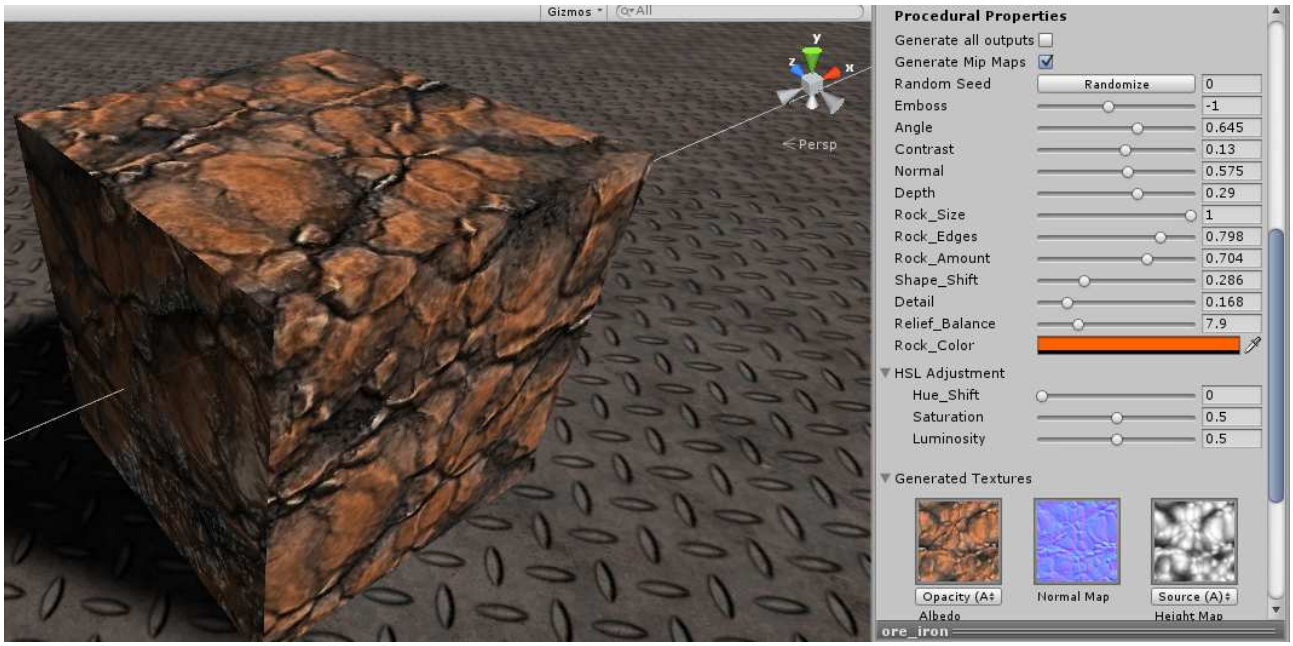
Při každém umístěném kusu materiálu do některého ze sběračů se zavolá funkce na kořenový úkol k vyhodnocení výhry nebo prohry. Ten pak rekurzivně projde všechny uzly a vyhodnotí podle konkrétních úkolů výsledek. Nejdříve se prochází strom pro podmínky prohry a následně až pro podmínky výhry. Podle výsledků se může zobrazit výsledná informace o prohře či výhře (prohra má přednost). Během běhu se také kontroluje aktuální čas scénáře s nastaveným limitem.

Pokud hráč stiskne tlačítko „spustit“ znovu, nebo když vyprší nastavený časový limit, vrátí se hra do režimu editace. Všechny kusy materiálu jsou odstraněny ze scény.

9.5 Vytvoření materiálu

Jednotlivé modely materiálu jsem vytvořil v Blenderu z nízko polygonové koule pomocí „sculpt mode“ a následného vyhlazení. Tím jsem dosáhl dostatečně organických tvarů pro kámen.

Všechny tři textury pak vznikly ze stejného zdroje, tzv. substance. Ta umožňuje tvořit tématickou texturu na základě řady parametrů. Rozdílným nastavením těchto hodnot lze vytvořit velice odlišné textury. Tento objekt pak vygeneruje vytvořenou texturu do všech potřebných souborů (včetně výškové mapy). Tímto způsobem vznikly kromě textur hornin také textury prostředí.



Obrázek 13: Vytváření textury pomocí "Substance Textures"

10 Vytvoření editoru scénářů

Všechny předem vytvořené úrovně byly vytvořeny ve scénáři editorů. Hráč tedy může vytvořit přesně scénář se stejnými možnostmi.

Samotný editor je řízen místo herního skriptu („GameControl.cs“) skripty určenými pro editor: „CreatorControl.cs“ a „CreatorComponents.cs“. Druhý jmenovaný se stará o vše okolo komponent, první o všechno ostatní. Původně jsem je chtěl pojmenovat „EditorControl“, ale Unity pak takovýto skript bere jiným způsobem (vztahuje ho k editoru). Samotný kód pro editor je upravená kopie hlavního skriptu pro hru. Ve většině funkcí jsou ale větší nebo menší rozdíly, proto jsem se rozhodl neslučovat vše do jednoho souboru. Uživatelské rozhraní je také z části zkopírováno, ale velká část je nová.

Pomocí editoru lze nastavit:

- prostředí scénáře
- přednastavené komponenty
- dostupnost komponent
- sběrače a tvůrci
- nastavení úrovně
- úkoly.

