UNIVERZITA PARDUBICE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

BAKALÁŘSKÁ PRÁCE

2024 Petr Vávra

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

3D stavební simulátor pro kutily

Bakalářská práce

2024 Petr Vávra

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2022/2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Petr Vávra**
Osobní číslo:	**I20167**
Studijní program:	**B0688A140009 Informační technologie**
Téma práce:	**3D stavební simulátor pro kutily**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování

Cílem bakalářské práce je vytvoření 3D simulátoru pro návrh kutilských stavebních projektů. V aplikaci bude možné z pohledu první osoby stavět z vybraných materiálů různé struktury. Výstupem z projektu bude soupis materiálu a stavební výkresy.

Rozsah pracovní zprávy: **cca 30 stran**
Rozsah grafických prací:
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

Jiří Žára, Bedřich Beneš, Jiří Sochor, Per Felkel, Moderní počítačová grafika, Computer Press, Brno, ISBN 80-251-0454-0

Tomáš HolanEdice. UNITY První seznámení s tvorbou počítačových her, CZ.NIC, z. s. p. o., Praha 2020, ISBN 978-80-88168-60-7

WATT, Alan. 3D computer graphics. 3rd ed. New York: Pearson Education, 2000. ISBN 0-201--39855-9.

LINOWES, Jonathan. Unity virtual reality projects: explore the world of virtual reality by building immersive and fun VR projects using Unity 3D. Birmingham, UK: Packt Publishing, 2015. ISBN 978-1-78398-855-6.

Vedoucí bakalářské práce:
- **Ing. Martin Pozdílek, Ph.D.**
  Katedra informačních technologií

Datum zadání bakalářské práce: **16. prosince 2022**
Termín odevzdání bakalářské práce: **12. května 2023**

L.S.

Ing. Zdeněk Němec, Ph.D. v.r.
děkan

Ing. Jan Panuš, Ph.D. v.r.
vedoucí katedry

V Pardubicích dne 28. února 2023

Prohlašuji:

Práci s názvem 3D stavební simulátor pro kutily jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 7/2019 Pravidla pro odevzdávání, zveřejňování a formální úpravu závěrečných prací, ve znění pozdějších dodatků, bude práce zveřejněna prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 4. 4. 2024

Petr Vávra v.r.

PODĚKOVÁNÍ

Rád bych vyjádřil svou upřímnou vděčnost Ing. Martinu Pozdílkovi, Ph.D. za cenné rady a odborné vedení, které mi bylo poskytnuto. Dále bych rád poděkoval své rodině a přátelům za podporu a pochopení při psaní této bakalářské práce.

ANOTACE

Tato práce se zabývá tvorbou počítačové aplikace, která slouží k vytvoření modelu hrubé stavby. Uživatel aplikace je schopný z pohledu první osoby budovat struktury z výběru stavebních materiálů, stavět střechy a podlahy pomocí nástrojů, exportovat půdorys a soupis materiálu. Je vytvořena v Unity v jazyce C#. V teoretické části jsou popsány alternativy stavebních simulátorů a herních enginů. V praktické části je vytvořena aplikace stavebního simulátoru, popsán její návrh a vývoj.

KLÍČOVÁ SLOVA

Stavební simulátor, Unity, C#, 3D, půdorys, soupis materiálu

TITLE

3D construction simulator for DYIers

ANNOTATION

This thesis deals with the development of a computer application that is used to create a model of a rough building. The user of the application is able to build structures from a selection of building materials from a first-person perspective, build roof and floors using tools, and export a floor plan and list of materials. It is developed in Unity in C#. The theoretical part describes alternative building simulators and game engines. The practical part describes the design and development of a building simulator application.

KEYWORDS

Building simulator, Unity, C#, floorplan, list of materials

OBSAH

SEZNAM ILUSTRACÍ A TABULEK

SEZNAM ZKRATEK A ZNAČEK

Prefab	Prefabrikovaný objekt
3DS	3D Studio
AABB	Axis-Alligned Bounding Box
CAD	Computer Aided Design
CNC	Computer Numerical Control
DWG	Drawing
GNU/GPL	GNU General Public Licence
JPG	Joint Photographic Experts Group
OBJ	Object File
PDF	Portable Document Format
PNG	Portable Network Graphics
STL	Stereolitografie
TIF	Tagged Image File Format
UI	User Interface
VR	Virtual Reality

ÚVOD

Tématem bakalářské práce je 3D stavební simulátor pro kutily. Cílem práce je vytvořit počítačovou aplikaci pomocí herního enginu Unity 3D a jazyka C#, která bude formou podobné hře pomáhat kutilům vytvářet hrubé stavby a zjišťovat počty potřebných materiálů – to vše zábavnější a interaktivnější formou než pomocí výpočtů na papíře. Cílem je také navrhnout aplikaci s herními prvky tak, aby se dala jednoduše ovládat za pomocí přiloženého návodu.

Nejdůležitější vlastností aplikace je pohyb hráče a kamery, která bude z pohledu první osoby. Dále implementace výběru stavebních materiálů a entit, které bude možné pokládat do simulátoru. Důležitý je systém přichytávání, který zajišťuje přichycení stavebních bloků k sobě pro snadné vertikální nebo horizontální pokládání. Další vlastností je výstup z aplikace, a to ve formě výpočtu přibližného počtu potřebných materiálů k realizaci stavby, např. objem kvádrové betonové podlahy, počet cihel nebo plocha střechy.

Toto téma zpracovávám z pozice vlastního zájmu, kde potřeba primitivního simulátoru s velmi jednoduchým ovládáním, který ale přináší reálný účinek z hlediska výpočtu materiálu a generování půdorysu mi přišla užitečná a podmětná.

Práce je rozdělena do dvou částí, teoretické a praktické. V teoretické části jsou popsány alternativy stavebních simulátorů, jejich klady a zápory a v poslední alternativě i porovnání s touto prací. Dále je popsán engine Unity 3D společně s jeho alternativami. V praktické části je popsán vývoj práce, s důkladným popisem kódů a vysvětlení funkcionality komponent.

1 TEORETICKÁ ČÁST

1.1 Alternativy stavebních simulátorů

Pro návrh stavebního projektu je možné použít různé typy nástrojů a programů, které se můžou výrazně lišit v způsobu použití. Pro jednoduché návrhy může být lepší volba použít nástroje specializované pro tyto účely, například níže zmíněný SketchUp. Pro návrhy vyžadující větší přesnost lze použít nástroje typu CAD, které ale mohou být komplikovanější na použití.

1.1.1 SketchUp

Sketchup je 3D modelovací software, který umožňuje snadné a intuitivní vytváření trojrozměrných modelů a nákresů. Původně vyvinutý společností @Last Software, v roce 2006 byl odkoupen společností Google, a nakonec v roce 2012 společností Trimble, která platformu vlastní doposud. [1]

Sketchup má široké spektrum aplikací, je vhodný např. pro architekty, truhláře a inženýry, ale je více než vhodný i pro osobní použití. Obsahuje velkou škálu nástrojů, které lze použít pro usnadnění práce, např. vestavěný systém osvětlení, který umožňuje simulovat stíny a jejich změny v čase. Další užitečná komponenta je 3D Warehouse, což je zabudovaná knihovna, která je otevřená uživatelům pro stahování obsahu, ale i jeho nahrávání. [2]

Nabízeny jsou celkem čtyři licenční plány. Sketchup Free, který je zcela zdarma a poskytuje přístup do webové aplikace. Sketchup Go, ten nabízí prémiový přístup do webové aplikace a navíc také aplikaci pro iPad. Třetí plán je Sketchup Pro, v něm je už zahrnuta desktopová aplikace a rozšíření LayOut, který lze použít pro snadné převedení trojrozměrného modelu do 2D a následnou editaci. Nejobsáhlejší plán je SketchUp studio, který obsahuje funkcionality všech předchozích plánů a podporuje generaci realistických výkresů v reálném čase a práci s mračnem bodů v cloudovém prostředí.[3]

Za zmínku určitě stojí možnost programování vlastních rozšíření v jazyce Ruby. Sketchup má v sobě již zabudovaný balíček ruby, včetně vývojové konzole. Existuje knihovna s názvem Trimble Extension Warehouse, což je podobná kolekce jako 3D Warehouse, ale s rozšířeními.[4]

Podporovaná je dlouhá řada formátů a jejich užití je limitováno aktuálním plánem. V případě nejdražšího plánu SketchUp Studio dostaneme na výběr velkou škálu formátů, kde nejdůležitější jsou PNG, JPG, DWG, TIF, STL, 3DS, OBJ a PDF. Důležité je zmínit, že import a export formátu PDF je možné pouze na operačním systému Mac.[5]

Obrázek 1: Ukázka webového rozhraní aplikace SketchUp

1.1.2 FreeCAD

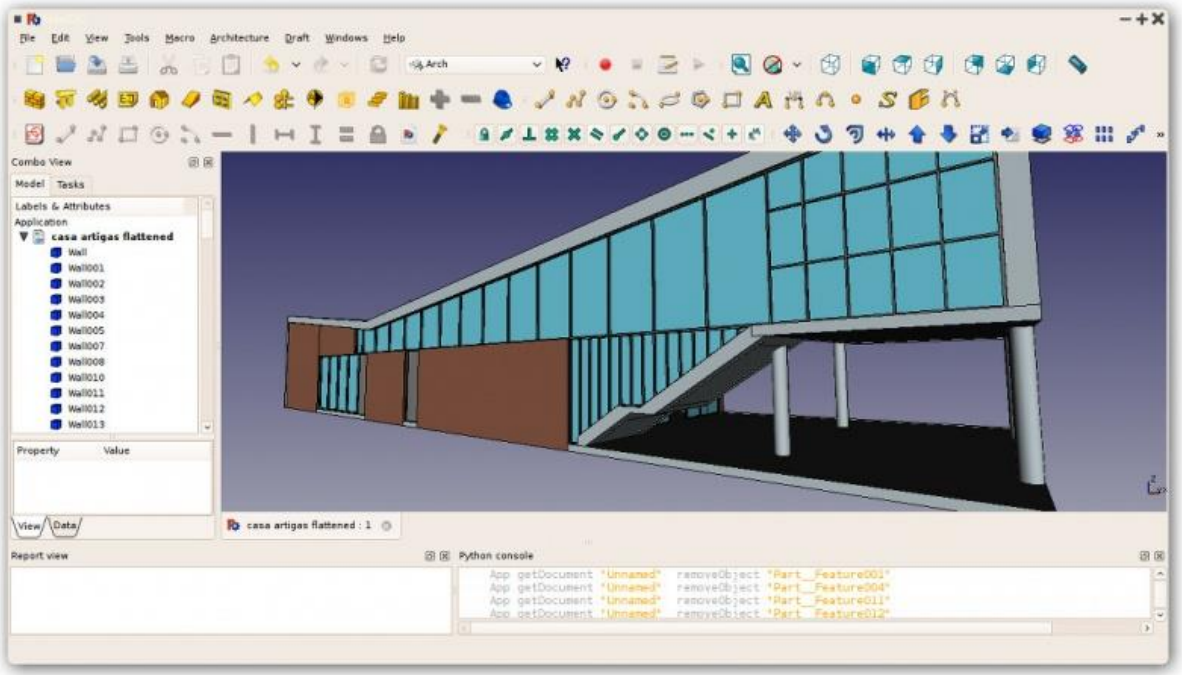
FreeCAD je zdarma 3D modelovací aplikace s otevřeným zdrojovým kódem. Spadá pod licenci LGPL a je zcela zdarma. Jedná se o CAD software s primární zaměřením na vývoj ve strojírenství, ale jeho univerzální funkce z něho dělají dobrou volbu i pro jiné aplikace, jako je např. modelování pro 3D tiskárny, produktový design nebo návrh architektury. FreeCAD začal jako projekt v roce 2001 a od té doby je doplňován a vyvíjen komunitou nezávislých vývojářů.[6]

FreeCAD je možné nainstalovat na většinu běžných systémů, mezi které patří Windows, Linux a Mac OS. Podporuje parametrické modelování, což je modelové paradigma, ve kterém je možné vytvářet části modelu, které na sebe logicky navazují a mohou být definovány funkcemi, které vychází právě z předchozích částí.[7]

Zajímavý koncept, který FreeCAD používá jsou tzv. pracovní plochy. Jedná se o skupiny nástrojů, které sdílí podobné funkcionality. Např. pracovní plocha „Path“ slouží ke generování instrukcí pro CNC stroje, typicky exportuje ve formátu gcode.[8] Existuje pracovní plocha s názvem architektura, kterou lze použít pro vytvoření stavebních modelů. Obsahuje užitečné komponenty pro vytváření zdí, střech, podlah, schodišť a dalších stavebních objektů.[9]

Celkově je FreeCAD velmi univerzální program s mnoha možnostmi, populární díky jeho dostupnosti. Mezi jeho omezení patří špatná optimalizace pro práci s 2D soubory, protože byl navržen primárně jako 3D aplikace. Pokud budeme pracovat s objemnými dvourozměrnými

soubory, můžeme očekávat zhoršení výkonu. Další nevýhodou je špatná podpora, jelikož se jedná o komunitní software. [9]



Obrázek 2: Ukázka stavebního projektu v aplikaci FreeCAD[10]

1.1.3 Floorplanner

Floorplanner je rozmanitý nástroj pro vytváření místností a půdorysů pomocí uživatelsky přívětivého rozhraní. Jeho editor umožňuje rychle a jednoduše vytvářet modely místností, a pomocí 3D zobrazení lze v reálném čase vytvářet reálné vykreslení projektu v rozlišení 7680 x 4320 pixelů. Floorplanner stejně jako ostatní výše zmíněné alternativy obsahuje rozsáhlou knihovnu trojrozměrných objektů, které jsou zdarma pro použití bez nutnosti připlácet za tuto funkcionalitu. Knihovna obsahuje modely dveří, oken a všeobecného nábytku známých značek.[11]

Co se týče licencování, Floorplanner nabízí možnost užívání softwaru zdarma při použití jejich základního účtu, který nabízí již od roku 2007. Nabízí také učitelský účet pro školy a jiné učební instituce, který je speciálně upravený pro edukativní účely. Existují také placené účty, a to dva druhy: Plus a Pro. Každý přidává více funkcionalit do softwaru, jako např. vlastní šablony a odebrání čekací doby při exportu (základní verze umožnuje pouze jeden export projektu za deset minut). Ve Floorplanneru jsou projekty rozděleny do různých tříd, kde každá třída stojí určitý počet tzv. „kreditů“. Standardní projekty mají malé rozlišení a obsahují vodoznak, a při použití kreditů se zvedá kvalita až na maximálních 7680 x 4320 pixelů.[11]

Samotná aplikace Floorplanner běží ve webovém rozhraní, kde podporuje 3D a 2D práci se scénou. Jednoduchý a doporučený přístup je navrhnout místnost z pohledu shora v 2D zobrazení, a poté kontrolovat a doladit v trojrozměrném režimu. [12]

Floorplan nabízí API, které nabízí možnost zaintegrovat zobrazovač přímo do vaší aplikace nebo webu. Na oficiálních stránkách je možné najít rozsáhlou dokumentaci a sbírku příkladů, jak API používat pro vlastní účely. API funguje podobně jaké jiné rozhraní, pomocí tokenů a požadavků pomocí POST, GET a DELETE.[13]

Celkově je Floorplanner oblíbený díky svému intuitivnímu a jednoduchému ovládání, dostupností a možnosti využití jeho služeb zdarma bez jakékoliv nutnosti platby. Hlavní nevýhodou je omezení v bezplatném režimu.[12]

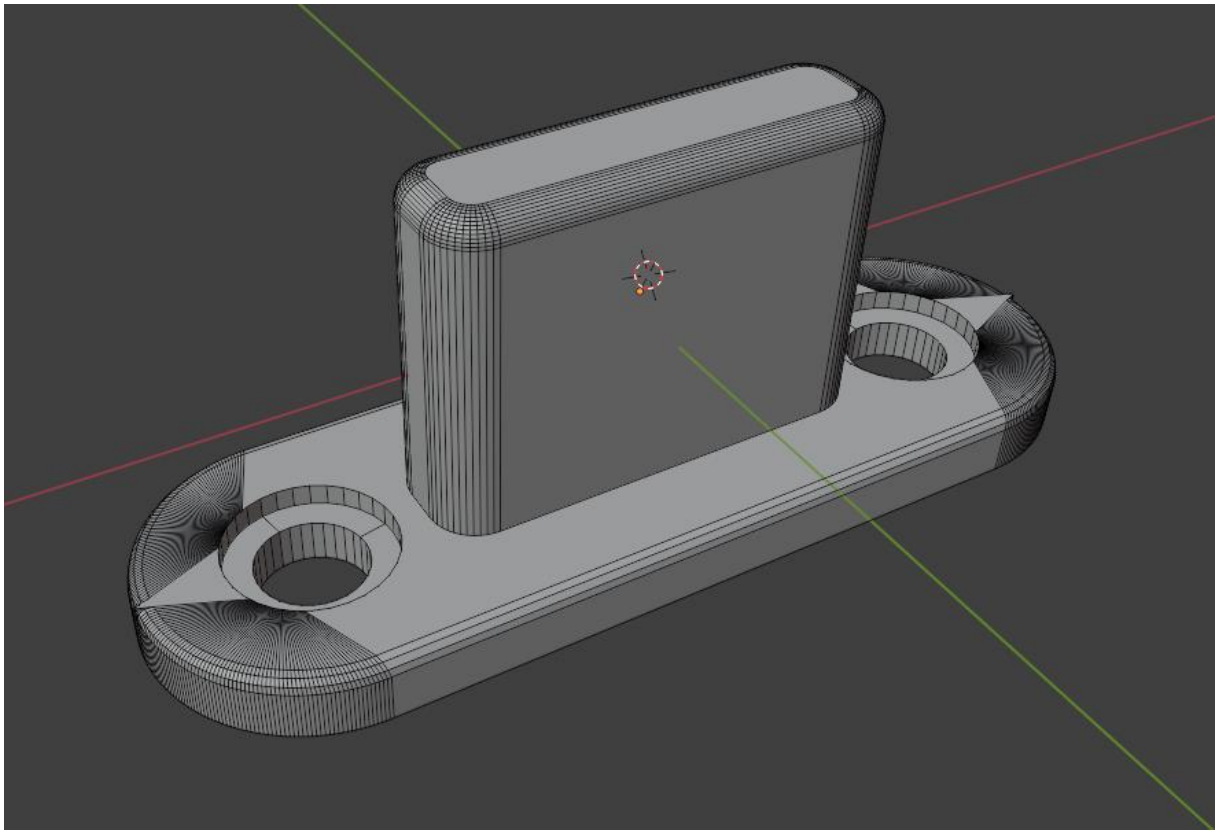
Obrázek 3: Příklad výkresu v aplikaci Floorplanner[14]

1.1.4 Blender

Blender je 3D modelovací software, který je zcela zdarma a volně stažitelný pod licencí GNU GPL. Jeho hlavní funkcionalitou je vytváření trojrozměrných modelů, vizualizací, animací a videí. Je dostupný pro všechny nejpopulárnější systémy, čímž je na mysli Linux, Windows a macOS. Jedná se o mocný a komplexní nástroj, který může vyžadovat určité technické znalosti. Nejedná se o software, který je jednoduchý k použití, pro nové uživatele může být obtížnější se zorientovat a naučit se v něm pracovat. Blender nemusí být tou nejlepší volbou, pokud hledáte jednoduché řešení. [15]

Software Blender obsahuje obsáhlou knihovnu funkcí pro práci s 3D objekty, zejména lze objekty vzájemně vyřezávat, zaoblovat, pracovat se světlem ale také používat fyziku např. pro simulaci tekutin a kolizí pevných objektů. [15]

Pro účely kutilských konstrukcí pro domácí použití může být použití Blenderu, ale ostatně také jiných složitějších 3D modelovacích softwarů přehnané. Osobně bych spíše použil Blender pro vykreslování složitějších tvarů a modelů, např. určených pro 3D tisk.



Obrázek 4: Model vytvořený v aplikaci Blender

1.1.5 Builder Simulator VR

Builder Simulator VR je software od herního vývojáře Epic VR, který se ze všech výše zmíněných aplikací podobá mé bakalářské práci nejvíce. Jak už z názvu vyplývá, celá hra je situovaná ve virtuálním prostředí, takže je nutné vlastnit VR headset, což se stává v době psaní této bakalářské práce čím dál dostupnější technologií. Builder Simulator VR si zakládá na realismu a vtažení hráče do stavby, která se celá realizuje po malých krocích cihla po cihle. Je možné stavět z mnoha materiálů a cihel různých velikostí, vkládání dveří a oken, vylévání podlah a pokládání střech. Výsledkem může být tedy hrubá stavba, stejně jako v praktické části této bakalářské práce. Na rozdíl od této práce ale Builder Simulator VR nenabízí generování půdorysu a ani soupis materiálu s přibližnými cenami. Builder Simulator VR se tedy naklání

spíše ke hře než ke kutilskému simulátoru, jako je např. tato práce. Obě aplikace mají jedno společné, pohled z první osoby. Ten může být velmi užitečný, protože uživateli poskytuje reálnou představu, jak to v postavených prostorech doopravdy vypadá a jak na člověka působí.

