Software pro interaktivní výuku algoritmizace

Bc. Jiří Čepela

Diplomová práce
2024

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2023/2024

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Jiří Čepela**
Osobní číslo:	**A22292**
Studijní program:	**N0613A140022 Informační technologie**
Specializace:	**Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Software pro interaktivní výuku algoritmizace**
Téma práce anglicky:	**Software for Interactive Education of Algorithmization**

Zásady pro vypracování

1. Proveďte informační rešerši shrnující dosažené výsledky v oblasti game-based learning software pro výuku algoritmizace a programování.
2. Porovnejte hlavní technologie využívané pro tvorbu her a zvolte, které využijete pro řešení s ohledem na koncept hry.
3. Formulujte základní herní koncepty, ovládání hry, motivaci hráče, obtížnost, scénář a cíle hry.
4. Vytvořte skripty obsluhující herní mechaniku a v práci popište jejich koncepci.
5. Testujte funkcionalitu hry a soustřeďte se na odstranění nedostatků hratelnosti a logických chyb.
6. Zhodnoťte naplnění vytyčených cílů a navrhněte směry budoucího vývoje projektu.

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

Seznam doporučené literatury:

1. KAPP, Karl M.; BLAIR, Lucas a MESCH, Rich. *The Gamification of Learning and Instruction Fieldbook: Ideas into Practice.* One Montgomery Street, Suite 1200, San Francisco, CA 94104-4594: Wiley, 2014. ISBN 9781118674437.
2. HOLAN, Tomáš. *Unity: první seznámení s tvorbou počítačových her*. CZ.NIC. Praha: CZ.NIC, z.s.p.o., 2020. ISBN 978-80-88168-57-7.
3. CORMEN, Thomas H.; LEISERSON, Charles Eric; RIVEST, Ronald L. a STEIN, Clifford. *Introduction to algorithms.* Fourth edition. Cambridge, Massachusetts: The MIT Press, 2022. ISBN 978-026-2046-305.
4. SCHELL, Jesse. *The Art of Game Design: A Book of Lenses.* 30 Corporate Drive, Suite 400, Burlington, MA 01803, USA: Elsevier, 2008. ISBN 978-0-12-369496-6.
5. SEDGEWICK, Robert a WAYNE, Kevin Daniel. *Algorithms.* Fourth edition. Boston: Addison-Wesley, 2011. ISBN 978-0-321-57351-3.
6. *Unity User Manual.* Online. 2023. Dostupné z: https://docs.unity3d.com/Manual/index.html. [cit. 2023-11-08].

Vedoucí diplomové práce: **Ing. Bc. Pavel Vařacha, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání diplomové práce: **5. listopadu 2023**
Termín odevzdání diplomové práce: **13. května 2024**

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 5. ledna 2024

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne

Jiří Čepela, v. r.
podpis studenta

ABSTRAKT

Tato práce se se zaměřuje na vývoj softwaru s cílem interaktivní výuky algoritmizace a programování pomocí herního konceptu. Práce začíná informační rešerší, která shrnuje dosavadní výsledky v oblasti vývoje softwaru využívajícího herní prvky pro výuku. V praktické části práce jsou porovnány různé frameworky pro vývoj videoher a vybrány nejvhodnější technologie pro tvorbu projektu. Cílem práce je vytvoření komplexního softwaru pro interaktivní výuku algoritmizace. V závěru práce jsou vyhodnoceny splněné cíle a návrhy směru budoucího vývoje projektu.

Klíčová slova: výukový software, algoritmizace, vizuální programování, hra, Unity

ABSTRACT

This thesis focuses on the development of software aimed at interactive teaching of algorithmization and programming using a gaming concept. The thesis begins with an information search summarizing existing results in the field of software development utilizing gaming elements for education. In the practical part of the thesis, various frameworks for game development are compared, and the most suitable technologies for the project are selected. The aim of the thesis is to create comprehensive software for interactive algorithmization teaching. In the conclusion of the work, achieved goals are evaluated, and suggestions for the future development direction of the project are provided.

Keywords: educational software, algorithmization, visual programming, game, Unity

Touto diplomovou prací chci vyjádřit poděkování vedoucímu mé diplomové práce Ing. Pavlu Vařachovi, Ph.D., za odborné vedení, rady a připomínky, které mi pomohly k vypracování této práce. Dále chci poděkovat své rodině, přítelkyni a přátelům za jejich neustálou podporu během celého studia.

Prohlašuji, že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD 8

I TEORETICKÁ ČÁST 9

1 ALGORITMIZACE 10

1.1 Algoritmus 10

1.2 Kroky algoritmizace 11

1.3 Výhody algoritmizace 11

1.4 Vliv výukových softwarů na výuku algoritmizace 12

2 VÝUKOVÝ SOFTWARE 13

2.1 Historie a vývoj výukových softwarů 14

2.2 Software pro výuku algoritmizace 15

2.2.1 Výhody výukového softwaru algoritmizace 15

2.3 Videohra jako výukový software 16

3 GAME-BASED LEARNING 17

3.1 Výhody GBL 17

3.2 Oblasti GBL 18

3.2.1 Herní principy v GBL 18

3.3 Gamifikace vs game-based learning 19

3.4 GBL pro výuku algoritmizace 19

3.4.1 Technologie pro výuku algoritmizace 20

3.4.2 Výsledky v GBL pro výuku algoritmizace a programování 21

3.4.3 Výzvy a omezení spojené s používáním GBL 22

4 HERNÍ ENGINY A TECHNOLOGIE PRO TVORBU HER 23

4.1 Unity 23

4.1.1 Rozhraní Unity 24

4.2 Unreal Engine 25

4.2.1 Rozhraní Unreal Enginu 25

4.3 Godot 26

4.3.1 Rozhraní Godot enginu 27

II PRAKTICKÁ ČÁST 28

5 ANALÝZA JEDNOTLIVÝCH HERNÍCH ENGINŮ 29

5.1 Volba vhodné technologie pro vývoj výukového softwaru 30

6 KONCEPT HRY 31

6.1 Ovládání 31

6.2 Úkoly a obtížnost 31

6.3 Konec hry 31

7 TVORBA HRY 32

7.1 Vytvoření Projektu a Příprava scény 32

7.2 Assety 32

7.3 Herní mechaniky 33

7.3.1 Pohyb hráče a kamery 33

7.3.2 Skripty pro jednotlivé bloky 34

7.3.3 Skripty pro propojení bloků 41

7.3.4 Systém ukládání a načítání hry 44

7.3.5 Otevírání a zavírání dveří 46

7.4 ROZHRANÍ VIZUÁLNÍHO PROGRAMOVÁNÍ ... 47
7.4.1 Rozhraní bloků pro tvorbu programu ... 47
7.4.2 Nápověda v rozhraní ... 49
7.4.3 Skript pro načtení a vypnutí rozhraní ... 51
7.5 VYTVOŘENÍ POSTAVY HRÁČE ... 52
7.6 TVORBA HERNÍHO PROSTŘEDÍ ... 53
7.6.1 Začátek ... 54
7.6.2 Vstupní místnost ... 55
7.6.3 Hlavní hala ... 57
7.6.4 Strojovna ... 58
7.6.5 Můstek ... 60
7.6.6 Laboratoř ... 62
7.7 UŽIVATELSKÉ ROZHRANÍ ... 64
7.7.1 Hlavní menu ... 64
7.7.2 Menu ve hře ... 65
7.7.3 Rozhraní aktuálního úkolu ... 67
7.7.4 Konec hry ... 68
7.8 ZVUK ... 68
7.8.1 Zvuk pohybu hráče ... 69
7.9 SVĚTLO ... 70
8 OPTIMALIZACE A TESTOVÁNÍ ... 72
8.1 PROBLÉMY PŘI VÝVOJI ... 72
9 NÁVRH BUDOUCÍHO VÝVOJE ... 73
10 ZHODNOCENÍ NAPLNĚNÝCH CÍLŮ PRÁCE ... 74
ZÁVĚR ... 75
SEZNAM POUŽITÉ LITERATURY ... 76
SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK ... 80
SEZNAM OBRÁZKŮ ... 81
SEZNAM TABULEK ... 83
SEZNAM PŘÍLOH ... 84

ÚVOD

V současné době, kdy se technologie neustále vyvíjí a software má stále větší význam, je potřeba hledat nové metody, jak motivovat a vzdělávat budoucí generace. Jedním z trendů v oblasti vzdělávání je software pro interaktivní výuku, který umožňuje zábavnou formu výuky daného problému.

Tato práce se zaměřuje na tvorbu softwaru pro interaktivní výuku algoritmizace, který bude vyučovat tvorbu algoritmů během průběhu hraní hry.

Teoretická část této práce se věnuje informační rešerši, která shrnuje dosažené výsledky v oblasti game-based learning softwaru pro výuku algoritmizace a programování. Úvodní část je zaměřena na seznámení se se samotným principem algoritmizace a jaké jsou její výhody. Dále jsou popsány možné přínosy a možnosti výukového softwaru pro algoritmizaci. Další část se věnuje výukovému softwaru a jeho historii a vývoji. Jsou zde popsány možnosti výukového softwaru a jeho výhody pro výuku algoritmizace. Následující část popisuje koncept game-based learning a jeho možnosti, výhody a výzvy při tvorbě hry pro výuku algoritmizace. V závěru teoretické části jsou krátce popsány herní enginy a technologie pro tvorbu her.

Praktická část práce se zaměřuje na tvorbu samotné hry pro interaktivní výuku algoritmizace. V úvodu jsou porovnány vybrané technologie využívané pro tvorbu her a následně zvolena nejvhodnější technologie pro vývoj hry s ohledem na koncept hry. Další část se věnuje vytvoření konceptu hry, který se zaměřuje na popis scénáře, ovládání hry, obtížnosti, mechaniky algoritmizace a cíl hry. Následující část popisuje postup při vytváření samotné hry. V této části jsou popsány využité nástroje a objekty při tvorbě hry. Také detailně popisuje jednotlivé vytvořené mechaniky s jejich skripty a použití těchto mechanik v samotné hře. Další část se věnuje testování a optimalizaci hry, kde jsou popsány postupy pro odstranění chyb a zmenšení nároků aplikace. Poslední část se věnuje zhodnocení dosažených výsledků a navržení směrů budoucího vývoje projektu.

I. TEORETICKÁ ČÁST

1 ALGORITMIZACE

Algoritmizace je popis procesu a implementace algoritmů pro řešení daného problému, který vede od vstupních údajů k požadovaným výsledkům. Jedná se tak o logický postup při rozdělení komplexního problému na menší a jednodušší kroky, které lze popsat samotným algoritmem. Algoritmizace se využívá v různých oblastech informatiky, ale také se může vyskytovat v samotných problémech v každodenním životě.[4]

1.1 Algoritmus

Samotný algoritmus se dá popsat jako jakýkoliv dobře definovaný postup, který přijímá určitou hodnotu ze vstupu, a za daný čas tak vyprodukuje určitou hodnotu jako výstup. Jedná se tedy celkově o posloupnost kroků, které mění daný vstup na výstup. Algoritmus lze také popsat jako nástroj, který napomáhá vyřešit výpočetní problém. Tento problém specifikuje obecně požadovaný vztah mezi vstupem a výstupem. Tímto pak algoritmus popisuje konkrétní postup pro dosažení tohoto vztahu.[4]

Při vytváření počítačového programu, se obvykle implementuje algoritmus, který byl předem vytvořen k řešení určitého problému, a který se program snaží řešit. Tato metoda je často nezávislá na konkrétním programovacím jazyce, který se používá, a je tak vhodná pro mnoho programovacích jazyků. Jedná se tedy o metodu, nikoliv samotný počítačový program, která specifikuje kroky, které lze podniknout k řešení problému. Termín algoritmus se v informatice používá k popisu konečné, deterministické a efektivní metody řešení problému, která je vhodná k implementaci jako počítačový program.[6]

Algoritmus musí splňovat několik základních kritérií. První z těchto kritérií je konečnost. To znamená, že algoritmus musí dokončit svůj úkol během konečného počtu kroků a nesmí zůstat uvězněn v nekonečném cyklu za žádných okolností. Druhou důležitou vlastností je resultativnost, která vyžaduje, aby algoritmus po skončení konečného počtu kroků vracel výsledek, případně chybové hlášení.[6]

Třetím kritériem je determinovanost, která je klíčovou vlastností algoritmu stanovující závislost pouze na jasně daných krocích a vstupních datech, aniž by závisel na osobě nebo zařízení, které ho provádí. Algoritmus musí být sestaven z elementárních aktivit, jejichž provádění je jasně definované a jednoznačné. Opakovatelnost je další důležitou vlastností algoritmu, která určuje, že pro stejný vstupní soubor hodnot vrací vždy stejný výsledek. Hromadnost algoritmu zase určuje, že je psán obecně a nezávisí na konkrétním souboru

hodnot. Dalším klíčovým prvkem vlastností algoritmu je srozumitelnost, která vyžaduje, že požadovaný vstup je nutné psát tak, aby byl jednoznačný pro jeho tvůrce i pro ty, kteří ho budou provádět. Jeho instrukce musí být přizpůsobeny cílové skupině uživatelů, kterými mohou být například děti, studenti nebo počítače.[6]

1.2 Kroky algoritmizace

Analýza problému či jednotlivých úloh je klíčovým prvkem procesu vývoje softwaru. Nejprve je nutné pečlivě definovat a analyzovat daný problém, který je potřeba řešit. To zahrnuje porozumění požadavkům uživatele, identifikaci hlavních cílů a potřeb, a také zhodnocení možných omezení a očekávaných výsledků. Poté následuje návrh algoritmu, který popisuje kroky a postupy, jak vyřešit daný problém. Tento algoritmus by měl být navržen tak, aby byl co nejefektivnější a zároveň aby byl jednoduše implementovatelný.[4]

Po dokončení návrhu algoritmu následuje fáze implementace. V této fázi se algoritmus převádí do konkrétního programovacího jazyka nebo prostředí. Je důležité zajistit správnou implementaci algoritmu a dodržení všech specifikací a požadavků. Po implementaci následuje fáze testování a ladění. Algoritmus je tak otestován s různými vstupy a je ověřeno, zda vrací správné výsledky a zda není náchylný na chyby nebo nekonzistence.[4]

Během testování je také možné odhalit případné chyby nebo nedostatky v implementaci, které je potřeba opravit. Po úspěšném testování a odstranění všech nalezených chyb je možné přistoupit k fázi optimalizace. Cílem optimalizace je dosažení maximální efektivity algoritmu. To může zahrnovat změny v implementaci nebo úpravy algoritmu tak, aby byl rychlejší a efektivnější.[4]

Je tedy důležité postupovat systematicky a metodicky v každé fázi vývoje softwaru. Díky správné analýze, návrhu, implementace, testování a optimalizace lze dojít k úspěšnému vytvoření a nasazení funkčního a spolehlivého softwaru.[4]

1.3 Výhody algoritmizace

Algoritmizace přináší řadu výhod v procesu řešení problémů. Jednou z hlavních výhod je zvýšení efektivity umožňující řešit problémy efektivněji a s menším úsilím. Další výhodou je snížení chybovosti, jelikož jasně definované kroky v algoritmu snižují riziko chyb a nesrovnalostí. Algoritmizace také přispívá k zvýšení srozumitelnosti, neboť umožňuje srozumitelně popsat a sdílet postupy řešení problémů. Důležitou výhodou je i zajištění

opakovatelnosti, díky tomu, že algoritmy umožňují opakovaně a konzistentně dosahovat požadovaných výsledků.[4]

1.4 Vliv výukových softwarů na výuku algoritmizace

Výukové softwary pro algoritmizaci mají pozitivní vliv na výuku algoritmizace v mnoha různých ohledech. Nejen, že zvyšují zájem a motivaci studentů tím, že nabízejí interaktivní a zábavný přístup k výuce algoritmizace, které studenty více zapojuje a motivuje k učení, ale také zlepšují pochopení algoritmů prostřednictvím vizualizací a interaktivních cvičení. Tyto prvky tak mohou napomoct studentům lépe porozumět složitým úkolům a umožňují jim si je tak i lépe zapamatovat. V kombinaci s interaktivním prostředím poskytují studentům možnost praktického odzkoušení a experimentování s algoritmickými principy, které je zásadní pro jejich učení a samotný rozvoj dovedností a zkušeností v oblasti algoritmizace.[13]

2 VÝUKOVÝ SOFTWARE

Výukový software představuje klíčovou roli v moderním a digitálním vzdělávání, který poskytuje širokou škálu možností pro výuku a samostudium. Tato kategorie počítačového softwaru má za cíl podpořit proces vzdělávání a samotný rozvoj dovedností u uživatelů. Jedná se nejen o počítačové programy, ale také online služby navržené speciálně pro podporu vzdělávání. Jeho hlavním cílem je usnadnit proces učení a výuky prostřednictvím interaktivních prvků a speciálně navržených funkcí. Edukační software má velké uplatnění jak ve školních prostředích, kde může být součástí výukového procesu, tak i při individuálním studiu doma. Jeho interaktivní charakter a různorodé funkce mohou významně přispět k efektivnějšímu a zábavnějšímu učení, a to jak studentů, tak i pedagogů, kteří mohou využít tento software jako doplňkový nástroj ke klasické výuce.[1]

V současné době vzdělávací software zahrnuje širokou škálu nástrojů a aplikací, které jsou nedílnou součástí moderního školství. Moderní vzdělávací software umožňuje rodičům sledovat pokrok svých dětí ve škole a poskytuje jim přístup k informacím a zprávám týkajících se vzdělávacího procesu. Existuje několik typů edukačního softwaru. V první řadě to jsou výukové programy, které jsou zaměřeny přímo na výuku konkrétního předmětu nebo dovednosti. Tyto programy často obsahují různé interaktivní prvky, jako jsou hry, online lekce, testy a další nástroje, které napomáhají studentům výuku lépe pochopit a procvičovat si nové koncepty. Další formou tohoto vzdělávání jsou také aplikace, které nabízejí širokou škálu možností. Mohou to být například jazykové aplikace, které pomáhají s učením cizích jazyků, aplikace zaměřené na matematiku, historii a další předměty nebo také i aplikace sloužící k rozvoji paměti, logiky, kreativity a dalších dovedností.[1]

Je také nutné podotknout, že vzniká stále více organizací, které se specializují na vývoj vzdělávacího softwaru, a to jak online, tak i offline. Tyto systémy se zaměřují na poskytování personalizovaných a interaktivních vzdělávacích zážitků pro studenty i učitele. Mezi hlavní výhody vzdělávacího softwaru patří integrace multimediálního obsahu a poskytování interaktivních prvků. Díky tomu se lze odklonit od tradičních výukových metod a nabízí studentům nové možnosti zapojení se a zaujetí například v podobě multimediálního obsahu, jako je například grafika, obrázky a zvuk. Tyto možnosti tak umožňují studentům interaktivní zážitek a zapojení se do výuky. Výhodou pro učitele je také lepší komunikace se studenty a udržení tak jejich zájmu o výuku. Díky výukovému softwaru lze vytvářet

produktivní prostředí pro vzdělávání a poskytuje moderní nástroje pro efektivní výuku a rozvoj dovedností.[1]

2.1 Historie a vývoj výukových softwarů

Samotný vývoj výukových softwarů spočívá v široké škále možností jako je například strukturování informací, hodnocení znalostí studentů a poskytování okamžité zpětné vazby, a to za cílem zvýšení automatizace výuky bez lidského zásahu, kromě fáze návrhu a implementace. Experimenty B.F. Skinnera s výukovými stroji v roce 1954, založené na behaviorismu, označily počátek výukových softwarů a raných počítačových výukových iniciativ. Tyto výukové stroje představovaly jednu z prvních forem počítačově podporované výuky a položily tak základ k automatizovaným vzdělávacím metodám.[10]

Jedním z nejvýznamnějších systémů, které vznikly, byl PLATO, systém počítačem podporované výuky původně vyvinutý na univerzitě v Illinois. Kolem konce 70. let měl PLATO tisíce terminálů po celém světě představující komplexní online prostředí s funkcemi jako fóra, testování a zasílání zpráv – předchůdce moderních e-learningových platforem.[8]

V 70. a 80. letech došlo k významnému pokroku v oblasti výukového softwaru díky rozvoji osobních počítačů. Software jako Logo a HyperCard umožnil učitelům a studentům vytvářet vlastní výukové materiály. Tyto nástroje také umožnily integraci multimediálních prvků, jako jsou obrázky a zvuk, které výrazně obohatily možnosti interaktivního učení.[9]

V polovině 80. let 20. století začaly snahy o replikaci výukových procesů prostřednictvím umělé inteligence (AI), zaměřené především na předměty, jako je například aritmetika. Navzdory značným investicím se aplikace umělé inteligence ve výuce potýkala s problémy při přizpůsobování se různým stylům učení a chování studentů. Nedávné pokroky v kognitivní vědě a neurovědě nabízejí slibné poznatky, ale také značné mezery stále přetrvávají mezi vědeckým výzkumem a praktickou implementací ve vzdělávacích prostředích.[10]

V 90. letech a na počátku 21. století přinesl internet další revoluci v oblasti výukového softwaru. Vznikly nové formy online výuky, jako jsou virtuální učebny a e-learningové platformy. Tyto platformy umožňují studentům přístup k široké škále výukových materiálů a interaktivních aktivit bez ohledu na čas a místo.[10]

V posledních letech se prosadilo také adaptivní učení, které analyzuje reakce studentů a přizpůsobuje poskytování obsahu na základě jejich výkonu. Související vývoj v oblasti

analýzy učení shromažďuje a analyzuje údaje o studujících s cílem informovat o výukových rozhodnutích ke zlepšení samotných výsledků. Tyto inovace představují významný pokrok v personalizovaném učení a vzdělávání založeném na datech.[10]

V současné době je výukový software nedílnou součástí moderního vzdělávacího procesu. S rozvojem technologií jako je umělá inteligence a virtuální a rozšířená realita se otevírají nové možnosti pro další vývoj a inovace v této oblasti. Cílem je vytvořit ještě poutavější a efektivnější výukové prostředky, které budou lépe odpovídat individuálním potřebám a učebním stylům.[10]

2.2 Software pro výuku algoritmizace

Výukový software pro algoritmizaci představuje inovativní a moderní nástroj pro výuku samotných základů algoritmizace a programování. Tyto softwary se odlišují od tradičních metod výuky interaktivním přístupem, který umožňuje studentům se aktivně zapojit do procesu učení a experimentovat s algoritmy a programovacími koncepty. Výukový software nabízí širokou škálu funkcí a aktivit, které studentům pomáhají lépe porozumět principům algoritmizace a rozvíjet tak jejich logické a programovací dovednosti.[11]

Existuje mnoho výukových softwarů pro algoritmizaci, které se odlišují svým zaměřením, funkcemi a cílovou skupinou. Mezi nejběžnější typy patří například simulační software, výukové hry a vzdělávací platformy. Simulační software umožňuje studentům experimentovat s algoritmy a programovacími koncepty v simulovaném prostředí. Tento typ softwaru je užitečný pro pochopení fungování algoritmů a umožňuje tak pozdější testování různých programovacích strategií.[11]

2.2.1 Výhody výukového softwaru algoritmizace

Výukové softwary pro algoritmizaci přináší řadu výhod ve srovnání s tradičními metodami výuky algoritmizace. První z těchto výhod je interaktivita, která umožňuje studentům aktivně se zapojit do procesu učení a experimentovat s algoritmy a programovacími koncepty. Tato interaktivita přispívá k hlubšímu porozumění algoritmizace a podporuje rozvoj logického myšlení u studentů. Dalším důležitým prvkem je vizualizace, kterou výukový softwary často využívají k zobrazení algoritmů a programových konceptů. Tento vizuální přístup umožňuje studentům lépe porozumět složitějším konceptům a sledovat celkový průběh algoritmů krok za krokem. Další významnou výhodou je personalizace, díky které se dokáže interaktivní výukový software lépe přizpůsobit individuálnímu tempu

a úrovni znalostí studentů. Tímto individuálním přístupem napomáhá studentům učit se vlastním tempem a zaměřit se na oblasti, ve kterých potřebují nejvíce podpory. Software také motivuje studenty k učení algoritmizace a programování díky své zábavné a interaktivní povaze. Tato motivace je klíčová pro udržení zájmu a angažovanosti studentů při procesu učení. Výukové softwary jsou dostupné online i offline, čímž mohou umožnit studentům přístup odkudkoliv. Tato dostupnost tak přináší flexibilitu nejen v samotném učení, ale také umožňuje studentům jej využívat v souladu s jejich individuálními preferencemi a potřebami. Díky těmto vlastnostem se interaktivní výukové softwary stávají efektivním nástrojem pro výuku algoritmizace a programování, který podporuje rozvoj dovedností, zkušeností a motivaci studentů.[11]