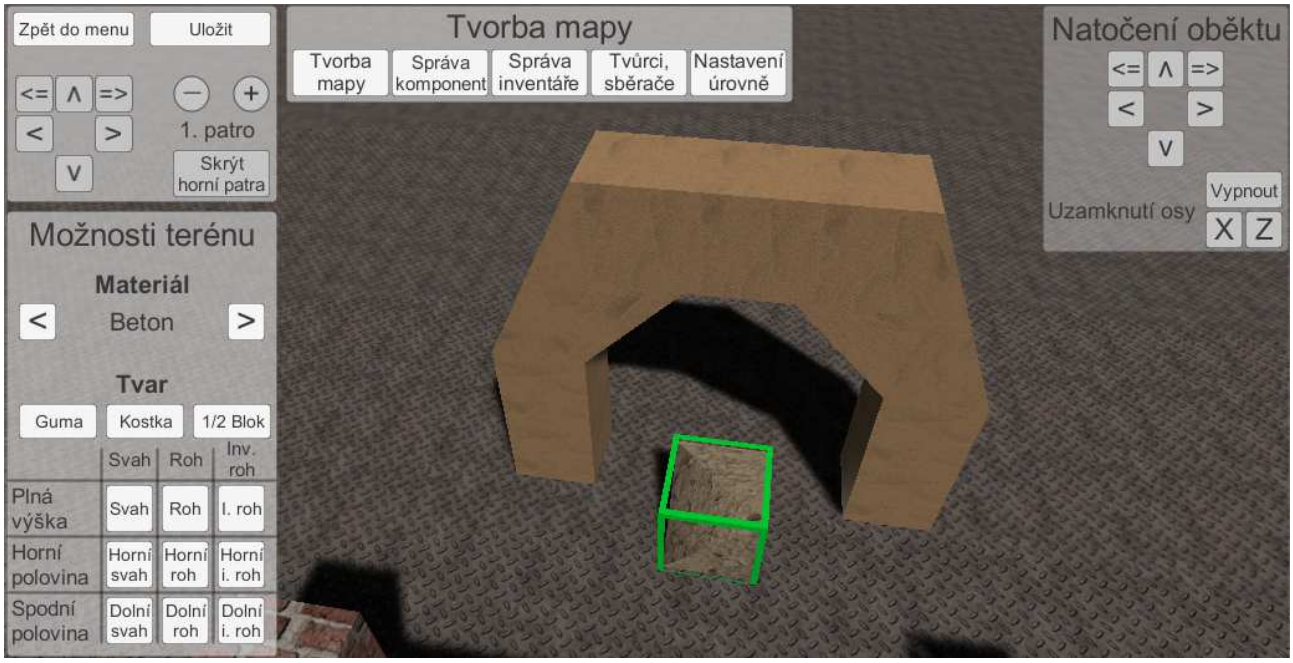
10.1 Tvorba prostředí

Editace prostředí je výchozí režim editoru a lze se na něj později přepnou tlačítkem „Tvorba mapy“. Pro editaci prostředí jsem v Blenderu vytvořil jednoduchý kursor, který pomáhá s lepší viditelností vybraného políčka. Umístění kurzoru se počítá stejně jako u umístění komponent, pouze se zaokrouhluje po větších kusech.

Do středu kurzoru je umístěn aktuálně vybraný tvar. Ten je vždy nastaven jako potomek kurzoru, takže se jeho pozice mění automaticky. Pro změnu tvaru jsem vytvořil jednak panel s tlačítky všech tvarů, tak i klávesové zkratky. Při změně tvaru se smaže původní tvar a změní se ukazatel. Ten pak z tabulky vybere odpovídající nový tvar a vytvoří ho. U vzniklého tvaru se nastaví rodič a jeho materiál podle aktuálně vybraného materiálu (lze ho měnit ze stejného panelu).

Pokud drží uživatel stisklé levé tlačítko myši (mimo oblast uživatelského rozhraní), volá se funkce pro umístění dílku „PlaceTileAtCursor“. Tato funkce ale neproběhne, když se od posledního vytvořeného dílku nezměnila vypočítaná pozice. Při puštění myši se poslední pozice nastaví mimo herní plochu. Pokud tedy bude klikat na stejné políčko, vždy se provede změna. Když by tlačítko myši pouze držel, provede se změna jen jednou. Před umístěním nového dílku se nejdříve projde seznam dosud uložených dílků a smaže se dílek na stejné pozici. Nakonec se nový dílek (pokud se nejedná o nástroj guma) uloží do seznamu dílků.

Každý dílek lze kromě změny tvaru a textury ještě natáčet. První řešení bylo provést natočení inkrementem k aktuální rotaci. To ale působilo problémy v rotacích kolem os X a Z, proto jsem vytvořil druhé řešení. Hodnoty rotace jsem uložil samostatně a při otáčení měním pouze tyto hodnoty. Po provedení úprav nakonec natočím objekt do vypočítané rotace.



Obrázek 14: Editor v režimu "Tvorba mapy"

10.2 Správa inventáře

Ve třetím režimu editoru je možné nastavit dostupnost jednotlivých komponent a umístit nějaké nové do scény. Panel s komponentami je generován podobně jako u hry, ale zde se vždycky vypíšou všechny druhy. Pro generování se také používá jiný vzorový panel. Stejně jako u hry je i zde tlačítko pro vytvoření nové komponenty. Každý panel s komponentou tak obsahuje také editační pole a přepínače pro stav inventáře a obchodu. Při načtení panelu komponent se tyto pole přednastaví do aktuálně uložených hodnot.

Editační pole i přepínače jsou registrovány na stejné funkce a volají se při změně parametru. Program nejdříve kontroluje stav přepínače (přepínač značí neomezený počet) a v případě negativní hodnoty se pokusí získat hodnotu z pole. Při neúspěchu se uloží hodnota „0“. Pokud je zaškrtnut přepínač, uloží se hodnota „-1“.

10.3 Správa komponent

Druhý režim editoru je totožný se základním režimem hry. Lze tedy vybírat jednotlivé komponenty a navzájem je propojovat nebo je editovat. Jedinými rozdíly je možnost zamezit další editaci od hráče a tlačítko pro přesun komponenty. Přepínač reaguje na změnu – zavolá se aktualizační funkce, která nastaví komponentě aktuální editovatelnost.

