Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Bakalářská práce

Vývoj aplikace 3D bludiště ve Windows Forms

Jan Cihelka

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Jan Cihelka

Informatika

Název práce

Vývoj aplikace 3D bludiště ve Windows Forms

Název anglicky

Development of a 3D maze application in Windows Forms

Cíle práce

Cílem práce implementovat aplikaci bludiště vytvořenou ve Windows Forms. Aplikace bude představovat jednoduchý herní engine s trojrozměrnou projekcí bludiště generovaného na základě jednoduchého obrázkového souboru se zvláštními funkcemi pro zajímavější tvorbu úrovní. Aplikace by měla sloužit jako platforma pro snadné sdílení nápadů, úrovní a rekordů, nebo případně jako výuková pomůcka.

Metodika

Práce sestává ze dvou částí, teoretické a praktické – vlastního řešení. V rámci zpracování teoretické části práce bude provedeno studium souvisejících odborných informačních zdrojů. Na jeho základě budou formulována teoretická východiska pro vlastní řešení.

Praktická část práce bude vycházet z teoretických východisek a v jejím rámci bude proveden návrh a implementace aplikace 3D bludiště. K implementaci bude využito jazyka C# a GUI frameworku Windows Forms. Aplikace bude otestována, budou shrnuty zkušenosti a poznatky získané během její implementace a budou navržena případná další rozšíření do budoucna.

Doporučený rozsah práce

35-40 stran

Klíčová slova

Bludiště, 3D, Windows Forms, C#

Doporučené zdroje informací

Buonanno, E. Functional Programming in C# : How to Write Better C# Code
Buonanno, E. Functional Programming in C#, Second Edition
Inc. BarCharts, Trigonometry
Sturm, O. Functional programming in C#-classic programming techniques for modern projects.

Předběžný termín obhajoby

2022/23 LS – PEF

Vedoucí práce

Ing. Jiří Brožek, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 31. 10. 2022

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 24. 11. 2022

doc. Ing. Tomáš Šubrt, Ph.D.

Děkan

V Praze dne 14. 03. 2023

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Vývoj aplikace 3D bludiště ve Windows Forms" jsem vypracoval(a) samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor(ka) uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 12.8.2022 _____

Poděkování

Rád bych touto cestou poděkoval Ing. Jiřímu Brožkovi

Vývoj aplikace 3D bludiště ve Windows Forms

Abstrakt

Tento projekt spočívá v rozvinutí aplikace 3D bludiště vytvořené v rámci závěrečné práce týkající se programování ve třetím semestru. Jedná se o aplikaci vytvářenou v programu Visual Studio v prostředí Windows Forms, psanou v jazyce C#. Aplikace samotná je ve své podstatě hra ve které je cílem hráče projít bludištěm, je ale obohacena o funkce které dovolují vytvářet iluzi toho, že se prostředí okolo hráče mění, a tím mu průchod ztěžuje. Hráč vidí bludiště z pohledu první osoby, a nezná tedy na první pohled jeho rozložení. Aplikace je schopná zobrazit 3D prostředí na základě jednoduchého obrázkového souboru se specifickým formátem. V tomto souboru lze specifikovat rozložení, vzhled, i interaktivní prvky bludiště. Díky tomu může tato aplikace sloužit jako výuková pomůcka, nebo platforma pro snadné sdílení vlastních herních úrovní. V rámci této práce by měla aplikace vylepšena, a rozšířena například o vestavěný editor úrovní, nebo systém zaznamenání průchodu bludištěm.

Klíčová slova:
Bludiště, 3D, Windows Forms, C#

Development of a 3D maze application in Windows Forms

Abstract

This project entails the development of a 3D maze application created as a final programming assignment in the third semester. The application is created in the program Visual Studio using the Windows Forms environment, and is written in the language C#. The application itself is in its nature a game where the objective of the player is to navigate a maze, it is however enhanced with functions which can create an illusion that the space around the player is changing, making the task more difficult. The player sees the maze from a first person perspective, and this does not know its layout at a glance. The application can display 3D environment based on a simple image file with a specific format. The file can specify layout, design, and interactive elements of the maze. The application can thus serve as a tool for education, or a platform for easy sharing of game levels. During this assignment, the application should be improved, and expanded. An example could be an inbuilt level editor, or a playthrough capture system.