2.3 Videohra jako výukový software

Videohry jsou prostředí vytvořené ve virtuálním světě s pevně stanovenými pravidly a cíli, které hráče vyzývají k plnění úkolu a současně zaměstnávají jejich pozornost velkým množstvím rozhodnutí. Flexibilita prostředí videoher je pro vzdělávací účely přínosná, neboť umožňuje vysokou míru kontroly nad funkcemi a vlastnostmi hry. Tato adaptabilita je pro organizace strategická, protože herní scénáře lze jednoduše přizpůsobit různým potřebám a specifickým cílům dle požadavků. V posledních letech je zaznamenáván vzrůstající trend využívání videoher i pro jiné účely než jen pro zábavu.[12]

Odborníci v těchto oblastech tak očekávají další rozvoj a rozšíření použití videoher do nových oblastí, jako je například výuka žáků na základních školách, interaktivní expozice v muzeích nebo školení a rozvoj zaměstnanců ve firmě.[12]

3 GAME-BASED LEARNING

Game-based learning se používá k motivaci studentů a zpříjemnění samotného procesu výuky za pomoci přidání herních prvků pro zábavu. Tento přístup má pozitivní vliv na kognitivní vlastnosti studentů. Implementace her do výukových kurzů je jedním možným způsobem, jak překonat stereotypní a pro někoho i nudný tradiční vzdělávací proces, jelikož prostředí hry může výrazně zlepšit motivaci studentů k učení. Když jsou studenti zapojeni do hry, jejich soustředěnost se zvyšuje a věnují tak více pozornosti dané látce. GBL přináší novou možnost využití počítačových her jako možný nástroj pro náročné učení, včetně oblastí jako je řízení zdrojů, programování, finance a zdravotnictví. Herní výuka umožňuje studentům prožívat samotné učení jako hru, což zvyšuje jejich zájem a motivaci k výuce dané látky. Během procesu herního učení jsou důležité dva prvky, kterými jsou zajímavost a zábava. Tyto aspekty pomáhají studentům vytvářet efektivní učící se prostředí, které je relaxační a zároveň motivující. Díky digitální herní výuce mohou studenti rozvíjet základní dovednosti a znalosti v oblastech, které jsou klíčové v současné digitální době.[14]

3.1 Výhody GBL

Výhody game-based learningu jsou mnohostranné a významně přispívají k učebnímu procesu. Implementace her do vzdělávacího prostředí má množství pozitivních efektů, které ovlivňují rozvoj studentů a jejich motivaci k učení.[14]

Jednou z hlavních výhod GBL je rozšiřování paměti studenta. Herní prostředí často vyžaduje zapamatování informací, pravidel a strategií, což stimuluje kognitivní funkce mozku a posiluje samotnou paměť. Kromě toho se v herním prostředí často setkáváme s různými úkoly a výzvami, což podporuje rozvoj kritického myšlení. Další významnou výhodou je rozvoj počítačové gramotnosti. Hraní her vyžaduje porozumění moderním technologiím a digitálnímu prostředí, což přispívá k lepšímu osvojení počítačových znalostí. GBL také podporuje strategické myšlení. V herním prostředí se často musí hráči rychle rozhodovat a plánovat své akce, což posiluje schopnost strategického myšlení a adaptace na různé situace. Kromě toho pomáhá herní prostředí rozvíjet rychlé vnímání studenta. Mnoho her vyžaduje precizní a rychlé pohyby, což podporuje synchronizaci mezi vizuálním vnímáním a pohyby rukou. Také motivuje studenty k učení zábavným a interaktivní herním prostředí, které vytváří příjemné prostředí pro učení. Toto má za cíl zvýšení motivace studentů a podněcování k aktivní účasti při učení se dané látky.[14]

3.2 Oblasti GBL

GBL nabízí mnoho možností uplatnění se v různých oblastech vzdělávání. Jednou z hlavních oblastí jsou technicky náročné a pro výuku nezábavné látky. Zde hry přicházejí jako nástroj pro oživení procesu výuky a zvýšení zájmu studentů. Například pro výuku matematiky, fyziky nebo programování mohou být hry velmi efektivním prostředkem k zapojení studentů a mohou usnadnit porozumění komplexním úkolům.[14]

GBL je také používáno pro výuku předmětů, které jsou náročné na pochopení. Hry v těchto případech mohou poskytnout interaktivní prostředí pro experimenty, zkoumání a aplikaci znalostní daného učiva, což může pomoci lépe a snadněji porozumět obtížným látkám. GBL může také poskytnout možnosti simulace různých scénářů a zkoumat jejich důsledky v digitálním prostředí, což rozvíjí kritické myšlení a strategické plánování.[14]

Další důležitou oblastí jsou případy, kde studenti mají určité specifické potřeby. Například pro vzdělávání studentů s různými typy zdravotních postižení mohou být hry přizpůsobeny tak, aby lépe odpovídaly jejich individuálním potřebám a zároveň pomohli pochopit výukovou látku.[14]

3.2.1 Herní principy v GBL

Herní principy jsou základem GBL. Výuka pomocí her často obsahuje jasně definované cíle a úkoly, které budou studenti plnit. Tyto úkoly mohou zahrnovat řešení hádanek, vytvoření funkční mechaniky nebo dokončení herního levelu. Hra většinou nabízí postupně zvětšující se obtížnost, kdy úkoly vyžadují pokročilejší dovednosti a znalosti dané látky. Tento princip podporuje postupné učení a rozvoj dovedností. Samotné prostředí hry také umožňuje okamžitou zpětnou vazbu na akce studentů, což umožňuje identifikování chyb a naučení se z nich.[15]

Motivace a odměny jsou další důležitou součástí hry. Pro motivování studenta mu mohou být udělovány body, odznaky a samotný postup ve hře. Tyto prvky podporují motivaci studentů k účasti a snaze dosáhnout lepších výsledků. Některá herní prostředí také podporují spolupráci a soutěžení mezi studenty, což může vést k zvýšenému zapojení do hry a tím i výuky dané látky.[15]

3.3 Gamifikace vs game-based learning

Gamifikace a game-based learning jsou dva přístupy, které využívají herní prvky nejen k motivaci, ale také k navýšení samotného angažování účastníků. Rozdíl mezi nimi pak spočívá v jejich zaměření a v samotném použití těchto prvků.[14]

Gamifikace je proces, při kterém jsou herní prvky, mechaniky a myšlení, integrovány do neherních aktivit s cílem motivovat účastníky. To znamená, že se tyto prvky využívají ke zvýšení angažovanosti a odměňování účastníků za jejich úspěchy a pokroky. Gamifikace se může vztahovat na různé aktivity, včetně vzdělávacích procesů, pracovního prostředí nebo dokonce zdravotní péče.[14]

Na druhé straně se GBL zaměřuje přímo na využití her jako prostředku k učení. Jednotlivé hry jsou integrovány do vzdělávacího procesu jako součást učebního plánu. GBL tak využívá herní prvky, jako jsou pravidla, bodové systémy, úrovně a mnoho dalších. Obvykle je upravuje tak, aby zapadaly do příběhu a scénářů dané hry, což umožňuje výukové materiály prezentovat a zpracovávat prostřednictvím interaktivních herních zážitků.[14]

Zatímco se gamifikace zaměřuje na celý proces a integruje herní prvky do neherních aktivit, GBL se soustředí přímo na použití her ke zlepšení učebního procesu a využívá tak herní prvky v rámci těchto her, aby podpořil samotné učení a angažovanost studentů.[14]

3.4 GBL pro výuku algoritmizace

Game-based learning se stává stále populárnější metodou výuky, která využívá herní principy a mechaniky pro podporu učení v různých oblastech. V kontextu algoritmizace a programování má GBL potenciál zpřístupnit a zprostředkovat abstraktní koncepty zábavným a interaktivním způsobem, čímž podporuje nejen motivaci, ale také hlubší pochopení u studentů.[16]

3.4.1 Technologie pro výuku algoritmizace

V game-based learningu jsou využívány různé technologie k vytvoření interaktivních a zábavných herních prostředí pro výuku algoritmizace.[16]

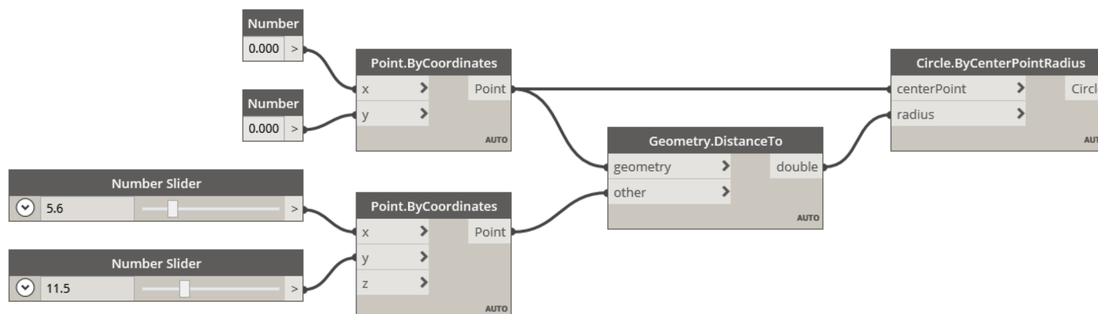
Mezi nejběžnější technologie patří vizuální programování, které zahrnuje nástroje jako Scratch[36], Alice[37] nebo Blockly[38]. Tyto nástroje tak umožňují studentům vytvářet algoritmy a programy pomocí grafických bloků, přetažením a spuštěním, což vede k vhodnějšímu přístupu pro začátečníky a studenty, kteří se teprve seznamují s algoritmizací.[16]

Kromě toho jsou využívána simulační prostředí, která umožňují studentům experimentovat s různými algoritmickými koncepty a sledovat tak jejich chování v reálném čase. Tato prostředí mohou simulovat různé situace, jako je například řízení robotů, procesy vývoje softwaru nebo dynamiku datových struktur. [16]

Vizuální programování

Vizuální programování představuje moderní metodu tvorby programů, která nahrazuje tradiční textové příkazy grafickými objekty. Těmito objekty, často zobrazovanými ve formě ikon, bloků nebo symbolů, se vytvářejí logické struktury, které definují chování programu. Jedním z hlavních atributů vizuálního programování je jeho intuitivnost a přístupnost, které ho činí atraktivním i pro začátečníky, kteří nemají zkušenosti s textovým programováním.[18]

Základní princip vizuálního programování spočívá v používání grafických bloků, které reprezentují různé funkce a operace (Obrázek 1). Tyto bloky se následně spojují do řetězců nebo mřížek, které definují postup instrukcí v programu. Vizuální programovací prostředí navíc často nabízí interaktivní nástroje pro ladění a testování programů.[18]



Obrázek 1. Vizualizace programu [17]

Mezi hlavní výhody vizuálního programování patří snadné učení, díky intuitivnímu grafickému rozhraní, které je srozumitelné i pro začátečníky. Grafické znázornění programu také usnadňuje pochopení jeho logiky a struktury, což vede k rychlejšímu vývoji aplikací bez nutnosti psát rozsáhlý kód. Kromě toho pomáhá vizuální programování snižovat počet chyb v programech, protože grafické bloky omezují možnost syntaktických chyb.[18]

Nicméně, existují i určité nevýhody spojené s vizuálním programováním. Mezi ně například patří omezená komplexnost, protože vizuální programovací jazyky mohou být méně vhodné pro tvorbu komplexních programů s rozsáhlou logikou. Další nevýhodou může být také omezená flexibilita, zejména pokud jde o specifické úpravy a optimalizace, které mohou být obtížně proveditelné v rámci vizuálního prostředí.[18]

3.4.2 Výsledky v GBL pro výuku algoritmizace a programování

Software pro výuku algoritmizace a programování představuje relativně nový směr v oblasti vzdělávání. Dosavadní výzkumy, avšak naznačují jeho slibné výsledky. Tato forma výuky má potenciál stát se nejen efektivním, ale také zábavným nástrojem, který pomůže studentům lépe porozumět principům algoritmizace a programování, a rozvíjet tak jejich dovednosti. Je ale nutné si uvědomit, že ne všechny GBL softwary jsou stejně účinné. Při tvorbě vhodného GBL softwaru pro výuku algoritmizace a programování je třeba vzít v úvahu několik faktorů. Jedním z nich je věk a úroveň znalostí studentů. Software by měl být navržen tak, aby byl přizpůsoben věkové kategorii a úrovni znalostí konkrétní skupiny studentů. Věková kategorie a úroveň znalostí studentů mohou mít významný dopad na účinnost GBL softwaru. Například software navržený pro mladší studenty by měl být jednodušší a intuitivnější, s důrazem na zábavnost a interaktivitu. Zatímco software určený pro pokročilé studenty by měl poskytovat vyšší úroveň náročnosti a podporovat rozvoj komplexních dovedností v oblasti algoritmizace a programování.[19]

Dalším faktorem, který je potřeba zvážit při tvorbě GBL softwaru, je jeho pedagogická hodnota a přizpůsobitelnost cílům výuky. Ideální software by měl poskytovat strukturovaný a cílený obsah, který je přizpůsoben vzdělávacím potřebám a cílům daného kurzu nebo programu. Zvážení věkové kategorie a úrovně znalostí studentů a pedagogické hodnoty může pomoci zajistit efektivní a poutavou výukovou zkušenost pro studenty.[19]

3.4.3 Výzvy a omezení spojené s používáním GBL

Výzvy a omezení spojené s používáním GBL představují klíčové faktory, které mohou ovlivnit úspěšnost a efektivitu výuky. Mezi ně lze například zařadit technické problémy, které mohou vzniknout při implementaci GBL do vzdělávacích prostředí. Tyto problémy zahrnují nedostatečnou dostupnost technického vybavení, nepředvídané technické chyby nebo nekompatibilitu s existujícími systémy. Dalším omezením je nedostatek teoretických rámců, které by podpořily efektivní design a implementaci GBL. Absence jasných metodik a standardů může ztížit plánování a realizaci výukových programů založených na hrách.[20]

Důležitým aspektem je také potřeba dobře navržených instrukčních kontextů, které by umožnily efektivní využití potenciálu GBL. Bez správného nastavení a podpory, ze strany pedagogů a vzdělávacích institucí, může být použití GBL neefektivní a neúčinné. Další výzvou může být odpor studentů vůči novým výukovým metodám. Někteří studenti mohou preferovat tradiční formy výuky a být skeptičtí vůči přijetí nových technologií a her jako vzdělávacího nástroje.[20]

Vzhledem k tomu, že mnoho her je navrženo pro individuální hraní, může být obtížné podporovat spolupráci a interakci mezi studenty. Toto omezení vyžaduje pečlivou pozornost při designu her a výběru vhodných výukových scénářů, které podporují spolupráci a sdílení znalostí mezi studenty.[20]

Všechny tyto výzvy by se měly zohlednit při plánování, implementaci a hodnocení výukových programů. Správné pochopení těchto faktorů a jejich řešení může přispět k úspěšnému využití GBL jako efektivního a účinného nástroje pro výuku a rozvoj dovedností studentů.[20]

4 HERNÍ ENGINY A TECHNOLOGIE PRO TVORBU HER

Herní engine je software umožňující vývoj a tvorbu her. Herní enginy jsou klíčovým prvkem v procesu vývoje her a mají zásadní vliv na konečný vzhled a funkčnost výsledného produktu. Každý herní engine má své specifické vlastnosti, které mu umožňují efektivně pracovat s různými typy her a různými herními mechanismy. Například některé enginy jsou optimalizovány pro vytváření 2D her s jednoduchou grafikou a ovládáním, zatímco jiné jsou zaměřeny na vývoj rozsáhlých 3D světů s pokročilými vizuálními efekty a fyzikální simulací. Je důležité si uvědomit, že herní enginy nejsou pouze nástrojem pro tvorbu her, ale také platformou pro tvorbu interaktivních projektů v různých oblastech, jako je vzdělávání, simulace, vizualizace a další. Proto mají herní enginy široké uplatnění nejen v zábavním průmyslu, ale i v akademickém a profesním prostředí. [24]

V současné době existuje mnoho různých herních enginů dostupných na trhu, které se liší svými funkcemi, cenovou politikou a podporou komunity. Mezi nejznámější herní enginy patří například Unity, Unreal Engine a Godot. Každý z těchto enginů má své specifické výhody a nevýhody, a proto je důležité pečlivě vybrat ten nejvhodnější pro konkrétní projekt a potřeby vývojářského týmu.[23]

4.1 Unity

Unity je jeden z nejpopulárnějších herních enginů. Své popularity dosáhl díky snadné použitelnosti a množství funkcí, které nabízí. Byl vyvinut společností Unity Technologies a poprvé představen v roce 2005 na konferenci Worldwide Developers Conference společnosti Apple. Původně byl prezentován jako herní engine pro platformu Mac OS. Od prvního představení prošel mnoha vývojovými fázemi, které přinesly podporu pro ostatní platformy.[24]

Popularita enginu spočívá nejen v jeho snadné použitelnosti, ale také v jeho flexibilitě. Unity je oblíbeným nástrojem pro tvorbu her pro začínající vývojáře, a je často používán při vývoji málo nákladových her. Díky svým schopnostem je vhodný pro tvorbu jak trojrozměrných (3D) her, tak i dvourozměrných (2D) her.[24]

Unity se používá jak pro menší mobilní hry, tak pro průmyslovou produkci na úrovni AAA, což svědčí o jeho všestrannosti. Nicméně, jeho použití není omezeno pouze na herní vývoj. Využívá se i v dalších odvětvích, jako je tvorba interaktivního obsahu pro filmy, automobilový průmysl nebo architekturu.[7][24]

4.1.1 Rozhraní Unity

Unity nabízí vizuální editor (Obrázek 2), který umožňuje rychlé vytváření scén. Jednoduché scény (bez kompletní herní mechaniky) lze dokonce vytvořit bez jediného řádku kódu. Jeho pracovní postup je založen na komponentech, což znamená, že je dostatečně modulární a flexibilní, aby bylo možné složité mechaniky sestavit z menších kousků logiky.[7]

Obrázek 2. Defaultní rozhraní Unity [7]

Základní herní entita se nazývá GameObject. Scéna je kontejnerem pro všechny GameObjecty. Prostorové vlastnosti GameObjectů, jejich grafické chování a herní mechaniky jsou definovány v různých komponentách. Kromě vestavěných komponent (pro vykreslování, animace, zvuk a podobných) může programátor definovat další logiku kódováním nových skriptových komponent (které všechny rozšiřují základní třídu MonoBehaviour).[7]

Sestavený prototyp lze testovat přímo ve vizuálním editoru a změny v kódu lze sledovat v reálném čase. Skriptování uvnitř Unity lze provádět v jazyce UnityScript (vlastní jazyk enginu se syntaxí podobnou JavaScriptu) nebo v jazyce C#, který je preferovanou volbou většiny uživatelů.[7]

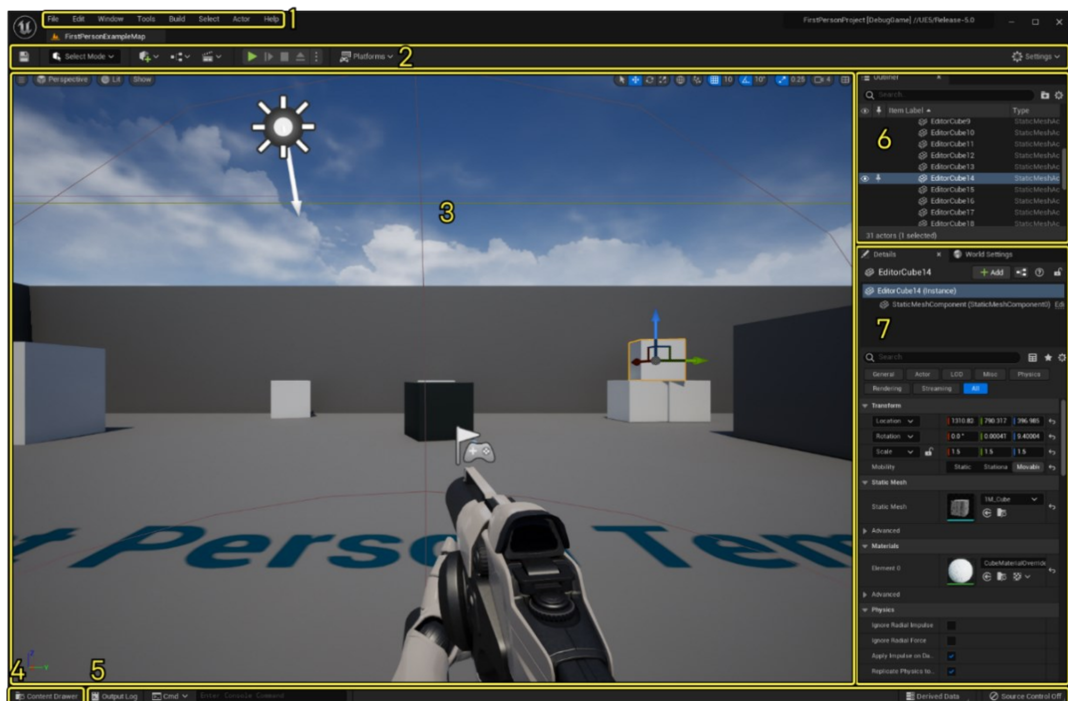
4.2 Unreal Engine

Unreal Engine je další velice populární herní engine. Byl vyvinut společností Epic Games a poprvé představen v roce 1998 ve hře Unreal. Původně byl vyvinut pro akční hry z pohledu první osoby na PC, od té doby byl však využitý v různých žánrech her a byl přijat i jinými průmyslovými odvětvími, zejména filmovým průmyslem. Nejnovější verze, Unreal Engine 5, byla spuštěna v dubnu 2022.[21]

Unreal Engine je napsán v C++ a podporuje vývoj pro různé platformy. Nástroj umožňuje vytváření fotorealistických scén. Unreal Engine je znám a používán především pro tvorbu videoher, ale jako univerzální 3D počítačový grafický engine je také používán pro prezentaci automobilů, v architektuře a pro tvorbu multimediálního obsahu. Ačkoli Unreal Editor může vycházet jako méně přívětivý pro nováčky, obecné názory komunity poukazují na to, že je schopen vytvářet působivější vizuální efekty ve srovnání s jinými enginy, jako je Unity.[21]

4.2.1 Rozhraní Unreal Enginu

Podobně jako Unity, Unreal Engine poskytuje vizuální editor (Obrázek 3), který umožňuje rychlé vytváření scén. Jeho pracovní postup je založen na komponentech, což umožňuje modulární a flexibilní tvorbu složitých herních mechanik z menších logických kousků.[21]