10.4 Sběrače a tvůrci

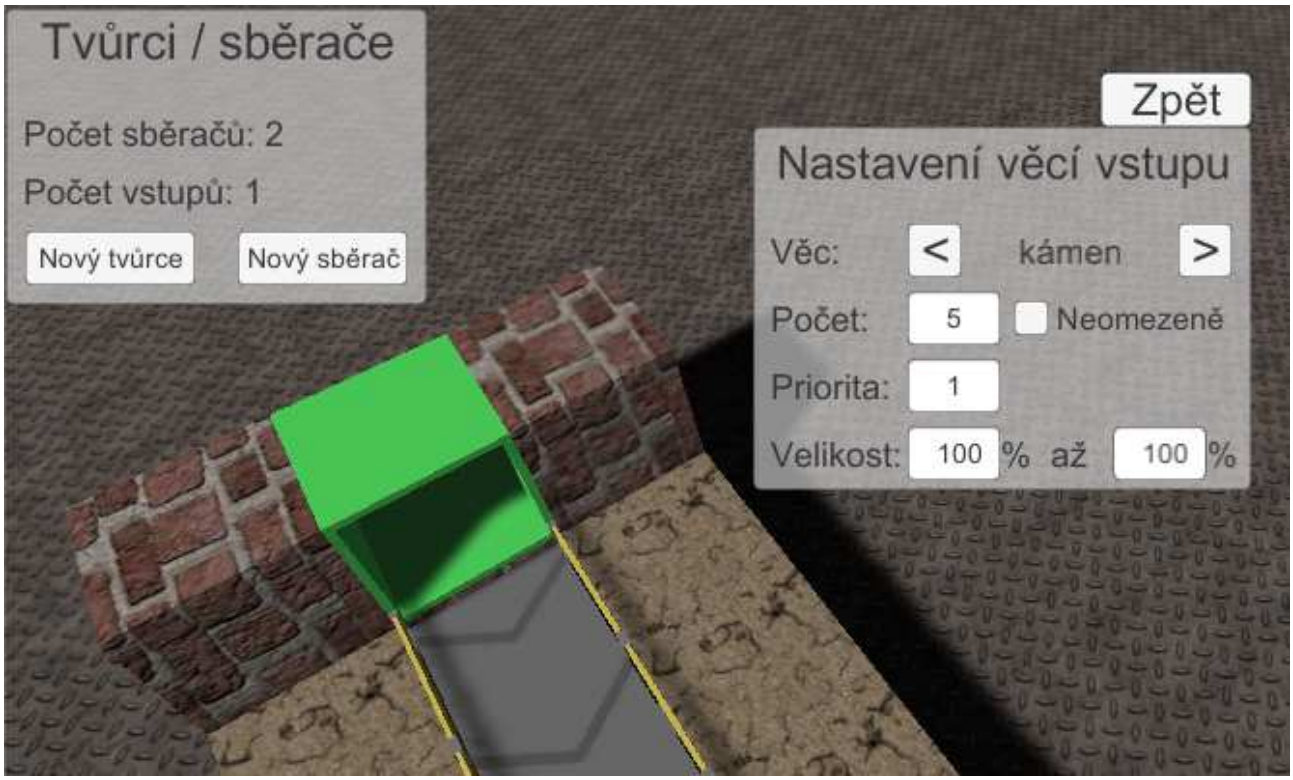
Tento režim je další unikátností oproti hře. V tomto režimu je možné umístit a spravovat sběrače i tvůrce, které tvoří základy úkolového systému. Základní ovládací panel obsahuje pouze tlačítka pro vytvoření nového prvku a jejich aktuální počet (aktualizován při načtení režimu).

Umísťování prvku používá kombinaci přístupů u tvorby mapy i přesunu komponent. Využívá stejný kurzor jako tvorba mapy. Také se pro umísťování zviditelní a správně natočí vymodelovaná šipka ukazující aktuální směr prvku. Stejně jako u tvorby mapy je i zde stejné ovládání natočení (na rozdíl od komponent, které se otáčejí jen kolem osy Y). Se systémem přesunu komponent sdílí toto umísťování stejnou mřížku a kontrolu kolizí.

Po vybrání umístění se prvek uloží do příslušného seznamu a automaticky se vybere. Pokud je nějaký prvek vybrán, zobrazí se další panel s editačními možnostmi. Panely se liší podle toho, jestli se jedná o sběrač nebo o tvůrce. V případě sběrače obsahuje panel pouze identifikační číslo a tlačítko pro smazání. Při mazání se kromě fyzického objektu musí také smazat položka ze seznamu a upravit indexy.

V případě tvůrce se v panelu zobrazí jednotlivé položky, které bude vytvářet. Z důvodu nedostatku místa jsem informace o jednotlivých položkách omezil jen na počet kusů. Při načtení panelu se vytvoří jednotlivé panely pro každou položku daného tvůrce společně s editačními a mazacími tlačítky. Pokud je jejich počet nižší než 4, vytvoří se další panel představující novou položku.

Editace nebo tvorba nové položky probíhá na novém panelu. Zde se buď načtou údaje z vybrané položky (editace), nebo jsou uvedeny ve výchozích hodnotách (nová položka). V okamžiku vytvoření nové sady se tato sada hned ukládá do tvůrce. Při každé změně hodnot, přepínačů nebo stisku tlačítek se volají automatizační funkce, které sadu uloží. Všechny hodnoty mají také nastavený rozsah hodnot, který se aplikuje při ukládání hodnot do tvůrce.



Obrázek 15: Editace konkrétní sady materiálů u tvůrce

10.5 Nastavení scénáře

Nastavení probíhá podobně jako ostatní editační záležitosti: zpracovává se a ukládá se po každé změně. Jedinou výjimkou je tabulka s popisem scénáře – ta se ukládá při ukončení editace. Jednotlivé položky nastavení se řeší přes stejnou aktualizační funkci a nepředstavují žádný komplikovaný problém. Editace popisu scénáře a editace úkolů pak probíhá v nových oknech.

Položka „hranice úrovně“ rozšiřuje úroveň o zadaný počet políček. Při změně se kromě uložení hodnot také zavolá funkce „RecalculateBorders“, která se stará o vytvoření viditelných hranic scénáře. Toho dosáhne získáním minimální a maximální polohy v ose X a Z. K tomu se přidá nastavený bonus z nastavení a podle výsledku se pak provede umístění a zvětšení jednotlivých obdélníků tvořící hranice.