Keywords:
Maze, 3D, Windows Forms, C#

Obsah

1 Úvod

Aplikace 3D bludiště je hra zasahující do sandbox, horor, a puzzle herních žánrů. Cílem hráče je projít v co nejkratším čase bludiště které se zdánlivě ne vždy řídí zákonitostmi euklidovského prostoru. Průchod bludištěm tedy není tak jednoduchý jako by se mohlo zdát, hráč musí počítat s tím, že se prostředí okolo něj může v nestřeženém okamžiku zcela změnit. Tato nejistota v kombinaci s ponurou atmosférou způsobenou technickými omezeními i designovými volbami propůjčuje hře nádech tajemna a pocit napětí. Kvůli tomu hra vystupuje pod pracovním názvem „Spooky." („Strašidelné")

Pokročilí uživatelé mohou také vytvářet vlastní bludiště ve kterém je poté možno se ve hře pohybovat. Velikost místností je pevně omezena, ale je možné definovat jejich počet, obsah, i vzhled.

Za normálních okolností by bylo lepší podobnou hru tvořit pomocí již existujícího herního enginu. Součástí tohoto projektu je ovšem snaha prozkoumat co prostředí Windows Forms, které pro podobné hry není příliš vhodné, dokáže. V rámci zprovoznění hry bylo nutné provést vlastní implementaci fyzikálních procesů, jako je pohyb hráče a jeho kolize s překážkami, vykreslování prostředí v pohledu z první osoby, a reakce interaktivních prvků na aktivaci hráčem.

2 Cíl práce a metodika

2.1 Cíl práce

Cílem práce je popsat, vylepšit a rozšířit aplikaci 3D bludiště. Je nutno odladit některé známé programové chyby, případně nalézt a identifikovat a eliminovat další. Některé části programu mohou být upraveny pro zjednodušení, zpřehlednění, zlepšení funkce, nebo snížení náročnosti programu na systémové zdroje.

Dále by měl program být obohacen o nové části které by měly vést k usnadnění jeho používání, a ke zvýšení jeho viability jako programu pro širší užívání. Program by měl ve finální fázi sloužit jako platforma pro snadné sdílení herních úrovní, nápadů a rekordů, případně jako výuková pomůcka.

Pro tento účel je žádoucí přidat vestavěný editor úrovní který by měl současně rozšířit možnosti interaktivních prvků bludiště, spolu se systémem na ověření dokončitelnosti vytvořených úrovní.

Při průchodu bludištěm je zaznamenáván čas který hráčovi průchod zabere, je tedy možné soutěžit o nejrychlejší možný průchod. K ověření validity je možné implementovat systém který by zaznamenával vstupy hráče do souboru, který by bylo možné poté přehrát a tím průchod replikovat. Možnost manuální úpravy tohoto souboru by také představovala nástroj pro tvorbu „tool assisted speedrun“ tedy uměle vytvořeného teoreticky nejlepšího možného dosažitelného průchodu.

2.2 Metodika

Aplikace je vytvářena v prostředí Windows Forms v jazyce C#. Bude popsán účel, funkce, a provedení existujících částí programu. Pomocí informací z odborných zdrojů mohou být sepsány a následně implementovány případné úpravy.

Na základě poznatků z odborných zdrojů budou také vytvořena teoretická východiska pro vytvoření nových častí programu které budou následně také implementovány.

3 Teoretická východiska

3.1 C#

Pro tvorbu programu byl zvolen programovací jazyk C#. Jedná se o vysokoúrovňový objektově orientovaný jazyk vhodný pro tvorbu aplikací pro Windows. Je charakterizován explicitním vyjádřením typů proměnných a rozdělení veškerého kódu do tříd. Tento jazyk je kompilovaný, což znamená že kód je před spuštěním převeden na instrukce zpracovatelné počítačem. Při tom je ověřena sémantická správnost kódu. Opakem jsou jazyky interpretované, které se převádějí za chodu postupně příkaz po příkazu.