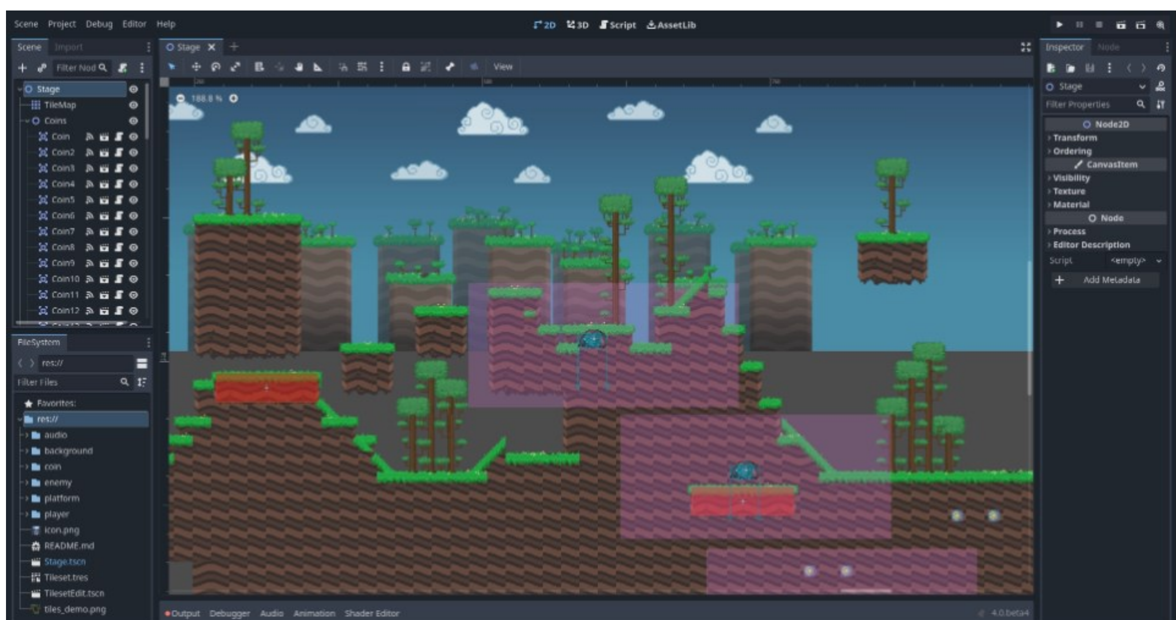
Obrázek 3. Defaultní rozhraní Unreal Editoru [22]

Základní herní entita v Unreal Engine se nazývá Actor. Scény jsou sestaveny pomocí Actorů, které jsou umístěny v prostoru hry. Každý Actor může mít několik komponent, které definují jeho chování a vlastnosti, včetně vizuálního zobrazení, fyzikálních vlastností a herní logiky. Unreal Engine poskytuje mnoho vestavěných komponent pro různé účely, a také umožňuje vývojářům vytvářet vlastní komponenty pomocí jazyka C++, který je preferovanou volbou pro většinu vývojářů díky své výkonnosti a flexibilitě. Kromě toho, Unreal Engine 5 nabízí programovací jazyk zvaný Blueprints, který umožňuje tvorbu herní logiky bez nutnosti psát kód v jazyce C++.[21]

4.3 Godot

Godot je open-source projekt pod licencí MIT, který je dostupný zdarma pro stažení, k použití pro osobní i komerční účely, úpravě a rozšíření zdrojového kódu. Godot je multiplatformní a umožňuje tvorbu her pro Windows, Linux, MacOS, mobilní telefony, webové prohlížeče a herní konzole, i když podpora pro konzole není velká. Podporuje vývoj 2D i 3D her, ale vyniká zejména ve 2D.[24]

Podporuje několik programovacích jazyků, včetně GDScriptu jazyk podobný Pythonu, C#, C++ a VisualScriptu. Godot je vhodný pro začátečníky i pokročilé a je používán pro výuku v rámci akademických kurzů.[24]



Obrázek 4. Defaultní rozhraní Godot enginu [26]

4.3.1 Rozhraní Godot enginu

Základním stavebním prvkem v Godot Engine (Obrázek 4) je herní entita nazývaná "Node". Scéna v Godotu slouží jako kontejner pro všechny uzly (Nodes). Prostorové vlastnosti uzlů, jejich chování a herní mechanismy jsou definovány v různých komponentách. Podobně jako v Unity, Godot Engine nabízí vestavěné komponenty pro vykreslování, animace, zvuk atd. Navíc umožňuje programátorům definovat vlastní logiku pomocí nových skriptovacích komponent, které mohou rozšiřovat základní třídu "Node" nebo "Control".[25]

Sestavené prototypy v Godot Engine lze testovat přímo ve vizuálním editoru a změny v kódu lze sledovat v reálném čase, což umožňuje efektivní a iterativní proces vývoje. Skriptování v Godot Engine lze provádět v jazyce GDScript, což je jazyk s podobnou syntaxí jako Python, nebo v jazyce C#. Pro většinu začínajících uživatelů je výhodnější volbou GDScript, který je integrován přímo do enginu a nabízí rychlý vývoj.[25]

II. PRAKTICKÁ ČÁST

5 ANALÝZA JEDNOTLIVÝCH HERNÍCH ENGINŮ

Prvním krokem při vývoji softwaru je nejdůležitější výběr vhodného vývojového prostředí. Pro tento účel byla provedena analýza a srovnání tří populárních herních enginů Unity, Unreal Engine a Godot. Každý z těchto enginů má své specifické vlastnosti, které mohou ovlivnit rozhodnutí při výběru. Aby bylo možné poskytnout celkový pohled na výhody a nevýhody jednotlivých enginů, byla vytvořena tabulka (Tabulka 1), která poskytuje srovnání klíčových vlastností. Při tvorbě tabulky bylo zvoleno několik klíčových vlastností, které jsou obecně považovány za důležité při výběru herního enginu. Kritéria byla vybrána s ohledem na jejich vliv při vývoji aplikace.

První vlastností v analýze byl programovací jazyk, který je klíčovou vlastností, neboť ovlivňuje způsob, jakým vývojáři píšou kód a implementují funkcionality do her. V tomto ohledu byly zahrnuty informace o jazycích, které jsou v jednotlivých prostředí podporovány.

Další vlastností byly podporované platformy, protože dostupnost na různých zařízeních může mít klíčový vliv na dostupnost vytvořeného softwaru. Byly zahrnuty platformy jako Windows, macOS, Linux, Android, iOS, Xbox, PlayStation, Nintendo a WebGL, aby byl poskytnut ucelený přehled možností.

Licenční modely byly další důležitou součástí při výběru herního enginu, neboť ovlivňují dostupnost funkcí a náklady spojené s vývojem her.

Velikost komunity je také velkým faktorem při výběru, protože síla a aktivita komunity může hrát klíčovou roli v podpoře vývoje her a řešení a nalezení problémů. Zahrnuty byly informace o velikosti a dynamice komunit jednotlivých prostředí.

V závěru porovnávací tabulky je uvedena vlastnost pro assety. Tato vlastnost popistuje možnost získání assetů do vývojového prostředí. Tato vlastnost je klíčová pro vývoj her, protože kvalitní assety mohou výrazně urychlit a zlepšit proces vývoje.

Tabulka 1. Srovnávací tabulka herních enginů

Vlastnosti	Unity	Unreal Engine	Godot
Programovací jazyk	UnityScript, C#	Blueprints, C++	GDScript, C#
Podporované platformy	Windows, macOS, Linux, Android, iOS, Xbox, PlayStation, Nintendo, WebGL	Windows, macOS, Linux, Android, iOS, Xbox, PlayStation, Nintendo	Windows, macOS, Linux, iOS, Android
Licence	Student, Personal, Pro, Enterprise, Industry	Standart, Custom	Open-source
Komunita	Velká	Velká	Rozrůstající
Assety	Asset Store	Marketplace	Asset Library

5.1 Volba vhodné technologie pro vývoj výukového softwaru

Po analýze a srovnání tří populárních herních enginů Unity, Unreal Engine a Godot se dospělo k závěru, že pro realizaci hry je nejvhodnější volbou herní engine Unity. Tento výběr byl proveden na základě klíčových faktorů, které byly zohledněny při posuzování jednotlivých enginů.

Jedním z hlavních důvodů, proč bylo rozhodnuto použít Unity, je jeho široká podpora a flexibilita v programování. Unity nabízí podporu pro jazyk C#, který je široce používaný a lehký na použití. Tato kombinace umožnila efektivně implementovat herní mechaniky pro propojování bloků a ostatních herních mechanik hry.

Dalším faktorem pro výběr vhodného herního enginu, byla velikost komunity. Komunita pro Unity poskytuje rozsáhlé zdroje dokumentace a tutoriálů, které umožnily rychlé získání potřebných dovedností pro tvorbu hry.

Poslední vlastností pro výběr Unity byl rozsáhlý Unity asset store, který nabízí velké množství assetů, které lze využít při vývoji. Asset store nabízí velké množství assetů, které jsou zdarma a zapadají do konceptů tvořené hry.

6 KONCEPT HRY

Hra byla zasazená do prostředí vesmírné lodi, kde je hráč probuzen ze spánku. Vesmírná loď vyžaduje, aby hráč vytvořil jednoduché algoritmy, které se budou starat o správný chod lodi. Hráč postupně prozkoumává různé části lodi a interaguje s prostředím. Využitím vizuálního programování pro tvorbu algoritmů plní jednotlivé úkoly a tím postupuje hráč skrze různé místnosti. Každá místnost obsahuje specifický úkol s jiným zadáním, který je potřeba splnit pro odemčení nové místnosti. Zároveň samotná místnost obsahuje objekty, s kterými hráč za pomoci tvorby algoritmu může interagovat a vidí tak aktuální změny podle vytvořeného programu. Po celou dobu hry je hráči ukazován popisek aktuálního úkolu pro lepší orientaci ve hře.

6.1 Ovládání

Ovládání hry je velice jednoduché, hráč se může pohybovat za pomoci kláves „WASD“ a šipek na klávesnici. S postavou lze také přejít do běhu klávesou „shift“. Samotné rozhraní pro vizuální programování algoritmu se otevře přistoupením k objektu počítače ve hře. V rozhraní pak hráč využívá myš pro přesun a propojení jednotlivých bloků a klávesnici pro zadávání potřebného textu. Pokud chce hráč hru opustit a uložit, může tak učinit za pomoci klávesy „escape“, která zobrazí menu během hry.

6.2 Úkoly a obtížnost

Samotné úkoly byly vytvořeny tak, aby vždy pracovaly s prvek v herním prostředí pro zlepšení interakce hry s hráčem. Tyto úkoly byly také vytvořeny tak, aby se neopakovaly a postupně se stěžovala jejich obtížnost.

Hra byla navržena tak, aby na začátku seznámila hráče s hlavní mechanikou hry pro tvorbu algoritmů. Pro lepší pochopení aktuálního úkolu byla do rozhraní také přidána možnost nápovědy, která popisuje aktuální úkol, proměnné a použitelné akce v dané místnosti. Do této nápovědy byl také vložen popis jednotlivých bloků, jejich vstupy a výstupy pro rychlejší pochopení mechaniky.

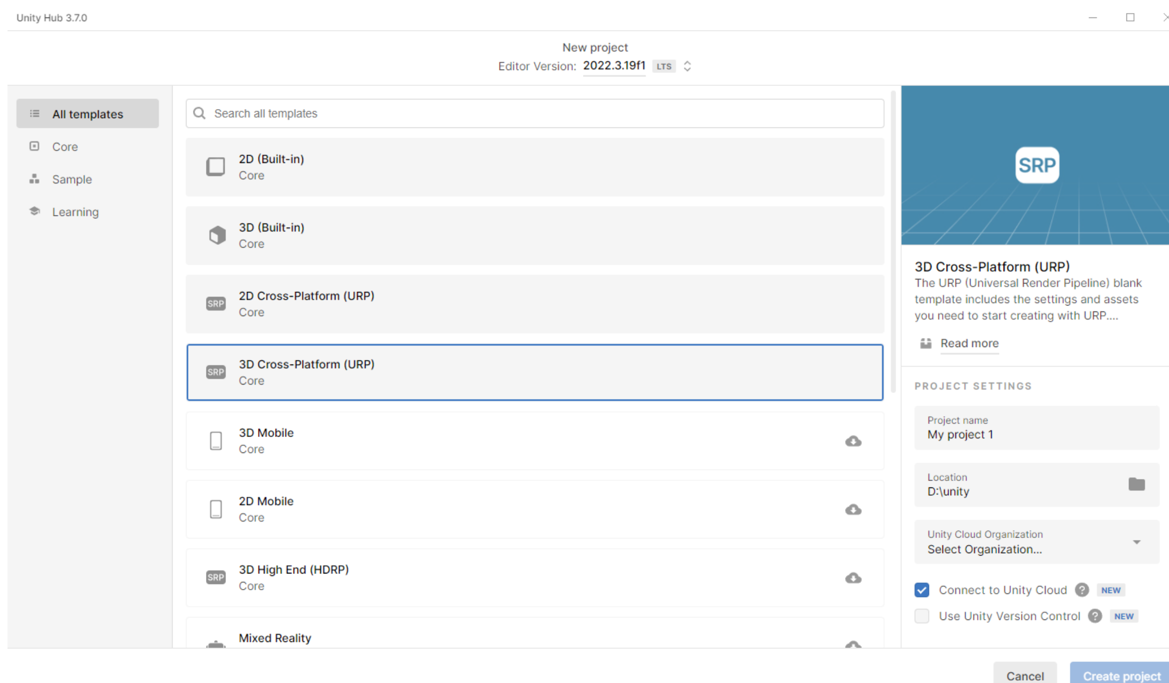
6.3 Konec hry

Cílem hry je splnění všech úkolu v každé místnosti na lodi, po úspěšném splnění všech úkolů je hráč poslán zpět na start, kde se nachází i konec hry.

7 TVORBA HRY

7.1 Vytvoření Projektu a Příprava scény

Prvním krokem bylo založení nového projektu v herním enginu Unity (Obrázek 5). Projekt je vytvořen ve verzi 2022.3.19f1 s využitím 3D Cross-Platform (URP) templatu.



Obrázek 5. Vytvoření Projektu

Po vytvoření projektu se načetl samotný editor se základní scénou „SampleScene“ s kamerou, globálním světlem a „Global Volume“, který slouží k postprocesu.

7.2 Assety

Pro vytvoření scény a jednotlivých levelů bylo potřeba využít assety, které by vyhovovaly zasazení do prostředí vesmírné lodi. Pro vyhledání volně dostupných assetů byl použit Unity Asset Store, který nabízí širokou škálu assetů za peníze, ale i zdarma.

K sestavení vesmírné lodi byl použit volně dostupný balíček modulárních prefabů[27]. Vzhledem k tomu že vesmírná loď, se nachází ve vesmíru, bylo potřeba najít vhodný skybox[28], který představuje prostor mimo stanici. Tyto assety byly přidány do editoru za pomoci *Package Manageru*.

7.3 Herní mechaniky

Aby hra fungovala a měla nějaké mechaniky, bylo potřeba implementovat jednotlivé herní mechaniky a vytvořit jejich skripty.

7.3.1 Pohyb hráče a kamery

Pro pohyb hráče byly vytvořeny skripty PlayerController.cs a PlayerInputs.cs. V těchto skriptech bylo vytvořeno ovládání hráče, které zohledňuje pohyb, skok, rotaci kamery a správu fyzikálních efektů, prostřednictvím *CharacterController* a vstupů hráče zpracovávaných prostřednictvím *PlayerInputs*.

Třída *PlayerController* slouží k ovládání hráče v Unity prostředí. Hráč se může pohybovat, skákat, rotovat kamerou a reagovat na uživatelské vstupy. Na začátku kódu jsou definovány různé parametry pro ovládání hráče, jako je rychlost pohybu, rychlost sprintu, rychlost rotace, rychlost změny rychlosti a velikost gravitace. Následně jsou inicializovány proměnné pro uchování rychlosti, rotace, vertikální rychlosti a dalších hodnot souvisejících s fyzikou hráče.

Metoda *Awake()* se používá k získání hlavní kamery ve scéně. Ve *Start()* metodě jsou inicializovány odkazy na *CharacterController* a *PlayerInputs*, které slouží k získávání vstupů od hráče. Hlavní části kódu jsou metody *Update()* a *LateUpdate()*, které řídí pohyb a rotaci hráče a kamery. V *Update()* metodě jsou volány další metody pro kontrolu zda je hráč na zemi a samotný pohyb. V *LateUpdate()* metodě je aktualizována rotace kamery na základě vstupu od hráče. Metoda *GroundedCheck()* provádí kontrolu, zda je hráč na zemi pomocí kolize s herním prostorem. *RotateCamera()* metoda slouží k rotaci kamery na základě vstupů hráče. *Move()* metoda se stará o pohyb hráče na základě vstupů. Určuje cílovou rychlost pohybu pro chůzi a běh, poté interpoluje mezi aktuální a cílovou rychlostí a aplikuje pohyb hráče. Metoda také aplikuje gravitaci na hráče. K dispozici jsou také pomocné metody jako *ClampAngle(),* která omezuje rozsah úhlu rotace kamery.

Třída *PlayerInputs* zajišťuje zpracování vstupů od hráče v Unity pomocí *Input System.* V této třídě jsou definovány proměnné pro uchování vstupů hráče. Dále jsou zde nastaveny vlastnosti kurzoru, které určují, zda je kurzor zamknutý a zda se vstupy kurzoru používají pro ovládání pohybu kamery.

Metody *OnMove()*, *OnLook()*a *OnSprint()* jsou označeny jako odběratelé událostí vstupu z *Input Systemu*. Tyto metody jsou volány při změně stavu příslušných vstupních událostí.

Metody *MoveInput()*, *LookInput()* a *SprintInput()* slouží k aktualizaci stavů vstupů na základě příchozích dat. *OnApplicationFocus()* je událost volaná při změně stavu aplikace. Tato událost zajistí nastavení stavu kurzoru na základě toho, zda je aplikace aktivní.

7.3.2 Skripty pro jednotlivé bloky

Pro samotnou mechaniku vizuálního programování, bylo potřeba vytvořit skripty pro každý blok tak, aby zajišťovaly funkčnost této mechaniky. Veškeré skripty byly napsány v programovacím jazyce C#.

Blok

Základním prvekem vizuálního programování jsou jednotlivé bloky obsahující funkci, kterou daný blok vykonává. Proto byl vytvořen skript Blok.cs, který obsahuje třídu *Variable* a třídu *Block*.

První třída reprezentuje proměnnou s názvem a hodnotou (Obrázek 6). Obsahuje vlastnosti (name a value), které lze číst a měnit. Tato třída slouží k vytvoření proměnných, které se deklarují za pomoci dalších bloků.

```
Počet odkazů: 9
public class Variable
{
    Počet odkazů: 7
    public string name { get; set; }
    Počet odkazů: 5
    public object value { get; set; }
}
```

Obrázek 6. Třída Variable

Samotná třída *Blok* poskytuje základní funkcionalitu pro interakci s bloky a jejich přidruženými proměnnými. Obsahuje vlastnosti *blockType*, *cantBeNext* a *nextBlock* pro určení typu bloku, možnosti připojení dalšího bloku a odkaz na následující blok v sekvenci.

Třída udržuje seznam deklarovaných proměnných pomocí statického pole objektů typu *Variable*. Metoda *Awake()* inicializuje vlastnost *blockType* na hodnotu *Default* při spuštění instance (Obrázek 7). Metoda *Execute()* simuluje provádění logiky bloku. Následně spouští provádění následujícího bloku. Třída také obsahuje metodu *WriteToDebugField*(), což je metoda, která slouží k vypsání aktuálního stavu do stavového řádku v rozhraní tvorby algoritmu.

Další metody v této třídě slouží k manipulaci s proměnnými, jako je hledání, zjišťování existence, aktualizace a získávání hodnoty proměnných. Třída *Block* funguje jako rodičovská třída pro ostatní typy bloků.

```
//Inicializace proměnných při načtení skriptu
Zpráva Unity | Počet odkazů: 4
protected virtual void Awake()
{
    //Přiřazení ID bloku
    blockID = blockId;
    blockId++;

    blockRectTransform = GetComponent<RectTransform>();

    blockType = BlockType.Default;

    //Najde body pro propojení
    blockConnector = FindChildWithTag(gameObject, "BlockConnector");
    nextBlockConnector = FindChildWithTag(gameObject, "NextBlockConnector");

    //Najde grayedImg a vypne ho
    grayedImg = FindChildWithTag(gameObject, "GrayedImg");
    grayedImg.SetActive(false);

    //Najde debug řádek v rozhraní
    debugField = transform.parent.parent.Find("DebugLog")?.GetComponentInChildren<TMP_InputField>();

}

//Spouští následující blok s delayem
Počet odkazů: 13
public virtual void Execute()
{
    StartCoroutine(ExecuteWithDelay(executionDelay));
}
```

Obrázek 7. Metody Awake() a Execute() ve třídě Block

StartBlock

První třídou, která je odvozená ze třídy *Block* je *StartBlock*. Tato třída byla vytvořena tak, aby sloužila jako první blok, který bude spouštět samotný kód z rozhraní vizuálního programování. Při spuštění bloku se kontroluje, zda existuje následující blok (*nextBlock*). Pokud ano, metoda *Execute()* je volána i na tomto následujícím bloku, čímž dochází k sekvenčnímu provádění bloků.

DeclarationBlock

Dalším blokem je *DeclarationBlock*, který byl navržen tak, aby umožňoval deklaraci a úpravu proměnné vázané k bloku. Každý *DeclarationBlock* může deklarovat pouze jednu proměnnou. Blok pracuje se vstupními poli pro název a hodnotu.

Při spuštění bloku (Obrázek 8) se tato pole kontrolují, zda nejsou prázdná. Pokud nejsou prázdná, název proměnné je přiřazen a hodnota je kontrolována vstupním datovým typem (boolean nebo float). Jestliže je předaná hodnota ve správném datovém typu, proměnná je deklarována a uložená do pole proměnných pomocí funkce D*eclareVariable()* s daným názvem a hodnotou. Pokud existuje proměnná se stejným názvem, je nejprve odstraněna pomocí funkce *RemoveVariable(),* aby mohla být vytvořena nová proměnná se stejným názvem a jinou hodnotou.

```
//Provede deklaraci a inicializaci proměnné
Počet odkazů: 8
public override void Execute()
{
    if (initialValue.text != "" && variableName.text != "")
    {
        string varName = variableName.text;

        //Odstraní proměnnou, aby mohla být znovu vytvořena
        RemoveVariable(varName);

        //Vytvoření za pomoci jiné proměnné
        if (IsVariable(initialValue.text))
        {
            object value;
            value = GetValueByName(initialValue.text);
            DeclareVariable(varName, value);
        }
        //Vytvoření bool proměnné
        else if (bool.TryParse(initialValue.text, out bool boolvalue))
        {
            DeclareVariable(varName, boolvalue);
        }
        //Vytvoření float proměnné
        else if (float.TryParse(initialValue.text, out float floatvalue))
        {
            DeclareVariable(varName, floatvalue);
        }
        //Pokud byl zadán input, který nešel uložit do proměnné, vypíše se error
        else
        {
            WriteToDebugField("Wrong input try to be declared", Color.red);
            return;
        }

    }
    //Provede následující blok
    base.Execute();
}
```

Obrázek 8. Metoda Execute() ve třídě DeclarationBlock

IfBlock

Třída *IfBlock* je další odvozenou třídou ze třídy *Block* a je použita k reprezentaci podmínky. Tento blok umožňuje porovnání dvou hodnot na základě zvoleného operátoru a následného provedení příslušných bloků v závislosti na výsledku podmínky. Ve třídě jsou definovány následující prvky: *falseBlock* (instance třídy Block, která bude provedena v případě, že podmínka není splněna), *variable1Input* (pole pro zadání první hodnoty podmínky), *variable2Input* (pole pro zadání druhé hodnoty podmínky) a *operatorDropdown* (seznam pro výběr porovnávacího operátoru). Při inicializaci instance třídy je nastaven typ bloku na *IfBlock*. Metoda *Execute()* je implementována pro vyhodnocení podmínky na základě zadaných vstupů a následného provedení příslušného bloku.

Během vykonávání bloku *IfBlock* se provádí následující kroky: Získání hodnoty a typu zadané první a druhé hodnoty podmínky z textových polí variable1Input a variable2Input, vyhodnocení typu zadaných hodnot a jejich převedení na odpovídající datový typ (*float* nebo *boolean*), získání vybraného operátoru porovnání ze seznamu *operatorDropdown*, porovnání hodnot na základě zvoleného operátoru a vyhodnocení podmínky (Obrázek 9). V případě splnění podmínky se provede příslušný *nextBlock*, jinak se provede definovaný *falseBlock*, oba bloky jsou spouštěny se zpožděním.