Vytvoření editačního panelu pro úkoly byl obtížný úkol. Protože jsou úkoly ve tvaru stromového grafu, není možné předem odhadnout potřebnou velikost. Proto jsem zvolil řešení, kdy je na panelu jen aktuální sada úkolů nebo konkrétní úkol. Kromě tlačítka pro přesun do kořenového uzlu (což je snadné) je zde i možnost přesunu o úroveň výše. Žádný uzel neobsahuje referenci na rodičovský uzel. Proto jsem vytvořil seznam obsahující posloupnost indexů popisující aktuální zanoření. Při postupu o úroveň výše se odebere poslední a od kořenového uzlu se projde pomocí indexového seznamu na požadovaný uzel.

Při změně uzlu se nejdříve vyhodnotí jeho druh (list nebo uzel). Pro oba druhy jsem pak předem vytvořil příslušné editační panely. Část obsahu panelů je pak generovaná (seznam úkolů, sběrače, typy). Jako u předešlých editací se i zde při každé změně volají ukládací funkce. Editace konkrétního úkolu jsou pak jediný případ, kdy jsem využil nástavby k přepínačům „Toggle Group“ (radio tlačítka).

Obrázek 16: Editace sady úkolů

11 Ukládání a načítání dat

Ukládání a načítání dat představovalo nejobtížnější část celé práce. Musel jsem postupně vyřešit celou řadu problémů, se kterými jsem ve fázi plánování vůbec nepočítal. Unity obsahuje možnost označit třídu jako „Serializable“. Takovéto třídy pak je možné snadno uložit do souboru pomocí nativního binárního kódovače. Stejně tak je snadné pak tyto soubory znovu načíst, pokud skript přesně ví, jakého datového typu jsou jednotlivé části.

Pro uložení dat jsem si tedy vytvořil dvě třídy. První obsahuje pouze hlavičku souboru: jméno scénáře, popis a status scénáře (od vývojáře nebo od uživatele). Druhá třída pak obsahuje všechny ostatní údaje potřebné k uložení a načtení. V následujících podkapitolách se pokusím nastínit jednotlivé problémy, na které jsem narazil a jak jsem je vyřešil (nebo obešel).

11.1 Problém 1: Vector3, Vector2 a Quaternion

Vector3 je datová proměnná obsahující trojrozměrný vektor a k němu příslušné funkce (3 float hodnoty). Vector3 se používá k uložení polohy, rychlosti, směru a v některých případech i rotace. Datový typ Vector2 je pak to samé v dvourozměrném prostoru. Quaternion se pak používá k uložení rotace pomocí 4 float hodnot. Obsahuje ale také funkce pro převod do a z Eulerovských úhlů zadaných přes Vector3.

Unity ale neumožňuje serializaci těchto datových typů. Vector3 přitom potřebuji ukládat dost často. První řešení proto bylo, že bych místo Vector3 používal 3 float hodnoty. To je ale velmi nepřehledné, nepraktické a v některých případech i špatně implementovatelné. Proto jsem se uchýlil k řešení jednoho z programátorů z webu Stack Overflow. Ten vytvořil serializovatelnou verzi Vector3 (tu jsem pojmenoval SVector3). Tuhle třídu jsem pak implementoval do své datové struktury. Sice je stále potřeba převádět z normálního Vector3 do serializovatelného SVector3 a naopak, ale už to není takový problém. Quaternion jsem vyřešil převedením na Vector3 a následně na SVector3.

11.2 Problém 2: MonoBehaviour

MonoBehavior je nativní třída v Unity, od které musí dělit každý skript, který se pak umísťuje na objekt ve scéně. Obsahuje některé předdefinované funkce jako například „Start“, „Update“ nebo „FixedUpdate“, které mají široké využití. Obsahuje také funkce volané fyzikálním enginem Unity.

Unity neumožňuje serializaci MonoBehaviour, tím pádem ani žádné třídy, která dědí od ní. Tento problém zasáhl především ukládání komponent. Proto jsem vytvořil novou serializovatelnou

třídu „SGameComponent“, která dokáže zastoupit všechny komponenty. Jednotlivé komponenty je totiž možné rozlišit pouze pomocí identifikace typu a parametru. Při načítání se pak jednotlivé komponenty vytvoří jako během hry za pomocí těchto několika uložených parametrů.

11.3 Problém 3: GameObject

V tomto momentě už dokáži uložit komponentu a ta se pak zase načte. Problém je ale v uložení propojení kontaktů. Propojení obsahuje odkazy na objekty (datový typ GameObject) a ty není možné serializovat. Proto jsem vytvořil novou třídu „Sconnection“, která vynechává tento seznam objektů (pro uložení ho stejně není potřeba). V případě obvodů se také ukládá jejich posloupnost bodů, která tvoří tvar spojení.

11.4 Problém 4: kopírování objektu

V případě úkolů je potřeba uložit celou strukturu. V tomto případě dochází ale k mělké kopii, takže případné reference odkazují na stejné proměnné. Struktura úkolů navíc obsahovala odkaz na GameObject, takže jsem opět musel vytvořit nové třídy obsahující jen potřebné vlastnosti. Do původní třídy jsem pak dal funkci pro uložení úkolů, která rekurzivně prochází stromem a vrací novou kopii v serializovatelném tvaru. Serializovatelný tvar pak má obdobnou funkci pro načítání.

12 Hlavní menu

Hlavní menu jsem vytvořil až nakonec. Neobsahuje nic, co bych už neudělal na jiném místě hry. To zahrnuje: kontrola uživatelského rozhraní, dynamické generování rozhraní a načítání ze souboru.

Při zvolení editoru nebo nové hry se zobrazí nabídka úrovní. Ta se vytváří prohlédnutím adresáře s úrovněmi a zobrazením všech souborů končící na „.lev“. Z každého souboru se deserializuje pouze hlavička, která obsahuje potřebné údaje k zobrazení do panelu úrovně. V případě nové hry se zobrazují pouze úrovně, které jsou označené jako validní (obsahují úkoly, sběrače a tvůrce).