C# je z uživatelského hlediska relativně jednoduchý a přívětivý jazyk, nabízí širokou řadu příkazů kterou je navíc možné ještě rozvinout použitím přídavných knihoven. Velmi užitečným nástrojem je třída Math která obsahuje goniometrické funkce které jsou potřeba k promítání bludiště do 3D pohledu.

Pro účely větvení kódu je možné kromě klasických podmíněných výroku if, else if, else které jsou vhodné pro komplexní rozhodování na základě více parametrů využít také výrok switch který je lépe uzpůsobený pro rozhodování mezi větším množstvím cest na základě jednoho parametru, nebo ternární operátor který umožňuje rychlé rozhodnutí mezi jednou nebo druhou hodnotou.

Pro iteraci kódu jsou k dispozici cykly while s podmínkou na začátku, do while s podmínkou na konci, for s určitým množstvím opakování, a foreach pro opakování na základě sbírky prvků.

Tato část je zpracována podle [4]

3.2 Objektově orientované programování

Jazyk C# je uzpůsobený k využívání paradigmatu objektového programování. To je přístup který rozděluje data a metody do jednotlivých logických celků které mají jasně vytyčené úkoly. Tyto celky se nazývají třídy. Třída může být statická nebo instanční.

Statické třídy slouží jako sbírky metod. Je možné je používat kdekoli v kódu bez nutnosti předchozího vytváření instance, ale zato nemohou uchovávat data.

Instanční třídy fungují jako předpis objektu který má nějaké stanovené vlastnosti a metody. Pro používaní instanční třídy je potřeba vytvořit její instanci. Třída může mít například stanoveno že by měla mít nějakou barvu, hmotnost, nebo jméno. Každá instance potom bude mít pro tyto vlastnosti stanoveny konkrétní hodnoty.

Mezi třídami je také možné vytvářet dědičné vazby. To znamená že třída může přejímat informace z nějaké nadřazené třídy. Například kdybychom vytvářeli třídy "Automobil", "Motocykl" a "Loď" mohli bychom vytvořit třídu "Vozidlo" která pro ně bude rodičovskou třídou. Tím pádem nám stačí všechny vlastnosti které mají Automobil, Motocykl a Loď společné deklarovat ve třídě Vozidlo a nechat je je zdětit, místo abychom je deklarovali v každé třídě zvlášť. Navíc je potom možné ke každé instanci některého z potomků přistupovat jako by se jednalo o instanci rodiče.

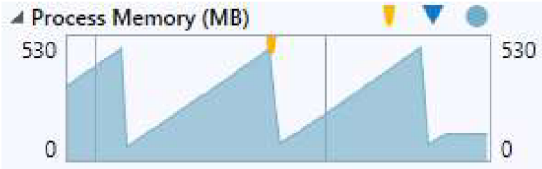
Tato část je zpracována podle [6]

3.3 Funkcionální programování

Odborná literatura vybraná k tomuto projektu pojednává z většiny o konceptu zvaném „funkcionální programování." Jedná se o přístup k tvorbě programu který staví funkce do role hodnot, a snaží se vyhýbat přepisování. Výhodou tohoto přístupu je jasnější posloupnost kódu, místo aby musel uživatel sledovat kde byla která proměnná jak upravena, jsou data v každém mezikroku ukládány do nové proměnné. Tím že původní data zůstanou nezměněna se také předchází chybám v případě toho že k datům přistupuje více částí programu které nemusí počítat s jejich pozměněním.

Na druhou stranu, tento styl tvorby programu má nevýhodu ve vyšší náročnosti na systémové zdroje. Tím že vytváří při každém mezikroku nové kopie dat spotřebovává větší množství paměti, to je do jisté míry řešeno systémem „garbage collection" který maže data která už nebudou v programu dále potřeba potom co zaniknou všechny odkazy na ně, ale i přesto je tento přístup náročnější.