Provedení bloků s časovým zpožděním je implementováno pomocí asynchronních metod *ExecuteNextBlockWithDelay()* a *ExecuteFalseBlockWithDelay()*, které využívají koprogram pro vyčkání určeného času před provedením následujícího bloku. Metody *CheckCondition()* a *CheckBoolCondition()* slouží k vyhodnocení podmínky na základě zvoleného operátoru a typu hodnoty.

```
//Metoda kontrolující podmínku pro float
Počet odkazů: 1
private bool CheckCondition(float value1, float value2, string @operator)
{
    switch (@operator)
    {
        case "==":
            return value1 == value2;
        case "!=":
            return value1 != value2;
        case "<":
            return value1 < value2;
        case ">":
            return value1 > value2;
        case "<=":
            return value1 <= value2;
        case ">=":
            return value1 >= value2;
        default:
            Debug.LogError("Invalid operand.");
            return false;
    }
}
```

Obrázek 9. Metoda CheckCondition() ve třídě IfBlock

LoopBlock

Třída *LoopBlock* byla vytvořena k reprezentaci bloku smyčky v herním prostředí Unity. Tento blok umožňuje opakování následujícího bloku podle zadaného počtu. Ve třídě byly definovány prvky *loopedBlock* (instance třídy *Block*, která má být opakovaně vykonávána), *loopCount* (vstupní pole pro zadání počtu opakování smyčky). Při inicializaci třídy byl v metodě *Awake()* nastaven typ bloku na *LoopBlock*.

Metoda *Execute()* byla implementována pro vykonání smyčky (Obrázek 10). Nejprve se načte počet opakování ze vstupního pole *loopCount*. Poté se zvýší počet iterací o jedna. Pokud je aktuální iterace větší než počet opakování (*loopNumber*), provede se následující blok se zpožděním a iterace se resetuje na nulu. Jinak se provede blok, který má být opakován. Metody *ExecuteNextBlockWithDelay()* a *ExecuteLoopedBlockWithDelay()* slouží ke spouštění následujícího bloku s časovým zpožděním pomocí asynchronního vykonávání koprogramu.

```
//Provádí smyčku podle daného počtu
Počet odkazů: 8
public override void Execute()
{
    float value;

    //Kontrola zda jsou inputy hodnota(float) nebo proměnná
    if (!float.TryParse(loopCount.text, out value))
    {
        if (IsVariable(loopCount.text))
        {
            value = (float)FindValue(loopCount.text);
        }
        else
        {
            Debug.Log("Invalid input. Please enter a valid number or variable name.");
            WriteToDebugField("Invalid input. Please enter a valid number or variable name.", Color.red);
            return;
        }
    }

    //Zaokrouhlení hodnoty
    loopNumber = (int)Math.Round(value);

    iteration++;

    //Pokud je iterace větší než požadovaný počet je zavolán následující blok, jinak se volá blok ve smyčce
    if (iteration > loopNumber)
    {
        WriteToDebugField($"Loop ended");
        ExecuteNextBlockWithDelay();
        iteration = 0;
    }
    else
    {
        WriteToDebugField($"Iteration of loop: {iteration}");
        ExecuteLoopedBlockWithDelay();
    }

}
```

Obrázek 10. Metoda Execute() ve třídě LoopBlock

MathBlock

Další odvozenou třídou od třídy *Block* je Třída *MathBlock*. Tato třída byla použita k provádění matematických operací na základě vstupních hodnot a zvoleného operátoru. Třída obsahuje *variable1Input* (textové pole pro zadání první hodnoty), *variable2Input* (textové pole pro zadání druhé hodnoty) a *operatorDropdown* (seznam pro výběr matematického operátoru).

Metoda *Execute()* získává a převádí textové vstupy na čísla typu float. Pokud není možné převést vstup na číslo, je ověřeno, zda se jedná o proměnnou, v případě proměnné se načte její hodnota. Pokud se nejedná o číslo ani o platnou proměnnou, metoda se ukončí. Následně se získává vybraný operátor z *operatorDropdown*. Provedená operace (sčítání, odčítání, násobení nebo dělení) se vykoná na základě zvoleného operátoru a vstupních hodnot *value1* a *value2* (Obrázek 11).

Výsledek operace je uložen do proměnné *result*. Pokud je vstupní hodnota z *variable1Input* nebo *variable2Input* platnou proměnnou, je do ní uložen výsledek operace pomocí metody *SaveValue()*.

```
//Provede vybranou matematickou operaci na vstupech
Počet odkazů: 1
private float PerformOperation(float variable1, float variable2, string @operator)
{
    //Operace je vybrána na základě operatoru
    switch (@operator)
    {
        case "+":
            return variable1 + variable2;
        case "-":
            return variable1 - variable2;
        case "*":
            return variable1 * variable2;
        case "/":
            if (variable2 != 0)
                return variable1 / variable2;
            else
            {
                Debug.LogWarning("Division by zero.");
                return float.NaN;
            }
        default:
            Debug.LogWarning($"Unknown operand: {@operator}");
            return float.NaN;
    }
}
```

Obrázek 11. Metoda PerformOperation() ve třídě MathBlock

ActionBlock

Posledním blokem odvozeným ze třídy *Block* je blok *ActionBlock*. Tento blok umožňuje provádění různých akcí na základě výběru ze seznamu. V této třídě byly definovány prvky *actionDropdown* pro výběr akce ze seznamu a *actionEvents*, která slouží jako seznam událostí typu *ActionEvent*, které obsahují název akce a odpovídající událost typu *UnityEvent* (Obrázek 12).

Metoda *Start()* se volá při startu instance třídy. Kontroluje, zda *actionDropdown* není prázdný a poté vytváří možnosti v rozbalovacím seznamu na základě obsahu *actionEvents*. Pro každou událost v *actionEvents* se přidá název akce do seznamu možností.

Metoda Execute() se volá pro provedení bloku. Nejprve zkontroluje, zda actionDropdown není null, a pak získá vybranou akci z rozbalovacího seznamu. Následně spustí událost pro tuto akci pomocí metody *TriggerEvent*(*selectedAction*).

Metoda *TriggerEvent*(string *actionName*) vyhledá událost v *actionEvents* podle zadaného názvu akce. Pokud je nalezena odpovídající událost, spustí tuto událost pomocí metody *Invoke()*. V opačném případě vypíše varování, že událost pro zadanou akci nebyla nalezena.

```
//Struktura pro akci s názvem a událostí
[System.Serializable]
Počet odkazů: 3
public class ActionEvent
{
    public string actionName;
    public UnityEvent actionEvent;
}
```

Obrázek 12. Vytvoření struktury pro událost

7.3.3 Skripty pro propojení bloků

Pro propojení jednotlivých bloků byly vytvořeny skripty BlockConnector.cs, BlockConnection.cs pro vizualizaci propojení, a BlockDragDrop.cs pro přetahování a volání funkce pro propojení při dvojkliku.

BlockConnector

Třída *BlockConnector* slouží k propojení bloků v editoru a správě jejich spojení. V této třídě byly definovány proměnné pro uchování informací o prvním a druhém kliknutém bloku, stejně jako pro specifické typy bloků (*ifBlock* a *loopBlock*). Metoda *SetBlockClicked()* je volána při kliknutí na blok a slouží k nastavení prvního a druhého bloku v závislosti na kliknutí a zajištění správného spojení mezi bloky.

Metoda *ConnectBlocks()* slouží k vytvoření vizuálního spojení mezi bloky pomocí objektu *blockConnectionPrefab* a instance třídy *BlockConnection* (Obrázek 13). Tato metoda také zajišťuje správné nastavení barvy spojení v závislosti na typu bloků. Metoda *RemoveConnection()* slouží k odstranění spojení pro určitý blok a zajišťuje, že příslušné spojení je odstraněno z editoru. Metody *UpdateConnectionLines()* a *RemoveAllConnections()* slouží k aktualizaci a odstranění všech spojení v případě, že je to nutné.

```
//Metoda pro propojení bloků
Počet odkazů: 3
private void ConnectBlocks()
{
    if (firstBlockClicked != null && secondBlockClicked != null)
    {
        if (blockConnectionPrefab != null)
        {
            //Pokud existuje propojení je odstraněno
            RemoveConnection(firstBlockClicked);

            //Vytvoří nové propojení mezi bloky
            GameObject blockConnectionObject = Instantiate(blockConnectionPrefab, transform.parent);
            BlockConnection blockConnection = blockConnectionObject.GetComponent<BlockConnection>();

            if (blockConnection != null)...
        }

        //Pokud jde o IfBlok vytvoří se druhé propojení pro falseBlock
        if (firstBlockClicked.blockType == Block.BlockType.IfBlock)...

        //Pokud jde o loopBlok vytvoří se druhé propojení pro loopedBlock
        if (firstBlockClicked.blockType == Block.BlockType.LoopBlock)...
        firstBlockClicked = null;
        secondBlockClicked = null;
        ifBlock = null;
    }
}
```

Obrázek 13. Metoda ConnectBlocks() ve třídě BlockConnector

BlockConnection

Třída *BlockConnection* (Obrázek 14) slouží k vizuálnímu zobrazení spoje mezi bloky. V této třídě byly definovány proměnné pro odkaz na *LineRenderer*, počáteční a konečný blok. Dále zde byly definovány barvy pro různé typy spojení. Zelená barva pro základní propojení bloků, červená barva pro blok, který se má spustit, když by podmínka v bloku nebyla splněna, a modrá barva pro propojení bloků uvnitř smyčky. Metoda *DrawConnection()* se používá k vykreslení spojení mezi *startBlock* a *endBlock* pomocí *LineRendereru*. Pokud by byly bloky správně nastaveny, metoda nastaví počáteční a koncové pozice čáry na pozice příslušných konektorů bloků.

```
//Třída pro vykreslení propojení
Skript Unity (počet odkazů na prostředky: 1) | Počet odkazů: 9
public class BlockConnection : MonoBehaviour
{
    public LineRenderer lineRenderer;

    //Bloky pro propojení
    public Block startBlock;
    public Block endBlock;

    public bool isTrueConnection;

    //Barvy pro různé typy propojení
    public Color trueConnectionColor = Color.green;
    public Color falseConnectionColor = Color.red;
    public Color loopConnectionColor = Color.blue;

    //Metoda pro vykreslení čáry mezi bloky
    Počet odkazů: 4
    public void DrawConnection()
    {
        if (startBlock != null && endBlock != null)
        {
            if (lineRenderer.positionCount != 2)
            {
                lineRenderer.positionCount = 2;
            }

            //Nastaví začátek a konec čáry
            lineRenderer.SetPosition(0, startBlock.nextBlockConnector.transform.position);
            lineRenderer.SetPosition(1, endBlock.blockConnector.transform.position);
        }
        else
        {
            //Vymaže čáru
            lineRenderer.positionCount = 0;
        }
    }
}
```

Obrázek 14. Třída BlockConnection

BlockDragDrop

Třída *BlockDragDrop* implementuje různé události pro funkčnost přetahování a ovládání bloků. V této třídě jsou definovány proměnné pro práci s grafickým rozhraním, plátnem, odkazem na rodičovské *RectTransform* a *BlockConnector* pro správu propojení bloků.

Metoda *Awake()* se používá k inicializaci odkazů na komponenty a získání rodičovského *RectTransform* bloku. Zde také dochází k přidání komponenty *CanvasGroup*, pokud není již připojena k objektu. Metoda *OnPointerClick()* reaguje na kliknutí na blok. Pokud by byl proveden dvojklik, zavolala by se metoda *SetBlockClicked()* na *BlockConnector*, aby byl blok označen jako aktivní pro spojení. Metody *OnBeginDrag(), OnDrag()* a *OnEndDrag()* implementují události pro začátek, průběh a konec mechaniky pro přetahování bloku (Obrázek 15). Tyto metody umožňují plynulé přesouvání bloku v rámci definovaném prostoru v rodičovském *RectTransform*. Metoda *RemoveConnection()* slouží k odstranění všech spojení pro daný blok.

```
//Výpočet pozice během tažení
Počet odkazů: 2
public void OnDrag(PointerEventData eventData)
{
    //Výpočet nové pozice bloku na základě aktuální pozice kurzoru a offsetu
    Vector2 newPos = eventData.position + offset;

    //Převod pozice kurzoru na lokální pozici
    RectTransformUtility.ScreenPointToLocalPointInRectangle(parentRectTransform, eventData.position, eventData.pressEventCamera, out newPos);

    //Výpočet mezí pro novou pozici
    float minX = parentRectTransform.rect.xMin + blockRectTransform.rect.width / 2; // Adjusted to account for block size
    float maxX = parentRectTransform.rect.xMax - blockRectTransform.rect.width / 2; // Adjusted to account for block size
    float minY = parentRectTransform.rect.yMin + blockRectTransform.rect.height / 2; // Adjusted to account for block size
    float maxY = parentRectTransform.rect.yMax - blockRectTransform.rect.height / 2; // Adjusted to account for block size

    //Omezení pozice do romezí
    newPos.x = Mathf.Clamp(newPos.x, minX, maxX);
    newPos.y = Mathf.Clamp(newPos.y, minY, maxY);

    //Nastavení nového kotvícího bodu
    blockRectTransform.anchoredPosition = newPos;

    //Aktualizace propojení
    if (blockConnector != null)
    {
        blockConnector.UpdateConnectionLines();
    }
}

//Po ukončení tažení povolí raycast
Počet odkazů: 2
public void OnEndDrag(PointerEventData eventData)
{
    canvasGroup.blocksRaycasts = true;
}
```

Obrázek 15. Metody OnDrag() a OnEndDrag() ve třídě BlockDragDrop

7.3.4 Systém ukládání a načítání hry

Pro hru byl také vytvořen systém, který se stará o ukládání a načítání dosaženého postupu ve hře. Pro systém byla vytvořena série skriptů pro správu herních dat. Skripty zahrnují třídy pro ukládání a načítání herních dat a datovou strukturu dat, které byly potřeba uložit pro zachování postupu ve hře.

Prvním vytvořeným skriptem byl SaveData.cs. který obsahuje samotnou datová strukturu. Struktura se skládá z několika tříd, které se deklarují v hlavní třídě *GameData.* Tato třída obsahuje informace o herním stavu, jako je pozice hráče, seznam bloků a dveří a aktivní úkol (Obrázek 16). Třída má konstruktor, který inicializuje seznamy bloků a dveří. Následují třídy pro ukládání dat o blocích: *BlockData*, *StartBlockData*, *DeclarationBlockData*, *IfBlockData*, *LoopBlockData*, *MathBlockData* a *ActionBlockData*. Každá třída obsahuje informace specifické pro daný typ bloku, jako jsou pozice, následující blok, vstupy, proměnné a akce.

Třída *DoorData* uchovává informace o dveřích, jako je jejich název a aktuální stav. Pro uložení informací o aktuálním úkolu, jako je název a popis, byla vytvořena třída *ActiveQuest*. Skript také obsahuje třídy pro serializaci a deserializaci vektorů *SerializableVector2* a *SerializableVector3*.

Dalším skriptem je SaveLoadManager.cs. Tento kód obsahuje třídu nazvanou *SaveLoadManager*, která se stará o ukládání a načítání herních dat. Třída má několik metod a proměnných pro manipulaci s herními objekty a daty.

Nejdříve jsou deklarovány seznamy *blocksInUI* a *doorsInScene*, které uchovávají bloky a dveře v herním prostředí. Metoda *Awake*() je volána při startu hry a inicializuje seznamy *doorsInScene* a *blocksInUI*. Nejprve jsou všechny komponenty *DoorController* v herním prostředí získány a uloženy do seznamu *doorsInScene*. Poté jsou všechny herní objekty s tagem "Block" nalezeny a jejich komponenty *Block* jsou přidány do seznamu *blocksInUI*. Metoda *SaveGame*() slouží k ukládání herního stavu. Nejprve se ukládá pozice hráče a pak se prochází všechny bloky v seznamu *blocksInUI*. Každý blok je převeden na data. Podobně jsou ukládány i dveře. Nakonec se ukládá aktivní úkol.

Metoda *LoadGame*() načítá uložený herní stav. Nejdříve se obnoví pozice hráče a poté se prochází seznam *doorDataList*, kde se každý záznam přiřadí odpovídajícímu objektu dveří. Stejným způsobem se obnovují bloky a aktivní úkol. Soukromá metoda *FindBlockByBlockID*() slouží k vyhledání bloku podle jeho identifikátoru.

Pro samotné ukládání a načítání ze souboru slouží skripty SaveSystem.cs a LoadSystem.cs. Skript *SaveSystem* obsahuje statickou metodu *SaveGame()*, která přijímá objekt typu *GameData* a ukládá ho do souboru ve formátu JSON. Nejprve se herní data převedou na JSON pomocí *JsonUtility.ToJson*(), poté se určí cesta k souboru ve složce *persistentDataPath* aplikace a JSON data se zapíší do souboru.

Třída *LoadSystem* obsahuje statickou metodu *LoadGame()*, která načítá herní data ze souboru. Nejprve se určí cesta k souboru s uloženými daty. Pokud soubor existuje, jeho obsah je načten a převeden zpět na objekt typu *GameData* pomocí *JsonUtility.FromJson*().

```
//Struktura obsahujicí data pro uložení hry
[System.Serializable]
Počet odkazů: 8
public class GameData
{
    public SerializableVector3 playerPosition;
    public List<BlockData> blockDataList;
    public List<DoorData> doorDataList;
    public ActiveQuest activeQuest;

    Počet odkazů: 1
    public GameData()
    {
        blockDataList = new List<BlockData>();
        doorDataList = new List<DoorData>();
    }

}
```

Obrázek 16. Struktura ukládaných data

7.3.5 Otevírání a zavírání dveří

Dveře ve hře slouží jako překážka, kterou je potřeba odstranit otevřením dveří. Tyto dveře může hráč otevřít za pomoci sestaveného algoritmu, nebo splněním daného úkolu v místnosti. K otevření dveří bylo potřeba vytvořit skript, který by kontroloval a měnil jejich stav. Za tímto účelem byl vytvořen skript DoorController.cs, který ovlivňuje chování dveří v herním světě. Umožňuje otevírání a zavírání dveří a ukládání a načítání stavu dveří. Ve třídě jsou deklarovány proměnné pro animátor, hodnotu potřebnou k otevření, zvuk dveří a další pomocné proměnné pro sledování stavu dveří.

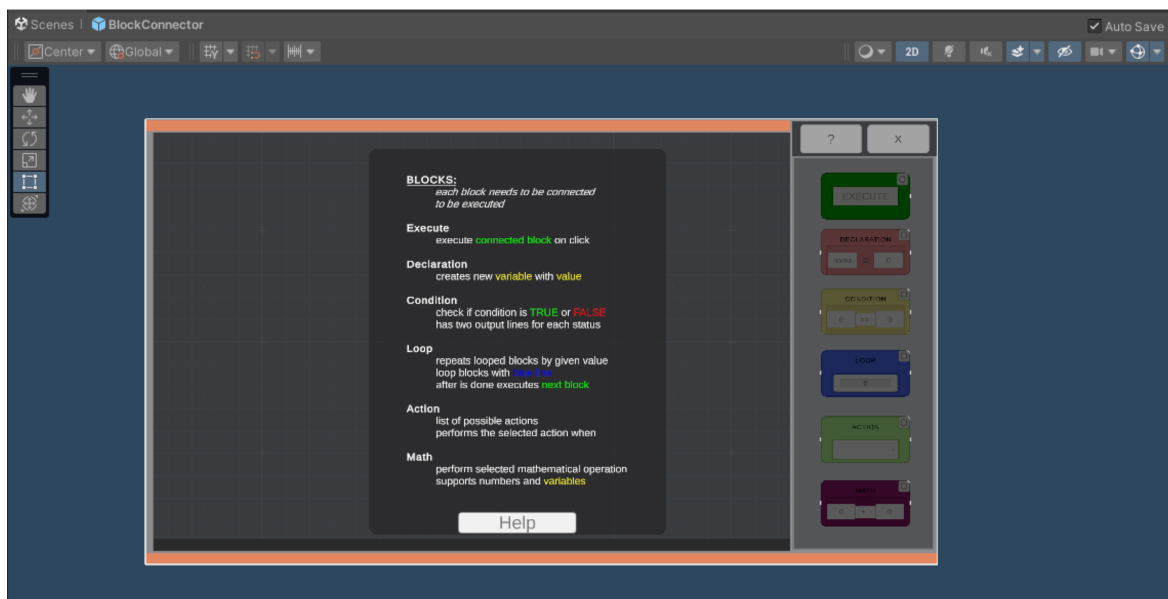
Metoda *Start*() inicializuje odkazy na komponenty animatoru, potřebné hodnoty, zvuk dveří a přiřazuje název dveří. Veřejné metody *OpenDoor*() a *CloseDoor*() slouží k otevírání a zavírání dveří. Pokud jsou dveře již otevřené nebo zavřené, metoda je ignorována. Pokud jsou dveře uzamčeny, je zkontrolována hodnota potřebná k jejich otevření. Pokud je tato hodnota v rozmezí mezi minimální a maximální hodnotou potřebnou k otevření, dveře se otevřou a přehraje se zvukový efekt. Stav dveří je následně aktualizován na otevřený. Metoda *GetDoorData*() ukládá aktuální stav dveří do objektu typu *DoorData*. Metoda *SetDoorData*() nastavuje stav dveří podle dat z objektu typu *DoorData*.

DoorController pracuje s třídou *NeedValue*, která slouží k určení hodnoty potřebné k otevření dveří. Obsahuje odkaz na blok s hodnotou, jméno hodnoty, minimální a maximální hodnotu potřebnou k otevření a zda je hodnota potřebná k otevření.

7.4 Rozhraní vizuálního programování

Pro samotnou mechaniku propojování bloků a tím tvoření jednotlivých algoritmů, bylo vytvořeno rozhraní, které slouží k sestavení algoritmu a splnění tak daného úkolu v levelu. Veškeré grafické prvky rozhraní byly vytvořeny ve volně dostupném softwaru GIMP. [29]

Samotné rozhraní (Obrázek 17) obsahuje plochu pro vytvoření programu, oblast obsahující jednotlivé bloky poskytnuty ke splnění daného úkolu, pole s textem nápovědy a dvě tlačítka pro zapnutí/vypnutí nápovědy a zavření samotného rozhraní. Ve spodní části plochy pro tvorbu programu se nachází stavový řádek, který vypisuje aktuální blok, který je spuštěn. Cele rozhraní bylo uloženo jako prefab, pro zjednodušení práce v editoru a dodržení stejného stylu pro každé rozhraní.