U editoru se zobrazí seznam dostupných úrovní a možnost vytvořit novou. Pokud se jedná o běžného uživatele (nemá vývojářskou verzi), nezobrazí se mu zde scénáře vyrobené mnou (v nové hře už vidět jsou). Jednotlivé scénáře se ukládají stylem „level[X].lev“, kde [X] představuje trojmístné číslo úrovně. Tento název hráči neovlivní, ale mohou si zvolit název, který se zobrazuje ve výběru úrovně.

Obrázek 17: Výběr úrovně k editaci v hlavním menu

13 Návod k použití

První obrazovkou hry je hlavní menu. Zde je možné si vybrat úroveň, kterou chcete hrát nebo editovat. Výběr se provádí kliknutím na příslušné položky („Nová hra“, „Editor“ a „Ukončit hru“). Při vybrání editoru nebo nové hry se zobrazí nová stránka obsahující jednotlivé dostupné úrovně. Načtení vybrané úrovně se provede tlačítkem „spustit“ v řádku s názvem úrovně. V případě editoru je zde na prvním řádku možnost vytvořit novou úroveň.

Krátká poznámka o hlavním menu

(nastavení kláves?)

13.1 Hra

Po vybrání úrovně se provede její načtení. První, co se ukáže, je panel obsahující název scénáře a jeho popis. V popisu by měly být vysvětleny cíle daného scénáře. Tlačítkem „pokračovat“ se hráč přepne do samotné hry. Tento panel s instrukcemi je možné znovu zobrazit přes tlačítko vlevo nahoře „O scénáři“. Ve stejném panelu je také tlačítko pro návrat do menu – dosavadní činnost se neukládá a je při návratu ztracena. Vedle tohoto kontrolního panelu je panel pro změnu režimu. Využití jednotlivých režimů je popsáno níže. Šikovnou klávesou je také „Escape“, který přerušuje právě prováděnou činnost (umísťování komponenty, výběr propojení atd.).

13.1.1 Pohyb po mapě

Pohyb po mapě je možný dvěma způsoby: přes kontrolní panel a s pomocí myši a kláves. Kontrolní panel obsahuje tlačítka pro pohyb kamery dopředu, dozadu a do stran (4 tlačítka ve obsahující „>“). Taktéž obsahuje tlačítka pro natočení kamery („<=“ a „=>“) a přiblížení kamery („+“ a „-“). Tlačítka pro pohyb a natočení fungují tak dlouho, jak dlouho je tlačítko stisknuté. Tlačítka pro přiblížení pracují s jednotlivými kliknutími. Myší je možné pohybovat s kamerou držením pravého tlačítka myši a posunováním. Zoom se provádí kolečkem myši a natočení se provádí šipkami do stran.

(obrázek s kontrolním panelem)

13.1.2 Obchod / inventář

První prioritou hráče je prohlédnout si prostředí scénáře a dostupné komponenty. Na scéně už mohou být nějaké komponenty předpřipravené. Zbytek komponent je v inventáři a obchodu. Do přehledu komponent, které může hráč získat, se hráč přepne změnou režimu na „Sklad, obchod“. V

levé části se zobrazí nový panel obsahující jednotlivé komponenty, které je možné nějakým způsobem získat. Pokud je možné komponentu získat, vytvoří se vedle dané komponenty tlačítko „Vzít“.

Aby si mohl hráč komponentu vzít, musí mít buď nějakou na skladě (položka „stav“), nebo musí být dostupná v obchodě (poslední řádek). Pokud chce brát hráč komponentu z obchodu, musí mít samozřejmě minimálně stejné množství měny, jako je uvedená cena. Aktuální stav herní měny je zobrazen v kontrolním panelu. V případě, že zde žádná měna zobrazena není, není možné používat obchod, ale pouze inventář.

13.1.3 Přesun komponent

Při pořízení nové komponenty nebo při přesunu stávající se vybraná komponenta přichytí ke kurzoru myši. V pravém horním rohu se zobrazí nový panel s možnostmi pro umístění komponenty. Polohou kurzoru je tak možné určit místo, kam chce hráč komponentu umístit. Kurzor určuje pouze dvě souřadnice, výsledná výška se určuje pomocí kolečka myši nebo tlačítek „+“ a „-“ na panelu umístění. Tlačítka „<=“ a „=>“ pak umožňují natočit komponentu po 90°.

Komponentu je možné umístit jen do oblasti vyznačené červenými pruhy. Také se komponenta nesmí překrývat s jinou komponentou nebo prostředím. Pokud je tohle vše splněno, je možné levým kliknutím komponentu umístit. Tlačítkem „Zrušit“ je možné přerušit umísťování komponenty. Už existující komponenta se vrátí zpátky na své místo a nově přidávaná komponenta se vrátí zpátky do inventáře nebo obchodu. Při vybrání režimu „Přesun komponent“ je pak možné jednotlivé komponenty přesunovat kliknutím na ně.

13.1.4 Propojení komponent

Pokud je hráč v prvním režimu „Informace, propojení“, může kliknutím na komponentu otevřít jeho vlastnosti. Zobrazí se dva panely: panel propojení a editace. Prostřední panel obsahuje všechny vstupy a výstupy vybrané komponenty a jejich napojení na ostatní komponenty. Na levé straně panelu jsou jednotlivé vstupy a výstupy (výstupy jsou červeně). Na pravé straně jsou pak tlačítka propojení.

Pokud tlačítko obsahuje text „Nové spojení“, je možné z daného konektoru vytvořit nový spoj. V opačném případě obsahuje tlačítko název připojené komponenty a jeho kontaktu. Kliknutím na takovéto tlačítko se pak toto spojení přeruší. Při tvorbě nového propojení se objeví výzva k vybrání komponenty k propojení. V této fázi si hráč kliknutím na komponentu vybere cílový objekt. V tom případě se zobrazí uprostřed panel obsahující název a popis vybrané komponenty a seznam

jejich kontaktů (ale ne jejich propojení). Pokud je některý z nich propojitelný, objeví se tlačítko „Propojit“, které obě komponenty propojí.

Aby bylo možné spojit dva kontakty, musí být každý opačného typu (vstup a výstup) a musí mít stejný počet bitů. Kontakt typu vstup navíc může mít jen jedno propojení. Pokud jsou tyto podmínky splněny, je možné vytvořit propojení (včetně propojení kontaktů stejné komponenty).