1.1.6 Porovnání s aplikací

Ani po rozsáhlé rešerši jsem nebyl schopný najít vydanou aplikaci, která by se funkcionalitou a rozhraním podobala této práci. Tato práce totiž balancuje mezi prvky, které bychom mohli nalézt spíše v herních simulátorech než v profesionálních softwarech, a prvky užitečného rázu, např. generování půdorysů a soupisy použitých materiálů. Je to právě kvůli této zajímavé kombinaci pohledu z první osoby a stavění objektů, které tuto práci dělají jedinečnou, protože většina simulátorů obsahuje volně pohyblivé zobrazovací prostředí, které typicky využívají ortografické nebo perspektivní kamery. Mám za to, že pohled z první osoby nemusí být užitečný pro všechny případy, ale poskytuje intenzivnější a bezprostřednější zážitek simulátoru nebo hry. Také je mnohem intuitivnější a způsob kterým jsem ho naimplementoval v této práci poskytuje jednoduché ovládání celého simulátoru pomocí klávesových zkratek a pohybu kamery myší, což je pro uživatele PC a hráče her velmi přirozený způsob ovládání.

1.2 Unity 3D Game Engine

Pro realizaci své bakalářské práce jsem si vybral herní engine Unity 3D. Jedná se o software typu game engine, což je software, který vývojářům umožňuje jednodušeji vytvářet a programovat hry nebo aplikace. Typicky je nabízena knihovna funkcí specifická pro daný engine, která vývojáři umožňuje přístup k proměnným, které můžou být užitečné pro aplikaci nebo hru, jako je např. práce s herním časem, osvětlením, uživatelským rozhraním, vstupem z myši nebo klávesnice a další. Pro každý skript jsou obvyklé dvě funkce, jedna, která se volá při spuštění skriptu, a druhá, která se volá každý snímek, což vývojáři umožňuje reagovat na herní události nebo vstup uživatele. Pro herní engine je obvykle specifický grafický editor se scénou a zabudovaným herním přehrávačem, kde je možné si hru vyzkoušet nebo spouštět v ladícím režimu. Unity 3D je zdarma pro studenty a jednotlivce, u kterých roční příjem nepřesáhl 100 000 dolarů. Pro firmy a větší organizace jsou již plány placené, kde se ceny pohybují v řádu tisíce dolarů za jednoho uživatele, který software používá.[16]

Unity jsem si vybral z více důvodů. Hlavním důvodem je to, že se softwarem Unity mám již zkušenosti, které jsem získal programováním jednoduchých her. Poměrně slušně se orientuji

v jeho editoru a ovládám jeho funkce. Druhý důvod je programovací jazyk, kde Unity používá pro své obslužné skripty jazyk C#, ve kterém jsem schopný programovat a znám jeho syntaxi. Třetí důvod je jeho dostupnost – jak už jsem výše zmínil, je zdarma pro jednotlivce s výdělkem pod 100 000 dolarů nebo pro studenty, kteří jsou zaregistrování pod akreditovaným programem edukativní instituce.[16] Výhodou Unity je jeho poměrná jednoduchost a vstřícnost pro začátečníky. Díky tomu, že je herní engine Unity velmi rozšířený, lze v internetových pramenech snadno najít velké množství dokumentace, návodů, příkladů a rad. Lze také čerpat z oficiální Unity dokumentace, API, nebo oficiálního fóra.

1.3 Alternativy herních enginů

1.3.1 Unreal Engine

Za vhodnou alternativu herního enginu pro vývoj mé bakalářské práce by určitě šel označit Unreal Engine. Jedná se primárně o nástroj k vytváření her, ale lze ho použít také jako software pro modelování architektury nebo tvorbu scén např. pro účely kinematografie. Jedná se o platformu, která je zdarma, vhodná pro začátečníky, ale i pro profesionály a je považována za standard pro vývoj AAA her.[17]

Pro vývoj se v Unreal Engine využívá programovací jazyk C++, v porovnání s Unity Game Engine, který používá C#. Obecně se dá říct, že C++ je jazyk nízké úrovně a poskytuje menší abstrakci nad hardwarovými prostředky. Na druhou stranu C# je jazyk vyšší úrovně, což znamená, že je jednodušší na pochopení a dal by se označit za lepší volbu pro začátečníky. Když budeme pokračovat v porovnání s Unity, Unreal Engine vyžaduje platbu 5% podílů z výdělku, pokud hra nebo produkt generuje více než 3000 dolarů za kvartální období.[17]

Stejně jako Unity Game Engine nabízí Unreal Game Engine on-line tržiště s komponenty, které je možné využít pro prodej nebo koupi modelů. Tyto komponenty poté možné použít pro vývoj her nebo aplikací. Pokud bych neměl zkušenosti s vývojem v Unity, určitě bych označil Unreal Engine jako silného kandidáta při volbě, v jakém nástroji bakalářskou práci uskutečnit.

1.3.2 Godot

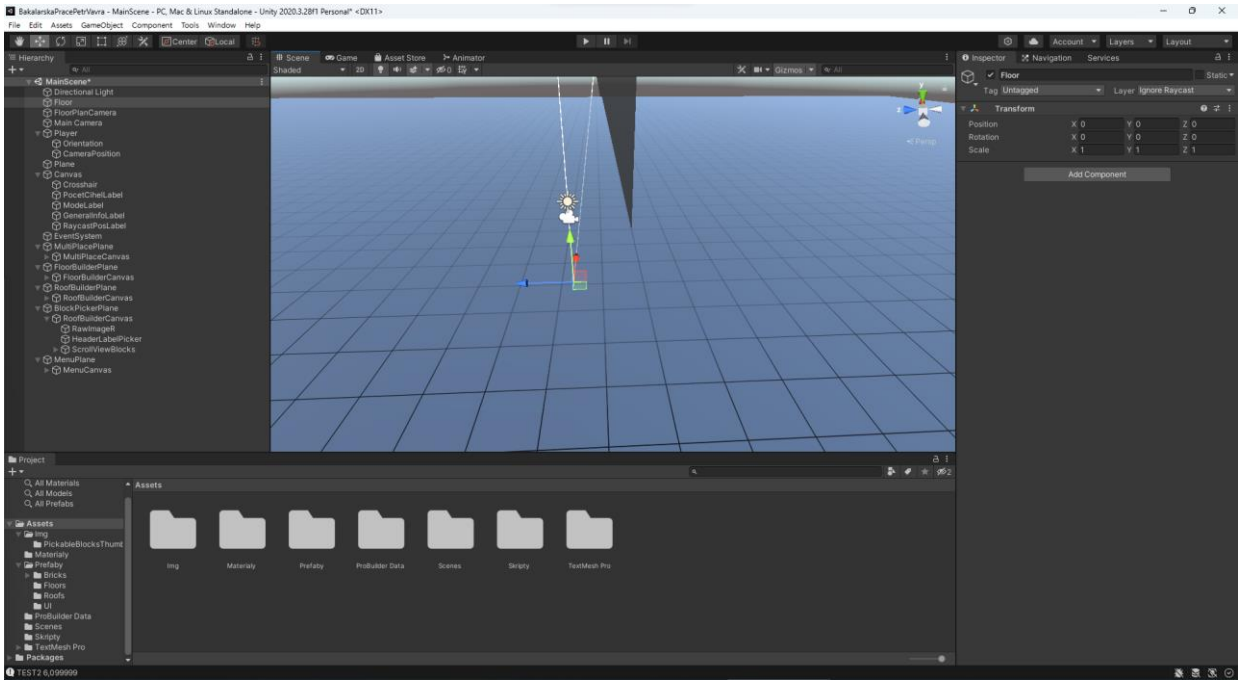
Jednou z dalších alternativ je herní engine Godot. Jedná se o engine, který je zdarma a open source pod licencí MIT. Podporuje vývoj her ve svém editoru, což mají všechny alternativy zmíněné v této sekci společné. Programovací jazyky pro vývoj nabízí GDScript a C#. GDScript je vytvořený specificky pro Godot a podporuje technologii zvanou GDExtension, která umožňuje např. napsat náročné části kódu v nízkém jazyce jako je např. C nebo C++. [18]

Godot nabízí rozsáhlou dokumentaci, která obsahuje mimo jiné sadu návodů pro kompletní začátečníky. Godot je určitě také dobrým kandidátem pro tvorbu jednoduché 3D aplikace, ale osobně jsem volil místo něj Unity, které má rozsáhlejší komunitu a podporu funkcí.

2 PRAKTICKÁ ČÁST

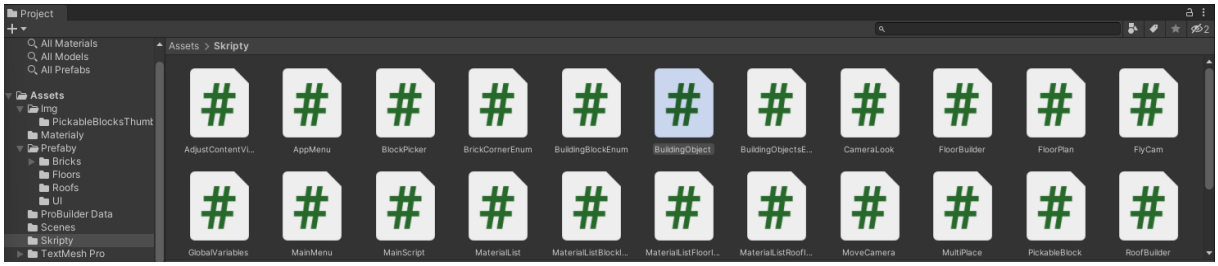
2.1 Metodika implementace

Vývoj většiny aplikací a her v Unity spočívá ve scénách, které obsahují herní objekty. Na tyto objekty můžeme poté navázat komponenty, což jsou vlastnosti unity představující určitou funkcionalitu, vlastnost nebo například skript obsahující kód. Objekty ve scéně fungují v hierarchickém režimu.



Obrázek 5: Vývojové prostředí Unity editor

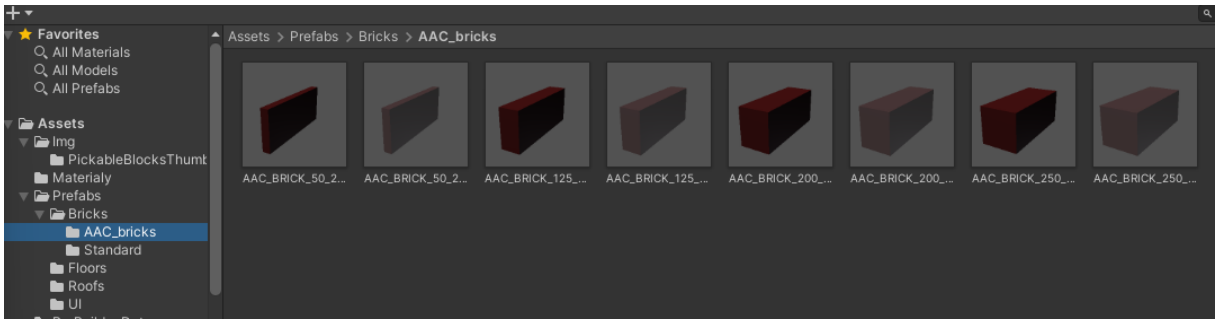
V této práci jsem využil způsob implementace, kde jsem vytvořil hlavní herní objekty a přidělil jim skripty a komponenty ručně. Jedná se primárně o objekty herní plochy, uživatele, světla, kamery a všech uživatelských rozhraní. Unity umožňuje si tyto objekty po vytvoření zobrazit, modifikovat nebo přiřadit skript, což usnadňuje vývoj. Součástí je také adresář „Assets“, který je v editoru dostupný v dolní části. Zde jsem si vytvořil souborovou strukturu projektu, kterou jsem rozdělil do obrázků, materiálů, před vytvořených objektů, scén a skriptů.



Obrázek 6: Adresář pro Unity skripty

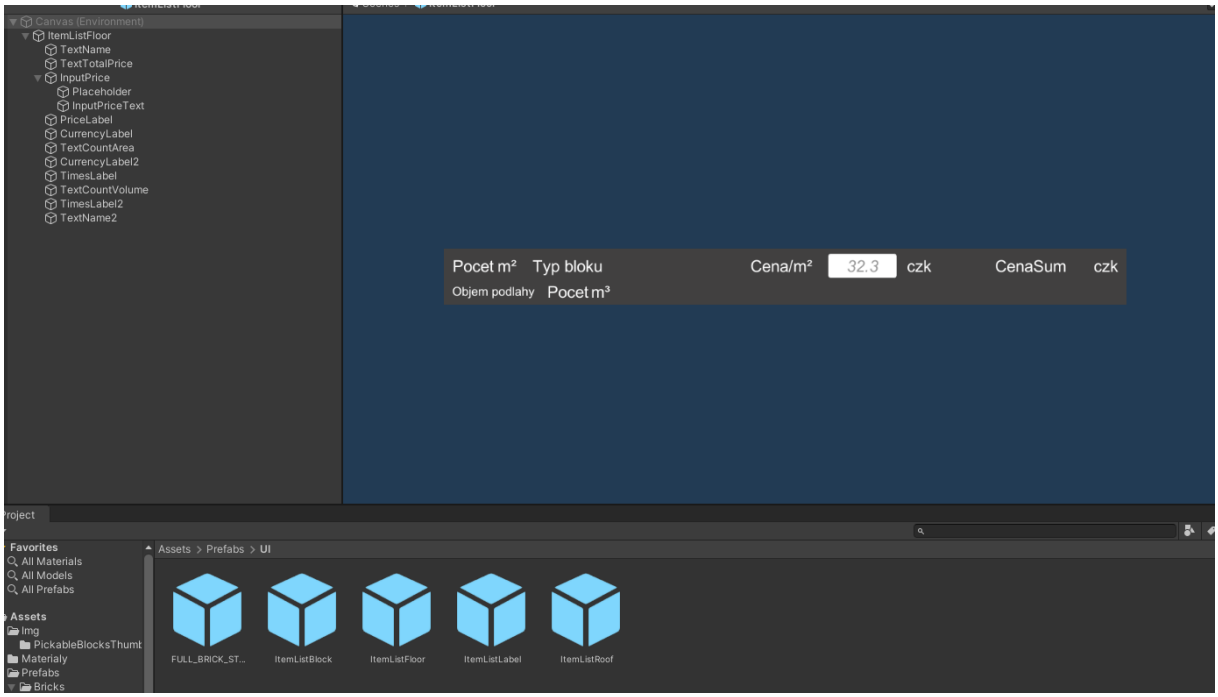
Když se podíváme do adresáře skriptů, najdeme zde přibližně 22 obslužných skriptů, které pohání celý simulátor. Tyto skripty jsou napsány mnou v jazyce C#, a aby je bylo možné použít, je nutné je přiřadit nějakému hernímu objektu. Samozřejmě to neplatí pro všechny, některé z nich jsou enumerace nebo čistě C# skripty bez unity knihovny, např. třída MaterialListBlockItem.cs, která je popsána v kapitole „Soupis materiálů“.

Ve složce „Prefabs“ se nachází objekty typu prefab, což v překladu znamená předem vytvořený. V této práci se nachází čtyři složky objektů, a to cihel, podlah, střech a uživatelského rozhraní. Tyto objekty jsou poté předávány skriptům, které je potřebují, a jsou instantizovány nebo např. přidávány do seznamů.



Obrázek 7: adresář s před vytvořenými objekty

Zajímavé je např. možnost vytvoření právě uživatelských rozhraní jako množinu předvytvořených objektů, což jsem využil při vytváření soupisu materiálů. V grafickém editoru jsem použil prvky uživatelského rozhraní, které nabízí unity a vytvořil šablonu, kterou jsem v kódu instantizoval a plnil daty. Více do hloubky jsem tuto problematiku popsal v kapitole soupis materiálu. Pro vývoj UI mi tato možnost přišla velmi užitečná, protože jsem byl schopný UI prvky vidět napřímo. Například menu pro výběr bloků v kapitole stavební materiál jsem programoval čistě v kódu bez použití šablony nebo editoru, a tento postup byl pomalejší a méně efektivní.



Obrázek 8: Předvytvořený objekt jako šablona grafického rozhraní

2.2 Seznam funkcionalit aplikace

Aplikace slouží primárně pro vytvoření plánu hrubé stavby, proto opomíjí detaily jako např. nábytek. V aplikaci je možné se pomocí klávesnice a myši libovolně pohybovat v trojrozměrném prostoru a stavět z jednotlivých cihel jakékoliv struktury. Cihly je možné mazat a rotovat s nimi po všech osách. Z jakých cihel chce uživatel stavět si vybere v grafickém menu, kde je vidět seznam s popisky a miniaturami všech stavebních cihel. Cihly do sebe zapadají a přichytávají se k sobě podle jejich rotace a podle toho, na kterou stranu nebo část se uživatel právě dívá. Cihla, na kterou se uživatel právě dívá, je pro usnadnění vyznačena jinou barvou a má celené okraje.

Uživatel nemusí stěny stavět po jednom, ale pomocí nástroje stěna si vytvoří libovolně velkou stěnu z vybraných cihel pomocí uživatelského rozhraní. Má také možnost stavět stěny na míru cihly.

Aplikace také nabízí nástroje pro tvorbu podlah a střech, kde pomocí uživatelského rozhraní může měnit jejich parametry, např. u střechy její rotaci nebo styl (sedlová, stanová, pultová).

Veškeré objekty je možné okótovat nástrojem kóta, a zjistit tak jejich přesnou vzdálenost mezi sebou nebo například šířku stěny. Tyto kóty se souvisejí s dalšími funkcionalitami, jako je půdorys, nárys a bokorys. V půdorysu má uživatel k dispozici rozhraní s grafickým plánkem, který se generuje ze stavební scény. Půdorys je možné různě upravovat, skrývat střechy a

podlahy, pohybovat se v něm a exportovat ho do formátu PNG. Tyto samé funkce nabízí bokorys i nárys.

Důležitou funkcionalitou je soupis materiálu, což je grafické rozhraní veškerých stavebních bloků ve scéně. Zde je přehledně vidět veškerý materiál s cenami, které je možné upravovat. U cihel se počítá počet a cena za kus, u podlah se počítá obsah podlahy v metrech čtverečních a u střech je relevantní obsah krytiny v metrech čtverečních. Tento seznam je možné vyexportovat do textového formátů, což se může hodit například pro tisk.

Aplikace obsahuje nástroj pro tvorbu stavebních otvorů, oken a dveří, který umožňuje v již vytvořených stěnách vytvořit otvor, vložit okno nebo dveře, vše libovolné velikosti.

Existuje možnost stavební scénu ukládat a načítat, a to buď v předem definovaných pozicích (slotech) nebo dialogem pro výběr souboru Windows.

Mezi další funkcionality se řadí možnost resetovat scénu, zobrazení vnořené nápovědy pro rychlý přehled nad klávesovými zkratkami, animace cihel při rotaci, aplikační menu pro přehledné ovládání všech funkcí simulátoru nebo možnost vypnout a zapnout mřížku přichytávání cihel a podlahy.

Všechny výše zmíněné funkcionality jsou hlouběji popsány v kapitolách níže.

2.3 Kamera

V aplikaci se uživatel pohybuje po stavební scéně v pohledu první osoby, kde existuje objekt připomínající kapsuli, který představuje uživatele a má přidělenou kameru vykreslující scénu. Uživatel pohybuje myší do stran, což je zaznamenáno a příslušně se aplikuje rotace na kameru. Tento přístup je velmi intuitivní pro uživatele a simuluje pohled z reálného života, což umožňuje rychlé seznámení s ovládáním.