Vzhledem neoptimalizovanosti prostředí Windows Forms pro tuto aplikaci, a ke způsobu implementace fyzikálních simulací a vykreslování pohledu je program už v tuto chvíli relativně náročný, proto je nutné ve vhodné míře kombinovat přehlednější funkcionální styl s úspornějším imperativním stylem.



Obrázek 1 - Využití paměti při vytváření nového objektu pro každý snímek

Obrázek 2 - Využití paměti při překreslování jednoho objektu

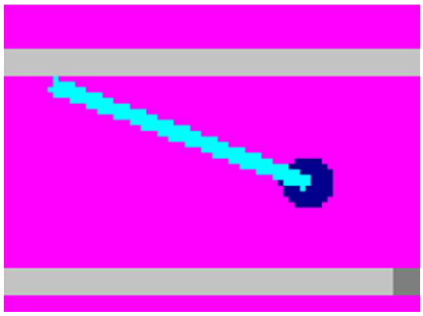
Obrázek 6 ukazuje využití operační paměti během chodu hry v případě že je pro každý snímek vytvářen nový objekt jak by napovídalo paradigma funkcionálního programování. Graf v některých bodech dosahuje až půl gigabytu. Na obrázku 7 je ukázána ta samá hodnota v případě že existuje pouze jeden objekt pro zobrazení pohledu hráče, který je ovšem každý snímek překreslen. Využití paměti narůstá pomaleji, a do výrazně nižší míry. Zlomy v obou grafech jsou způsobovány uvolněním nevyužitých zdrojů garbage collectorem.

Příkladem vhodného použití funkcionálního programování může být vlastnost pro vykreslování hráče v debugovací minimapě. Objekt hráč obsahuje proměnnou „pozice“ která uchovává jeho souřadnice v místnosti. Metoda pro vykreslování elipsy která by ho na minimapě měla zastupovat ale jako parametr přijímá levý horní roh, výšku a šířku elipsy. Aby byla elipsa správně vykreslena na středu pozice hráče, je nutné parametr pro její vykreslení upravit, a to tak že od souřadnice X odečteme polovinu šířky hráče, a od souřadnice Y polovinu jeho výšky. Při tom ale nemůžeme pozměnit skutečnou pozici hráče. Pro tento účel je použita následující vlastnost:

```
public Point DrawPos { get { return new Point((int)position.X - Thick / 2, (int)position.Y - Thick / 2); } }
```

Obrázek 3 - Vlastnost pro určení vykreslení pozice hráče

Při zavolání této vlastnosti se provede potřebný výpočet, a odešle se pouze jeho výsledná hodnota bez nutnosti přepisovat původní hodnotu.



Obrázek 4 - Světlý paprsek vychází ze skutečné pozice hráče, ve středu tmavé elipsy

Tato část je zpracována podle [1]

3.4 3D projekce

Princip zobrazování je inspirován hrou Wolfenstein 3D z roku 1992 která byla jednou z prvních her z pohledu první osoby ve 3D. Počítače v té době neměly dostatečný výkon aby dokázaly zpracovat hru ve skutečném trojrozměrném prostoru, proto byla použita metoda trojrozměrné projekce. Hráč se skutečně pohybuje ve dvourozměrném bludišti, to je ale promítáno z jeho pohledu. Hráč se vždy pohybuje ve stejné výšce, a nemůže svůj pohled naklánět nahoru a dolu, protože hra tento rozměr ve skutečnosti neobsahuje.[5] Hra Doom vydaná v roce 1993 sice změnu výšky umožňuje, ale nikde se v ní nevyskytují dvě místnosti které by byly přímo nad sebou, to proto, že ačkoli se jedná o pokročilejší systém, hra je stále pouhou projekcí ze dvou rozměrů, a proto by se místnosti nad sebou musely překrývat.[7]

Průchod bludištěm na které má hráč pohled z ptačí perspektivy není příliš složitý ani zábavný úkol. Použitím perspektivy první osoby je omezeno množství informací které jsou hráči okamžitě přístupné, a nutí ho tedy prozkoumat okolní terén, z toho vychází výzva a podstata hry.