13.1.5 Editace komponent

Druhý panel (vlevo dole), který se objevuje v režimu „Informace, propojení“, jsou informace a editace. První položka je editační pole se jménem komponenty. Přepsáním je možné toto jméno změnit. Následuje hrubý popis komponenty a tlačítka „Editovat“ a „Smazat“. Smazání odstraní komponentu ze hry a vrátí ji do inventáře nebo obchodu. Editace otevře nový panel, kde je možné nastavit parametr komponenty (liší se podle jejich druhu). V případě elektronických obvodů se otevře kompletní editor.

V případě předem připravených komponent je možnost, že je tvůrce mapy nastavit jako needitovatelné. V tom případě není možné měnit jejich jméno, polohu a parametry. Stále je ale možné měnit jejich propojení.

13.1.6 Editor obvodů

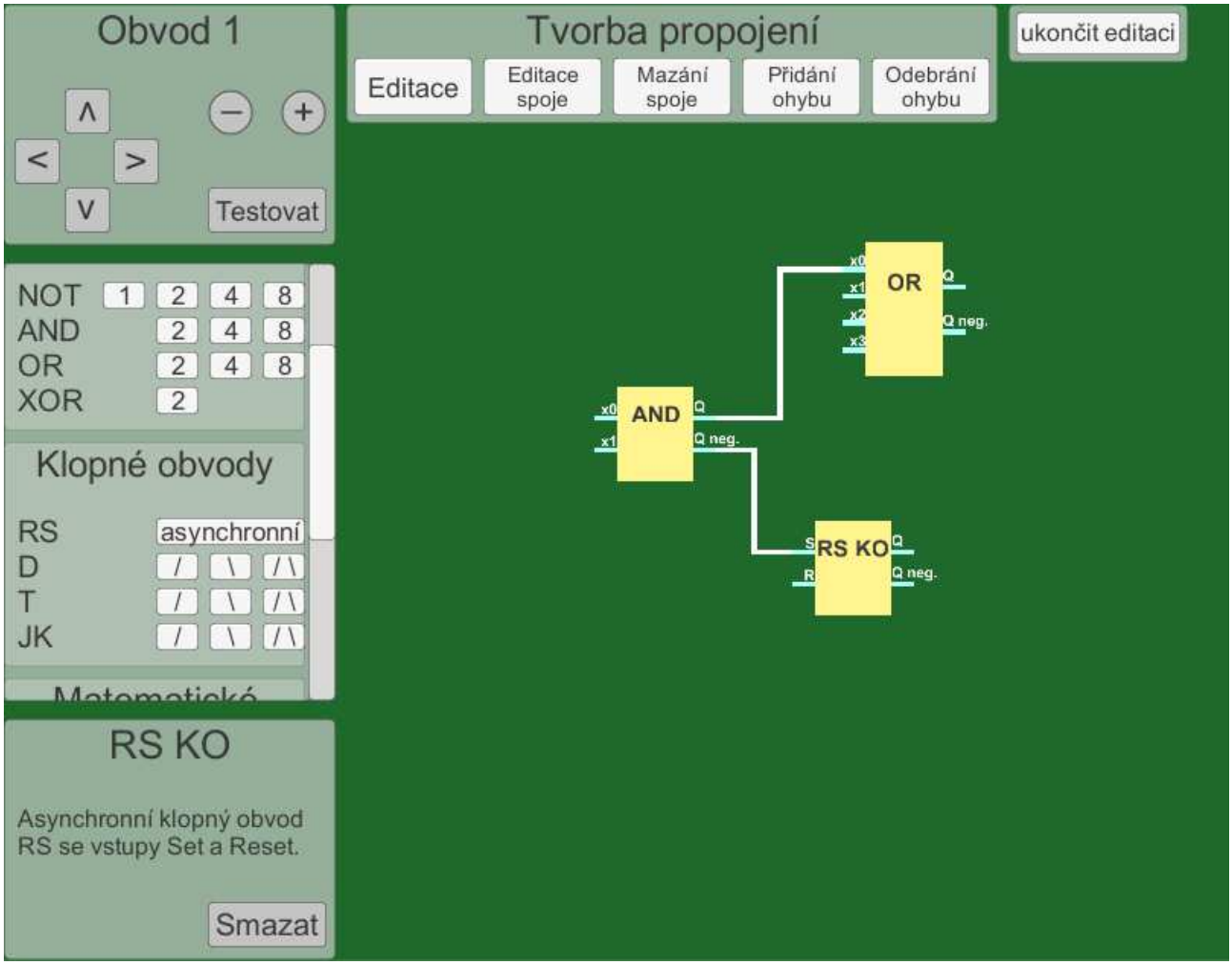
Pohyb po obrazovce obvodu se provádí stejně jako u normální hry, pouze není možné se otáčet. Levý prostřední panel obsahuje jednotlivé bloky, které je možné použít. Většina bloků má více variant (například velikost u hradel, typ hrany u klopných obvodů atd.). Kliknutím na žádaný blok se vytvoří tento blok ve středu obrazovky. V kterémkoliv režimu pak je možné tento blok přesunovat pomocí kliknutí a táhnutí levým tlačítkem myši.

V prvním režimu „Editace“ je možné vybrat jednotlivé bloky nebo kontakty kliknutím na ně. V případě celého bloku se zobrazí název, popis a možnost smazání. V případě konkrétního kontaktu se zobrazí základní informace o něm a v případě vstupů i výchozí hodnota. Tato hodnota se používá v případě, když kontakt není připojen k jinému kontaktu.

Ve druhém režimu „Editace spoje“ je možné kliknutím na kontakt (pokud to není už propojený vstup) začít propojování. To se značí vykreslenou linií od kontaktu ke kurzoru. Dalším kliknutím na jiný kontakt se vytvoří propojení (opět zde platí stejná pravidla jako u komponent). V tomto režimu je pak také možné toto spojení upravit tažením jedné z linií spojení (nesmí se jednat o krajní linii).