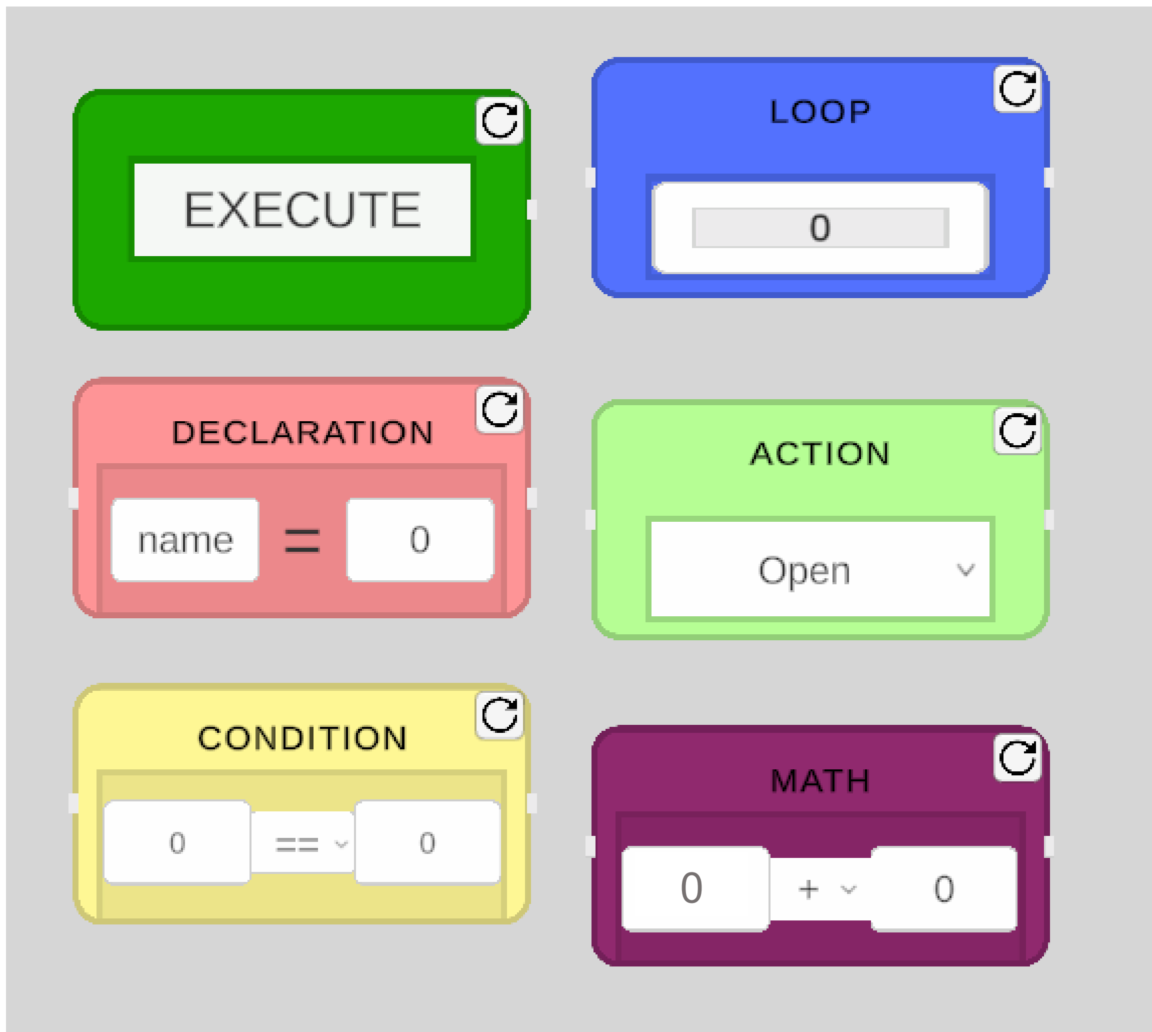


Obrázek 17. Rozhraní pro stavbu algoritmu

7.4.1 Rozhraní bloků pro tvorbu programu

Bloky byly tvořeny za pomoci Unity editoru a byly odlišeny různými barvami pro jednoduší rozpoznání funkce daného bloku (Obrázek 18). Každý blok obsahuje skript *BlockDragDrop.cs* pro možnost pohybu a propojování bloků. Další společnou vlastností jsou body pro ukotvení začátku a konce propojovací úsečky mezi bloky. Bloku *Execute* bylo ukotvení pro konec propojení odebráno, aby nemohl být následujícím blokem jiného bloku a sloužil tak pouze jako začáteční blok programu. Bloky také obsahují tmavé překrytí, které slouží pro efekt ztmavení bloku při dvojkliku na blok. Poslední viditelnou vlastností, kterou má každý blok, je tlačítko pro resetování propojení.

Samotné propojení bloků se automaticky resetuje, když hráč chce propojit blok s jiným blokem než s již propojeným, ale pokud chce hráč vymazat propojení daného bloku, bez přidání nového následujícího bloku, může k tomu použít toto tlačítko. Tlačítko bylo vytvořeno za pomoci ikony[30] znázorňující resetování stavu.



Obrázek 18. Bloky

Jelikož každý blok představuje jinou funkcionalitu, bylo potřeba přiřadit jednotlivým blokům vstupy a vstupní pole podle požadavků přiřazeného skriptu. Prvním blokem, který byl vytvořen, je blok *Execute*. Tento blok obsahuje tlačítko, za pomoci, kterého spouští funkci následujícího bloku.

Blok *Declaration* slouží k deklarování jednotlivých proměněných, které se budou v rámci programu používat, proto byl vytvořen s dvěma vstupními poli pro název a hodnotu deklarované proměnné.

Vstupní pole pro hodnotu a název se pro jednotlivé úkoly dají zamykat za pomoci editoru tak, aby hráč řešil problém s již vytvořenou proměnnou, s kterou příklad pracuje, a využíval ji pro správnou funkcionalitu.

Dalším poskládaným blokem v editoru byl blok *Condition,* který slouží jako porovnávací blok zadané kondice. Bloku byly přiřazeny dvě vstupní pole pro každou proměnou a hodnotu a rozbalovací seznam pro výběr operátoru.

Pro vytvoření bloku *Loop* bylo přidáno pouze jedno vstupní pole, které bylo nastaveno pouze pro příjímání vstupu celých čísel. Toto číslo reprezentuje hodnotu, kolikrát se blok zopakuje předtím, než půjde na následující blok.

Pro blok *Action* stačilo přidat rozbalovací menu pro výběr akcí, které byly vytvořeny pro daný úkol. Jednotlivé akce lze vytvořit za pomoci rozhraní editoru a vytvořeného skriptu *ActionBlock.cs* pro tento blok. Každá akce má vlastní název a přiřazený event, který volá při vybrání a spuštění bloku.

Posledním blokem vytvořeným v editoru byl blok *Math*. Tomuto bloku byly podobně jak v bloku *Condition* přiřazeny dvě vstupní pole pro jednotlivé hodnoty a proměnné a rozbalovací seznam pro výběr operátoru. Blok pak na dvou zadaných hodnotách provádí vybranou operaci.

7.4.2 Nápověda v rozhraní

Rozhraní pro tvorbu algoritmu bylo také rozšířeno o nápovědu (Obrázek 19), která hráče detailně seznamuje s jejich aktuálním úkolem, a poskytuje podrobný popis jednotlivých proměnných a akcí, které jsou v rámci úkolu definovány. Tato nápověda má za cíl usnadnit hráčům porozumění úkolu a jeho jednotlivých prvků, aby mohli efektivněji vytvářet svůj algoritmus. Kromě toho obsahuje nápověda také podrobné informace o funkcích jednotlivých bloků, aby hráči bylo jasné, jaký účel má každý blok a jak ho využít v rámci jejich algoritmu.

Samotná nápověda se zobrazí po stisknutí tlačítka v rozhraní pro nápovědu. Po stisknutí se uprostřed rozhraní objeví objekt s textem, který popisuje nejprve aktuální úkol, který je potřeba splnit. Pod úkolem jsou popsány dané proměnné, se kterými hráč pracuje pro splnění daného úkolu, a jejich typ. Poslední část, kterou tento text popisuje, jsou samotné akce, které může hráč zvolit za pomoci bloku *Action*.

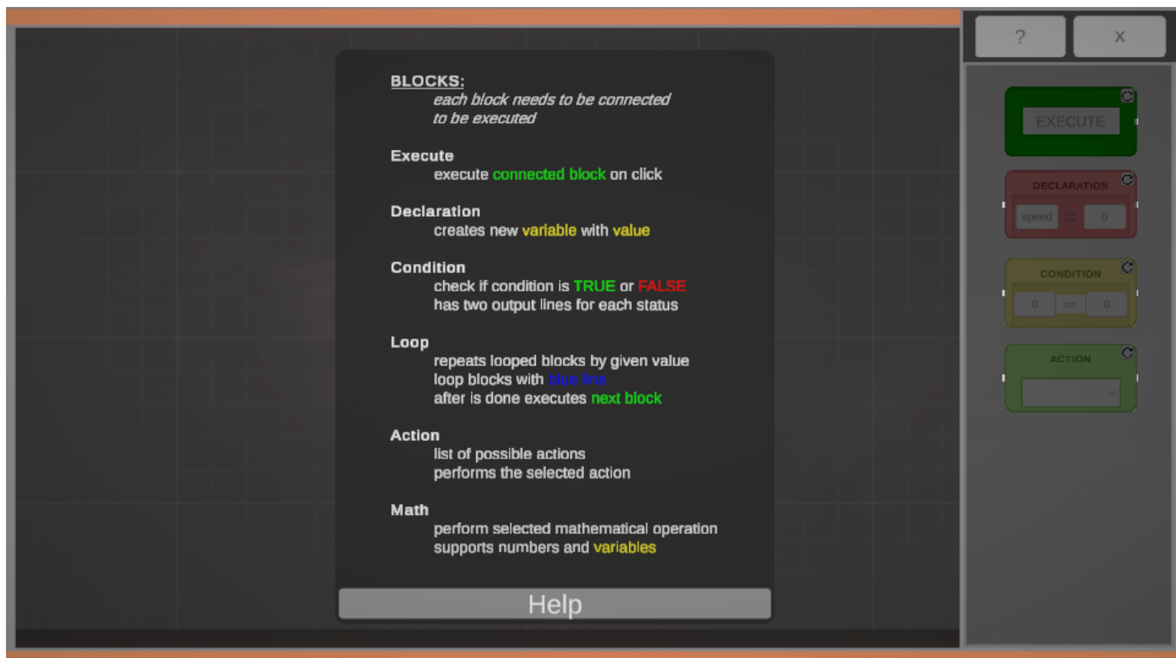
Každá akce obsahuje krátký popis toho, co dělá a jakou proměnnou případně vrací. Do objektu s nápovědou bylo dále přidáno tlačítko, které slouží k přepnutí na nápovědu pro jednotlivé bloky.

Obrázek 19. Nápověda k úkolu

Pro nápovědu, která popisuje jednotlivé bloky byl vytvořen podobný objekt jako pro první nápovědu. Do samotné nápovědy však místo popisu úkolu a jednotlivých akcí, byly popsány jednotlivé bloky, které hra obsahuje (Obrázek 20).

Na začátek nápovědy bylo napsáno že každý blok, který má být spouštěn, musí být propojený, aby mohl být proveden. Poté byl popsán blok *Execute*, který se používá jako základní blok každého algortimu. Následně byl popsán blok *Declaration*, který deklaruje a inicializuje novou proměnnou se zadaným názvem a hodnotou. Dalším popsaným blokem byl blok *Condition*, který kontroluje, zda je daná podmínka pravdivá nebo nepravdivá. Blok *Loop* byl dalším blokem, který byl popsán do nápovědy. Blok opakuje smyčku podle zadané hodnoty. Po tomto bloku se nachází blok *Action*, který obsahuje seznam možných akcí a vykonává vybranou akci. Posledním popsaným blokem byl blok Math, který vykonává matematickou operaci podle výběru operátora na dvou vstupních operandech. Blok byl vytvořen tak, aby podporoval vstupy jak v číslech, tak i v inicializovaných proměnách.

Na konci nápovědy bylo vytvořeno tlačítko, které přepne rozhraní nápovědy zpět na nápovědu k danému úkolu.



Obrázek 20. Nápověda pro bloky

7.4.3 Skript pro načtení a vypnutí rozhraní

Pro zobrazení rozhraní během hry, když hráč přistoupí k objektu počítače, byl vytvořen skript CallUI.cs. Tento skript řídí zobrazení uživatelského rozhraní ve hře a manipuluje s různými prvky rozhraní a herními objekty. Ve třídě jsou deklarovány proměnné pro odkazy na prvky rozhraní, hráče a kamery, které budou ovlivněny aktivací rozhraní.

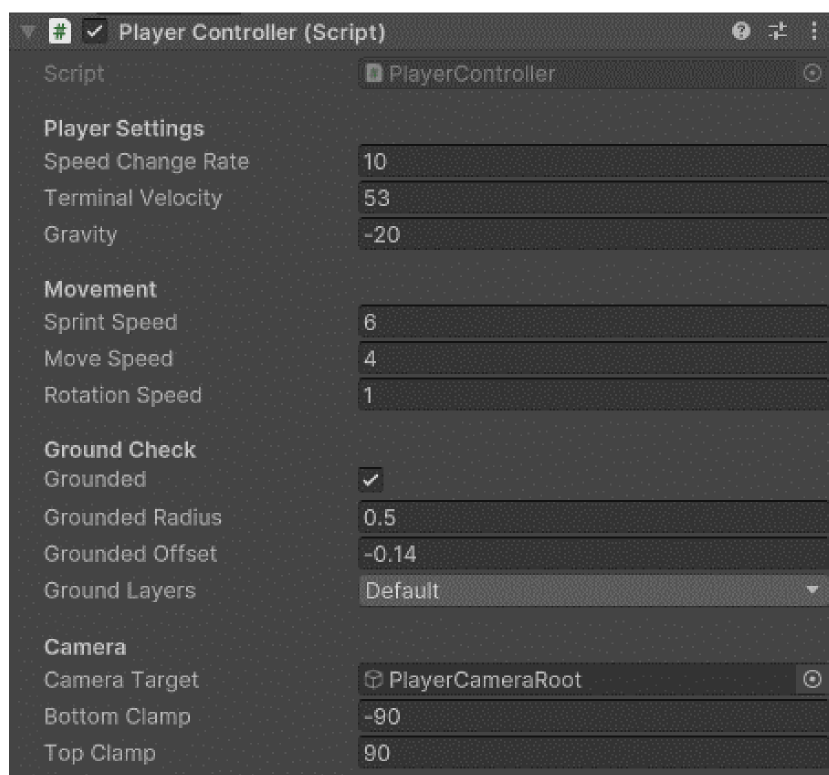
Metoda *Awake*() inicializuje odkazy na prvky, jako je hlavní plátno, kamera pro rozhraní, hráč a kamera hráče. Metoda *Start*() vypíná rozhraní při spuštění scény a deaktivuje kameru rozhraní. Také získává odkaz na ovládání hráče.

Metoda *OnTriggerEnter*(*Collider other*) se spouští, když hráč vstoupí do zóny spouštěče. Pokud hráč vstoupí do této zóny, rozhraní se zapne a provádí se různé úpravy, jako je přepínání prvků a kamer, povolení kurzoru myši a deaktivace pohybu hráče. Metoda *TurnOffUI*() vypíná rozhraní a provádí opačné úpravy jako při zapnutí rozhraní. Tato metoda byla následně přiřazená v editoru do unity event funkce *OnClick()* tlačítka pro vypnutí samotného rozhraní. To znamená že při kliknutí na tlačítko pro vypnutí rozhraní, se zavolá tato metoda a rozhraní se vypne. Metody *ToggleLines*() a *ToggleInputs*() slouží k zapnutí nebo vypnutí různých prvků v rozhraní, jako jsou linky a vstupní pole.

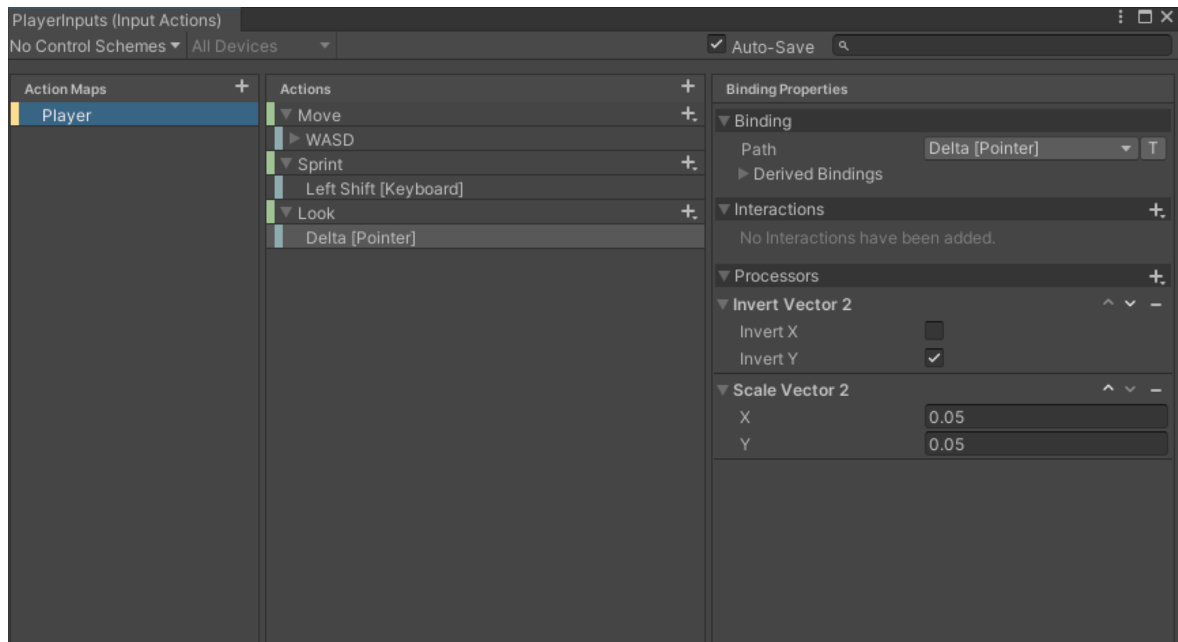
7.5 Vytvoření postavy hráče

Pro pohyb hráče a kamery v samotné hře bylo potřeba vytvořit nový objekt s názvem „Player", který obsahuje všechny potřebné skripty pro pohyb a samotnou kameru, která reprezentuje pohled hráče.

Pohyb hráče je řešen v objektu „PlayerCapsule", který byl vytvořen pod objekt hráče. Do tohoto objektu byly přidány skripty pro pohyb hráče *PlayerController* a *PlayerInputs*. Následně byly do *PlayerController* zadány hodnoty tak, aby pohyb byl plynulý (Obrázek 21). Objektu byl také přidán vestavěný komponent *CharacterController* který se používá ve skriptu *PlayerController* a zajišťuje pohyb a limity pro pohyb postavy. Posledním komponentem, který se stará o pochyb hráče, je komponent *PlayerInput*, ve kterém byly namapovány jednotlivé klávesy a jejich akce při zmáčknutí klávesy (Obrázek 22). Dále byl vytvořen nový objekt „Capsule", který reprezentuje velikost hráče a zajišťuje, aby se hráč zastavoval o zdi a překážky.



Obrázek 21. Hodnoty *PlayerController* v editoru



Obrázek 22. Mapování kláves pro pohyb

Pro vytvoření kamery byl použit volně dostupný nástroj „CinemaMachine“ [35], který umožňuje jednoduší práci s kamerou. Do objektu „MainCamera“ byl přidán komponent *CinemamachineBrain*, který se stará o funkcionalitu nástroje kamery. Do objektu „Player“ byl vložen nový objekt „PlayerFollowCamera“ s komponentem *CinemamachineVirtualCamera*, který přepisuje nastavení kamery, která bude sloužit jako hlavní kamera hráče. Pro tento komponent bylo potřeba vytvořit do objektu „PlayerCapsule“ prázdný objekt, který slouží jako ukotvení pro kameru. Kamera tak bude následovat postavu při pohybu. Dále byla kameře nastavena velikost zorného pole na šedesát. Do objektu „MainCamera“ byl také přidán komponent *PostProcessLayer*, který vylepšuje vzhled hry.

Pod objekt hráče byl přidán objekt s komponentem *AudioSource*, který se používá pro přehrávání zvuků při pohybu postavy.

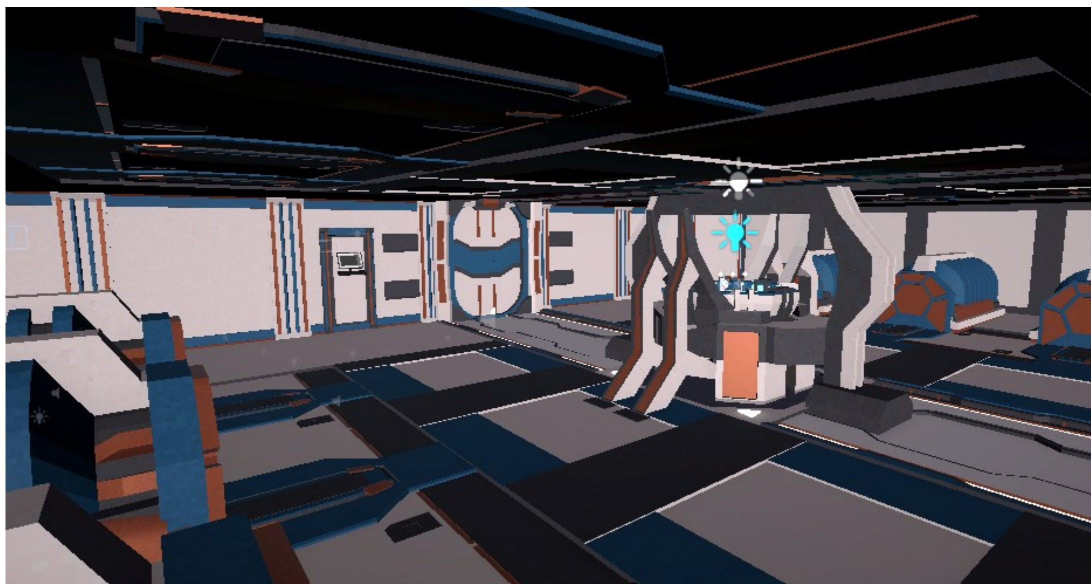
7.6 Tvorba herního prostředí

Pro vytvoření prostředí vesmírné lodi byl použit balíček modulárních prefabů[27]. Tyto prefaby bylo potřeba poskládat v editoru tak, aby představovaly prostředí vesmírné lodi.

Loď byla vytvořena tak, aby se skládala z několika místností, které hráč postupně prochází. Každá místnost představuje jinou část lodi a má přiřazený úkol, který hráč musí splnit, aby mohl pokračovat.

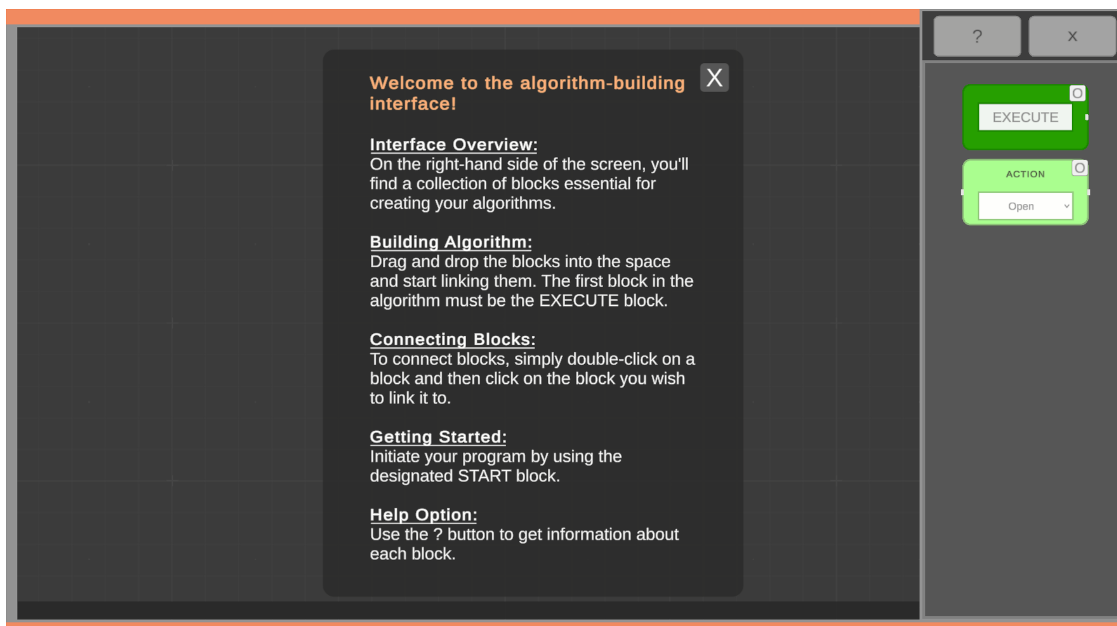
7.6.1 Začátek

Hra začíná probuzením hráče z hibernace v místnosti, ze které se musí dostat, aby mohl pokračovat. Za tímto účelem byla vytvořena místnost, která představuje místnost vesmírné lodi se speciálně upraveným zařízení pro kryoniku (Obrázek 23). Do místnosti byl přidán počítač s rozhraním pro tvorbu programu a dveře, které se dají otevřít, nebo zavřít, podle vybrané akce ve vytvořeném programu.