Pro pohyb uživatelské kamery a implementaci pohledu z první osoby je potřeba obslužných skriptů, zejména MoveCamera.cs a CameraLook.cs. MoveCamera.cs je primitivní skript, který má za úkol přesouvat kameru na pozici hráče. Toto je sice možné docílit přidělením kamery jako potomka objektu uživatele, kde si poté kamera automaticky bere jako svůj nulový vektor pozici uživatele, ale s tímto přístupem se v mé práci objevily problémy jako jsou například trhavé pohyby a pomalý pohyb kamery. Vytvoření dedikovaného skriptu pro posun kamery na pozici uživatele tento problém vyřešilo.[19]

Podíváme se na skript CameraLook.cs, který zajišťuje pohyb samotné kamery na bázi vstupů z metod Input.GetAxisRaw(). Při prvním spuštěním skriptu se v metodě Start() změní proměnná Screen.lockCursor na pravdivou hodnotu, což způsobí přesun systémového kurzoru na střed obrazovky, jeho zamknutí na této pozici a jeho skrytí. Skript při každé iteraci metody Update() nejdříve kontroluje globální proměnnou GlobalVariables.lookEnabled, čímž umožňuje ostatním skriptům mít kontrolu nad vypnutím a zapnutím pohybu kamery. Dále jsou načteny dvě hodnoty, a to mouseX a mouseY, které vzniknou vyčtením osy kamery z metod Input.GetAxisRaw("Mouse X ") a Input.GetAxisRaw(" Mouse Y "). Tyto hodnoty jsou navíc vynásobeny desetinným číslem z proměnné cameraSensitivity. Tyto dvě hodnoty jsou poté přičteny/odečteny z proměnných rotationX a rotationY. Důležité je ošetření rotace x pomocí metody Mathf.Clamp(), která omezí rotaci na hodnoty -90 až 90 stupňů a tím zamezí překulení kamery směrem nahoru a dolů. Na konci skriptu se vypočítané hodnoty aplikují na objekt kamery, pomocí Eulerovských kvarterionů. Kvarteriony jsou způsob, kterým lze aplikovat rotace v Unity. Jsou užitečné hlavně díky tomu, že předchází problému zvanému jako kardanový zámek, který vzniká, pokud se dvě ze tří os objektu stanou k sobě paralelní. Důsledkem je nemožnost rotace po všech třech osách, ale pouze po dvou.[20]

```
if (GlobalVariables.lookEnabled)
 {
     GlobalVariables.cameraRotationY = transform.eulerAngles.y;

     mouseX = Input.GetAxisRaw("Mouse X") * cameraSensitivity;
     mouseY = Input.GetAxisRaw("Mouse Y") * cameraSensitivity;

     rotationY += mouseX;
     rotationX -= mouseY;

     rotationX = Mathf.Clamp(rotationX, -90, 90);

     transform.rotation = Quaternion.Euler(rotationX, rotationY, 0);
     orientation.rotation = Quaternion.Euler(0, rotationY, 0);
 }
```

2.4 Pohyb uživatele

Pohyb uživatele ve scéně je realizován technikou volné kamery, kde je možné se libovolně pohybovat do stran po scéně pomocí kláves W, A, S, D. Pro pohyb nahoru se používá mezerník, a pro vzestup klávesa Q. Tato logika je realizována ve skriptu FlyCam.cs, který využívá primárně metodu GetKeyDown() třídy Input, díky které rozpozná, jakou klávesu uživatel stiskl. Další užitečnou metodou je GetAxis() třídy input, která přijímá jako parametry řetězce „Horizontal“ a „Vertical“, kde řetězec „Horizontal“ představuje stisk kláves A nebo D, a řetězec „Vertical“ představuje stisk kláves W a S.[19]

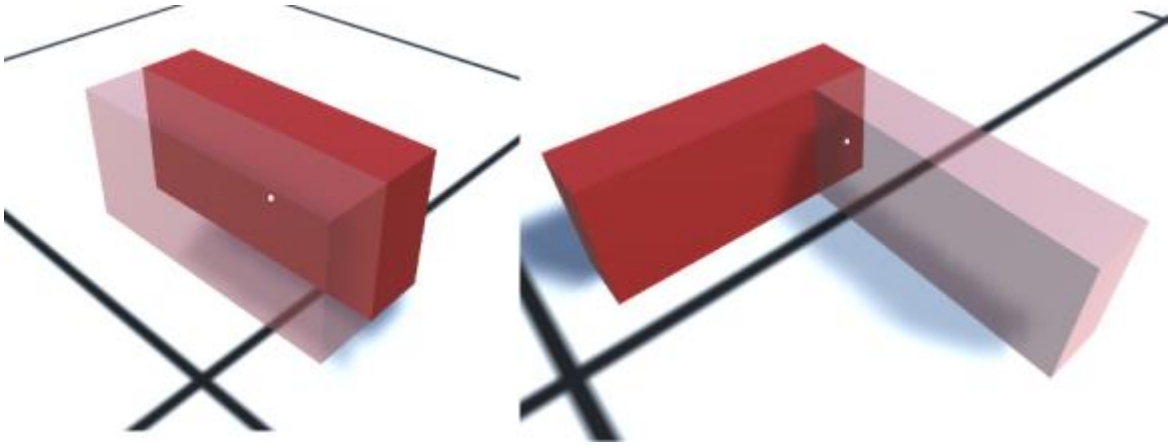
```
float moveHorizontal = Input.GetAxis("Horizontal") * speed;
float moveVertical = Input.GetAxis("Vertical") * speed;

Vector3 move = transform.right * moveHorizontal + transform.forward *
moveVertical;

if (Input.GetKey(KeyCode.Q))
{
    move -= transform.up * upDownSpeed;
}
if (Input.GetKey(KeyCode.Space))
{
    move += transform.up * upDownSpeed;
}
```

Z této ukázky kódu lze vyčíst, že uživatelská pozice se vypočítává vynásobením pohybového vektoru směrem dopředu a do stran, hodnot získaných z vynásobení Input.GetAxis() rychlostí. Rychlost je předem definovaná proměnná.

Pro výpočet vzestupu a klesání se používá velmi podobný výpočet, ale s vektorem představující směr nahoru/dolů.

2.5 Přichytávání stavebních bloků

Pro snadné používání stavebního simulátoru bylo nutné navrhnout a implementovat způsoby, kterými může uživatel přichytávat bloky k ostatním blokům. Tento režim přichytávání je možný vypnout nebo zapnout stiskem tlačítka M. Implementovány jsou dva hlavní druhy přichytávání, kolmé a rovnoběžné. V případě rovnoběžného přichytávání jsou bloky, jak už z názvu vyplývá, přichytávány rovnoběžně k bloku, na který uživatel aktuálně směruje kurzor, podle jeho strany. Např. pokud se podíváme směrem na širší boční stranu standardní cihly, silueta pokládané cihly bude zarovnána rovnoběžně, viz. Obrázek č. 2. Kolmé přichytávání je užitečné při vytváření pravých úhlů u vytvářených struktur. Aplikuje se při pohledu na krajní strany pokládaného bloku, a to pouze v případě, kdy je pokládaný blok kolmý k bloku, na který se uživatel dívá. Tyto dva způsoby přichytávání jsou implementovány způsobem, který je dynamický a nepoužívá žádné konstantní hodnoty, vždy vychází z rozměrů daných objektů. To znamená, že jsou univerzální pro bloky jakékoliv velikosti. To umožňuje jednoduchou implementaci nových stavebních bloků. V prvních fázích této práce bylo přichytávání implementováno „natvrdo“ pro každý blok, ale z důvodu výše zmíněných jsem od této techniky upustil.



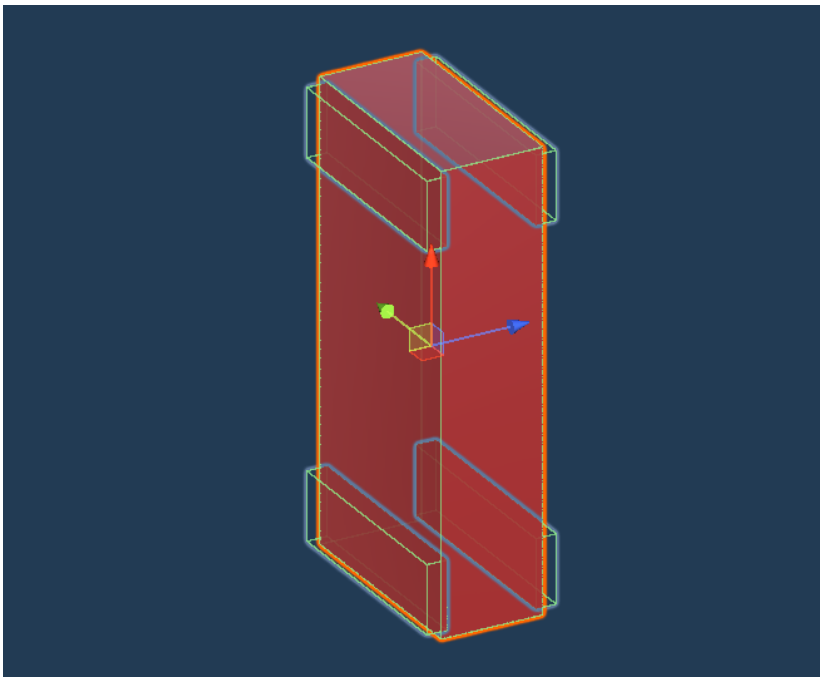
Obrázek 9: Ukázka režimů připnutí siluety

Metody přichytávání jsou implementovány v hlavním skriptu. Nejdříve se vyšle objekt typu raycast[1], kterým se zjišťuje objekt, na který dopadl. Jsou tři možnosti, první je dopad na jednu ze stran stavebního bloku (rovnoběžné připnutí), druhá je dopad na kraj bloku (kolmé připnutí) a třetí je dopad na podlahu scény. Toto se zjišťuje pomocí struktury switch[2] a tagů[3], které mají objekty typu prefab[4] přidělené. Na tomto základě se pouští metody CalculateBrickSnap() a BrickFullSnap(), které popisuji níže.[19]

2.5.1 Kolmé přichytávání

Pro kolmé připnutí stavebních bloků se používá metoda BrickFullSnap(). Kolmé přichytávání funguje na principu čtyř kolizních domén/rohů, které jsou rozmístěny v objektu prefab každého bloku. Každá doména/roh má vlastní značku, kterou skript rozpozná a posílá v parametru této metodě. Rohy jsou definované jako objekty a mají hodnotu, kterou definuje enumerace[5] BrickCornerEnum. Díky tomuto rozdělení je možné rozpoznat, o který roh se jedná pomocí konstrukce switch. Pro snadnější výpočet skript ukládá postupně informace do dočasného vektoru. Pozice siluety se počítá odečtením/přičtením poloviny délky cílového bloku, což efektivně posune siluetu na kraj cílového bloku. Poté přičte/odečte polovinu šířky siluety k dočasnému vektoru, což posune siluetu a zarovná ji perfektně s cílovým blokem.

[1] Metoda představující paprsek, typicky používaná pro detekci kolizí nebo objektů v 3D prostoru.
[2] Rozhodovací konstrukce programovacího jazyka, vykonává akci na základě hodnoty proměnné.
[3] Značka, používá se jako identifikátor přidělený objektům.
[4] Předem vytvořený objekt představující šablonu, který může být vytvářen opakovaně.
[5] Datový typ v programování, definuje seznam konstantních hodnot.



Obrázek 10: Kvádrové kolizní objekty stavebního bloku

2.5.2 Rovnoběžné přichytávání

Tento způsob přichytávání je implementován v metodě CalculateBrickSnap(). Na začátku získává střed a velikost cílového objektu targetBrick, a velikost objektu který má být umístěn. Následně generuje všechny možné vektory směru, kterými může být stavební blok posunutý a ukládá je do proměnných directionFront, directionBack, directionRight, directionLeft, directionUp a directionDown. Tyto směry určují, jakým směrem a jak daleko se silueta posune od středu cílového bloku. Je nutné zmínit, že tyto vektory se generují ve dvou variantách, a to podle rotace cihly, specificky dle hodnoty 0 nebo 90 stupňů. V další části funkce rozpoznává, který směr využít pro výpočet. To jsem docílil použitím metody Vector3.Dot(), která kalkuluje skalární součin dvou vektorů. Výsledkem je skalár, ze kterého lze vyčíst, na kterou část bloku se uživatel aktuálně dívá. Stačí tři proměnné, pro všechny směry.

```
Vector3 Normal = _hit.normal;
float dotUp = Vector3.Dot(_hit.normal, _hit.transform.up);
float dotForward = Vector3.Dot(_hit.normal, _hit.transform.forward);
float dotRight = Vector3.Dot(_hit.normal, _hit.transform.right);
```

Pokud je proměnná dotUp velmi blízko jedné, znamená to, že směruje nahoru. Pokud je velmi blízko mínus jedné, směruje dolů. Tento princip je použitý pro všechny tři proměnné, a díky tomu je možné určit na kterou stranu se uživatel dívá kurzorem a patřičně aplikovat posun a přichycení.

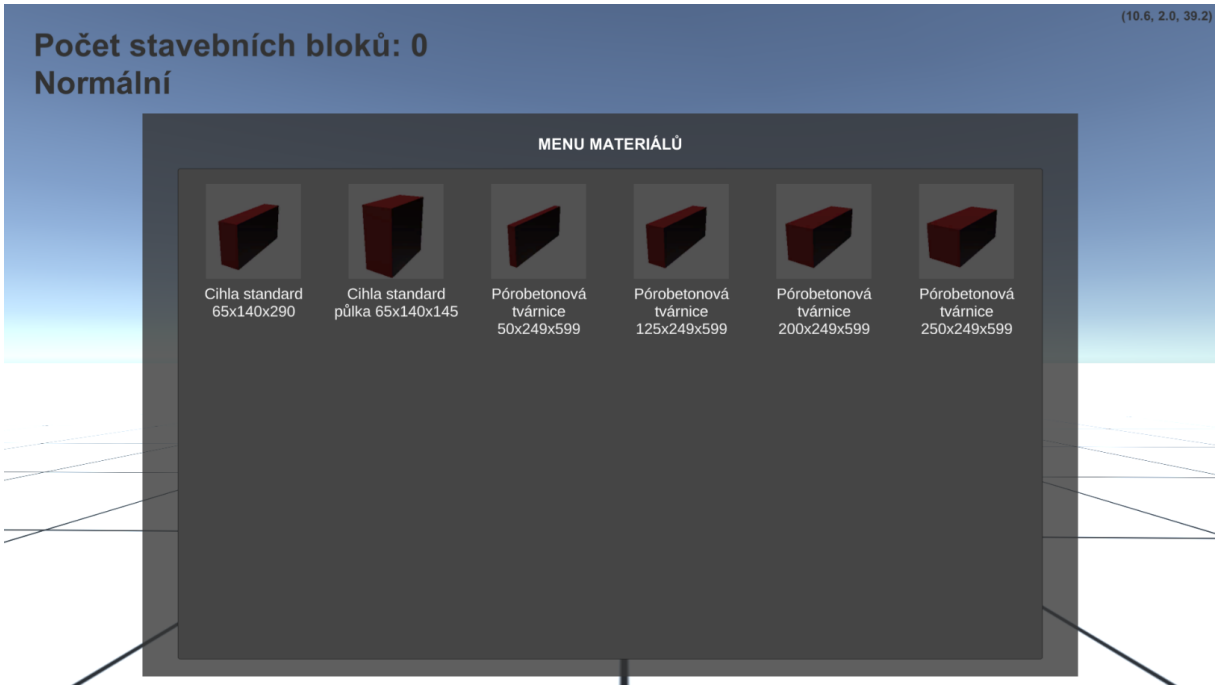
2.5.3 Připnutí k zemi

V této části kód kontroluje metodou raycast vzdálenost od uživatelské kamery k zemi, a pokud je menší než specifikovaná hodnota, připne stavební blok k podlaze scény. Silueta stavebního bloku je posunuta směrem nahoru po ose y o polovinu své výšky a přičtena k hodnotě 2, což zajišťuje připnutí k podlaze univerzálně pro bloky všech velikostí.

2.6 Stavební materiál

2.6.1 Výběr stavebních bloků

Uživatel má na výběr ze dvou způsobů výběru stavebních bloků. Jedním způsobem je pomocí numerické klávesnice, kde klávesy jedna až devět jsou připnuté k vybraným stavebním blokům. Druhý způsob, který je z hlediska implementace a použití zajímavější, je přes grafické rozhraní, které jsem nazval „menu materiálů“. Toto grafické rozhraní se spouští klávesou P a má za úkol zobrazit výběr stavebních bloků společně s jejich náhledem a rozměry. Uživatel má možnost kliknutím vybrat blok, což způsobí ukončení grafického rozhraní a změnu siluety bloku.



Obrázek 11: Menu výběru stavebních bloků

Nyní se podíváme na implementaci této komponenty. Je implementována skriptem BlockPicker.cs a jako obslužnou třídu využívá PickableBlock.cs. Třída PickableBlock obsahuje definici bloků (popis, obrázek, typ), které se používají pro generování obsahu uživatelského rozhraní. Obsahuje dva konstruktory, kde jeden je používaný a druhý je zakomentovaný.

Zakomentovaný konstruktor lze použít pro generování obrázků herních objektů z jejich náhledu v Unity editoru pomocí třídy AssetPreview. Ta umožnuje přistoupit k textuře daného náhledu a jeho převodu na obrázek. Je nutné, aby tato třída byla zakomentovaná a používala se pouze pro obslužné účely, protože třída AssetPreview je dostupná pouze z editoru, ne v sestavení aplikace.

Přesuneme se ke třídě BlockPicker. V metodě start se deklarují a inicializují instance třídy PickableBlock, které se vloží do listu. Tento list je poté iterován cyklem for-each a z každého prvku je vytvořen herní objekt, který obsahuje náhled a popisek daného bloku. Zajímavá je implementace změny kurzoru při přejetí kurzorem napříč bloky, což jsem docílil přidělením tagu „PickableBlockUI“ každému hernímu objektu reprezentujícímu blok, a v metodě update metodou raycast a kontrolou objektu, který je pod kurzorem myši. Při kliknutím myši se použije stejná logika, ale v takovém případě se zavolá pomocí struktury switch, která rozhodne, o jaký blok se jedná. Je to možné díky tomu, že při generování bloky dostanou také přidělené jméno, které odpovídá enumeraci BuildingObjectsEnum. Nakonec struktura switch volá metodu ChangeBlock() která má v parametru typ bloku. Tato metoda změní typ bloku v hlavní třídě a zavolá obslužnou metodu pro její změnu.

2.7 Rotace materiálu

Stavební bloky je možné otáčet vzhledem ke každé ose x, y, z. Pomocí tlačítka R je možné rotovat okolo osy y, po ose x je možné otáčet pomocí tlačítka F. K uchovávání slouží proměnné rotationX a rotationY, které jsou používány průběžně v hlavním skriptu, primárně pro výpočet přichytávání.

2.7.1 Animace rotací

Rotace bloků není okamžitá, ale má průběžný efekt přechodu, který trvá desetinásobek časového rozmezí mezi posledním a aktuálním snímkem. Toto je zacíleno v hlavním skriptu v metodě update, která periodicky kontroluje aktuální rotaci x a y, ze které vypočítá vektor:

```
Vector3 to = new Vector3(rotationX, rotationY, rotaceZ);
```

Který je použit při transformaci metodou Vector3.Lerp():

```
mainSilhouette.transform.eulerAngles =
Vector3.Lerp(mainSilhouette.transform.rotation.eulerAngles, to, Time.deltaTime *
10);
```

Metoda Lerp() slouží v Unity pro lineární interpolaci mezi dvěma vektory po časový úsek, jak je patrné z ukázky kódu. Time.deltaTime je proměnná Unity prostředí, která představuje časový úsek mezi aktuálním a předchozím snímkem.

2.8 Způsob pokládání

Ve stavebním simulátoru je způsob pokládání implementovaný pomocí siluety pokládaného objektu, kde po stisknutí levého tlačítka myši se ze siluety instantizuje klon stavebního bloku, který silueta představuje. Silueta mění pozici dle pohybu myši, přichytává se k zemi, pokud je zapnutá mřížka, a připíná se k již postaveným stavebním blokům pomocí metod, které jsou popsány v kapitole „Přichytávání stavebních bloků“ výše. Uživatel samozřejmě může i veškeré objekty scény mazat, a to stisknutím pravého tlačítka myši. Tato část kódu se nachází v hlavním skriptu MainScript.cs v metodě Update(). Nejdříve je vyslán paprsek, který dopadá kam se uživatel právě dívá.

```
Ray = Camera.main.ScreenPointToRay(Input.mousePosition);
```

Z této proměnné je možné zjistit tag herního objektu, na který dopadl. Toto je kontrolováno, a pokud se jedná o stavební blok, objekt podlahy nebo objekt střechy, je tento objekt nalezen v globálním poli proměnných a je smazán.