Zbylé tři režimy slouží každý k jednomu konkrétnímu účelu, který se provádí kliknutím na spojení. V případě režimu „Mazání spoje“ se vybraný spoj zničí. Režim „Přidání ohybu“ pak přidá další dva rohy do spojení, takže je možné ho víc přizpůsobit situaci. Poslední režim „Odebrání ohybu“ pak odstraňuje přebytečné rohy.

Poslední užitečnou pomůckou je testování obvodu. Testovací tlačítko je na kontrolním panelu (vlevo nahoře) a spustí simulaci běhu obvodu. Červené linie nebo kontakty představují nenulové hodnoty. Kliknutím na 1-bitové vstupy je možné invertovat jejich hodnotu a tím otestovat obvod. Jakmile je hráč s obvodem spokojen, stiskem tlačítka „Ukončit editaci“ se vrátí zpět do hry. Tabulka s propojeními se automaticky aktualizuje podle použitých prvků.



Obrázek 18: Kompletní uživatelské rozhraní u editoru elektronických obvodů

13.1.7 Spuštění běhu hry

Pokud si hráč myslí, že je jeho zařízení hotové, nebo chce otestovat aktuální funkčnost, může spustit hru tlačítkem „Spustit“ v kontrolním panelu. Stejným tlačítkem je také možné proces za-

stavit. Když je hra v běhu, tvůrci (zelené krabice) postupně tvoří předem nadefinovaný materiál. Podle popisu scénáře je pak potřeba materiál roztřídit a rozmístit do jednotlivých sběračů (modré krabice). Během běhu je možné ovlivňovat některé komponenty kliknutím na ně (spínače a přepínače). Také se zobrazí časomíra s aktuálním časem tohoto běhu.

V případě, že hra vyhodnotí výhru, se objeví panel s gratulací. Pokud vyprší časový limit (byl-li nějaký), časomíra změní barvu na červenou a už není možné scénář dokončit (v tomhle běhu). Pokud se neděje nic a už nezbývá materiál k umístění, tak to může znamenat buď špatné splnění instrukcí, nebo špatně nastavený scénář.

13.2 Editor

V editoru je možné vytvořit stejné úrovně, jako jsou ty ukázkové. Pokud se edituje už existující scénář, provede se na začátku automaticky načtení úrovně. V opačném případě se začíná s prázdnou scénou. Pohyb po mapě funguje úplně stejně, jako pohyb ve hře. Kdykoliv je pak možné mapu uložit přes kontrolní panel.

13.2.1 Tvorba prostředí

V prvním režimu „Tvorba mapy“ se provádí konstrukce herního prostředí. V levém panelu se pomocí tlačítek vybere materiál dílku a jeho tvar. Guma (mazání), kostka a blok jsou umístěny samostatně. Ostatní tvary jsou pak umístěny v tabulce. Podobně jako u umístění komponent je i zde pomocný panel s možnostmi umístění. Oproti komponentám je zde také možnost rotovat dílek ve všech 3 osách. Zamknutí osy pak omezí pohyb jen do jedné osy. Terén se do prostředí přidává držením levého tlačítka myši.

13.2.2 Komponenty

Režimy „Správa komponent“ a „Správa inventáře“ odpovídají běžné editaci a obchodu ze hry. V editaci je tu pouze rozdíl v přidání přepínače, jestli bude moct hráč komponentu jakkoliv editovat. Režim přesunu komponent je zde nahrazen dalším tlačítkem v panelu editace. Správa komponent obsahuje všechny možné komponenty a u všech je možné jich libovolné množství umístit. Pomocí přepínačů a editačních polí je pak možné určit, kolik jich bude mít hráč ve skladě nebo obchodě.

13.2.3 Tvůrci a sběrače

Nový panel umístěný pod kontrolním zobrazuje aktuální počet tvůrců a sběračů. Tlačítky pod tím je možné přidat do scény nové. Umístění je podobné jako u komponent, akorát je možné nato-

čení do všech stran. Po umístění se automaticky zobrazí editační panel tohoto prvku. Další způsob, jak zobrazit editační okno, je kliknutí na žádaný tvůrce nebo sběrač.

V případě sběrače je zde pouze možnost smazání. U tvůrce se zobrazí aktuální seznam materiálu, který bude produkovat. Můžou zde být až 4 položky. Tlačítkem „smazat“ se daná položka odstraní, tlačítkem „Editace“ se otevře editační okno vybrané položky. V editačním okně je možné snadno nastavit všechny potřebné parametry. Ty se automaticky ukládají při každé změně. Priorita ovlivňuje pravděpodobnost, s jakou bude vybrána (čím vyšší číslo, tím je větší pravděpodobnost).

13.2.4 Nastavení úrovně

V nastavení úrovně se zobrazí hranice úrovně a editační panel. V editačním panelu je možné nastavit všechny základní parametry scénáře. Pro popis se otevře nové okno s editačním polem. Také v případě nastavení cílů se otevře nové editační okno. Cíle úrovně mají strukturu stromového grafu. Na obrazovce se v každém okamžiku zobrazuje pouze jeden konkrétní uzel nebo úkol.

Pokud se právě nalézá v uzlu, vidí seznam jednotlivých potomků (podúkolů). Poslední položka (pokud jich není příliš mnoho) obsahuje tlačítka pro vytvoření nového úkolového uzlu nebo konkrétního úkolu. Tlačítky „Editovat“ u jednotlivých položek se hráč dostane hlouběji do struktury. Horními tlačítky „Hlavní uzel“ a „O úroveň výše“ je pak možné se vrátit zpět. Tlačítka „Smazat“ pak smažou celou větev. U uzlu je také možnost nastavit styl splnění (horní přepínač). Za normálních okolností je potřeba splnit všechny položky. Pokud je políčko zaškrtnuto, stačí splnit jen jedinou.

Pokud se hráč nalézá v konkrétním úkolu, zobrazí se mu všechny možnosti nastavení úkolu. Prvním krokem je vybrání typu podmínky. Následně vybrat sběrače a typy materiálu, kterých se má podmínka týkat. Výběr symbolu pak udává znaménko rovnice a parametr určuje její pravou stranu. V případě podmínky „čas“ nezáleží na výběru sběračů i předmětů.