Obrázek 23. Začáteční místnost

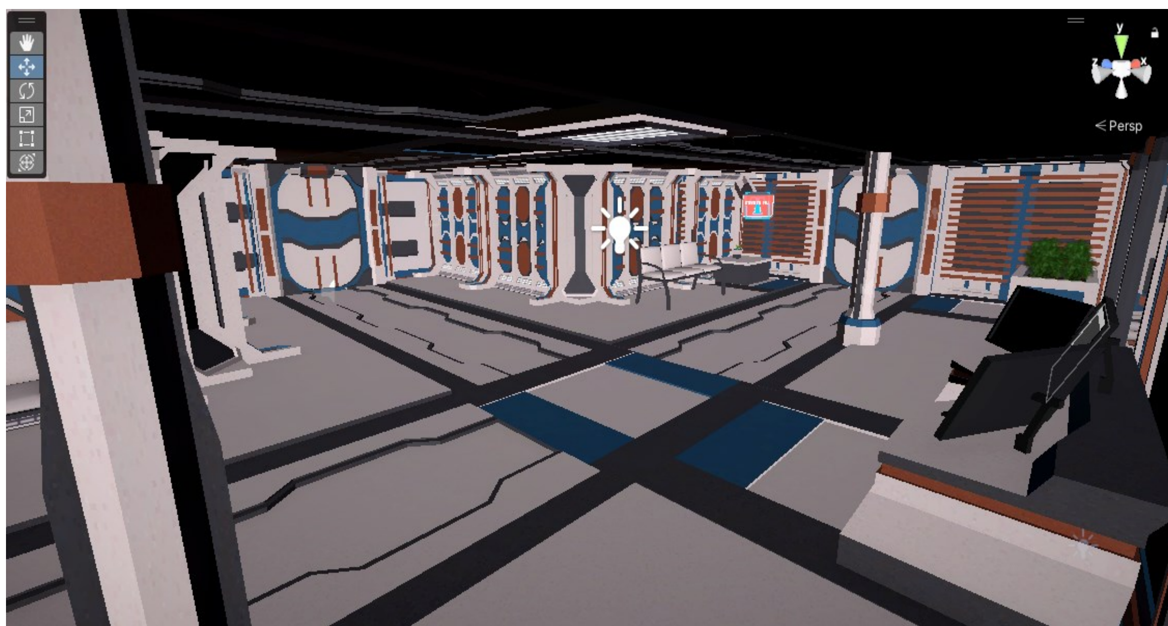
Jelikož se jedná o první místnost ve hře bylo rozhraní pro vizuální programování v této místnosti upraveno tak, aby obsahovalo nápovědu ovládání a práce s rozhraním pro stavbu programů (Obrázek 24). V nápovědě byly zmíněny informace popisující samotné rozhraní, jak vytvořit algoritmus za pomoci propojování bloků, a jak se program spouští a odkazuje na pomocné tlačítko, které zobrazí nápovědu pro daný úkol a jednotlivé bloky. Protože se jedná o první úkol, na který hráč narazí, je tento úkol vytvořen tak, aby byl velice jednoduchý a seznámil hráče se základy herní mechaniky pro tvorbu algoritmů. Proto byly k tomuto úkolu poskytnuty pouze bloky *Execute* a *Action* s akcí pro otevření a zavření dveří.



Obrázek 24. Nápověda pro rozhraní

7.6.2 Vstupní místnost

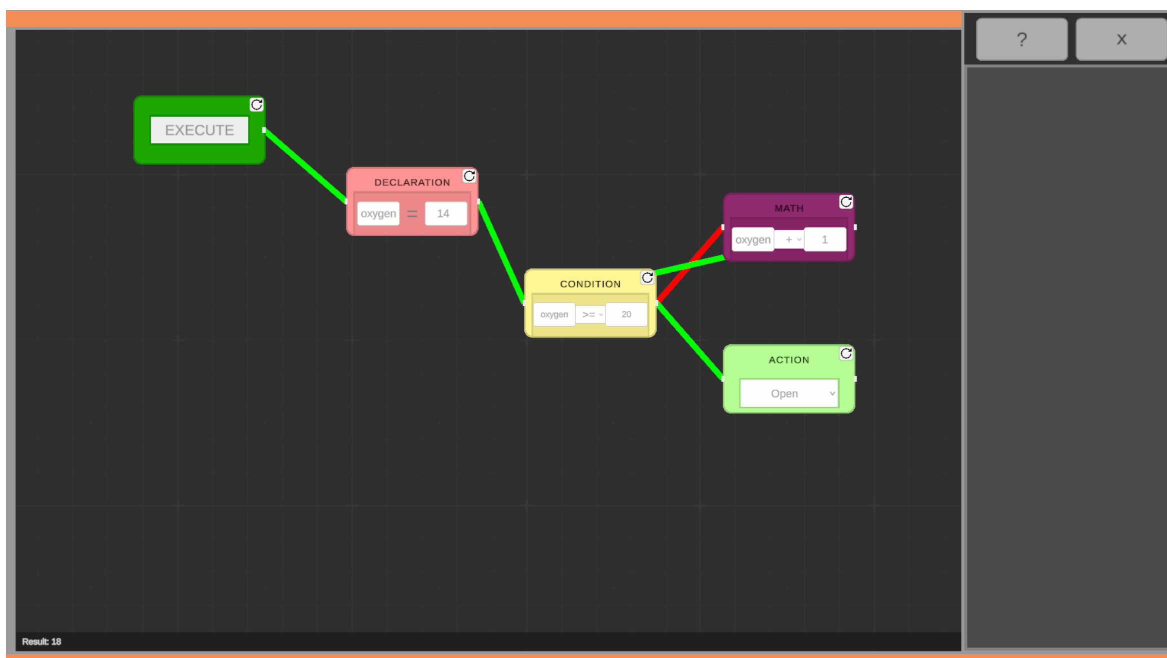
Po otevření dveří se hráčovi naskytne nová místnost, která slouží jako přechodová místnost do hlavní části lodi (Obrázek 25). Za dveře do místnosti byl vložen spouštěč, který změní text aktuálního úkolu otevření dveří, na úkol pro doplnění kyslíku ve zbytku lodi, aby do ní mohl hráč bezpečně vstoupit.



Obrázek 25. Vstupní místnost

Pro splnění tohoto úkolu je v místnosti umístěn stůl s počítačem, který obsahuje další rozhraní pro tvorbu algoritmu. Toto rozhraní bylo oproti prvnímu rozšířeno o bloky *Declaration*, *Conditon* a *Math* (Obrázek 26). Blok deklarace obsahuje pevný název a hodnotu proměnné, která představuje aktuální hodnotu kyslíku ve zbytku lodi. Vstupy tohoto bloku byly proto uzamčeny, aby je hráč nemohl upravovat. Pro blok matematické operace byla uzamčena jedna vstupní hodnota, aby se hodnota kyslíku dala měnit pouze o jednu úroveň. Do místnosti byly přidány dveře, které se ovládají za pomoci vybrané akce v programu.

Dveře se otevřou pouze, když je kyslík na bezpečné úrovni. Vedle dveří byl přidán displej, který ukazuje aktuální hodnotu kyslíku. Pokud je kyslík v bezpečné úrovni pro hráče, tak se pozadí displeje rozsvítí zeleně, jinak by zůstalo pozadí červené. Pro tuto funkcionalitu, byly vytvořeny skripty SetValueToInput.cs a ChangeScreenOnValue.cs. První skript se stará o aktualizování hodnoty zobrazované na displeji a druhý o změnu barvy pozadí při dosažení hodnoty v rozmezí, které se definuje za pomoci proměnných.



Obrázek 26. Algoritmus pro navýšení hladiny kyslíku

Skripty pro vstupní místnost

Pro mechaniku vypsání aktuální hodnoty kyslíku na displej bylo potřeba napsat skript SetValueToInput.cs.

Skript obsahuje proměnnou pro textové pole, do kterého se vypisuje hodnota kyslíku, proměnnou pro blok obsahující hledanou proměnnou a proměnnou pro název hledané proměnné. Následuje metoda *Update()*, ve které se každý snímek hry získává aktuální hodnota hledané proměnné z bloku a následně se zapíše do textového pole.

Dalším vytvořeným skriptem pro tuto místnost byl skript ChangeScreenOnValue.cs, který obsahuje proměnnou pro textové pole s hodnotou, kterou bude porovnávat, proměnnými pro rozsah minimální a maximální hodnoty v rozsahu a dvě proměnné pro objekt s barvou pozadí displeje, jeli rozsah splněn, nebo pokud splněn nebyl. Skript obsahuje jedinou metodu a to metodu *Update()*, ve které se kontroluje zda hodnota z textového pole je v rozsahu daným minimální a maximální hodnotou. Pokud se hodnota nalézá v daném rozsahu je pozadí displeje změněno na zelené pozadí, jinak je pozadí displeje červené.

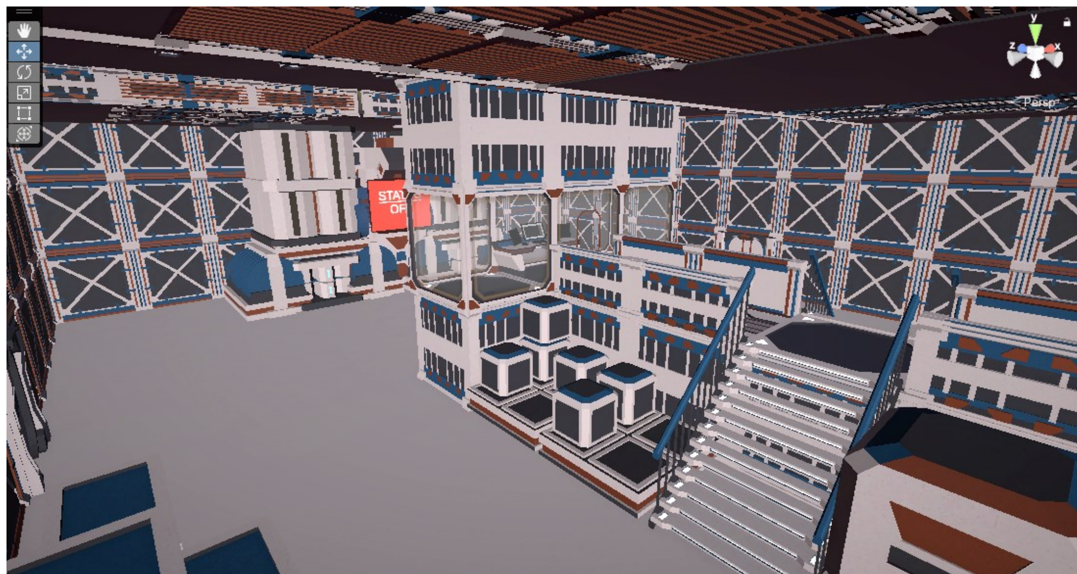
7.6.3 Hlavní hala

Když hráč projde dveřmi ve vstupní místnosti dostane se ke schodišti a chodbě, která ho dovede až do hlavní haly lodi (Obrázek 27). Tato hala slouží k propojení všech zbylých místností ve hře. V hale se nachází vstupy do strojovny, laboratoře a můstku. Každá místnost byla schována za dveře, které je potřeba otevřít splněním aktuálního úkolu. Nad každé dveře byla umístěna cedule s názvem místnosti pro lepší orientaci. Dveře do strojovny jsou otevřeny, společně s dveřmi od hlavní haly, jelikož jde o místnost, do které se hráč vydá při prvním vstupu do haly. Při vstupu do haly se změní název a popis aktuálního úkolu na úkol opravy motoru ve strojovně.

Obrázek 27. Hlavní hala lodi

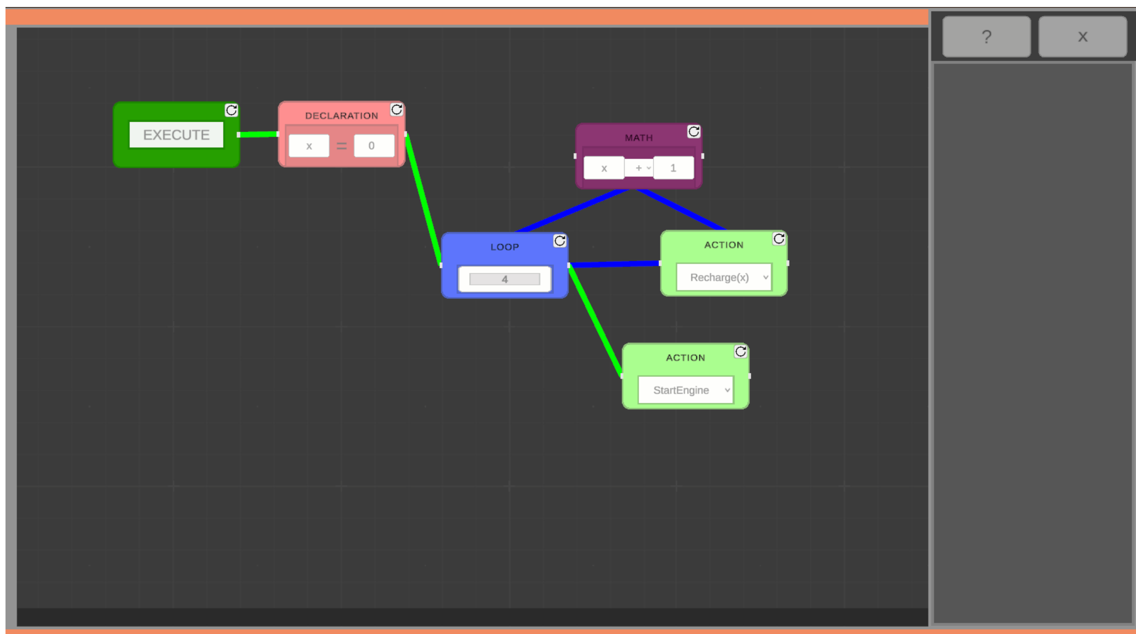
7.6.4 Strojovna

Při vstupu do strojovny (Obrázek 28) se před hráčem objeví prosklená místnost s počítačem, který otvírá rozhraní pro tvorbu algoritmu. V místnosti se také krom samotného počítače nachází motor s displejem ukazující aktuální stav motoru. Kolem motoru jsou čtyři vybité baterie. Úkolem pro tuto místnost je sestavení algoritmu, který nabije vybité baterie a poté spustí motor lodi. Pokud je úkol splněn, baterie se rozsvítí pro efekt nabití a stav motoru na displeji se změní společně s barvou displeje. Po nastartování motoru jsou otevřeny dveře k můstku a je změněn název a popisek aktuálního úkolu.



Obrázek 28. Strojovna lodi

K sestavení algoritmu pro dobití baterií a spuštění motoru byly hráči poskytnuty bloky *Execute*, *Declaration*, *Loop*, *Math* a dvakrát blok *Action* (Obrázek 29). Blok *Declaration* deklaruje proměnnou „x“, která se využije jako index pro dobití dané baterie. Jelikož se v místnosti nachází čtyři baterie, které hráč musí před spuštěním motoru dobít, byl poprvé ve hře hráči poskytnut i blok *Loop*, za pomoci, kterého může vytvořit smyčku, která bude procházet všechny baterie a postupně je dobíjet za využití akce *Recharge(x)*, po uplynutí všech smyček může být zavolán blok s akcí *StartEngine*. Akce zkontroluje, zda jsou všechny baterie nabity a poté spustí motor.



Obrázek 29. Algoritmus pro zapnutí motoru

Skripty pro strojovnu

Pro jednotlivé akce bylo potřeba napsat nové skripty. Prvním vytvořeným skriptem byl skript BatteryManager.cs, který řídí správu baterií a provádí plynulé přechody mezi materiály baterií na základě jejich úrovně nabití. Ve třídě je vnořená třída *BatteryData*, která uchovává informace o baterii, jako je odkaz na její vykreslování, materiál pro nízkou a plnou úroveň nabití a samotná úroveň nabití. Dále jsou deklarovány proměnné pro seznam baterií, délku trvání přechodu mezi materiály a pole koprogramů pro synchronizaci přechodů mezi materiály baterií. Metoda *ChargeBattery*() zvyšuje úroveň nabití určité baterie na maximum na základě vstupu. Pokud je vstup správný a baterie existuje, úroveň nabití se nastaví na jedna. Metoda *SetBatteryLevel*() aktualizuje úroveň nabití určité baterie a spouští koprogram pro plynulý přechod mezi materiály baterií. Metoda *TransitionBatteryMaterial*() je koprogram, který provádí plynulý přechod mezi materiály baterií. Nejprve získává startovní a cílový materiál pro baterii na základě její úrovně nabití. Poté postupně interpoluje barvu materiálu baterie, aby vytvořila efekt plynulého přechodu. Na konci nastaví materiál baterie na cílovou barvu a ukončí koprogram.

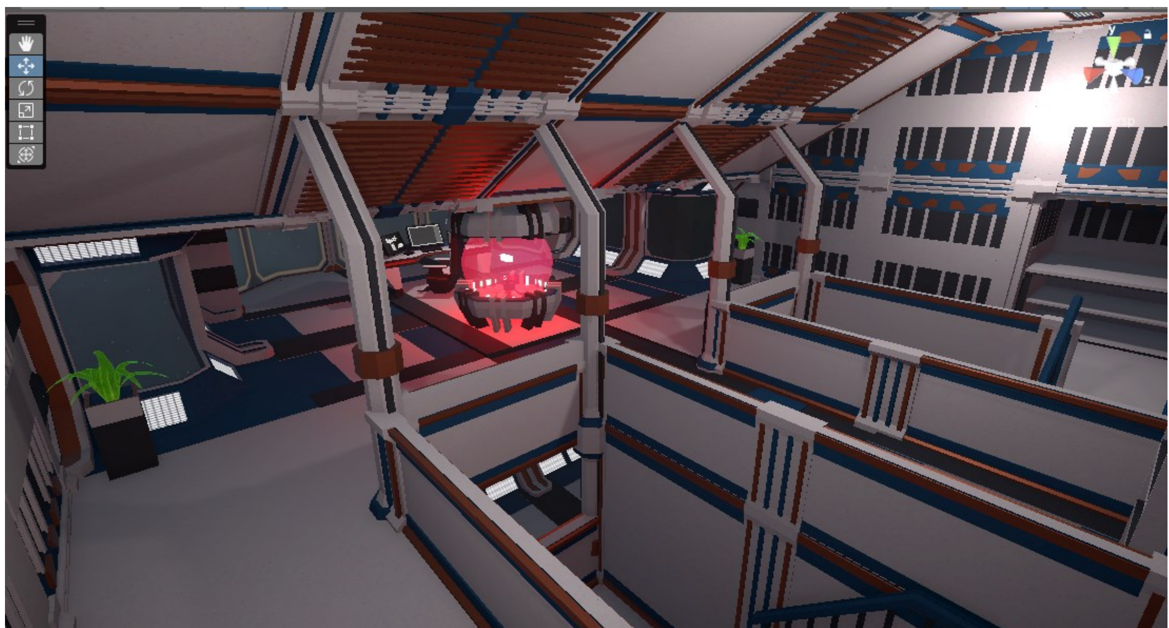
Dalším vytvořeným skriptem byl skript pro zapnutí motoru lodi. Tento skript je EngineSwitch.cs a řídí přepínání mezi dvěma stavovými UI pro motor, ovládá otevírání dveří a mění text úkolu v závislosti na stavu baterií. Proměnné *engineUiOn* a *engineUiOff* odkazují na UI prvky, které zobrazují stav motoru zapnutý a vypnutý.

Proměnná *doorToOpen* a *questChanger* odkazují na objekty DoorController a ChangeQuest, které se mají provést při spuštění motoru. Metoda *StartEngine*() spouští motor a zjišťuje, zda jsou baterie dostatečně nabité voláním metody *CheckBatteries*(). Poté volá metodu *ToggleEngine*(). Tato metoda zapíná nebo vypíná UI prvky motoru a volá metody pro otevření dveří a změnu textu úkolu v závislosti na stavu motoru. Metoda *CheckBatteries*() kontroluje stav nabité baterie. Prochází všechny baterie a zjišťuje, zda je jejich úroveň větší než zadaná hodnota. Pokud jsou všechny baterie nabité, vrátí „true", jinak vrací „false".

7.6.5 Můstek

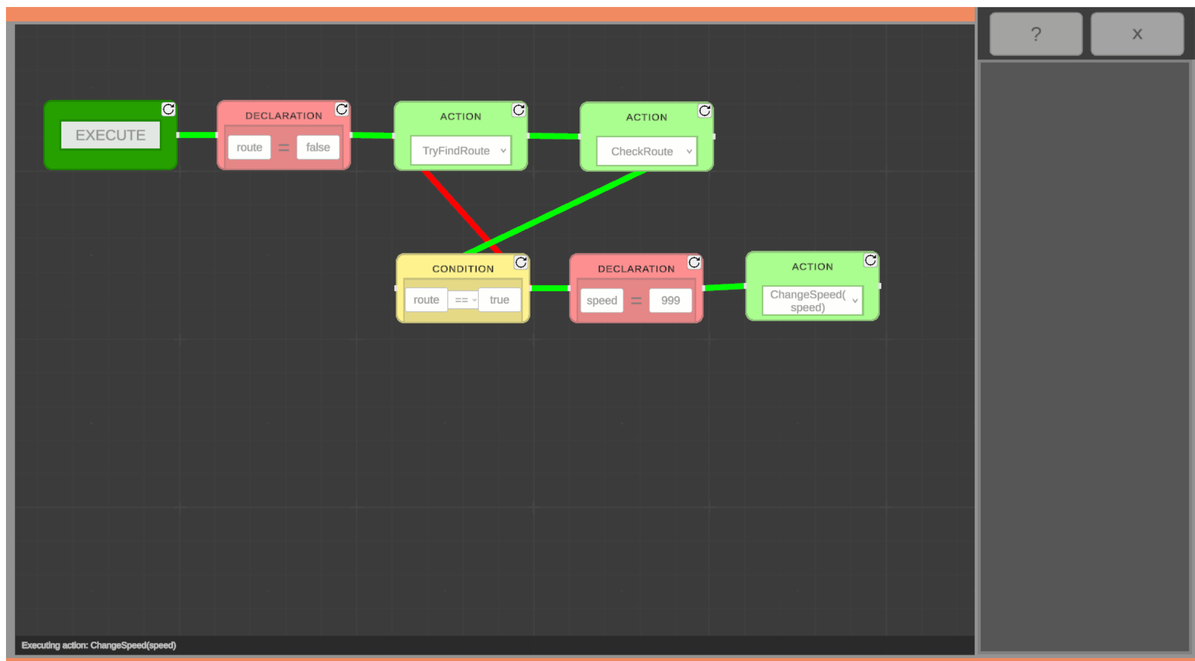
Po úspěšném nastartování motoru se otevře nová místnost a změní se aktuální úkol na úkol pro uvedení lodi do pohybu. Za tímto účelem byla vytvořena místnost, která reprezentuje můstek vesmírné lodi (Obrázek 30), ve kterém hráč může vytvořit algoritmus pro nalezení bezpečné trasy a uvede loď do pohybu za pomoci změny rychlosti lodi v programu. Poté co hráč rozpohybuje loď tak se otevře místnost laboratoře a změní se název a popis úkolu.

Pro tento účel byl do místnosti umístěn objekt počítače, který zobrazuje nejen rozhraní pro tvorbu algoritmu, ale také displej, který ukazuje aktuální rychlost lodi, a druhý displej pro zobrazení textu, zda je cesta bezpečná nebo ne. Dále byl do místnosti přidán objekt projektoru, který mění barvu projekce podle toho, zda byla nebo nebyla nalezena bezpečná cesta.



Obrázek 30. Můstek lodi

K sestavení programu byly hráči poskytnuty bloky *Execute*, bloky pro deklaraci *route* a *speed*, blok *Condition* a tři bloky *Action* (Obrázek 31). Pro úkol byla také vytvořena akce *TryFindRoute*, která vygeneruje novou náhodnou cestu. Další vytvořenou akcí byla akce *CheckRoute,* která vrací do proměnné *route* pravdu nebo nepravdu podle toho, zda byla nalezena bezpečná cesta. Poslední přidanou akcí do úkolu je akce *ChangeSpeed*, která nastavuje rychlost lodi podle proměnné *speed*.



Obrázek 31. Algoritmus pro nalezení trasy

Skripty pro můstek

Funkcionalitu jednotlivých akcí obsahuje vytvořený skript BridgePcActions.cs. Skript obsahuje funkce pro interakci s herními objekty na můstku lodi. Ve třídě byly deklarovány proměnné pro kontrolu bezpečnosti trasy, odkazy na textová pole v UI a proměnná pro rychlost lodi. Metoda *CheckRoute*(*Block block*) zjišťuje, zda je trasa bezpečná, a aktualizuje text, který vypisuje, zda je trasa bezpečná, nebo ne. Také aktualizuje proměnnou *route* v bloku a volá změnu *SwapColor(bool routeSafe)*. Metoda *NewRoute*(*Block block*) generuje náhodně zda je trasa bezpečná nebo není. Metoda *ChangeSpeed*(*Block block*) mění rychlost lodi podle hodnoty proměnné *speed* z bloku. Pokud je rychlost správně zadaná, postupně se mění rychlost lodi a aktualizuje se textové pole ukazující rychlost lodi. Také aktualizuje úkol a otevírá dveře za pomoci *doorController*. Metoda *ChangeSpeedCoroutine*(*float targetSpeed*) provádí postupnou změnu rychlosti lodi. Postupně se aktualizuje textové pole s rychlostí lodi.