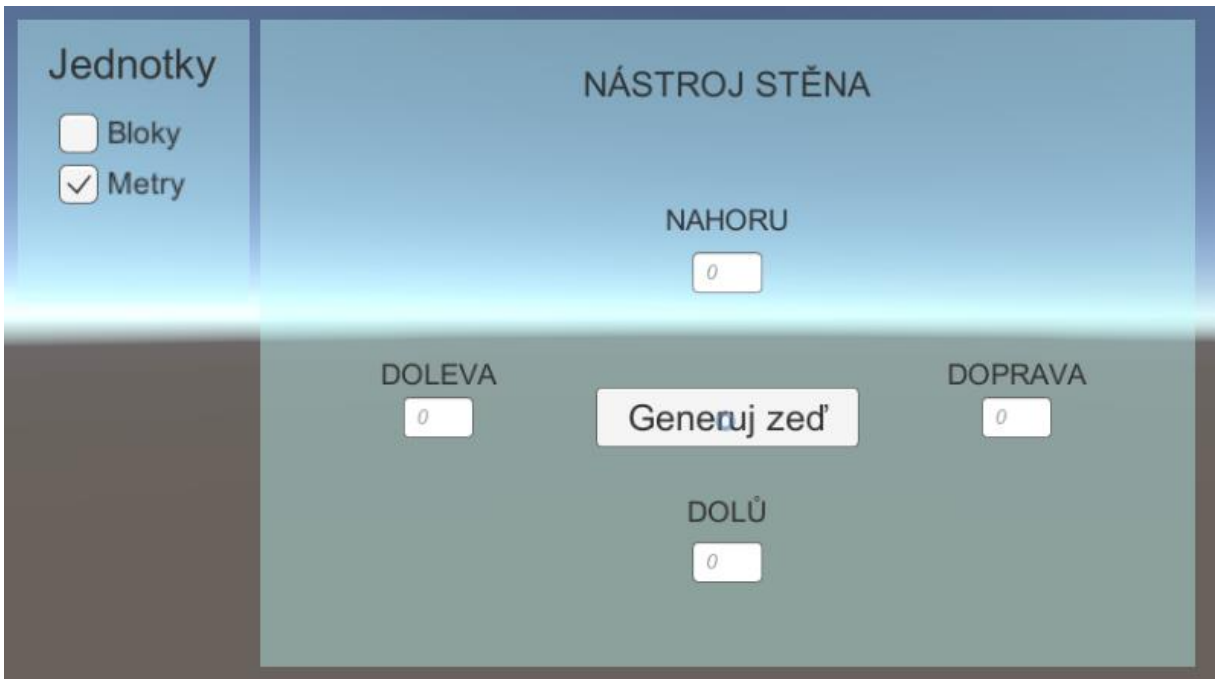
2.8.1 Jednotlivý způsob

Jak už bylo výše zmíněno, bloky je možné pokládat do scény stisknutím levého tlačítka myši, kde logika je implementována v hlavní třídě MainScript.cs v metodě Update(). Kontroluje se stisk myši pomocí metody Input.GetKeyDown() s parametrem KeyCode.Mouse0, který představuje levé tlačítko. Tato logika je poměrně primitivní, pouze se instantizuje objekt siluety a je z něho vytvořen objekt typu BuildingObject, který se vloží do globálního pole. Ostatní složitější náležitosti jsou již vyřešeny pohybem kamery a jinými metodami, které zajišťují změnu siluety nebo například její přichycení k jiným blokům.

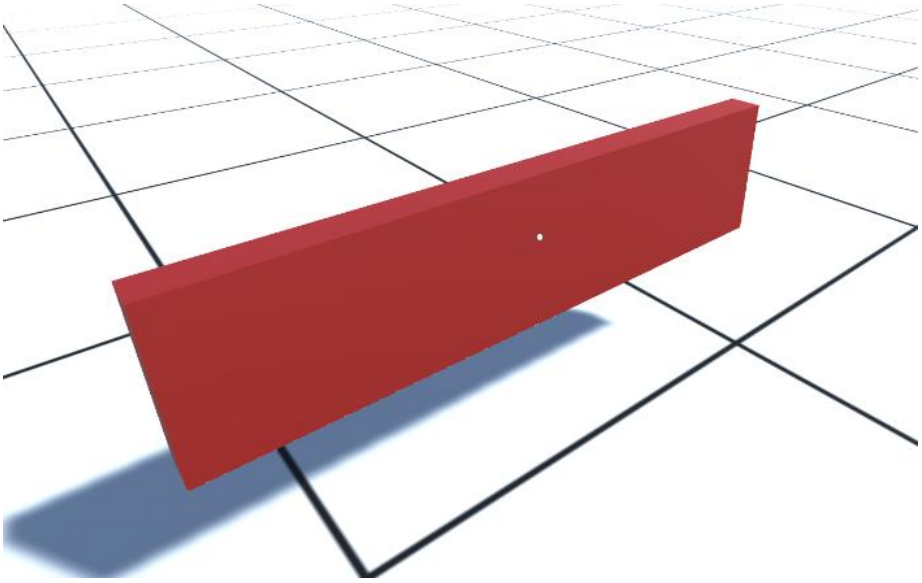
2.8.2 Plošný způsob

Protože pokládání cihel po jednom není vhodné pro větší struktury, bylo nutné implementovat způsob, jak je pokládat plošně. Hlavní myšlenkou plošného způsobu pokládání bylo vytvořit uživatelské rozhraní, kde by uživatel mohl zadat vstupní údaje a po stisku tlačítka bude vygenerována příslušná struktura. Plošný způsob pokládání je možné vyvolat stisknutím tlačítka N. Zobrazí se jednoduché a intuitivní uživatelské rozhraní skládající se ze čtyř vstupních polí a tlačítka pro spuštění. Vstupní pole představují směry nahoru, dolů, doleva a doprava. Vždy je nutné zadat dvě kladné hodnoty, a to do sousedících polí. Například při zadání

čísla 2 do vrchního pole, a čísla 4 do pole levého bude vygenerována stěna, která je dva vybrané bloky vysoká a čtyři bloky široká. Při zadání hodnot do nekompatibilních polí, zejména vrchní + spodní a levé + pravé nebude nic vygenerováno. Je to z důvodu implementace, kde stěna by měla představovat dva směry, které tvoří obdélník nebo čtverec. Například směry doleva a doprava v této aplikaci nepředstavují obdélník, protože chybí výška nahoru nebo dolů. Jako stavební blok bude použita aktuální silueta. Uživatel má možnost si vybrat ze dvou možných režimů, a to režimu bloků nebo metrů. Pokud uživatel vybere režim metrů, je vygenerována stěna v metrech. V opačném případě je vygenerována stěna na míru stavebních bloků. Například při vložení čísla 1,2 do textového pole „nahoru“ a čísla 2 do textového pole „doprava“ bude mít stěna velikost 1,2 metru do výšky a 2 metry do šířky směrem vpravo.



Obrázek 12: Uživatelské rozhraní nástroje stěna



Obrázek 13: Výstup nástroje stěna

Tato funkce je implementována v třídě MultiPlace.cs. V metodě Update() skript reaguje na stisk tlačítka, pomocí Input.GetKeyDown(). Při stisku tlačítka „Generuj zeď“ je zavolána metoda OnConfirmButtonClicked(), která si vytvoří pole čtyř čísel(vstupů). Třída má přístup k těmto polím pomocí veřejných proměnných typu InputField, které jsou přiřazeny v Unity editoru. Tyto proměnné jsou iterovány v cyklu, pokud jsou prázdné nebo záporné, vloží se místo nich do pole nula. V unity editoru lze vstupnímu poli přidělit vlastnost „Content type“, která definuje, jaký typ dat může uživatel vložit. Pro tyto pole jsem použil typ Integer Number, tudíž není nutné ošetřovat vstup pro text či jiné znaky, jelikož takové znaky nelze v žádném případě vložit.

Pro určení, které dva vstupy jsou vyplněny existují ve skriptu čtyři podmínky, kde každá obsahuje dva vnořené cykly for. Nyní budeme uvažovat případ, kdy uživatel zadal hodnotu vrchní a levou. V cyklu je nutné zkontrolovat o kolik stupňů je natočen stavební blok a zároveň je kontrolováno skrze globální proměnnou cameraRotationY směr, kterým se uživatel právě dívá. Díky tomu je možné zjistit, kterým směrem generovat stěnu. Pokud je hodnota menší než 270 stupňů a zároveň větší než 90 stupňů, uživatelská kamera směruje na jih, a pokud je větší než nula a zároveň menší než 180 stupňů, směruje na sever. Při použití dvou cyklů je poté možné poměrně jednoduše generovat jednotlivé cihly vytvořením nového vektoru. Když se uživatel bude dívat na jih, spustí se tato část kódu:

```
if (GlobalVariables.cameraRotationY > 90 && GlobalVariables.cameraRotationY <
270)
{
    Debug.Log("facing south");
    newPosition = new Vector3(
    currentPosition.x + (GlobalVariables.brickPrefab.transform.localScale.x * j),
    currentPosition.y + (GlobalVariables.brickPrefab.transform.localScale.y * i),
    currentPosition.z);
}
```

Zde je patrné vytvoření nového trojrozměrného vektoru, kterému vynásobíme hodnoty x a y proměnnými j a i z cyklů ve kterých se nachází. Tím je zajištěno posunutí cihly při každé iteraci cyklem, a to do šířky pomocí proměnné j, a do výšky díky proměnné i.

2.8.3 Nástroj střecha

Nástroj střecha, v aplikaci nazýván také jako „Roof builder“ slouží pro generování střech, jak už z názvu vypovídá. Skládá se z uživatelského rozhraní RoofBuilderPlane, a obslužného skriptu RoofBuilder.cs. Když jsem tento nástroj programoval, hlavní myšlenkou byla jednoduchost vytvoření, proto jsem zvolil přístup dvou bodů. Uživatel spustí nástroj střecha pomocí klávesy V, je o tom informován změnou módu v levém horním rohu. Kliknutím levého tlačítka myši uživatel vybere dva body, které jsou vypočítány vrhnutím paprsku na daný povrch. Mezi těmito paprsky je vygenerován herní objekt, který odpovídá velikosti vypočítané z těchto dvou bodů. Poté se otevře uživatelské rozhraní, kde je možné vybrat typ střechy, a to: stanová, sedlová a pultová. Pokud je vybrána střecha pultová, objeví se navíc rozevírací seznam s typy, kterými je možné měnit její rotaci. Samozřejmostí je textové vstupní pole pro definici výšky střechy. Po zadání údajů je možné je libovolně aktualizovat a sledovat změnu objektu pomocí tlačítka změny hodnot v textovém poli nebo rozevíracím seznamu. Změna reaguje na událost typu onChange. Další dvě tlačítka Potvrdit a Zrušit slouží k uložení střechy a vložení do seznamu objektů, a ke zrušení objektu, pokud nevyhovuje.

Obrázek 14: Uživatelské rozhraní nástroje střecha

Přesuňme se ke skriptu RoofBuilder.cs. Zde je uložena logika překreslování a instancování objektu střechy dle zadaných parametrů. Obsahuje také logiku skrývání a zobrazování objektu typu canvas[6] , uzamčení pohybu a kamery, skrytí siluety cihly a zobrazování zpráv uživateli.

Celá třída funguje jako jedna sekvence, kde metoda Update() volá metodu OpenCanvas(). V otevřeném objektu canvas už se pouze naslouchá událostem, které generují prvky uživatelského rozhraní. Třída končí stisknutím klávesy Escape, které naslouchá metoda Update, nebo metodou OnSubmitButtonClick() či OnCancelButtonClick().

Metoda OnSubmitButonClick() je významná, protože generuje objekt typu BuildingObject, který se přídává do globálního seznamu allBuildingObjects ve třídě GlobalVariables. Tento seznam se používá pro generování výstupních dat a soupisu materiálů.

```
BuildingObject buildingObject = new BuildingObject();

switch (roofType)
{
    case 0:
        buildingObject.type = BuildingObjectsEnum.ROOF_OBJECT_TENT;
        break;
    case 1:
        buildingObject.type = BuildingObjectsEnum.ROOF_OBJECT_GABLE;
        break;
    case 2:
        buildingObject.type = BuildingObjectsEnum.ROOF_OBJECT_COUNTER;
        break;
}

buildingObject.GameObject = generatedRoof;
GlobalVariables.allBuildingObjects.Add(buildingObject);
GlobalVariables.modeLabel = "Normální";
isRoofBuilderEnabled = false;
GlobalVariables.mainBrick.gameObject.SetActive(true);
```

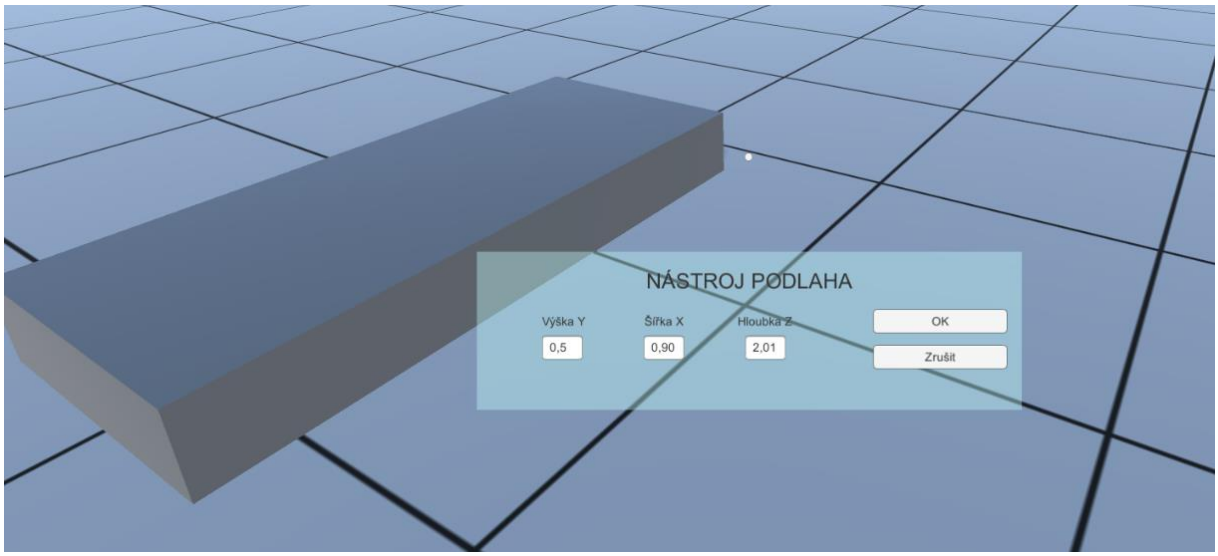
Nejkomplikovanější metoda, která stojí za zmínku je metoda OnUpdateButtonClick(), která ve své podstatě vyčítá data z uživatelského rozhraní a patřičně na něj reaguje. Při detekci změny maže starý objekt střechy a generuje nový. Jelikož je metoda rozsáhlá v příkladu uvedu pouze první případ struktury switch, ostatní případy jsou velmi srovnatelné a liší se hlavně v druhu objektu který generují. Z kódu níže lze vyčíst, že případ vždy smaže starý objekt střechy a vygeneruje nový. Přiřadí mu délku pomocí nového trojrozměrného vektoru (parametry jsou vyčtené z prvních dvou metod raycast, které určil uživatel při spuštění nástroje střecha), instantizuje ho a pomocí komponenty transform mu nastaví vypočítanou velikost a pozici.

[6] Herní objekt představující prostor pro prvky uživatelského rozhraní.

```
case 0:
     Destroy(generatedRoof);
     GameObject tent;
     Vector3 size = new Vector3(Mathf.Abs(firstPoint.x - secondPoint.x) ,
yVectorTemp , Mathf.Abs(firstPoint.z - secondPoint.z) );

     tent = Instantiate(roofTent);
     tent.transform.position = midpoint;
     tent.transform.localScale = size;
     generatedRoof = tent;
     break;
```

2.8.4 Nástroj podlaha

Pro vytváření objektů, které reprezentují podlahu jsem v této práci vytvořil nástroj s uživatelským rozhraním. Spouští se stiskem tlačítka B, a spočívá v systému, kde uživatel vybere dva body, podle kterých se vypočítá a instantizuje objekt. To je doprovázeno pomocnými ukazateli ve formě textu, kde při zapnutí nástroje podlaha je uživateli uprostřed obrazovky zobrazen text „Kliknutím vyberte dva body“. Tyto body se vybírají paprskem metody raycast, tudíž se bere objekt ve středu kurzoru, na který se uživatel právě dívá. Po vybrání dvou bodů je vytvořen objekt podlahy, a zároveň je zobrazeno uživatelské rozhraní pro její editaci. Na výběr jsou tři vstupní pole, kde lze měnit šířku výšku a hloubku. Při změně údajů se automaticky pomocí události typu onChange zavolá funkce, která validuje vstupy a patřičně upraví objekt.



Obrázek 15: ukázka nástroje podlaha s uživatelským rozhraním

Co se týče kódu, nástroj podlaha je implementovaný ve třídě FloorBuilder.cs. V této třídě jsou tři hlavní části, a to: sběr dvou bodů, tvorba objektu, reakce na tlačítko OK. Sběr bodů probíhá v metodě update, pomocí třídy Physics a metody Raycast(). Metoda Raycast() přijímá jako

parametr objekt paprsku ray, který je vytvořen pomocí metody ScreenPointToRay(Input.mousePosition) z třídy Camera.main.

```
if (Input.GetMouseButtonDown(0) && rayCount < 2)
{
    Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
    RaycastHit hit;

    if (Physics.Raycast(ray, out hit))
    {
        rayCount++;
        Vector3 hitPosition = hit.point;
        if (rayCount == 1)
        {
            firstPoint = hitPosition;
            GlobalVariables.generalInfoLabel = "Bod 1: " + hitPosition;
        }
        else if (rayCount == 2)
        {
            secondPoint = hitPosition;
            GlobalVariables.generalInfoLabel = "Bod 2: " + hitPosition;
            CreateBoxBetweenPoints(firstPoint, secondPoint);
        }

    }
}
```

Po zavolání metody CreateBoxBetweenPoints() je nejdříve vypočítán střed mezi dvěma body, a to vzorcem (bod1 + bod2) / 2. Dále je vytvořen AABB(Axis-Aligned Bounding Box), což je objekt, který se používá pro označení kvádrového bloku zarovnaného s osami x, y, z.[21]

```
Vector3 size = new Vector3(Mathf.Abs(point1.x - point2.x), Mathf.Abs(point1.y - point2.y), Mathf.Abs(point1.z - point2.z));
```

Dále je vytvořen primitivní herní objekt typu cube, kterému je přiřazen vypočítaný střed, rozměr a vykreslovací komponentě je přiřazen materiál. Pokud uživatel vybere dva body ve stejné výšce, může se stát, že vytvořený objekt bude mít téměř nulovou výšku a bude se překrývat s již existující podlahou, což může způsobovat vizuální chyby. Pro ošetření je každému objektu podlahy kontrolována výška osy y, a pokud je nulová, je jí automaticky nastavena hodnota 0.02f.

Pokud uživatel odklikne tlačítko OK, je vytvořen nový objekt třídy BuildingObject, je mu přiřazena hodnota FLOOR_OBJECT z enumerace BuildingObjectsEnum a odkaz na herní objekt. Poté je přidán do globálního seznamu stavebních objektů pro pozdější zpracování třídou MaterialList.cs.

2.9 Způsob mazání

V aplikaci je možné jakýkoliv položený nebo vygenerovaný objekt jednoduše smazat, a to stisknutím pravého tlačítka myši. Z hlediska implementace se vyšle paprsek typu raycast, díky

kterému je možné zjistit, na který stavební objekt se uživatel právě dívá. Dále se iteruje globální seznam všech stavebních bloků, a pokud je nalezena shoda, objekt je vymazán ze scény pomocí metody Destroy(), a zároveň je smazán ze seznamu proměnných.[19]

```
foreach (BuildingObject item in GlobalVariables.allBuildingObjects)
{
    if (hitDestroy.collider.gameObject == item.GameObject)
    {
        Destroy(hitDestroy.collider.gameObject);
        GlobalVariables.allBuildingObjects.Remove(item);
        break;
    }
}
```

2.10 Aplikační menu

Aplikační menu je komponenta uživatelského rozhraní, kterou uživatel spouští stisknutím klávesy escape. Slouží k třem hlavním účelům, přejít do menu výpisu materiálů, spustit generování půdorysu a ukončit aplikaci.