14 Závěr

14.1 Cíle práce

1. Vyberte automatizační technologie vhodné pro implementaci do herních mechanismů.
2. Ve zvoleném 3D vývojovém prostředí vytvořte algoritmy pro podchycení všech herních mechanismů.
3. Vytvořte ergonomické uživatelské rozhraní aplikace.
4. Vytvořte sadu edukativních scénářů včetně jednoho průvodce řešením vybraného zadání.
5. Zdokumentujte vývoj, obsluhu a tvorbu uživatelských scénářů.

14.2 Postup

Prvním krokem při vytvoření práce bylo navrhnutí celkové koncepce projektu. Simulátor jsem tedy pojal jako hru, která bude obsahovat jednotlivé scénáře (vytvořené mnou nebo samotnými hráči). Každý scénář pak obsahuje svoje vlastní prostředí, herní prvky i vlastní cíle.

Herní prostředí je tvořené 3D voxelovým systémem obohaceným o možnost rotace jednotlivých bloků. Herní prvky jsou tvořené tzv. Komponentami, které reprezentují jednotlivé reálné technologie. Součástí práce tak bylo vybrání z reálných technologií ty, které budou přínosem pro hru. Tyto technologie se pak zjednoduší a upraví pro potřeby hry – stanou se z nich komponenty. Cílem hráče je poskládat a propojit tyto komponenty tak, aby došlo ke splnění cílů scénáře. Toho se dosáhne především správným přesunem materiálu do sběrných bloků.

Pro hru jsem také vytvořil dva editory. První slouží k editaci a testování elektronických bloků (jedna z komponent). Druhý slouží k vytváření a úpravy jednotlivých scénářů. Pro celou hru včetně editorů jsem také vytvořil uživatelské rozhraní pomocí nativního systému v Unity.

14.3 Výsledný stav práce

Výsledná hra splňuje všechny výše zmíněné cíle. Celá hra je plně funkční a mnohokrát otestována. Během běhu poslední verze jsem nenarazil na žádnou chybu. Pouze při velmi specifických podmínkách může dojít ke špatnému pohybu materiálu, což je způsobené některými z metod pro optimalizaci hry.

14.4 Budoucnost

Přestože tento projekt obsáhl několik různých automatizačních technologií a je plně funkční, existuje tu stále několik možností pro jeho rozšíření. Pokud bych nadále pokračoval ve vývoji této hry, zaměřil bych se na následující vylepšení:

- Přidání více různých automatizačních komponent (písty, vertikální transport, písty atd.).
- Možnost uživatelských účtů.
- Vytvoření systému důlních vozíků pro přepravu materiálu.
- Přidání více druhů materiálu (například výrobky) a s nimi spojené komponenty.
- Možnost upravovat terén v průběhu hry.

15 Seznam použitých zdrojů

- Serializing Vector3 in Unity. 2014. *Wobbleboxx* [online]. [cit. 2015-05-08]. Dostupné z: http://www.wobbleboxx.com/wordpress/?p=395
- 18 FREE Substances. 2014. *Unity Asset Store* [online]. [cit. 2015-05-13]. Dostupné z: https://www.assetstore.unity3d.com/en/#!/content/1352
- UNITY TECHNOLOGIES. 2015. *Unity: Game engine, tools and multiplatform* [online]. [cit. 2015-05-08]. Dostupné z: https://unity3d.com/unity
- Unity: game engine. 2015. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation [cit. 2015-05-08]. Dostupné z: https://en.wikipedia.org/wiki/Unity_%28game_engine%29
- C Sharp. 2015. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation [cit. 2015-05-08]. Dostupné z: https://cs.wikipedia.org/wiki/C_Sharp
- *Blender.org: Home of the Blender project - Free and Open 3D Creation Software* [online]. 2015. [cit. 2015-05-08]. Dostupné z: http://www.blender.org/
- Jak se dělá hra: nesmyslné či přehnané nápady zkrotí herní návrh - iDNES.cz. 2014. *Bonusweb* [online]. [cit. 2015-05-08]. Dostupné z: http://bonusweb.idnes.cz/jak-se-dela-hra-game-design-091-/Magazin.aspx?c=A140219_142315_bw-magazin_oz
- The 13 Basic principles of Gameplay Design. 2009. *Gamasutra: The Art & Business of Making Games* [online]. [cit. 2015-05-08]. Dostupné z: http://www.gamasutra.com/view/feature/132341/the_13_basic_principles_of_.php
- Designing Video Games. 2013. *For Dummies: Making Everything easier* [online]. [cit. 2015-05-08]. Dostupné z: http://www.dummies.com/how-to/content/designing-video-games.html
- The Fundamentals of Game Design. 2010. *Raph's Website* [online]. [cit. 2015-05-08]. Dostupné z: http://www.raphkoster.com/2010/10/12/the-fundamentals-of-game-design/
- Programovatelný logický obvod. 2015. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation [cit. 2015-05-08]. Dostupné z: https://cs.wikipedia.org/wiki/Programovateln%C3%BD_logick%C3%BD_obvod

- Jednoduché programovatelné logické obvody. 2003. *HW* [online]. [cit. 2015-05-08]. Dostupné z: http://www.hw.cz/navrh-obvodu/software/jednoduche-programovatelne-logicke-obvody.html
- Pásový dopravník. 2015. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation [cit. 2015-05-08]. Dostupné z: https://cs.wikipedia.org/wiki/P%C3%A1sov%C3%BD_dopravn%C3%ADk
- Laserové senzory/Senzory pro měření vzdálenosti. 2015. *Ifm* [online]. [cit. 2015-05-08]. Dostupné z: http://www.ifm.com/ifmcz/web/pmain/010_070_030.html
- Váhy. 2015. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation [cit. 2015-05-08]. Dostupné z: https://cs.wikipedia.org/wiki/V%C3%A1hy
- Hopper: particulate collection container. 2015. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation [cit. 2015-05-08]. Dostupné z: https://en.wikipedia.org/wiki/Hopper_%28particulate_collection_container%29