Pro změnu projektoru slouží skript ProjectorLightChange.cs, který mění materiál a světlo objektu projektoru. Ve třídě jsou deklarovány proměnné pro odkazy na světlo a herní objekt hologramu, jakož i pro materiály pro projekci v normálním a upozorňujícím režimu. Metoda *SwapColor(bool routeSafe)* provádí výměnu barev světla a materiálu projektoru na základě aktuálního stavu bezpečnosti cesty. Pokud je zvolen materiál pro normální režim a stav bezpečnosti je nastaven na pravdivý, materiál hologramu a barva světla jsou nastaveny na modrou. Pokud je přiřazený materiál pro nebezpečí a stav bezpečnosti je nastaven na nepravdivý, materiál hologramu a barva světla jsou nastaveny na červenou. Nakonec jsou nové materiály a barvy přiřazeny k odpovídajícím prvkům hologramu a světla.

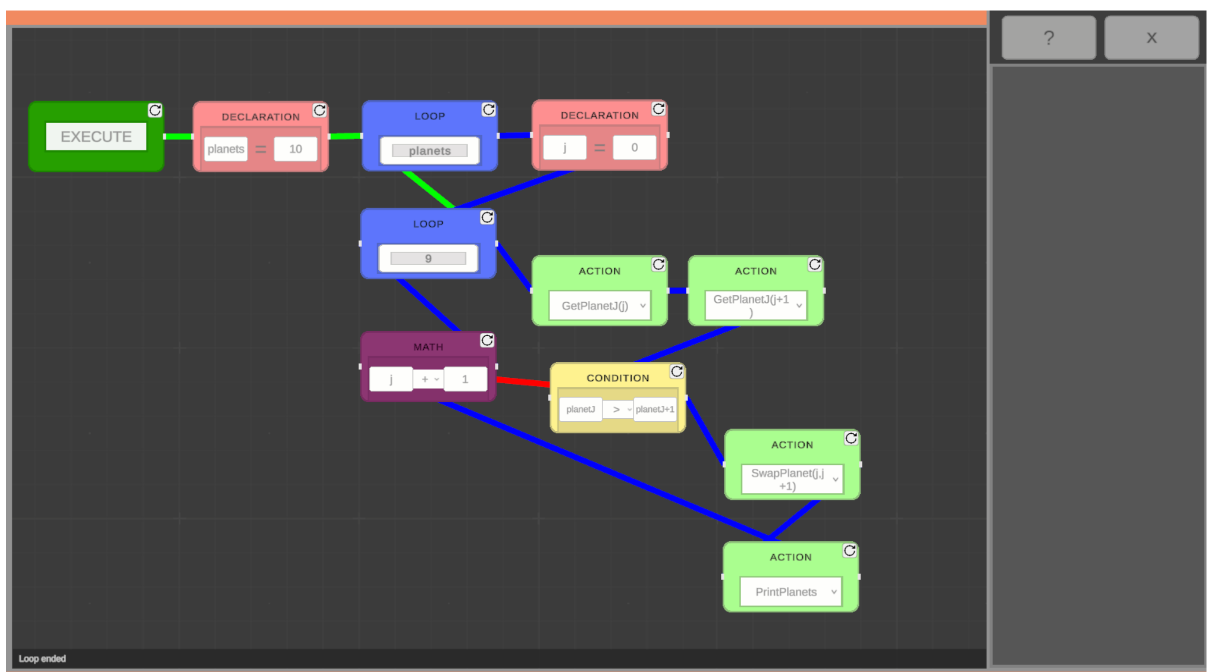
7.6.6 Laboratoř

Poslední vytvořenou místnosti je laboratoř (Obrázek 32). Do této místnosti byl vložen objekt počítače pro otevření rozhraní k tvorbě programu. Dále byla do místnosti umístěna velká obrazovka, která vypisuje název a vzdálenost všech nalezených planet pro lepší orientaci v úkolu. Pro splnění daného úkolu v této místnosti je potřeba, aby hráč seřadil nalezené planety podle vzdálenosti od nejbližší po nejvzdálenější. Když jsou všechny planety seřazeny podle správného pořadí, aktualizuje se hráči aktivní úkol na vrácení se na začátek hry, kde se nachází zároveň i konec hry.

Obrázek 32. Laboratoř

K vyřešení tohoto úkolu byly hráčovi poskytnuty všechny bloky v rozhraní (Obrázek 33). Úkol obsahuje deklaraci pro proměnnou počtu planet *planets* s pevně daným počtem planet

a pomocnou proměnnou *j*, která se používá jako index pro jednotlivé akce. Pro tento úkol byly do bloku *Action* vytvořené akce *GetPlanet(j)* a *GetPlanet(j+1)*, které najdou planetu podle indexu z pole nalezených planet a inicializují proměnnou *planetJ* a *planetJ+1*, která je pak používána ve vstupech v bloku *Condition*. Další vytvořenou akcí pro tento úkol byla akce *SwapPlanet(j, j+1)*, která prohodí planety v poli nalezených planet. Poslední přidanou akcí byla akce *PrintPlanets*, která vypisuje aktuální seznam planet na obrazovku v místnosti.



Obrázek 33. Algoritmus pro seřazení planet

Skripty pro laboratoř

Pro funkcionalitu jednotlivých akcí byl pro tuto místnost napsán skript LabActions.cs, který obsahuje několik metod pro manipulaci s informacemi o planetách a provádění různých akcí v laboratoři. Ve skriptu byly vytvořeny dvě třídy. První třída *PlanetDistance* obsahuje informace o jménu planety a vzdálenosti od lodi. Druhá třída *LabActions* obsahuje metody pro jednotlivé akce. Také deklaruje seznam *planetDistances*, který uchovává jednotlivé planety s jejich vzdáleností.

Metoda *Awake()* inicializuje seznam *planetDistances* s názvy planet a jejich vzdálenostmi. Metoda *Update()* kontroluje, zda jsou planety ve správném pořadí podle vzdálenosti. Pokud ano, spustí se změna úkolu pomocí metody *ChangeQuestText*() ze třídy *ChangeQuest*. Metody *GetPlanetJ()* a *GetPlanetJplus1()* získávají vzdálenosti dvou planet na základě

proměnné *j* a ukládají je do proměnných *planetJ* a *planetJ+1*. Tyto vzdálenosti jsou také ukládány do bloku. Metoda *SwapPlanets()* prohodí pozice dvou planet v seznamu *planetDistances* na základě proměnné *j*. Metoda *PrintPlanets()* vypisuje informace o planetách, včetně jejich jmen a vzdáleností, do textového pole v UI. Metoda *CheckPlanets()* kontroluje, zda jsou planety seřazeny podle nejkratší vzdálenosti od lodi.

7.7 Uživatelské rozhraní

Do hry bylo potřeba přidat také uživatelské rozhraní, které by hráčům poskytovalo možnost ovládat stav hry a současně by jim pomáhalo orientovat se ve hře tím, že ukazuje aktivní úkol, který hráč musí splnit.

7.7.1 Hlavní menu

Po spuštění hry se hráči zobrazí hlavní menu, s výběrem možností pro pokračování hry, novou hru a ukončení hry. Pro hlavní menu byla vytvořena, v editoru Unity, nová scéna s názvem *MainMenu*, do které byly přidány objekty pro efekt zničené lodi ve vesmíru v pozadí menu. Pozadí pro menu pak tvoří zavěšený displej. Samotné hlavní menu je vytvořeno pomocí komponenty *Canvas* a tlačítek umístěných na obrazovce, která umožňuje hráčům interagovat s menu. Pro správu těchto tlačítek byl vytvořen skript MainMenuController.cs. Tento skript obsahuje funkce, které se aktivují po stisknutí každého tlačítka zvlášť, což umožňuje správné řízení navigace a chování hlavního menu. Do scény byl také přidán objekt *GameManager* se skriptem GameManager.cs, který se stará o načte-níložené hry po načtení scény.

Obrázek 34. Hlavní menu hry

Skripty pro hlavní menu

Pro správnou funkcionalitu hlavního menu byl napsán skript MainMenuController.cs, který řídí chování hlavního menu ve hře a obsahuje metody pro změnu scény, ukončení hry, pokračování ve hře a aktualizaci tlačítka pokračování.

Tento skript začíná deklarací tlačítka *continueButton*, které slouží k pokračování ve hře. První vytvořenou metodou byla metoda *Start*(), která se spouští při startu scény a aktualizuje stav tlačítka pro pokračování na uzamčený. Metoda *ChangeScene(string sceneName)* se použila k přepínání mezi scénami. Před změnou scény se schová kurzor a uzamkne se jeho pozice do středu obrazovky. Metoda *ExitGame()* ukončuje hru. Další vytvořenou metodou byla
ContinueGame(), která načítá hru z uloženého souboru, pokud se podaří najít instance *GameManager*. Poslední metodou je *UpdateContinueButton()*, která aktualizuje stav tlačítka pokračování podle existence uloženého souboru. Pokud soubor existuje, tlačítko je aktivní, jinak je deaktivováno.

Dalším vytvořeným skriptem byl skript GameManager.cs, který se stará o načtení scény samotné hry a následně načtení uloženého postupu. Ve třídě jsou deklarovány proměnné pro instanci GameManager a pro správce ukládání a načítání *SaveLoadManager*.

Metoda *Awake()* zajistí, že existuje pouze jedna instance *GameManager* v celé hře, a to tak, že se nenechá zničit při přechodu mezi scénami. Metoda *LoadGame()* slouží k načtení scény hry. Před načtením scény se přidá do událostí scény nová událost za pomoci *SceneManager.sceneLoaded*, aby mohla být provedena po načtení scény. Metoda *OnGameSceneLoaded(Scene scene, LoadSceneMode mode)* se spustí po načtení herní scény. Nejprve se odhlásí z události scény, aby nedošlo k opětovnému spuštění této metody. Poté se hledá instance *SaveLoadManager* ve scéně a pokud je nalezena, zavolá se metoda *LoadGame()*, která načte uložený postup.

7.7.2 Menu ve hře

Pro uložení a vypnutí hry bylo do hry přidané rozhraní pro menu (Obrázek 35), které hráč může zobrazit pomocí klávesy *escape* během hraní hry. Rozhraní bylo sestaveno za pomoci komponentu *Canvas,* ve kterém se nachází text pro pozastavenou hru, tmavé pozadí a dvě tlačítka. Pro tlačítka byl vytvořen skript GameMenu.cs obsahující jejich funkcionalitu. První tlačítko slouží pro pokračování ve hře, proto tlačítko při stisknutí volá metodu *Continue()*,

která vypne rozhraní a schová a uzamkne kurzor. Druhé tlačítko bylo vytvořeno tak, aby při stisknutí zavolalo metodu *SaveAndQuit()*, která uloží aktuální postup ve hře a následně vypne celou hru.

Obrázek 35. Menu ve hře

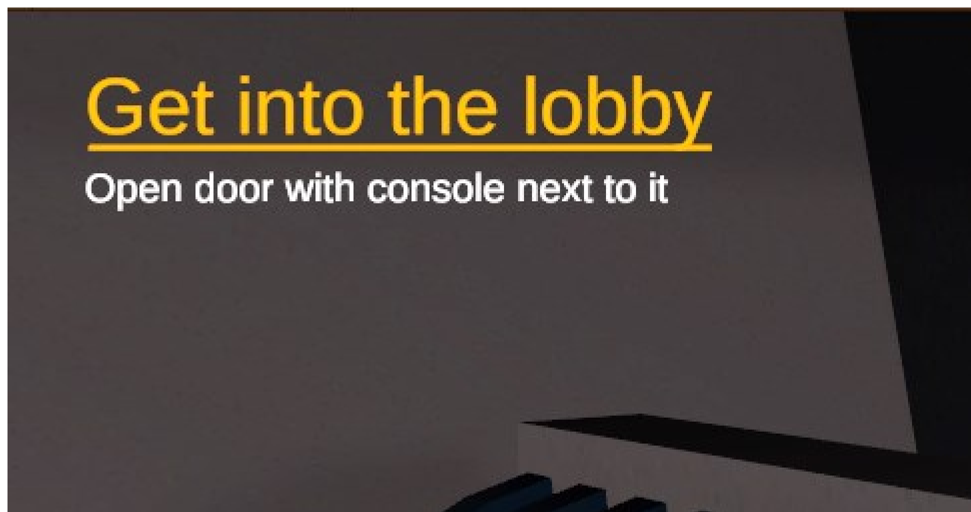
Skript pro menu hry

Veškerou funkcionalitu menu ve hře obstarává skript GameMenu.cs, který byl vytvořen tak, aby řídil zobrazení herního menu a interakce s ním.

Proměnná *gameMenu* odkazuje na objekt s herním menu, které je řízeno tímto skriptem. Metoda *Update()* je volána každý snímek hry a zjišťuje, zda hráč stiskl klávesu *escape*. Pokud ano, kurzor myši je odemčen ze středu obrazovky a je nastaven na viditelný. Následně je samotné menu aktivováno. Další metodou je *Continue()* která se volá, když hráč klikne na tlačítko pokračovat v herním menu. Tato metoda vypíná herní menu a mění stav kurzoru na neviditelný a uzamčený ve středu obrazovky. Poslední napsanou metodou ve skriptu je metoda *SaveAndQuit()*, která je volána, když hráč klikne na tlačítko uložit a ukončit v herním menu. Nejprve se získává instance třídy *SaveLoadManager*. Pokud je tato instance nalezena, herní postup se uloží a aplikace se ukončí.

7.7.3 Rozhraní aktuálního úkolu

Pro lepší orientaci hráče ve hře bylo také do hry přidáno rozhraní (Obrázek 36), které hráči sděluje aktuální úkol, který má splnit. Za tímto účelem byl do scény hry přidán nový objekt *QuestUI* s komponentem *Canvas*, který obsahuje dvě textová pole, jedno pro název úkolu a druhé pro samotný popis úkolu.



Obrázek 36. Aktuální úkol

Skript pro změnu aktuálního úkol

Pro změnu textu aktuálního úkolu bylo potřeba vytvořit nový skript ChangeQuest.cs. Skript se stará o změnu názvu a popisu úkolu ve hře při interakci hráče s určitým objektem.

Proměnné *questTitle* a *questDescription* slouží k zadání názvu a popisku úkolu v komponentu, který obsahuje tento skript. Ve třídě jsou deklarovány textová pole, do kterých se zapisuje název a popis úkolu. První metodou ve skriptu je metoda *Start*(), která inicializuje textové pole názvu a popisu úkolu. Další metodou je *OnTriggerEnter(Collider other)*, která se spouští, když hráč vstoupí do zóny objektu se skriptem. Pokud hráč vstoupí do zóny, metoda nastaví textová pole názvu a popisu úkolu podle hodnoty proměnných *questTitle* a *questDescription*. Poté se zničí objekt se skriptem, aby se úkol nemohl nastavit podruhé. Poslední metoda *ChangeQuestText()* slouží k okamžité změně textu úkolu bez nutnosti interakce hráče se zónou objektu a je volána v rámci ostatních skriptů.

7.7.4 Konec hry

Když hráč splní všechny úkoly, kde je potřeba sestavit algoritmus, je hráči aktualizován úkol, aby se vrátil do postele, kde se probudil na začátku hry. Poté co hráč dojde na místo, kde hra končí, vrátí se zpět do hlavního menu. Pro ukončení hry bylo vytvořeno nové rozhraní v objektu *EndUI*, do objektu byl přidán komponent *Canvas*, který obsahuje tmavé pozadí a text oznamující konec hry. Po 3 sekundách, po zobrazení rozhraní pro konec hry, se hra ukončí a vrátí hráče zpět do hlavního menu.

Skript pro konec hry

Pro tuto funkcionalitu byl vytvořen skript EndGame.cs, který se stará o končení hry a zobrazení rozhraní po splnění podmínky. Na začátku třídy se deklaruje proměnná *endUI* pro objekt rozhraní a proměnná *questTitle*, která slouží pro porovnání názvu aktuálního úkolu. První napsanou metodou byla metoda *Awake()*, která inicializuje *questTitle* nalezením objektu s názvem "QuestName" a získáním jeho komponentu *TMP_Text*. Hlavní metoda skriptu je *OnTriggerEnter(Collider other)*, která se spouští, když hráč vstoupí do zóny objektu, který má tento skript. Když hráč vejde do zóny a aktuální název úkolu se shoduje s požadovaným názvem pro ukončení hry, aktivuje se rozhraní pro ukončení hry a spustí se koprogram *ReturnToMainMenu()*, který čeká tři sekundy a poté načte hlavní menu hry.

7.8 Zvuk

Pro přidání zvuků do hry byl využit komponent *AudioSource*., který slouží k přehrání přiřazeného zvuku.

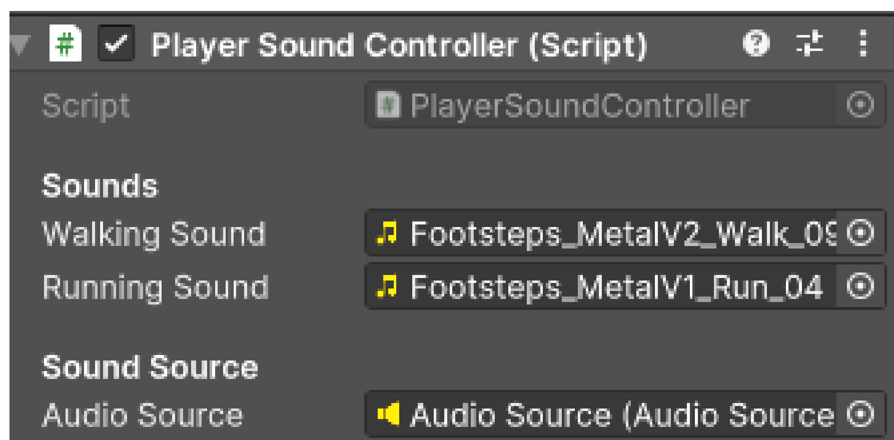
Do hry byl přidán zvuk pro otevření a zavření dveří, pro tento účel byl použit volně dostupný zvuk otevírání dveří[31]. Pro přehrání zvuku byl ke každým dveřím, které obsahují skript *DoorController,* přidán komponent *AudioSource* s tímto zvukem. Zvuk byl následně v komponentu ztlumen, aby nebyl moc hlasitý. Skript *DoorController* pak tento komponent nalezne a přehraje ho pokaždé, když se otevřou nebo zavřou dveře

Zvuk v pozadí byl vytvořen za pomoci volně dostupného zvuku „hlasitého ticha“[32]. Ten byl vložen do komponentu *AudioSource* v objektu *Ambient* a jelikož byl zvuk moc hlasitý, bylo potřeba ho v nastavení komponentu snížit na pět procent původní hlasitosti. Zvuk se přehrává od načtení scény hry a po skončení se opakuje tak, aby nikdy neskončil.

Do rozhraní pro tvorbu algoritmu byl přidán zvuk pro interakci s tlačítky v rozhraní. Pro tento zvuk byl vybrán volně dostupný efekt stisknutí tlačítka[33]. Efekt se přehraje pokaždé, když hráč klikne na tlačítko pomoci, ukončení rozhraní a při spuštění bloku *Execute*. Pro tuto funkcionalitu bylo potřeba přidat komponent *AudioSource* do prefabu *BlockConnector*, který obsahuje rozhraní pro tvorbu algoritmu. Tento komponent je následně volán ve funkci *OnClick()* každého tlačítka, které má přehrát daný efekt při kliknutí.

7.8.1 Zvuk pohybu hráče

Pro pohyb hráče byly použity zvuky z volně dostupného balíčku zvuků pro kroky na různém povrchu[34]. Z tohoto balíčku byl vybrán zvuk chůze a běhu po železném povrchu. Tyto zvuky byly následně přiřazeny do skriptu (Obrázek 37) PlayerSoundController.cs, který byl vytvořen tak, aby přehrával zvuk podle pohybu hráče. Do komponentu byl také přiřazen *AudioSource*, který se nachází v objektu *Player*



Obrázek 37. Skript PlayerSoundController v Editoru

Skript pro přehrání zvuku postavy

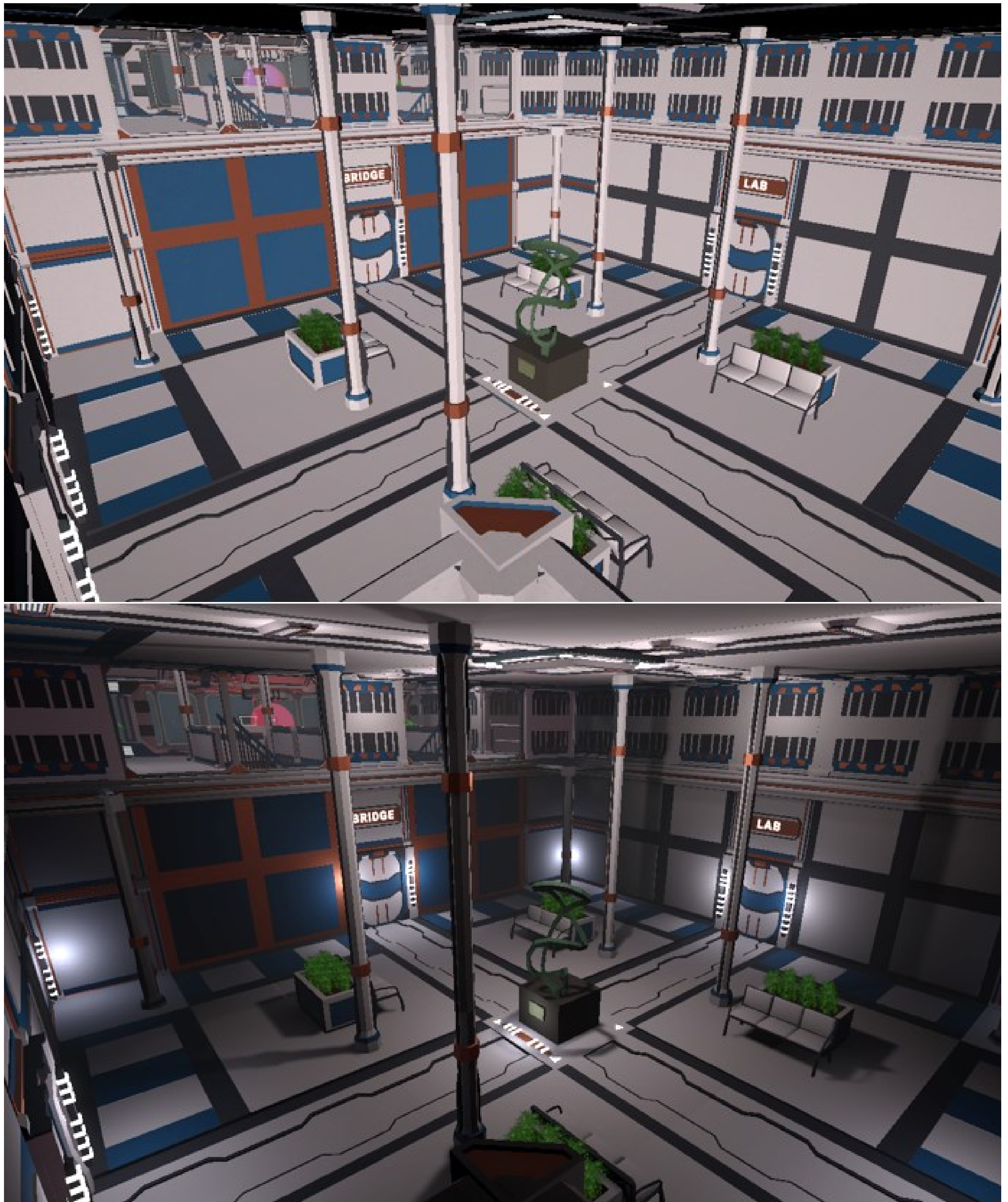
Skript *PlayerSoundController* řídí přehrávání zvuků ve hře v závislosti na pohybu hráče. Proměnné *walkingSound* a *runningSound* obsahují zvuky chůze a běhu hráče. Proměnná *audioSource* určuje zdroj zvuku, pomocí kterého se zvuky přehrávají.