Obrázek 16: Aplikační menu

Logika této komponenty je popsána ve skriptu AppMenu.cs. Tento skript je poměrně stručný a jednoduchý, a má na starost pouze zobrazení objektu canvas, ukončení hry pomocí metody Quit třídy Application při stisku tlačítka ukončit a zobrazení dvou dalších oken pro výpis materiálu a zobrazení generace půdorysu. Tyto dvě okna mají samostatné komponenty uživatelského rozhraní a vlastní obslužné skripty, které jsou komplexnější, a proto jsou popsány níže

v kapitolách Soupis materiálu a Generace půdorysu. Obě třídy ale mají společnou kostru. Ve výchozím stavu pomocí jejich metody Start jsou iterovány všechny jejich potomci a jejich stav je přepnutý do stavu false.

```
void Start()
{
    foreach (Transform child in transform)
    {
        child.gameObject.SetActive(false);

    }
}
```

Tím je zajištěno jejich skrytí při startu aplikace. Obě třídy mají také svoji obdobu metod OpenUI() a CloseUI() které stejným způsobem iterují své potomky. Nakonec obsahují obslužnou metodu tlačítka zpět, která vrací uživatele zpět do aplikačního menu.

```
public void OpenListUI() {
    foreach (Transform child in transform)
    {
        child.gameObject.SetActive(true);
    }
}

public void CloseListUI() {
    foreach (Transform child in transform)
    {
        child.gameObject.SetActive(false);
    }
}

public void OnBackButtonClicked() {
    CloseListUI();
}
```

V prostředí Unity je možné iterovat veškeré potomky každé třídy přímo z jejího kontextu, a to odkazem na vlastnost transform.

2.11 Soupis materiálu

Užitečná komponenta stavebního simulátoru vytvořeného v této práci je generování soupisu materiálů. Soupis materiálů má své vlastní grafické rozhraní, které je přístupné po vstupu do aplikačního menu. Soupis je rozdělen do čtyř částí v tabulkovém rozhraní, ve kterém je možné se pohybovat kolečkem myši nebo stisknutím tlačítka myši a tahem. Každá část má své vlastní ovládací prvky a funkcionality.

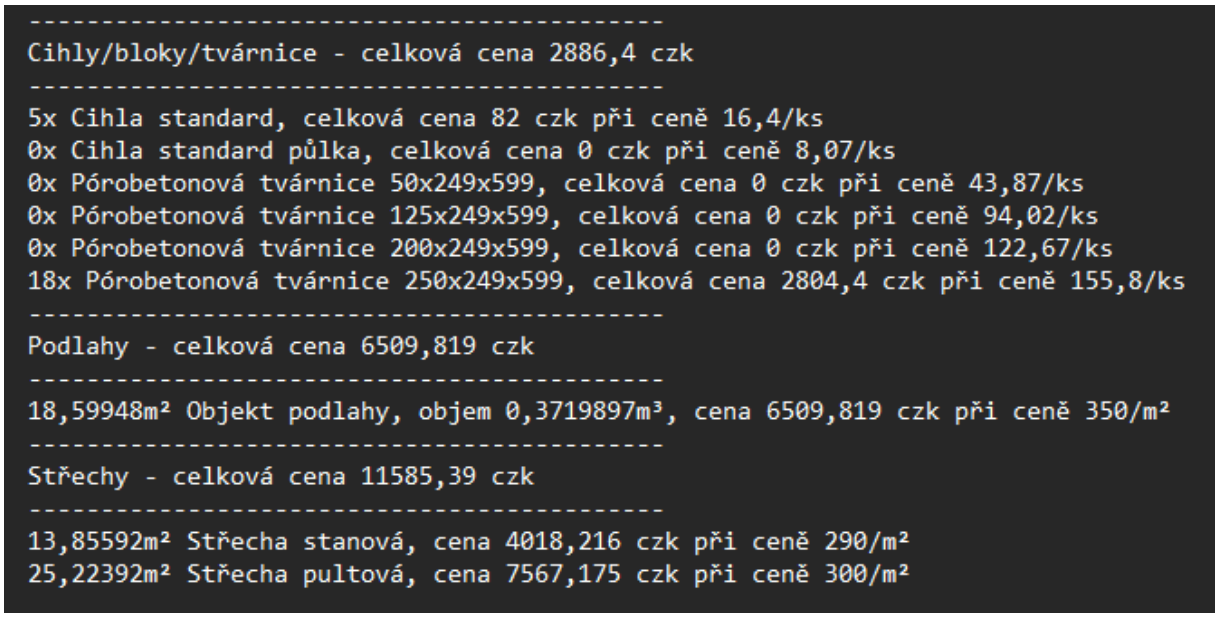
2.11.1 Počet jednotlivých kusů

Kdykoliv uživatel položí stavební blok, je uložen do globálního seznamu všech herních objektů. Toho využívá soupis jednotlivých kusů, což je jedna z více komplexních tříd v této práci.

Skládá se z tabulky, která obsahuje záhlaví obsahující celkovou cenu všech bloků a jednotlivých záznamů, které jsou dynamicky generovány dle globálního seznamu. Záznamy jsou seskupené podle typu stavebního bloku. Skládají se z počtu bloků, názvu bloku dle enumerace, výchozí ceny bloku také dle enumerace a celkové ceny za skupinu bloků. Uživatel má možnost pomocí zaškrtávacího políčka zobrazit nebo skrýt prázdné položky, což může být užitečné, pokud uživatel používá ke stavbě pouze pár druhů stavebních bloků.

Obrázek 17: Grafické rozhraní seznamu materiálu

Uživatel má možnost si soupis materiálu vyexportovat do textového formátu pomocí tlačítka, které otevře dialogové okno pro ukládání. Tato funkce se může hodit například pro tisk na papír.



Obrázek 18: textový výstup seznamu materiálu

Skript který generuje soupis jednotlivých bloků se nazývá MaterialList.cs, a pro své fungování používá pomocnou třídu MaterialListBlockItem. Třída MaterialListBlockItem představuje položku seznamu bloků, a to již v uskupeném formátu, tedy každá instance této třídy představuje všechny objekty jednoho typu. Například MaterialListBlockitem typu FULL_BRICK_STANDARD představuje všechny položené plné cihly ve scéně. Tato třída tedy obsahuje potřebné vlastnosti, které lze vidět v seznamu, a to zejména počet všech bloků, jejich cenu za kus, název a celkovou cenu. Pro generování uživatelského rozhraní dynamicky jsem vytvořil dva objekty typu prefab, které slouží jako šablona, která se vyplní daty a vloží do seznamu. Ve skriptu je hlavní metodou GenerateList(), která nejdříve iteruje enumeraci BuildingObjectsEnum a pro každou položku enumerace vytvoří nový MaterialListBlockItem. Do této enumerace jsem pro zjednodušení přidal dvě statické metody, a to ToFriendlyString() a ToFriendlyPriceInCZK(), které vrací na základě typu cenu, nebo popisek.

```
foreach (BuildingObjectsEnum item in Enum.GetValues(typeof(BuildingObjectsEnum)))
{
    MaterialListBlockItem newItem = new MaterialListBlockItem(item)
    {
        TotalPrice = 0,
        BuildingObjectCount = 0,
        PiecePrice = item.ToFriendlyPriceInCZK(),
        Name = item.ToFriendlyString(),
    };
    MaterialListBlockItems.Add(newItem);
}
```

V další části kódu proběhne sčítání všech objektů, a to opět dynamicky. Iterují se všechny položky globálního seznamu všech objektů, a pomocí metody GetBlockTypeFromList() je nalezena položka seznamu odpovídajícího typu a její proměnná s počtem bloků je inkrementována. Metoda GetBlockTypeFromList() přijímá jako parametr hodnotu z enumerace typů bloků a vrací MaterialListBlockItem ze seznamu který jí odpovídá. Při přidání nových bloků tedy není potřeba editovat žádný kód a přidávat podmínky např. do struktury switch, protože vše je snímáno dynamicky.

V závěru metody jsou iterovány všechny položky MaterialListBlockItem a dle jejich parametrů jsou vytvořeny a instantizovány herní objekty dle šablony zmíněné výše. Protože seznam obsahuje více položek dle stejné šablony, které jsou dynamicky generovány, pro přidělení jsem použil metodu Find(), která umožňuje najít potomky herního objektu pomocí jejich názvu. To mi umožnilo získat jednotlivé herní objekty, ze kterých se šablona skládá a přistoupit k jejich komponentám pomocí metody GetComponentInChildren().

```
newBlockItem.transform.Find("InputPrice").transform.GetComponent<InputField>().on
EndEdit.AddListener(delegate { OnEdited(item, newBlockItem); });
```

Za zmínku stojí vstupní pole ceny, kterému je přidělen delegát s metodou OnEdited(). Tato metoda se spouští, pokud uživatel změní výchozí hodnotu vstupního pole. Metoda OnEdited() přijímá jako parametr herní objekt celé položky, ale i MaterialListBlockItem, ze kterého je položka generovaná, což umožňuje absolutní kontrolu. Metoda přepočítá cenu a aktualizuje ji, a nakonec volá metodu RecalculateTotalBlockPrice(), která jak už z názvu vypovídá, přepočítá celkovou cenu za všechny stavební bloky a přepíše ji v záhlaví.

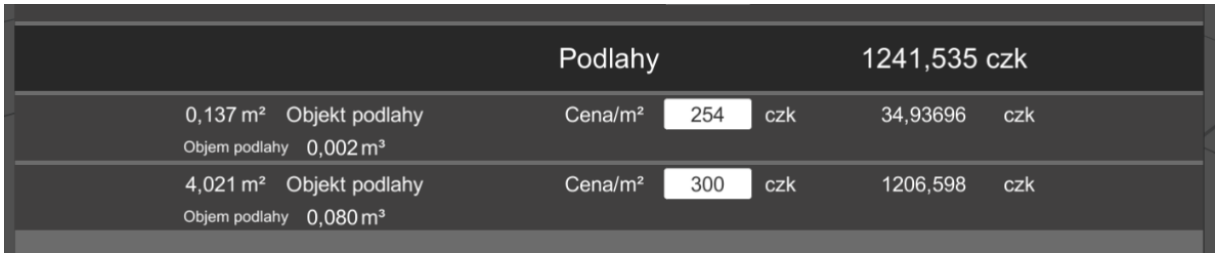
```
private void OnEdited(MaterialListBlockItem listItem, GameObject blockItem)
 {
     float newPrice =
float.Parse(blockItem.transform.Find("InputPrice").transform.GetComponent<InputFi
eld>().text);
     listItem.PiecePrice = newPrice;
     listItem.TotalPrice = newPrice * listItem.BuildingObjectCount;
```

```
blockItem.transform.Find("TextTotalPrice").transform.GetComponentInChildren<Text>
().text = "" + listItem.TotalPrice;
     RecalculateTotalBlockPrice();
 }
```

2.11.2 Výpočet objektů střecha a podlaha

2.11.2.1 Výpočet podlahy

Objekty podlahy jsou v seznamu separovány hlavičkou, která stejně jako hlavička pro jednotlivé bloky obsahuje celkovou cenu za všechny podlahy. V této části se vypočítává pro každý objekt podlahy jeho obsah a objem, kde obsah se používá pro výpočet celkové ceny. U každé podlahy je textové pole s výchozí cenou 300 korun za metr čtvereční, a tuto cenu je možné měnit. Při změně dynamicky přepočítána celková cena dané podlahy, ale také celková cena všech za všechny podlahy zobrazující se v hlavičce. Pro každý objekt podlahy, který uživatel ve scéně položil je vytvořena vlastní UI komponenta, která obsahuje její obsah.



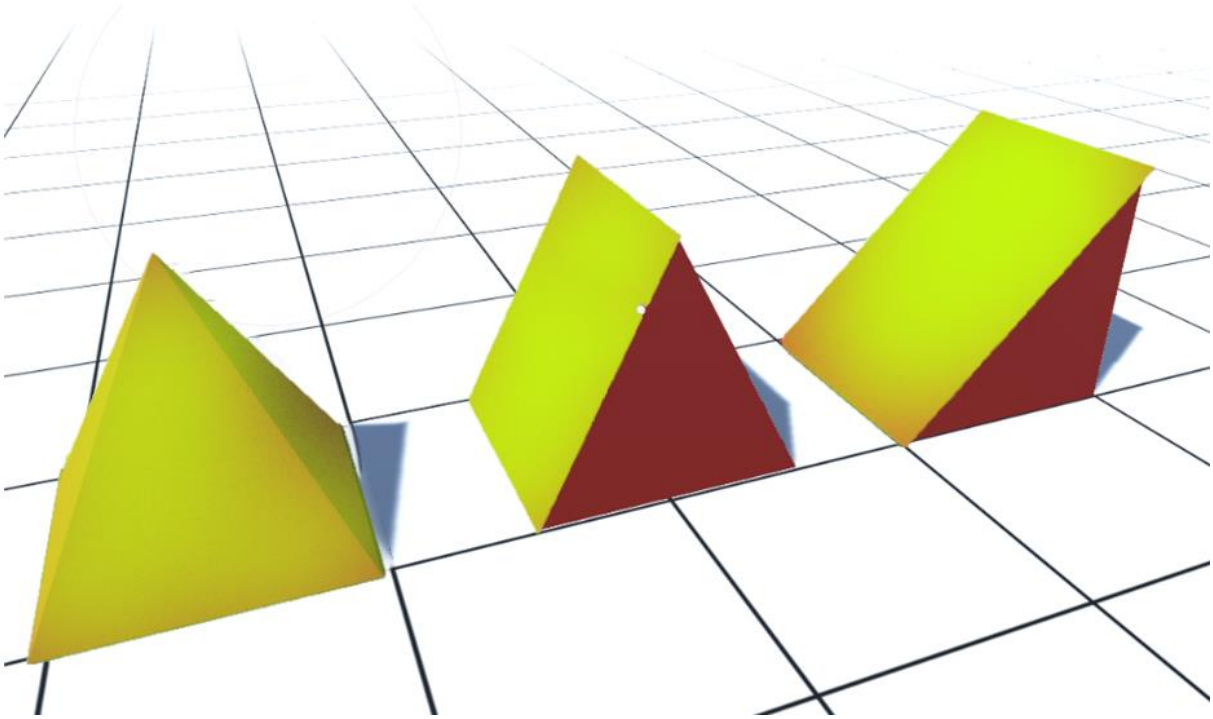
Obrázek 19: Soupis objektů typu podlaha

Tato funkcionalita je stejně jako výpočet soupisu jednotlivých bloků implementována ve třídě MaterialList.cs a má také svou vlastní pomocnou třídu MaterialListFloorItem.cs. V třídě MaterialList.cs je z hlediska podlah důležitá hlavně metoda RegenerateFloors(), která se spouští vždy při generování seznamu materiálu. Princip je velmi podobný jako výše popsaná generace soupisu jednotlivých bloků, kde je nejdříve vytvořena hlavička, která značí začátek seznamu podlah. Iteruje se globální seznam všech stavebních bloků, kde jsou nalezeny všechny podlahy a jsou podle nich vytvořeny objekty dle třídy MaterialListFloorItem.cs a jsou vloženy do seznamu. Tento seznam je poté iterován a pro každý objekt podlahy je vytvořen z objektu typu prefab objekt s jejími vlastnostmi. Tento objekt je na konci vložen do seznamu ListViewContentsFloors, který obsahuje již finální objekty uživatelského rozhraní. Je důležité je vložit do seznamu, aby mohly být později smazány při opětném generování. Logika pro přepočet celkové ceny za všechny podlahy a reakci na změnu v textovém poli zůstala totožná jako pro výpočet soupisu bloků. Pro výpočet obsahu a objemu jsem vytvořil primitivní metody CalculateFloorArea() a CalculateFloorVolume() které přijímají jako parametr proměnnou typu BuildingObject.

```
private float CalculateFloorVolume(BuildingObject item)
{
    GameObject gameObject = item.GameObject;
    if (gameObject != null)
    {
        float length = gameObject.transform.localScale.x;
        float width = gameObject.transform.localScale.z;
        float height = gameObject.transform.localScale.y;
        return length * width * height;
    }
    return 0;
}
```

2.11.2.2 Výpočet plochy střechy

Objekty střechy jsou zobrazeny totožným způsobem v grafickém rozhraní jako bloky a podlahy zmíněné výše. Kód pro jejich zobrazení je také velmi podobný, pouze z drobnými rozdíly. Je implementovaný v třídě MaterialList.cs a jako vlastní pomocnou třídu používá MaterialListRoofItem.cs. Při výpočtu obsahu střechy se objevila řada problémů, a to z důvodu více možných druhů střech a jejich rotací, kde existuje mnoho variací. Ve skriptu je nutné rozpoznat, o který typ střechy se jedná, na výběr máme ze tří možností: stanová, sedlová a pultová. Například u střechy stanové je potřeba vypočítat její obsah, ale bez podstavy. U střechy sedlové je potřeba vypočítat obsah sedla, ale bez podstavy, a navíc bez krajních stěn, které se obvykle neobkládají a jedná se o fasádu. V případě střechy pultové je nutné vypočítat pouze obsah „rampy“, a vynechat podstavu a krajní stěny. Na následujícím obrázku jsou pro představu zeleně vyznačené žádané oblasti pro výpočet obsahu.



Obrázek 20: Ukázka žádaných ploch pro výpočet obsahu střechy

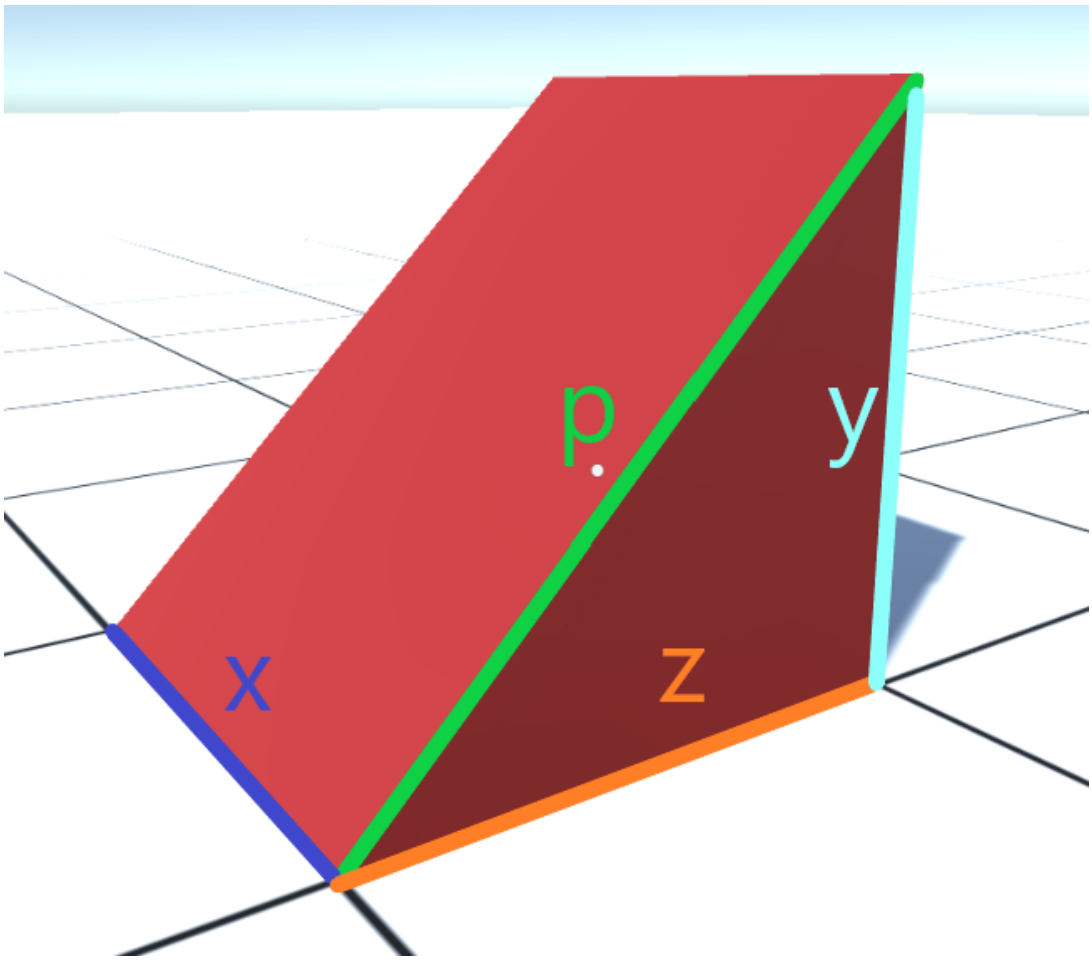
V kódu tyto výpočty zařizuje metoda CalculateRoofArea(), která přijímá jako parametr proměnnou typu BuildingObject, která představuje objekt střechy. Nejdříve jsou vytvořeny proměnné x, y, z, které obsahují hodnoty šířku, výšku a délku objektu a proměnná typu mesh[7]. Ve struktuře switch je rozhodnuto, o který typ střechy se jedná, protože třída BuildingObject v sobě uchovává hodnotu enumerace BuildingObjectsEnum. Pokud se jedná o střechu stanovou, použil jsme přístup, kde jsem přistoupil ke komponentě mesh objektu, která se skládá z vektorů a trojúhelníků. V unity se většina objektů skládá ze sítě trojúhelníků, které se skládají vždy ze tří vektorů, které na sebe navazují. Tyto trojúhelníky je možné si zpřístupnit a vypočítat jejich obsah. To se děje v cyklu for, kde se iterují všechny hrany trojúhelníků a z každé je vytvořen vektor. V každém cyklu se tedy vytvoří trojúhelník ze tří vektorů, a je vypočítán jeho obsah pomocí metody Vector3.Cross(), která představuje vektorový součin. Vektorový součin vrací magnitudu dvou vektorů, kterou si lze představit jako rovnoběžník, a proto je nutné magnitudu vydělit dvěma, abychom získali obsah trojúhelníku. Potom, co získáme veškeré obsahy trojúhelníků objektu, odečteme od nich obsah podstavy, který je vypočítán jednoduše vynásobením lokálních rozměrů objektu střechy x a z. Díky tomuto přístupu je výpočet univerzální pro jakoukoliv stanovou střechu. [21]

```
for (int i = 0; i < triangles.Length; i += 3)
```

[7] Trojrozměrný model objektu, skládající se z geometrických trojúhelníků.

```
{
    Vector3 vertex1 =
    roofObject.transform.TransformPoint(vertices[triangles[i]]);
    Vector3 vertex2 = roofObject.transform.TransformPoint(vertices[triangles[i +
    1]]);
    Vector3 vertex3 = roofObject.transform.TransformPoint(vertices[triangles[i +
    2]]);
    float triangleArea = Vector3.Cross(vertex2 - vertex1, vertex3 -
    vertex1).magnitude * 0.5f;
    surfaceArea += triangleArea;
}
```

Ostatní typy střech již ale nebylo možné tímto způsobem vypočítat, a musíme se uchýlit k jiným metodám. Jako další příklad budeme uvažovat střechu pultovou, protože z principu, který jsme použili pro její výpočet jsme využili i u střechy sedlové. Pro výpočet použijeme Pythagorovu větu, kde vypočítáme délku přepony, která představuje jednu stranu obdélníku představující požadovaný obsah. Když tuto přeponu vynásobíme s šířkou podstavy získáme tento obsah. Tento scénář funguje bohužel pouze u střech s čtvercovou podstavou, a u střech s obdélníkovou podstavou musíme rozlišovat šířku nebo délku dle rotace. Díky tomu, že objekty pultové střechy již měly přidělený tag dle jejich rotace, toto nebylo příliš obtížné a stačilo použít strukturu switch která dle případu pouze prohodí hodnoty x a z.

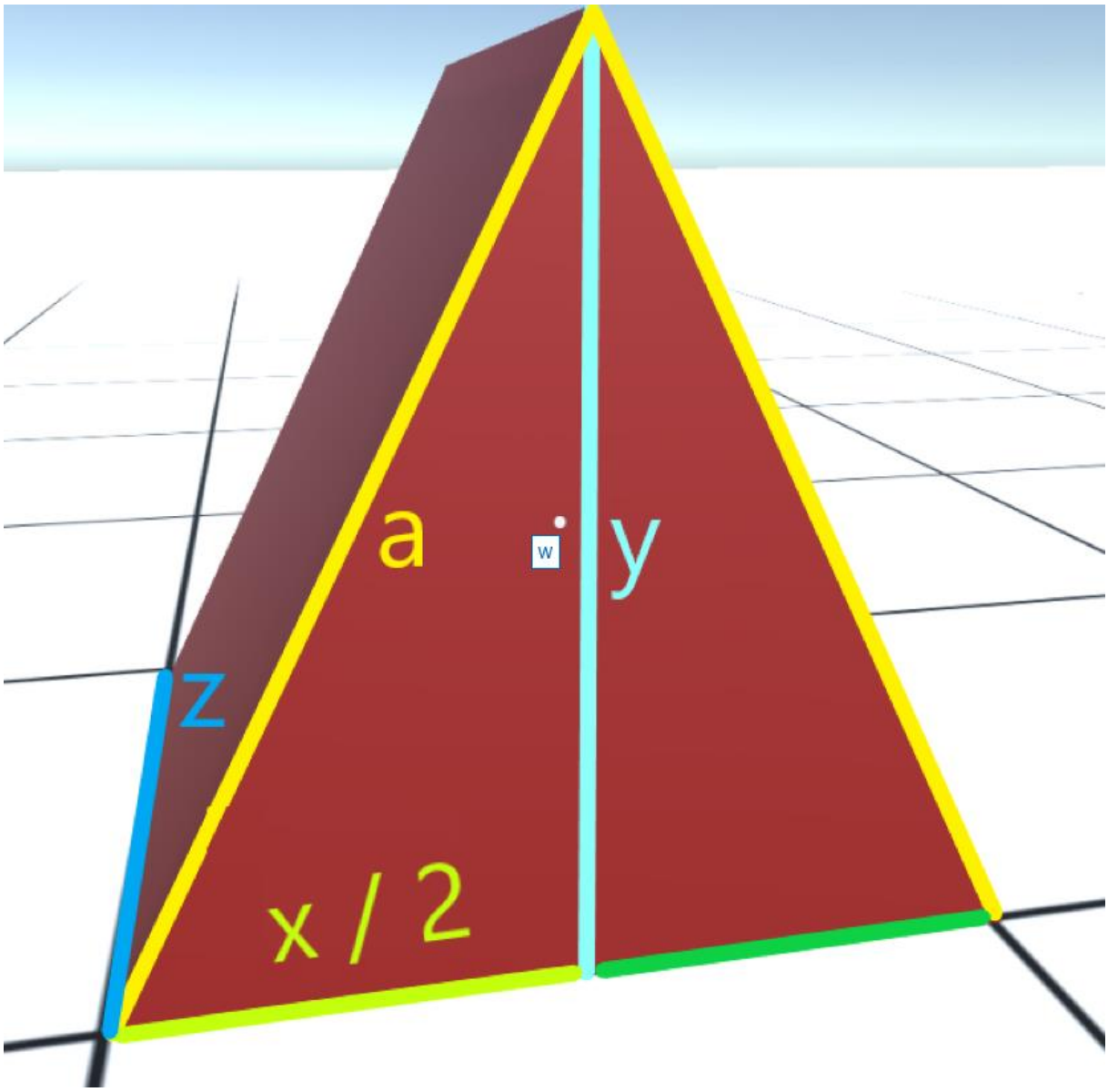


Obrázek 21: Vizualizace výpočtu obsahu pultové střechy

Pro snadnější vysvětlení jsem vytvořil jednoduchý obrázek pomocí stavebního simulátoru. Zelená strana p vznikne použitím Pythagorovy věty na strany y a z. Poté se vynásobí strana p se stranou x a výsledek je požadovaný obsah střechy, který se dál zpracovává. Je nutné zmínit, že se dle rotace objektu mění osa, kterou se přepona p násobí, a násobí se buď osou x nebo v druhém případě osou z.

```
counterRoofArea = Mathf.Sqrt(x * x + y * y) * z;
```

Jak už jsem výše zmínil, pro výpočet obsahu sedlové střechy můžeme použít podobnou metodu jako pro výpočet střechy pultové. Opět vypočítáme obsah pomocí výpočtu přepony, kterou vynásobíme se stranou podstavy, ale místo strany podstavy použijeme pouze polovinu strany, a poté celkový obsah vynásobíme dvěma. Objasnit by nám to mohl následující obrázek.



Obrázek 22: Vizualizace výpočtu obsahu sedlové střechy

Z obrázku je patrné, proč je nutné vynásobit výsledek dvěma. Opět se vypočítá žlutá přepona a, použitím výšky objektu y a polovinou osy x nebo z. To záleží na rotaci objektu, která v tomto případě má pouze dva stavy, které je možné zjistit z tagu objektu střechy, který má každá střecha od své inicializace.

```
counterGableArea = (Mathf.Sqrt((x / 2 * x / 2) + y * y) * z) * 2;
```

Tyto výsledky jsou zpracovávány v grafickém rozhraní a jsou použity pro výpočet celkové ceny za střešní krytinu, v metrech čtverečních.

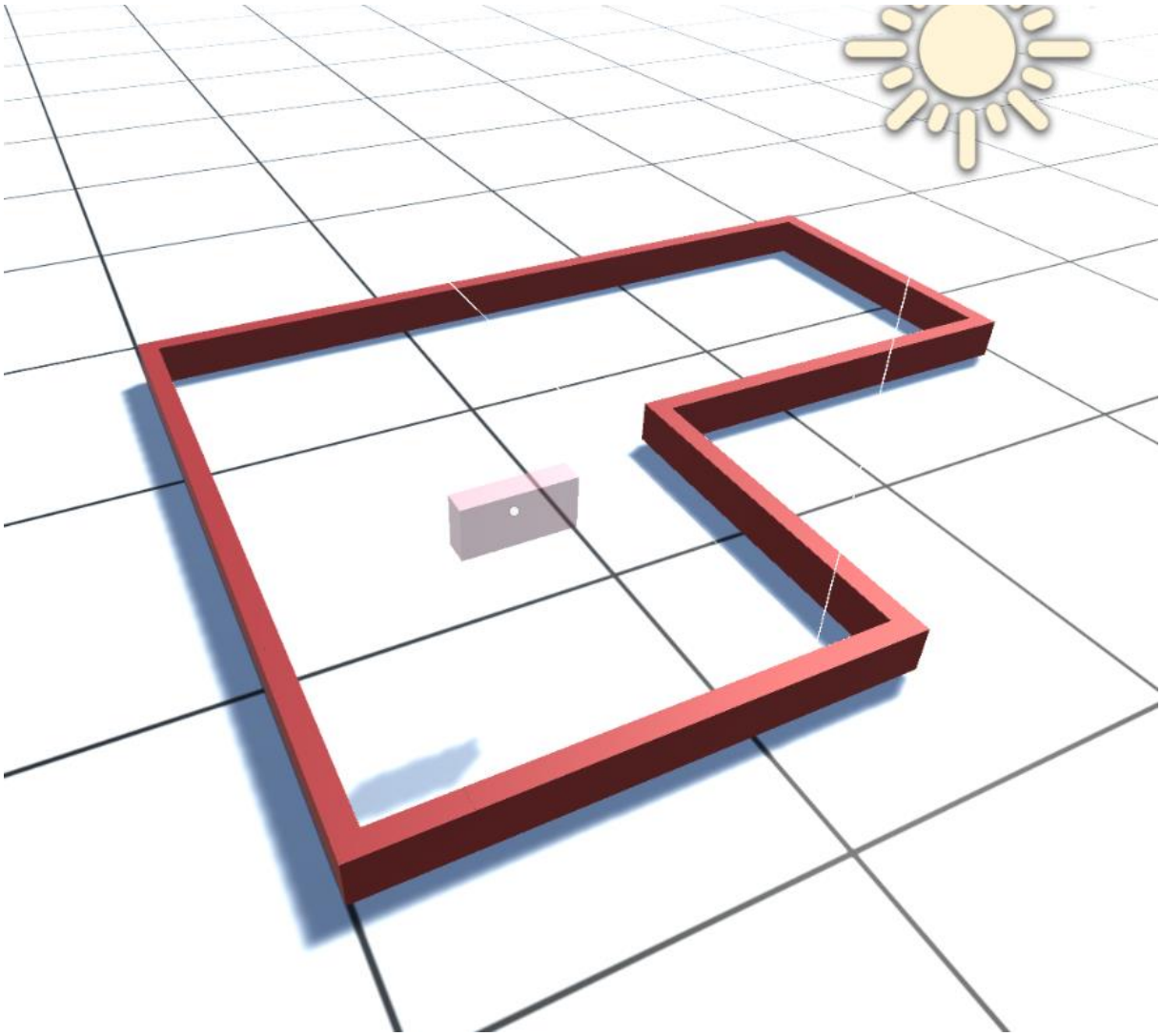
2.12 Generace půdorysu

Stavební simulátor nabízí možnost generování půdorysu postavených staveb a struktur, včetně podlah a střech. Je přístupný z aplikačního menu kliknutím na tlačítko „Generovat půdorys“.

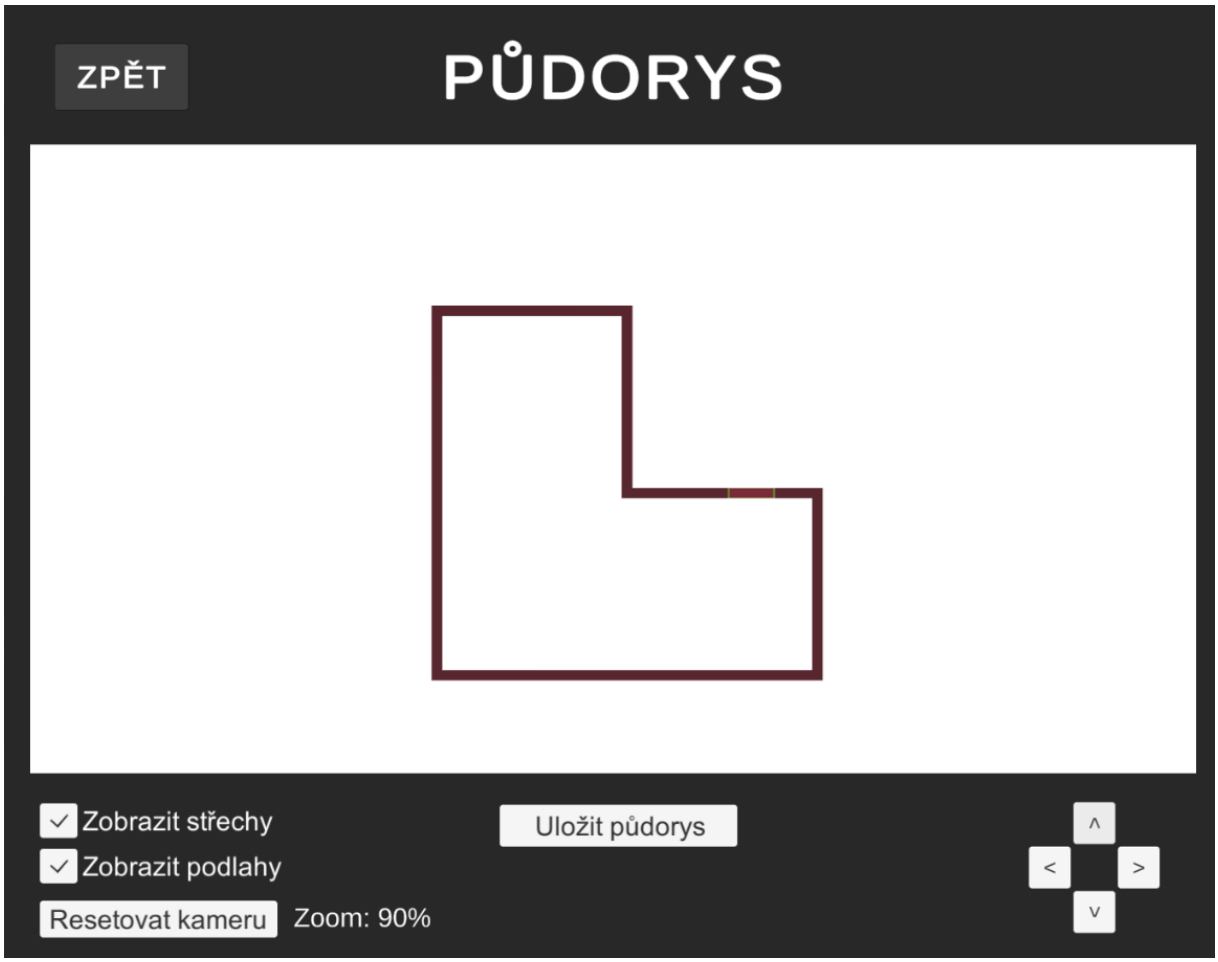
Při zapnutí je uživateli zobrazeno nové rozhraní s ovládacími prvky, které umožňuje uživateli přizpůsobit půdorys dle jeho potřeb. Uprostřed je hlavní komponenta, která vykresluje půdorys v reálném čase. V dolním levém rohu je možné pomocí dvou zaškrtávacích políček zakázat nebo povolit vykreslování střech či podlah, protože to někdy může být nežádoucí. Také je možné obnovit kameru do výchozí pozice Vektoru3(0,0,0), který označuje střed stavební plochy. Pomocí kolečka myši je možné půdorys přiblížit či oddálit, a aktuální hodnota přiblížení je zobrazena vedle tlačítka k obnově kamery.

V dolní prostřední části uživatelského rozhraní se nachází jedna z nejdůležitějších částí, a to možnost vyexportovat půdorys do formátu Portable Network Graphics (PNG) pro uložení do počítače. Uložení je možné spustit tlačítkem, což otevře klasický Windows dialog ve kterém je možné vybrat cestu pro uložení. Pokud bylo uložení úspěšné, zobrazí se hláška informující uživatele, společně s cestou, kam se podařilo soubor uložit. V opačném případě se zobrazí chybová hláška, z jakého důvodu nebylo možné soubor uložit.

V pravé dolní části se nachází ovládání pohybu kamery. Skládá se z čtyř tlačítek, které je možné intuitivně ovládat kameru směrem nahoru, dolů, doleva a doprava. Pro ukázku jsem vytvořil pomocí nástroje stěna jednoduchou strukturu ze standardních cihel.



Obrázek 23: Ukázka primitivního půdorysu



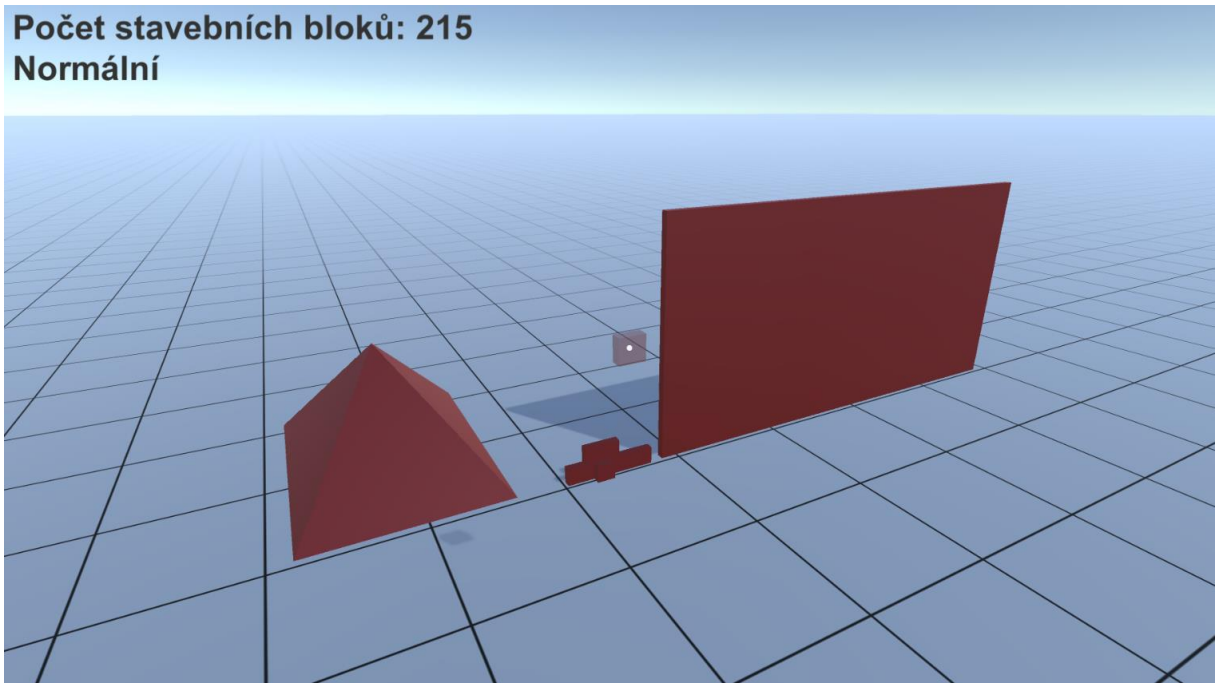
Obrázek 24: Uživatelské rozhraní generace půdorysu