Skript obsahuje metodu *Awake()*, která hledá a inicializuje objekt *PlayerInputs*. V metodě *Update()*, která se volá každý snímek, se kontroluje, zda se hráč pohybuje a zda není přehráván zvuk. Pokud se hráč pohybuje a zvuk není přehráván, zjistí metoda, zda hráč běží. Pokud ano, přehraje se zvuk běhu. Pokud se hráč pohybuje, ale neběží, přehraje se zvuk chůze. Metoda *PlaySound(AudioClip clip)* slouží k přehrání tohoto zvuku.

7.9 Světlo

Pro zlepšení vzhledu herního prostředí a vylepšení atmosféry vesmírné lodi byly do jednotlivých místností přidány objekty světel (Obrázek 38) s komponentem *Light*, který simuluje efekt světla. Pro každou místnost byla použita různá nastavení komponentu, podle potřeby síly osvětlení.

Do scény bylo následně přidáno nastavení světel za pomoci okna v editoru *Lighting*. Toto nastavení se postaralo o vykreslení jednotlivých světel a jejich stínů ve hře. V nastavení byl do možnosti *Enviroment* vybrán materiál pro *skybox*, který představuje prostor mimo vesmírnou loď, který lze vidět skrz okna lodi. Pro tento materiál byl použit volně dostupný materiál[28]. Pro nastavení světla byl také vytvořen nový světelný objekt *Directional Light*, který reprezentuje vzdálený zdroj světla. Toto světlo bylo následně přidáno do možnosti *Sun Source* v nastavení světla. Samotné hodnoty v nastavení byly následně upraveny pro dosažení nejlepšího efektu. Po výběru hodnot a přiřazení proměn bylo světlo vykresleno.



Obrázek 38. Porovnání prostředí bez světel a se světly

8 OPTIMALIZACE A TESTOVÁNÍ

Pro snížení nároků na grafický procesor byla nastavena maximální vzdálenost vykreslování objektů pro kameru na hodnotu sedmdesát pět, což je nejnižší možná hodnota, která šla nastavit, aby objekty nemizely z pohledu hráče. Nejvíce k optimalizaci výkonu hry však pomohlo vykreslování jednotlivých světel do scény, které snížilo zátěž na grafický procesor. Díky vykreslování světel byla dosažena vyšší plynulost hry. Vykreslení světel bylo vytvořeno za pomoci nastavení pro světlo v Unity editoru.

Dále byly upraveny i jednotlivé skripty, ve kterých byly odstraněny zbytečné řádky a přidány komentáře pro lepší orientaci v kódu hry. Na závěr proběhla kontrola a optimalizace jednotlivých úkolu při které se odstranily přebytečné bloky, které nebyly potřeba pro splnění úkolu. Veškeré testování finální hry probíhalo na počítačích s operačním systémem Windows 11.

8.1 Problémy při vývoji

První velký problém nastal při testování mechaniky tvorby algoritmu. Algoritmus šel sestavit za použití jednotlivých bloků a šel spustit, ale při vypnutí rozhraní se program ukončil a nedokončil tak požadovanou akci. Problém nastal, jelikož skript pro vykonávání akce, byl volán z bloku *Action*. Samotný blok je součásti prefabu *BlockConnector*, který se při vypnutí rozhraní vypnul. Tím se vypnul i samotný skript, který tak nemohl pokračovat. Pro vyřešení tohoto úkolu proběhly úpravy ve vypínání samotného rozhraní, kde místo vypnutí celého prefabu se pouze nevykreslí na Canvas. Tím zůstaly veškeré skripty načteny a mohly interagovat s prostředím, i když bylo rozhraní schované. Do skriptu CallUI.cs však musela být ještě přidána metoda pro schování a znovu načtení již vykreslených propojení mezi bloky. A metoda pro vypnutí a zapnutí možnosti vložení textu do vstupního pole.

Dalším nalezeným problémem bylo, když hráč vytvořil nekonečnou smyčku, která pak následně zasekla celou hru. Problém byl v tom, že bloky neustále volaly následující blok a zasekly tak celou hru, proto nešel zavolat žádný jiný skript. Tento problém byl vyřešen přidáním prodlevy při volání následujícího bloku. Tímto krokem se problém vyřešil a zároveň vylepšil herní zážitek, jelikož hráč stihne zaregistrovat právě probíhající blok a jeho akci. Toto řešení však není úplně ideální, protože přidává další menší problém, který může nastat, když je potřeba vytvořit složitější algoritmus, který má větší časovou náročnost. Algoritmus by pak trval příliš dlouho.

9 NÁVRH BUDOUCÍHO VÝVOJE

Hlavním cílem pro budoucí vývoj je nalezení zbylých chyb a nedostatků. Ty je potřeba následně opravit a upravit skript tak aby se neopakovaly. Mezi tyto nedostatky se řadí i aktuální systém pro ukládání a načítání hry, který nefunguje tak, jak by měl, a občas nenačte jednotlivé bloky zpět do pozice, ve které byly uloženy.

Dalším důležitým cílem je vytvoření více místností s jednotlivými úkoly, které hráč bude plnit. Také je potřeba provést změnu seřazení úkolů a přidat více úkolů, které budou hráče seznamovat s jednotlivými bloky pomaleji než v současné podobě hry, kde je náročnost pro druhý úkol podstatně těžší než pro první úkol, který seznamuje hráče se základní mechanikou propojování bloků a stavění algoritmů.

Samotné rozhraní pro tvorbu programů by bylo také dobré vylepšit o přívětivější grafiku, která by zapadala do stylu vesmírné lodi.

Hra má momentálně podporu pouze pro Windows, ale díky možnostem herního enginu Unity je možné hru rozšířit na více platforem.

10 ZHODNOCENÍ NAPLNĚNÝCH CÍLŮ PRÁCE

Hlavním cílem této diplomové práce bylo vytvoření interaktivního softwaru pro výuku Algoritmizace. Prvním krokem byla informační rešerše, která shrnuje dosažené výsledky v oblasti GBL softwaru pro výuku algoritmizace a programování. Tato rešerše je popsána ve třetí kapitole teoretické části práce a obsahuje popis GBL, jeho výhody, možné oblasti využití a využití GBL pro výuku algoritmizace.

Dalším bodem bylo porovnání hlavních technologií využívaných pro tvorbu her a zvolení vhodné technologie pro vývoj hry. Ve čtvrté kapitole teoretické části práce byly popsány nejpoužívanější herní technologie pro vývoj her. Tyto technologie následně byli porovnány v paté kapitole praktické části práce a na základě analýzy byla vybrána vhodná technologie pro vývoj softwaru.

Herní koncept softwaru byl popsán v šesté kapitole praktické části práce a popisuje prostředí, ovládání, úkoly a cíl samotné hry. Následně byl v sedmé kapitole popsán samotný postup při tvorbě hry. V této kapitole byly podrobně popsány všechny vytvořené skripty a jejich mechanika v samotné hře.

Ve vytvořené hře byla následně testována funkcionalita samotné hry za účelem nalezení nedostatků hratelnosti a logických chyb a zlepšení optimalizace. Postup a největší nalezené problémy byly popsány v osmé kapitole praktické části práce.

Posledním bodem práce bylo zhodnocení vytyčených cílů a navržení směru budoucího vývoje softwaru. Návrh budoucího vývoje hry byl popsán v deváté kapitole.

Vytvořená hra byla z projektu vyexportována tak aby mohla být spuštěna přes exe souboru. Tento soubor se nachází ve složce „Hra“, která je umístěna v přílohách práce. Samotný projekt je umístěn ve složce „HraAlgoritmizace“ v příloze práce.

ZÁVĚR

Hlavním cílem diplomové práce bylo vytvořit software pro interaktivní výuku algoritmizace. Za tímto účelem byla vytvořena hra, ve které hráč sestavuje různé algoritmy, a následně pozoruje jejich chování v herním prostředí.

K naplnění tohoto cíle bylo nejprve potřeba provést informační rešerši shrnující dosažené výsledky v oblasti game-based learning softwaru pro výuku algoritmizace a programování. V teoretické části práce byl proto nejprve popsán samotný princip algoritmizace a také možný vliv výukového softwaru pro výuku algoritmizace. Dále byl v teoretické části popsán výukový software a jeho vývoj a historie. Také jsou zde popsány samotné výhody využití výukového softwaru pro výuku algoritmizace. Další sepsanou kapitolou byla kapitola pro game-based learning software, ve které byl zmíněn jeho koncept, možnosti a oblasti jeho využití, výhody a výzvy, kterým tento software může čelit. V této kapitole byly také popsány možnosti výuky algoritmizace za pomoci vizuálního programování, které bylo v praktické části zvoleno jako optimální cesta pro hlavní mechaniku hry. Poslední popsanou kapitolou v teoretické části je kapitola popisující populární technologie pro tvorbu her.

Začátek praktické části byl věnován pro porovnání popsaných herních technologií a zvolení vhodné technologie podle výsledků porovnání. V další kapitole byly formulovány jednotlivé herní koncepty vytvořeného softwaru. V následující kapitole byl popsán kompletní postup při tvorbě samotného softwaru v editoru Unity. Zde byl kladen důraz hlavně na samotné herní mechaniky a jednotlivé skripty, které bylo potřeba vytvořit pro požadovaný průběh hry. Další popsanou částí bylo testování a optimalizace hry, která popisuje postupy pro zlepšení běhu softwaru a největší problémy při vývoji, které bylo potřeba vyřešit. Následuje část, ve které jsou návrhy a doporučení pro budoucí vývoj aplikace. V poslední části se popisuje a vyhodnocuje splnění vytyčených cílů práce.

Výsledkem práce je funkční hra s interaktivními prvky pro výuku algoritmizace, která se dá použít při výuce algoritmizace a pro vyzkoušení znalostí programování. Samotná aplikace seznamuje hráče se základy tvorby algoritmů. Aplikace byla vytvořena tak, aby byla připravena pro budoucí úpravy a případné rozšíření.

SEZNAM POUŽITÉ LITERATURY

[1] What You Need To Know About Educational Software, 2017. Online. *Elearningindustry*. Dostupné z: https://elearningindustry.com/need-know-educational-software. [cit. 2024-02-28].

[2] KAPP, Karl M.; BLAIR, Lucas a MESCH, Rich. *The Gamification of Learning and Instruction Fieldbook: Ideas into Practice.* One Montgomery Street, Suite 1200, San Francisco, CA 94104-4594: Wiley, 2014. ISBN 9781118674437.

[3] HOLAN, Tomáš. *Unity: první seznámení s tvorbou počítačových her*. CZ.NIC. Praha: CZ.NIC, z.s.p.o., 2020. ISBN 978-80-88168-57-7.

[4] CORMEN, Thomas H.; LEISERSON, Charles Eric; RIVEST, Ronald L. a STEIN, Clifford. *Introduction to algorithms.* Fourth edition. Cambridge, Massachusetts: The MIT Press, 2022. ISBN 978-026-2046-305.

[5] SCHELL, Jesse. *The Art of Game Design: A Book of Lenses.* 30 Corporate Drive, Suite 400, Burlington, MA 01803, USA: Elsevier, 2008. ISBN 978-0-12-369496-6.

[6] SEDGEWICK, Robert a WAYNE, Kevin Daniel. *Algorithms.* Fourth edition. Boston: Addison-Wesley, 2011. ISBN 978-0-321-57351-3.

[7] *Unity User Manual.* Online. 2023. Dostupné z: https://docs.unity3d.com/Manual/index.html. [cit. 2023-11-08].

[8] PLATO computer-based education system, 2015. Online. In: Britannica. Dostupné z: https://www.britannica.com/topic/PLATO-education-system. [cit. 2024-02-28].

[9] The forgotten software that inspired our modern world, 2019. Online. In: Britannica. Dostupné z: https://www.bbc.com/future/article/20190722-the-apple-software-that-inspired-the-internet. [cit. 2024-02-28].

[10] BATES, A.W. Tony, 2015. *Teaching in a Digital Age: Guidelines for Designing Teaching and Learning*. BCcampus. ISBN 9780995269255.

[11] OSHANOVA, N.; ANUARBEKOVA, G.; SHEKERBEKOVA, S. a ARYNOVA, G., 2019. *Algorithmization and Programming Teaching Methodology in the course of Computer Science of Secondary School*. Online. Dostupné z: http://www.journal.acce.edu.au/index.php/AEC/article/view/189. [cit. 2024-02-28].

[12] SANCHEZ, Diana R., 2022. *Videogame-Based Learning: A Comparison of Direct and Indirect Effects across Outcomes*. Online. Dostupné z: https://www.mdpi.com/2414-4088/6/4/26. [cit. 2024-02-28].

[13] BROLL, Briann; LÉDECZI, Akos; VOLGYESI, Peter; SALLAI, Janos; MAROTI, Miklos et al., 2017. *A Visual Programming Environment for Learning Distributed Programming.* Online. Dostupné z: https://dl.acm.org/doi/abs/10.1145/3017680.3017741. [cit. 2024-02-28].

[14] AL-AZAWI, Rula; AL-FALITI, Fatma a AL-BLUSHI, Mazin, 2016. *Educational Gamification Vs. Game Based Learning: Comparative Study*. Online. International Journal of Innovation, Management and Technology. S. 131-136. ISSN 20100248. Dostupné z: https://doi.org/10.18178/ijimt.2016.7.4.659. [cit. 2024-02-28].

[15] James Gee's Principles For Game Based Learning, 2016. Online. *In: Legendsof-learning*. Dostupné z: https://www.legendsoflearning.com/blog/james-paul-gee-game-based-learning/. [cit. 2024-02-28].

[16] VIDENOVIK, Maja; VOLD, Tone; KIØNIG, Linda; BOGDANOVA, Ana Madevska a TRAJKOVIK, Vladimir, 2023. Game-based learning in computer science education: a scoping literature review. Online. *International Journal of STEM Education*. Roč. 10, article 54, s. 1-23. Dostupné z: https://doi.org/10.1186/s40594-023-00447-2. [cit. 2024-05-02].

[17] DYNAMO PRIMER, 2021. *Co je vizuální programování?* Online. Dynamo. Dostupné z: https://primer.dynamobim.org/cs/01_Introduction/1-1_what_is_visual_programming.html. [cit. 2024-05-02].

[18] RASHID, Mohammad Mamunur a ISLAM, Md. Nazrul, 2014. Visual Programming. In: *Visual Programming*. Bangladesh Open University: Publishing, Printing and Distribution Division Bangladesh Open University, s. 1-17. ISBN 978-984-34-4030-3.

[19] LU, Zhuotao; CHIU, Ming Ming; CUI, Yunhuo; MAO, Weijie a LEI, Hao, 2022. Effects of Game-Based Learning on Students' Computational Thinking: A Meta-Analysis. Online. *Journal of Educational Computing Research*. Roč. 61, č. 1, s. 235-256. Dostupné z: https://doi.org/10.1177/07356331221100740. [cit. 2024-05-02].

[20] PELLAS, Nikolaos a MYSTAKIDIS, Stylianos, 2020. A Systematic Review of Research about Game-based Learning in Virtual Worlds. Online. *JOURNAL OF*

UNIVERSAL COMPUTER SCIENCE. Roč. 26, č. 8, s. 1-26. Licence: CC BY-ND 4.0. Dostupné z: https://doi.org/10.3897/jucs.2020.054. [cit. 2024-05-02].

[21] *Unreal Engine 5.4 Documentation*, 2024. Online. Dostupné z: https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-engine-5-4-documentation. [cit. 2024-05-02].

[22] *Unreal Editor Interface*, 2024. Online. Dostupné z: https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-editor-interface. [cit. 2024-05-02].

[23] ANDRADE, A., 2015. Game engines: a survey. Online. *EAI Endorsed Transactions on Game-Based Learning*. 2015-11-05, roč. 2, č. 6, s. 10.4108/eai.5-11-2015.150615. ISSN 2034-8800. Dostupné z: https://doi.org/10.4108/eai.5-11-2015.150615. [cit. 2024-05-02].

[24] SOBOTA, Branislav a PIETRIKOVÁ, Emília, 2023. The Role of Game Engines in Game Development and Teaching. Online. *Computer Science for Game Development and Game Development for Computer Science*. 2023-10-11, s. 26. ISBN 978-1-83769-733-5. Dostupné z: https://doi.org/10.5772/intechopen.1002257. [cit. 2024-05-02].

[25] *Godot Docs*, 2014. Online. Dostupné z: https://docs.godotengine.org/en/stable/. [cit. 2024-05-02].

[26] *First look at Godot's editor*, 2014. Online. Godot Docs. Dostupné z: https://docs.godotengine.org/en/stable/getting_started/introduction/first_look_at_the_editor.html. [cit. 2024-05-02].

[27] *Sci-Fi Styled Modular Pack*, 2018. Online. Unity Asset Store. Dostupné z: https://assetstore.unity.com/packages/3d/environments/sci-fi/sci-fi-styled-modular-pack-82913. [cit. 2024-05-02].

[28] *Starfield Skybox*, 2017. Online. Unity Asset Store. Dostupné z: https://assetstore.unity.com/packages/2d/textures-materials/sky/starfield-skybox-92717. [cit. 2024-05-02].

[29] *GIMP*, 2024. Online. Dostupné z: https://www.gimp.org/. [cit. 2024-05-02].

[30] *Reset Icon*, 2024. Online. ICONS8. Dostupné z: https://icons8.com/icon/12491/reset. [cit. 2024-05-02].

[31] *Spaceship Door Opening*, 2022. Online. Pixabay. Dostupné z: https://pixabay.com/sound-effects/spaceship-door-opening-43604/. [cit. 2024-05-02].

[32] *Loud Silence*, 2022. Online. Pixabay. Dostupné z: https://pixabay.com/sound-effects/loud-silence-75803/. [cit. 2024-05-02].

[33] *Button*, 2022. Online. Pixabay. Dostupné z: https://pixabay.com/sound-effects/button-124476/. [cit. 2024-05-02].

[34] *Footsteps – Essentials*, 2022. Online. Unity Asset Store. Dostupné z: https://assetstore.unity.com/packages/audio/sound-fx/foley/footsteps-essentials-189879. [cit. 2024-05-02].

[35] *Cinemachine package*, 2024. Online. Unity. Dostupné z: https://docs.unity3d.com/Packages/com.unity.cinemachine@3.1/manual/index.html. [cit. 2024-05-02].

[36] *Scratch*. Online. Dostupné z: https://scratch.mit.edu/. [cit. 2024-05-02].

[37] *Alice*, 2020. Online. Dostupné z: http://www.alice.org/about/. [cit. 2024-05-02].

[38] *Blockly Games*. Online. Dostupné z: https://blockly.games/. [cit. 2024-05-02].

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

GBL	Game-based learning
UI	Uživatelské rozhraní
2D	Dvourozměrný prostor
3D	Trojrozměrný prostor
EXE	Spustitelný soubor

SEZNAM OBRÁZKŮ

Obrázek 1. Vizualizace programu [17]...20
Obrázek 2. Defaultní rozhraní Unity [7]...24
Obrázek 3. Defaultní rozhraní Unreal Editoru [22]...25
Obrázek 4. Defaultní rozhraní Godot enginu [26]...26
Obrázek 5. Vytvoření Projektu...32
Obrázek 6. Třída Variable...34
Obrázek 7. Metody Awake() a Execute() ve třídě Block...35
Obrázek 8. Metoda Execute() ve třídě DeclarationBlock...36
Obrázek 9. Metoda CheckCondition() ve třídě IfBlock...38
Obrázek 10. Metoda Execute() ve třídě LoopBlock...39
Obrázek 11. Metoda PerformOperation() ve třídě MathBlock...40
Obrázek 12. Vytvoření struktury pro událost...41
Obrázek 13. Metoda ConnectBlocks() ve třídě BlockConnector...42
Obrázek 14. Třída BlockConnection...43
Obrázek 15. Metody OnDrag() a OnEndDrag() ve třídě BlockDragDrop...44
Obrázek 16. Struktura ukládaných data...46
Obrázek 17. Rozhraní pro stavbu algoritmu...47
Obrázek 18. Bloky...48
Obrázek 19. Nápověda k úkolu...50
Obrázek 20. Nápověda pro bloky...51
Obrázek 21. Hodnoty *PlayerController* v editoru...52
Obrázek 22. Mapování kláves pro pohyb...53
Obrázek 23. Začáteční místnost...54
Obrázek 24. Nápověda pro rozhraní...55
Obrázek 25. Vstupní místnost...55
Obrázek 26. Algoritmus pro navýšení hladiny kyslíku...56
Obrázek 27. Hlavní hala lodi...57
Obrázek 28. Strojovna lodi...58
Obrázek 29. Algoritmus pro zapnutí motoru...59
Obrázek 30. Můstek lodi...60
Obrázek 31. Algoritmus pro nalezení trasy...61
Obrázek 32. Laboratoř...62

Obrázek 33. Algoritmus pro seřazení planet 63
Obrázek 34. Hlavní menu hry...64
Obrázek 35. Menu ve hře...66
Obrázek 36. Aktuální úkol..67
Obrázek 37. Skript PlayerSoundController v Editoru 69
Obrázek 38. Porovnání prostředí bez světel a se světly..71

SEZNAM TABULEK

Tabulka 1. Srovnávací tabulka herních enginů..30

SEZNAM PŘÍLOH

Příloha P I – Obsah DVD

Příloha P II – Část kódu Block.cs

Příloha P III – Část kódu Block.cs

PŘÍLOHA P I: OBSAH DVD

- fulltext.pdf
- \prilohy
    - \Hra (Build hry)
        - Hra.exe
    - \HraAlgoritmizace (Projekt Unity)

PŘÍLOHA P II: ČÁST KÓDU BLOCK.CS

```
//Metoda pro nalezení hodnoty proměnné
Počet odkazů: 5
protected object FindValue(string variable)
{
    foreach (Variable value in variables)
    {
        if (value.name == variable)
        {
            return value.value;
        }
    }
    return 0;
}

//Ověřuje, zda proměnná existuje
Počet odkazů: 8
protected bool IsVariable(string variable)
{
    foreach (Variable value in variables)
    {
        if (value.name == variable)
        {
            return true;
        }
    }
    return false;
}

//Uloží hodnotu do proměnné
Počet odkazů: 3
public void SaveValue(string variable, object valueToSave)
{
    foreach (Variable value in variables)
    {
        if (value.name == variable)
        {
            value.value = valueToSave;
        }
    }
    return;
}

//Najde a vrati hodnotu dané proměnné
Počet odkazů: 7
public object GetValueByName(string variable)
{
    foreach (Variable value in variables)
    {
        if (value.name == variable)
        {
            return value.value;
        }
    }
    return 0;
}
```

PŘÍLOHA P III: ČÁST KÓDU BLOCK.CS

```
//Načtení dat bloku
Počet odkazů: 13
public virtual void SetBlockData(BlockData data)
{
    if (data != null)
    {
        blockID = data.blockID;
        blockRectTransform.anchoredPosition = data.position.ToVector2();
        inLoop = data.inLoop;
    }

    if (data.nextBlockID != -1)
    {
        nextBlock = FindBlockByBlockID(data.nextBlockID);
    }
    else
    {
        nextBlock = null;
    }

}

//Najde blok podle ID
Počet odkazů: 1
protected Block FindBlockByBlockID(int blockID)
{
    //Pokud se blok shoduje vrátí svoji instanci
    if (this.blockID == blockID)
    {
        Debug.Log("Next Block with ID: " + blockID + " was found");
        return this;
    }

    //Hledá blok v potomcích
    foreach (Transform child in transform)
    {
        Block childBlock = child.GetComponent<Block>();
        if (childBlock != null && childBlock.blockID == blockID)
        {
            Debug.Log("Block with ID: " + blockID + " was found");
            return childBlock;
        }
    }
    return null;
}

//Hledá objekt v potomcích s daným tagem
Počet odkazů: 3
private static GameObject FindChildWithTag(GameObject parent, string tag)
{
    Transform t = parent.transform;

    for (int i = 0; i < t.childCount; i++)
    {
        if (t.GetChild(i).gameObject.tag == tag)
        {
            return t.GetChild(i).gameObject;
        }

    }

    return null;
}
```