Skript na generování půdorysu se nazývá FloorPlan.cs. Hlavní myšlenkou bylo použití ortografické kamery, která bude směřovat na stavební plochu. Bylo nutné použít kameru ortografickou, která snímá uniformní velikosti objektů a ignoruje hloubku scény. Druhým typem kamery je kamera perspektivní, která zachycuje také hloubku scény a úhlové zkreslení. Perspektivní kamera tedy nebyla v této situaci žádaná. V metodě Update() jsou kontrolovány pomocí třídy Input.GetAxis() pohyby kolečkem myši dopředu a dozadu, a dle toho se patřičně aktualizuje velikost kamery. Přiblížení není implementováno pomocí škálování herního objektu, ale pomocí vlastnosti kamery s názvem orthographicSize. Změnou této vlastnosti určujeme, jak velké pole kamera snímá, a tím je tedy zároveň možné regulovat i přiblížení. Protože jsem chtěl v aplikaci zobrazovat aktuální hodnotu přiblížení, a čím menší je velikost kamery tím je větší přiblížení, bylo nutné implementovat škálu. Tuto škálu je poté nutné invertovat, abychom získali požadovanou hodnotu přiblížení. To je možné použitím vzorce minimum + maximum – hodnotaPriblizeni. Tlačítka pro pohyb kamery jsou implementovány čtyřmi příslušnými metodami OnBtnClicked(), kde každá z nich mění vektorovou pozici kamery o jeden metr. To je dosaženo získáním hodnot X, Y, Z z aktuálního vektoru pozice a

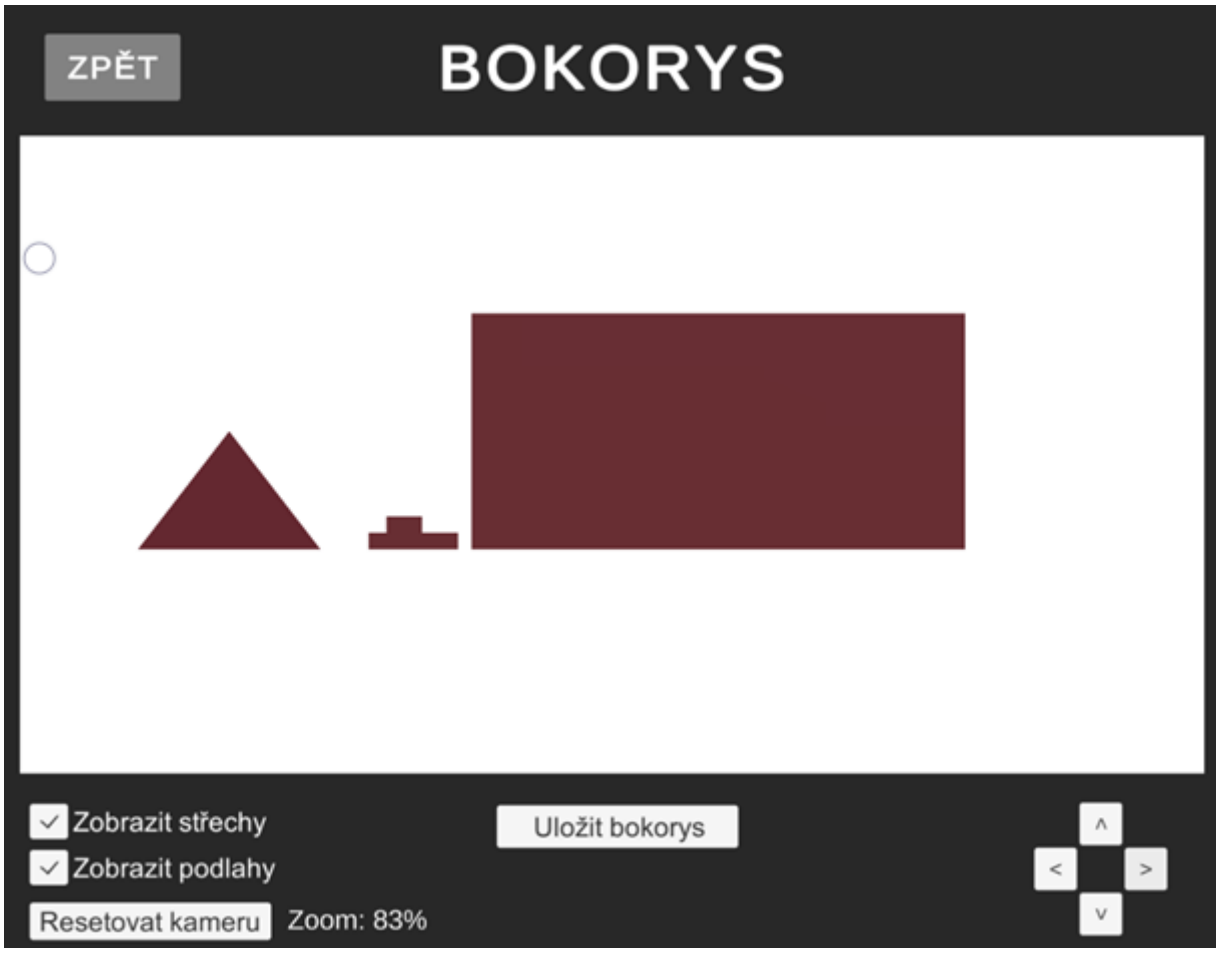
přidělením nového vektoru, kde jsou použity tyto hodnoty, ale s příslušným inkrementem/dekrementem. Zobrazení a skrytí střech/podlah zaškrtávacími políčky je poměrně jednoduché, a to iterací globálního seznamu všech stavebních objektů, a pokud se jedná o objekt typu střecha/podlaha je dočasně skryt. Nejzajímavější metodou je SaveFloorplanButton(), která naslouchá na stisk tlačítka „Uložit půdorys“. Použil jsem postup převodu RenderTexture do Texture2D, kde Texture2D je poté možné přímo převést do obrázku a uložit. Nejdříve je tedy vytvořen nový objekt Texture2D, poté nastavena aktivní vykreslovací textura vlastností RenderTexture.active. Tím řekneme dvourozměrné textuře odkud má načíst pixely. Pixely načteme metodou ReadPixels(), které stačí vložit šířku a výšku obrázku jako parametry. Po zavolání metody Apply() jsou aktuální pixely z vykreslované textury uloženy do textury a je možné s nimi dále pracovat. Další částí je převod na pole bajtů, metodou EncodeToPNG(). Je nutné zmínit, že metody ReadPixels(), Apply() a EncodeToPNG() jsou součástí každé třídy Texture2D, odkud je nutné je zavolat. Pro uložení je použita třída File a metoda WriteAllBytes() která přijímá dva parametry, cestu a pole bajtů. Pro odchytávání chyb jsem použil blok try-catch ve kterém patřičně měním popisek stavu úspěšnosti uložení.[19]

2.12.1 Bokorys a nárys

Aplikace nabízí uživateli vygenerovat nejen půdorys, ale i bokorys a nárys. Tyto funkce jsou dostupné z aplikačního menu, a každá z nich má své vlastní grafické rozhraní velmi podobné rozhraní půdorysu. Jsou implementované v třídách SideView.cs a FrontView.cs, kde implementace se velmi podobá třídě FloorPlan.cs



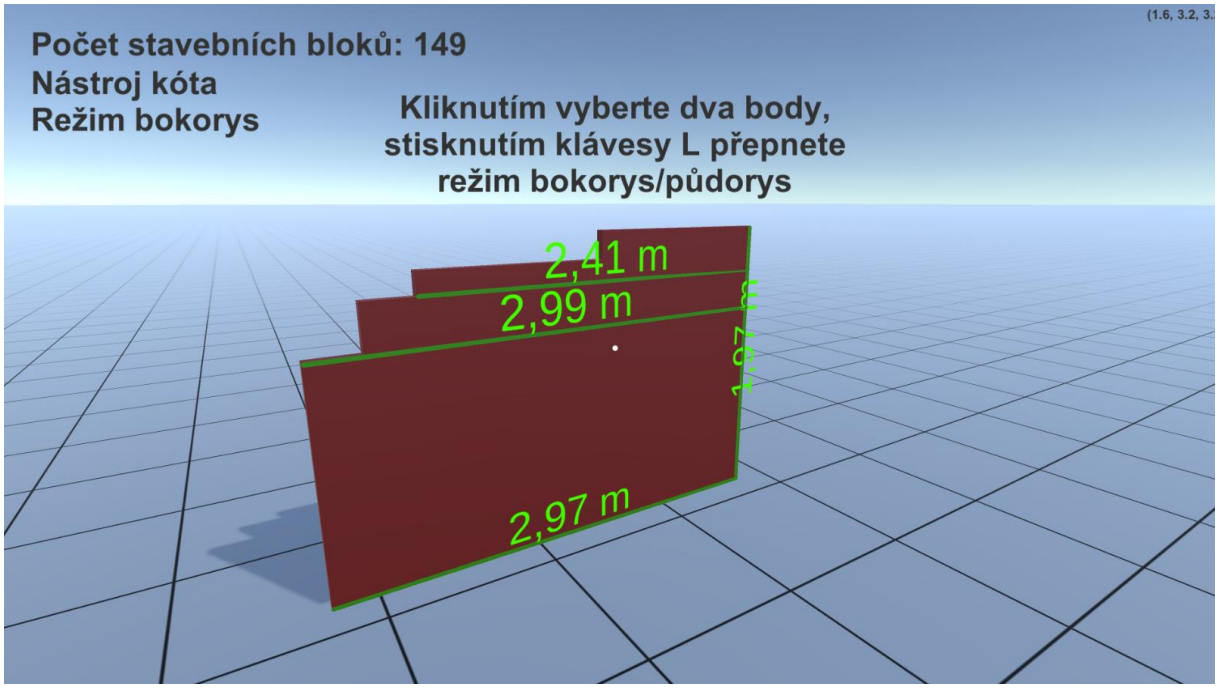
Obrázek 25: Ukázka funkce bokorys č. 1



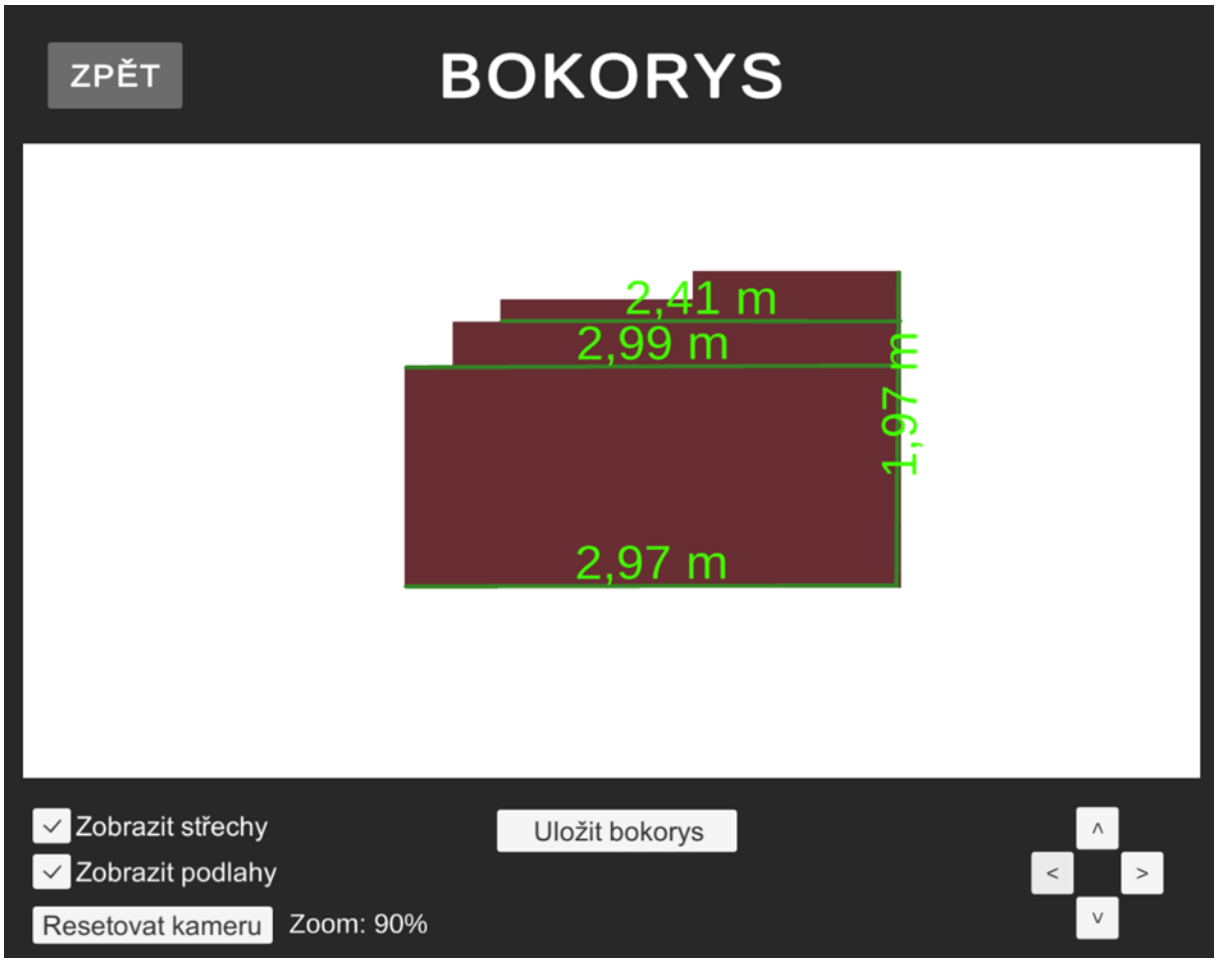
Obrázek 26: Ukázka funkce bokorys č. 2

2.13 Nástroj kóta

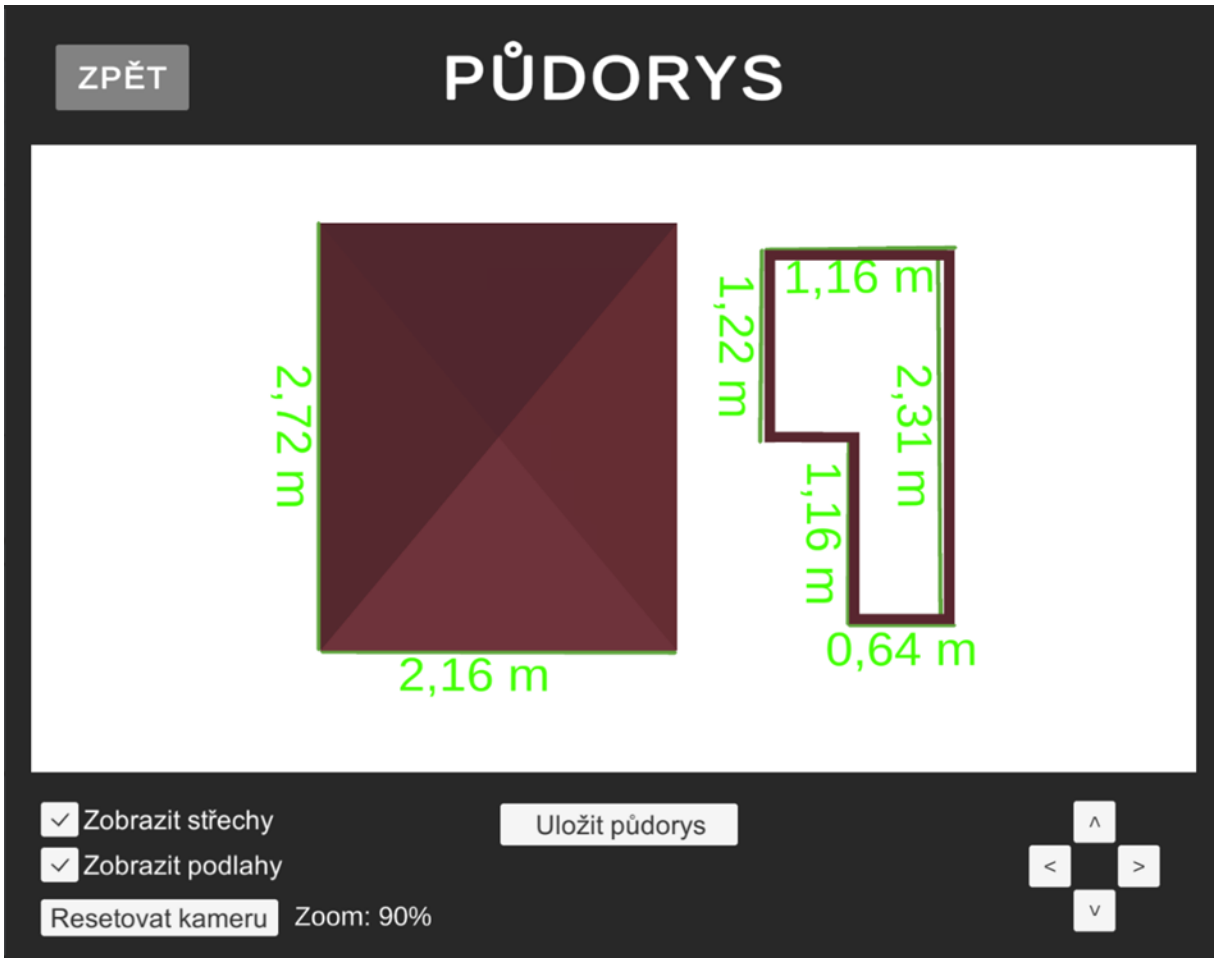
Aplikace nabízí uživateli výkresy libovolně okótovat, a to nástrojem kóta, který se spouští stiskem tlačítka „K“. Uživatel vybere dva body kliknutím levého tlačítka myši a program automaticky vypočítá délku kóty a vytvoří objekt který ji představuje. Okótovat je možné jakýkoliv objekt a libovolnou vzdálenost, vše závisí na dvou bodech, které vybere uživatel. Protože se nacházíme v trojrozměrném prostoru, kóta nabízí dva režimy, které lze přepnout tlačítkem „L“. Na těchto režimech závisí pozice popisku kóty. V režimu bokorys bude popisek kolmý ke kótě směrem nahoru, v režimu půdorys bude kolmý směrem do strany a rovnoběžný s podlahou. Při kótování půdorysu je tedy dobré zvolit kóty v režimu půdorys, aby byly při vygenerování půdorysu viditelné. Kóty je možné mazat stisknutím pravého tlačítka myši.



Obrázek 27: Ukázka nástroje kóta, režim bokorys



Obrázek 28: Ukázka nástroje kóta v generovaném bokorysu



Obrázek 29: Ukázka nástroje kóta v generovaném půdorysu

Tento nástroj je implementovaný ve třídě QuotationTool.cs. Nejdříve se pomocí třídy RayCast a jejích metod zjistí pozice dvou bodů, kam se uživatel dívá při stisknutí tlačítka myši. Vytvoří se instance objektu typu linePrefab, což je předvytvořený objekt s komponentou zvanou LineRenderer. Tato komponenta se v Unity používá pro vykreslování čar. Objektu čáry jsou přiděleny vektory pozic bodů, které vybral uživatel. Podobným způsobem se vytvoří instance textového objektu, který bude obsahovat délku mezi začátkem a koncem čáry. Tuto vzdálenost je možné vypočítat pomocí metody Vector3.Distance(). Jsou kontrolovány dva režimy, bokorysu a půdorysu. Podle režimu je objektu textu přidělena patřičná rotace. Zajímavým problémem bylo vypočítání rotace textu, když se kóta vytvoří. Nastávala totiž situace, kdy uživatel vytvoří kótu a text je otočen o 180° směrem od uživatele. Toto jsem vyřešil vytvořením pomocné metody IsTextFacingPlayer(), která přijímá jako parametr textový objekt a vrací booleovské hodnoty true nebo false. K výpočtu používá metodu Vector3.Dot(), která vrací skalární součin dvou vektorů. Pokud je součin menší než 0, text je otočený směrem k uživateli.

```
bool IsTextFacingPlayer(Transform textTransform)
{
    Vector3 directionToPlayer = (mainCamera.transform.position -
    textTransform.position).normalized;
    float dotProduct = Vector3.Dot(textTransform.forward, directionToPlayer);
    return dotProduct < 0;
}
```

2.14 Nástroj pro tvorbu stavebních otvorů, oken a dveří

Aplikace obsahuje nástroj stavební otvor, který se spouští stiskem tlačítka „Z“ a má vlastní grafické rozhraní. Je užitečný pro tvorbu objektů, které představují otvory ve stěnách. Je s ním také možné tvořit objekty představující okna a dveře. Uživatel je po stisknutí tlačítka Z vyzván k výběru dvou bodů stěny. Body stačí vybrat z jedné strany stěny. Po vybrání druhého bodu je zobrazeno uživatelské rozhraní, kde uživatel může změnit texturu objektu v rozevíracím seznamu, potvrdit svůj výběr nebo ho zrušit. Objekty je možné smazat kliknutím pravého tlačítka myši. Pokud uživatel vybere dva body, mezi kterými nelze vytvořit otvor, je o tom informován chybovou hláškou ve středu obrazovky.

Obrázek 30: Ukázka rozhraní nástroje pro tvorbu otvorů, oken a dveří

Obrázek 31: Ukázka objektů otvor, čerchovaný otvor, okno a dveře

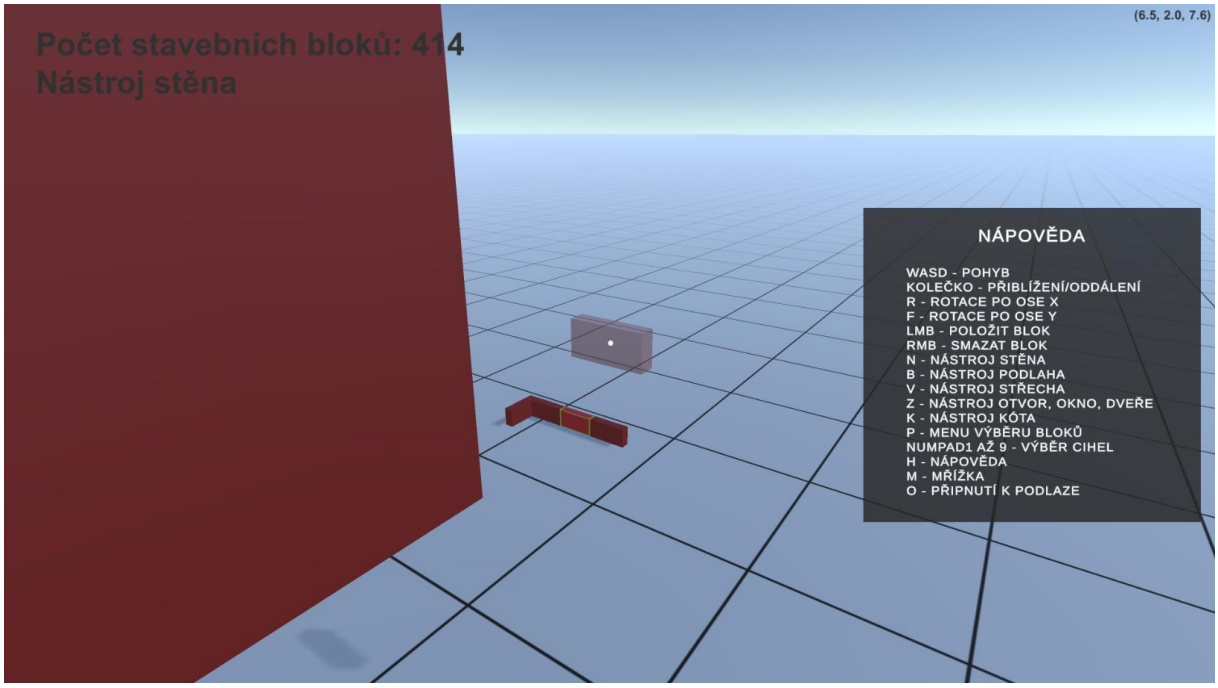
Tento nástroj je implementovaný v třídě HoleTool.cs. Hlavní logika probíhá v metodě update, kde skript kontroluje stisknutí levého tlačítka myši. Při stisku vyšle objekt typu raycast, který zjistí, na který objekt se uživatel aktuálně dívá. Po získání dvou bloků vypočítá střed mezi nimi, kam vloží klon stavební cihly, na kterou dopadl druhý paprsek objektu raycast. Klon stavební cihly představuje objekt otvoru. Tomuto objektu je patřičně upravena šířka, a výška dle dvou vybraných bodů. Jeho hloubka je automaticky nastavena jako šířka dané stěny, na které se otvor generuje. Objekt otvor ve skutečnosti nevytváří otvor ve zdi, jen vytváří objekt, který ho představuje. Je to z důvodu extrémní komplikace takového řešení. Jelikož se zeď typicky skládá z více cihel, které jsou na sebe naskládané, bylo by potřeba ze dvou bodů určit o jaké cihly se jedná a jaký přesný výřez by musel být odebraný. Stavební cihly se v této aplikaci skládají z objektu typu Mesh Filter, což je svým způsobem trojrozměrný model skládající se z vektorů, hran a trojúhelníků. U každé cihly by tedy bylo nutné upravit její objekt typu Mesh Filter, což je čistě matematická záležitost vyžadující velmi pokročilé znalosti v 3D modelování a matematice. Pro účely této aplikace jsem tedy zvolil místo editace objektů Mesh Filter strategii objektu, který otvor nevytváří, ale pouze představuje.

2.15 Resetování scény

Reset scény je možné spustit v aplikačním menu, kde má vyhrazené tlačítko. Stisk tlačítka vyvolá metodu OnResetSceneButtonClick() v hlavním skriptu MainScript.cs, což smaže veškeré stavební objekty z globálního pole a vyresetuje pozici uživatele.

2.16 Rozhraní nápovědy

Aplikace obsahuje vlastní nápovědu, aby uživatel nemusel neustále otevírat přiložený dokument s nápovědou. Jedná se o objekt typu canvas, který se spouští klávesou „H“. Po stisknutí se zobrazí překrytí, které je opět možné klávesou H vypnout. Užitečné je, že nápovědu může uživatel vyvolat kdykoliv z jakékoliv části aplikace.



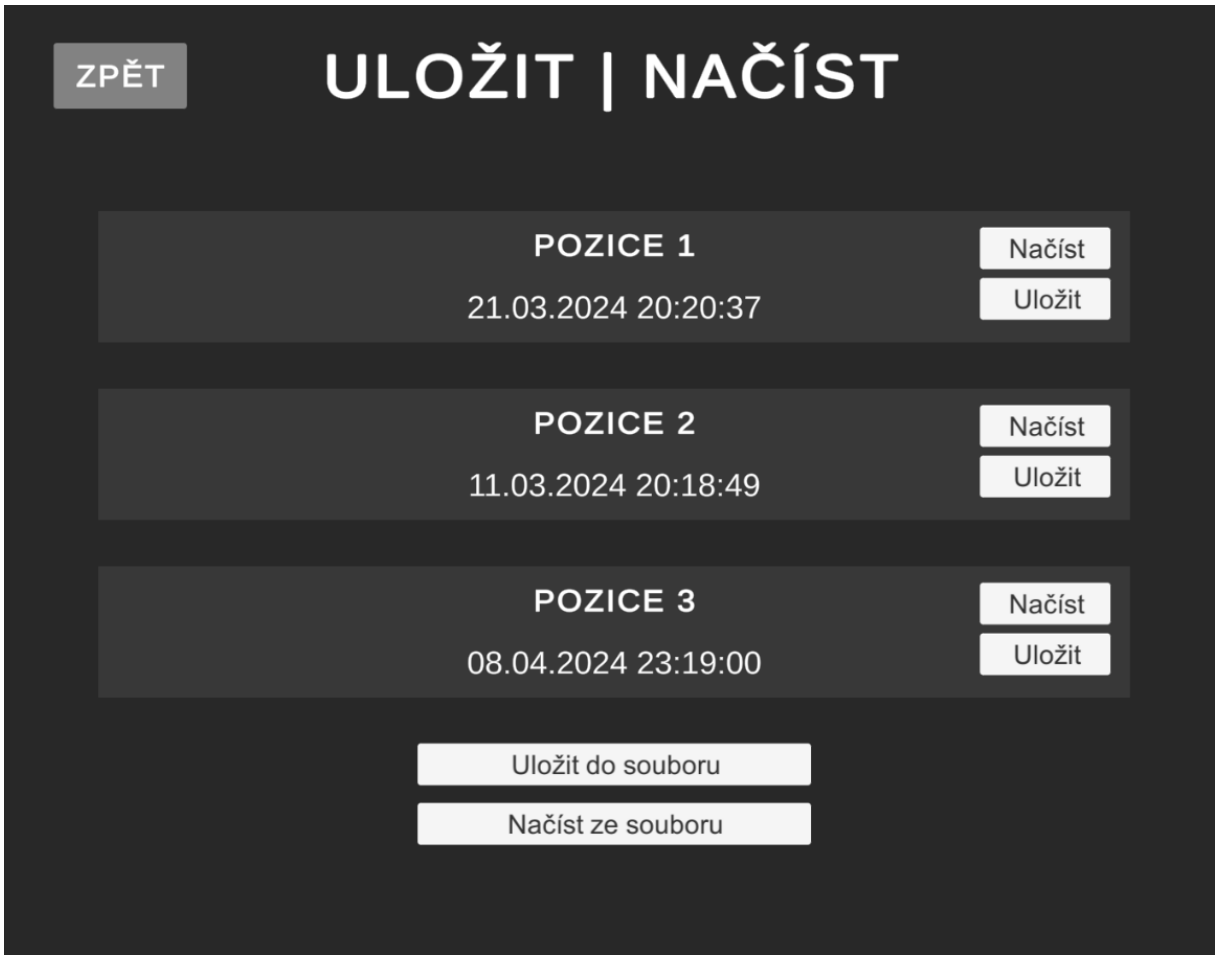
Obrázek 32: Ukázka překrytí nápovědy

Implementace nápovědy je uložena ve skriptu HelpUI.cs, který obsahuje poměrně primitivní kód, který naslouchá stisku klávesy a zapíná nebo vypíná objekt typu canvas.

```
void Update()
{
    if (Input.GetKeyDown(KeyCode.H))
    {
        if (isHelpEnabled)
        {
            canvas.enabled = false;
            isHelpEnabled = false;
        }
        else
        {
            canvas.enabled = true;
            isHelpEnabled = true;
        }
    }
}
```

2.17 Ukládání a načítání

Aplikace umožňuje ukládat stavební scény do třech předdefinovaných pozic, které ukládání a načítání značně usnadňují, protože uživatel nemusí hledat uložené soubory v průzkumníkovi souborů, stačí kliknout na tlačítko. Při uložení do pozice se změní její popis na datum uložení. Tři pozice nemusí být dostačující, proto navíc aplikace nabízí dvě tlačítka, kde je možné pomocí dialogu Windows jednoduše vybrat cestu pro uložení nebo načtení. Načítat a ukládat je možné z hlavní scény aplikace. Načíst je ale možné i z hlavního menu při prvním spuštění aplikace, pro rychlé načtení rozpracované scény.



Obrázek 33: Menu ukládání a načítání stavební scény

Implementace byla komplikovanější než ostatní části této práce, hlavně z důvodu, že objekty typu GameObject není možné přímo serializovat[8]. Hlavní logiku obsahuje skript SaveLoad.cs.

```
[System.Serializable]
public class BuildingObjectSerializable
{
    public string prefabPath;
    public float[] position;
    public float[] rotation;
    public float[] scale;
    public BuildingObjectsEnum objectType;
    public DateTime dateSaved;
}
```

Protože stavební bloky jsou typu GameObject, který není možné implicitně serializovat, bylo nutné vytvořit pomocnou třídu BuildingObjectSerializable, která obsahuje proměnné, které jsou vhodné pro serializaci. Z ukázky kódu lze vyčíst proměnnou prefabPath. Použil jsem strategii, ve které aplikace podle typu objektu, který ukládá vygeneruje cestu do složky „Prefabs“. Při deserializaci[9] je možné tuto cestu vyčíst, vytvořit nový objekt typu GameObject a přiřadit mu

[8] Převod datové struktury typicky do binárního formátu, pro snadné ukládání a načítání.
[9] Převod binárních dat zpět do podoby datové struktury.

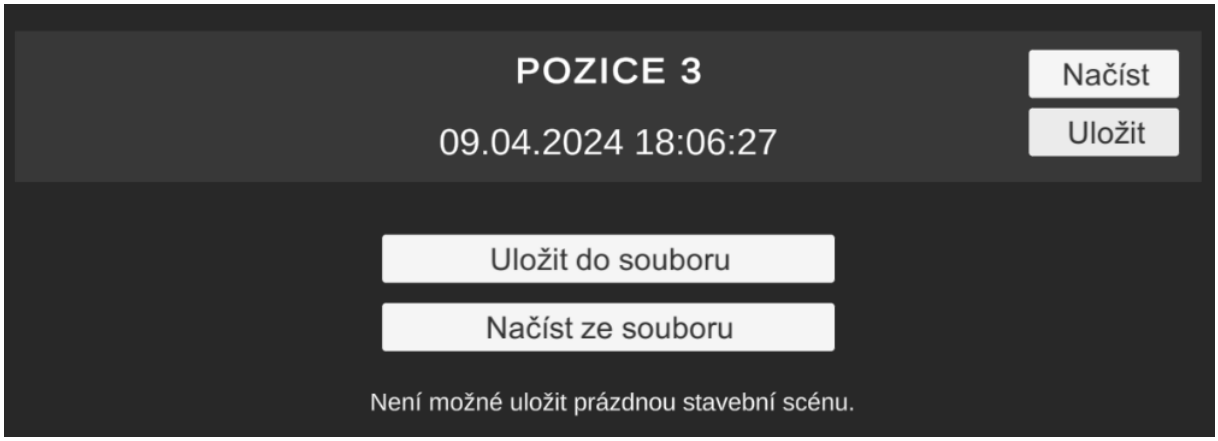
parametry dle objektu typu prefab vyčteného z cesty. Pomohl jsem si vytvořením statické metody ToPrefabPath() v enumeraci BuildingObjectsEnum, která vrací řetězec cesty k danému objektu.

Při ukládání do předdefinovaných pozic se vytvoří složka „Saves“ v pracovním adresáři aplikace. Do ní se poté pomocí třídy BinaryFormatter serializuje seznam všech stavebních objektů v dané scéně. Tento seznam připravuje metoda SaveGameObjects(), která seznam všech stavebních objektů převede na serializovatelné objekty typu BuildingObjectSerializable, které jsem zmínil výše.

```
List<BuildingObjectSerializable> buildingObjectSerializables = SaveGameObjects();
```

V případě načítání se opět používá třída BinaryFormatter a načte se seznam objektů typu BuildingObjectsSerializable. Smaže se veškerý obsah scény, a je znovu načten z deserializovaného seznamu, což zajišťuje metoda LoadGameObjects(). Využívá princip načtení objektu typu prefab pomocí cesty, který jsem zmínil výše.

Pokud uživatel načítá nebo ukládá prázdnou scénu, nebo načítá soubor který neexistuje, je o tom upozorněn zprávou, která se zobrazí po dobu tří vteřin.



Obrázek 34: Ukázka chybové hlášky při uložení prázdné scény

ZÁVĚR

Aplikace stavební simulátor uživateli umožňuje navrhnout hrubou stavbu. Je možné stavět z výběru stavebních cihel a tvárnic jednotlivě pomocí přichytávání, nebo pomocí nástroje vytvářet celé stěny. V hrubé stavbě je možné vytvořit si objekty jako podlahy, střechy, okna, dveře a stavební otvory pro účely výpočtu materiálů, nikoliv pro tvorbu interiéru nebo podkroví. Nástroj kóta umožňuje jakékoliv objekty okótovat a zjistit vzdálenosti, které se propisují do půdorysu, bokorysu a nárysu, které aplikace generuje. V soupisu materiálu je přehled všech stavebních objektů ve scéně a jejich ceny. Projekty je možné libovolně načítat a ukládat. Aplikace je určena pro domácí kutily na tvorbu jednoduchých staveb, například pergol a kůlen, kde vyniká svou jednoduchostí. Není vhodná pro komplikované projekty. Pokud ale uživatel přibližně ví, jakou stavbu by si představoval, je schopný vytvořit hrubou stavbu například pergoly řádově během minut, včetně celkové ceny za krytinu, dlažbu a stavební cihly.

Jelikož je aplikace určena na hrubou stavbu pro kutily, má své limitace. Například nástroj podlaha generuje podlahu libovolné kvádrové velikosti a spočítá její obsah s cenou dlažby, ale není možné si navrhnout její skladbu. Obdobně je to s nástrojem střecha, který je velmi užitečný pro rychlé vytvoření různých druhů střech, ale slouží primárně k vizualizaci a výpočtu ceny za krytinu, nelze si navrhnout nosnou konstrukci střechy. Zároveň nedisponuje žádným systémem pro výpočet zatížení nebo nosností struktur.

Aplikace má dobrý základ, a je naprogramovaná tak, aby nebylo těžké ji rozšířit. Určitě by bylo možné přidat další stavební bloky, nebo nástroj, který by uživateli umožnil vytvořit si vlastní knihovnu cihel, které by se ukládaly do souboru. Zajímavým rozšířením by mohl být nástroj plot, který by generoval mezi určitými body dle parametrů sloupky, sledoval jejich počet, přibližný objem betonu potřebný pro zabetonování a délku pletiva. Dalším rozšířením by mohl být nástroj pro automatický výpočet objemu omítky, spojovacího materiálu nebo lepidla. Aplikaci bych rád dál vyvíjel jako koníček, jelikož mě její implementace bavila a poměrně solidně mě seznámila s prostředím Unity. Vzhledem k jedinečnému stylu aplikace, kde spojuje prvky typicky používané ve videohrách s prvky z modelovacích programů by se dala označit jako díra na trhu, nepodařilo se mi najít aplikaci, která by se provedením podobala této.

POUŽITÁ LITERATURA

[1] History of Sketch Up. *MasterSketchUp* [online]. 2011, 17. 10. 2011 [cit. 2024-01-04]. Dostupné z: https://mastersketchup.com/history-of-sketchup/

[2] What is SketchUp. *How-To Geek* [online]. 2018, 4. 9. 2018 [cit. 2024-01-04]. Dostupné z: https://www.howtogeek.com/364232/what-is-sketchup/

[3] Sketchup Plan Overview. *SketchUp* [online]. ©2022 [cit. 2024-01-04]. Dostupné z: https://help.sketchup.com/en/admin/subscriptions

[4] Sketchup Developers FAQ. *Sketchup Developers* [online]. ©2022 [cit. 2024-01-04]. Dostupné z: https://developer.sketchup.com/developers/faq

[5] File import and export capabilities. *SketchUp* [online]. ©2023 [cit. 2024-01-04]. Dostupné z: https://www.sketchup.com/plans-and-pricing/compare/file-import-and-export-capabilities

[6] About FreeCAD. *FreeCAD Documentation* [online]. [cit. 2024-01-04]. Dostupné z: https://wiki.freecad.org/About_FreeCAD

[7] Parametrické modelování. *Fusion 360* [online]. ©2022 [cit. 2024-01-04]. Dostupné z: https://f360.cz/cad/parametricke-modelovani/

[8] Path Workbench. *FreeCAD Documentation* [online]. [cit. 2024-01-04]. Dostupné z: https://wiki.freecad.org/Path_Workbench

[9] Architektura výukový program. *FreeCAD Documentation* [online]. [cit. 2024-01-04]. Dostupné z: https://wiki.freecad.org/Arch_tutorial/cs

[10] Arch tutorial. In: *FreeCAD Documentation* [online]. 2014 [cit. 2024-03-18]. Dostupné z: https://wiki.freecad.org/File:Arch_tutorial_44.jpg

[11] Floorplanner. *Floorplanner* [online]. c2024 [cit. 2024-01-11]. Dostupné z: https://floorplanner.com/

[12] Floorplanner review. *Techradar* [online]. 2021, 7.10.2021 [cit. 2024-01-11]. Dostupné z: https://www.techradar.com/reviews/floorplanner

[13] Floorplanner API reference. *Floorplanner* [online]. [cit. 2024-01-11]. Dostupné z: https://floorplanner.readme.io/reference/getting-started

[14] Top view of a coloured 2d floorplan made by Floorplanner. In: *Floorplanner* [online]. c2024 [cit. 2024-01-11]. Dostupné z: https://static.floorplanner.com/_next/static/media/page-header.d3bde788.png

[15] Blender 4.0 Manual. *Blender* [online]. [cit. 2024-01-11]. Dostupné z: https://docs.blender.org

[16] Unity plans and pricing. *Unity* [online]. c2024 [cit. 2024-01-11]. Dostupné z: https://unity.com/pricing#plans-student-and-hobbyist

[17] What is Unreal Engine? *BairesDevBlog* [online]. c2024 [cit. 2024-01-20]. Dostupné z: https://www.bairesdev.com/blog/what-is-unreal-engine/

[18] Introduction to Godot. *Godot Engine 4.2 documentation* [online]. c2014-2024 [cit. 2024-01-20]. Dostupné z: https://docs.godotengine.org/en/stable/getting_started/introduction/introduction_to_godot.html

[19] HOLAN, Tomáš. *Unity: první seznámení s tvorbou počítačových her*. Praha: CZ.NIC, 2020. CZ.NIC. ISBN 978-808-8168-607.

[20] Rotation and orientation in Unity. *Unity Documentation* [online]. 2024 [cit. 2024-02-17]. Dostupné z: https://docs.unity3d.com/Manual/QuaternionAndEulerRotationsInUnity.html

[21] ŽÁRA, Jiří. *Moderní počítačová grafika*. 2., přeprac. a rozš. vyd. Brno: Computer Press, 2004. ISBN 80-251-0454-0.

SEZNAM PŘÍLOH

PŘÍLOHA A*:* Návod k použití aplikace stavební simulátor