Jak již bylo zmíněno, herní pole je zaznamenáno z půdorysu. Je rozlišen volný prostor a stěny. Hráč je definován jako bod uvnitř tohoto pole. Hráč má zorné pole, které je

skenováno pomocí metody známé jako raycast. Raycast představuje paprsek vystřelený určitým směrem. Tento paprsek může předad informaci o tom jak daleko doputoval než narazil na překážku, a o jakou překážku se jednalo. Tato metoda je běžně používána i v moderních hrách.

V programu je tato metoda řešena cyklicky. Na základě pohledu hráče se stanoví vektor směru a v tom je poté zkontrolován každý pixel dokud se nenarazí na pixel označený jako stěna, poté předá kolik pixelů prošel. Tímto je také vyřešen problém vykreslování zakrytých objektů. Raycast zaznamená pouze první překážku na kterou narazí.

Raycast probíhá v rozpětí sedmdesát stupňů, pro každý stupeň jednou. Obrazovka je rozdělena na sedmdesát sloupců, každý odpovídající jednomu raycastu. Podle zákona optiky se vzdálené objekty jeví menší než objekty bližší, proto výška sloupce která vychází z raycastu klesá s vstoupající vzdáleností. Pro správné vykreslení je tento vztah upravený matematickou rovnicí.

$$Výška = H/\frac{1 + D * \cos\alpha}{20}$$

Parametr H v tomto vzorci představuje referenční výšku sloupce. Je to taková výška jakou by měl sloupec mít pokud se na něj hráč dívá přímo ze vzdálenosti rovné 1. V kódu je stanoven na 120% výšky hráčova pohledu.

Parametr D je vzdálenost zjištěná raycastem.

Parametr α je rozdíl úhlu raycastu oproti úhlu ve středu zorného pole hráče. Tedy pro raycast měřící vzdálenost k objektu který je přímo rovně před hráčem je α rovno nule. Hodnota funkce kosínus pro úhel nula je rovna jedné, pro odchylující se úhly potom klesá. Je zde použit pro to, aby se předešlo efektu rybího oka. Pokud bychom se dívali kolmo na rovnou stěnu, byla by vzdálenost ke stěně uprostřed našeho zorného pole kratší než na jeho okrajích. Prostředek stěny by se tedy jevil oproti krajům vypouklý. Součinem vzdálenosti a kosínu α docílíme normalizace tohoto rozdílu[3], tím pádem se bude stěna jevit stejně vysoká po celé své šířce.

Konstanta 1 v horní části zlomku je přidána aby se předešlo potencálnímu dělení nulou ke kterému může dojít při vstupu do cíle který je vykreslován, ale nekoliduje s hráčem, tedy umožnuje se přiblížit na vzdálenost 0. Při normálních hodnotách není rozdíl způsobený touto přidanou konstantou znatelný.

Konstanta 20 ve spodní části zlomku upravuje výslednou hodnotu za účelem optimalizace zobrazení, a byla vybrána na základě experimentace. Nižší hodnoty způsobují smrštění výšky objektů v porovnání s jejich šířkou, vyšší hodnoty naopak výšku natahují. Při změně vzdálenosti se tím pádem výška a šířka mění v jiném poměru, a objekt se zdánlivě deformuje. Hodnota 20 způsobuje optimální poměr mezi výškou a šířkou díky kterému si objekty zachovávají svůj tvar.

4 Vlastní práce

4.1 Formulář

Formulář ve kterém se hra zobrazuje obsahuje pouze dva ovládací prvky (plus dva skryté panely pro zobrazení debug pohledu na místnost), a to Panel a Timer. Panel funguje jako plátno na které se vykreslují prvky menu a snímky hry. Také zaznamenává vstup myši, je schopen zaznamenat pozici kliknutí pro přístup k ovládacím prvkům. Timer se stará o běh hry, jedná se o objekt který v pravidelných intervalech dává programu impuls k vypočtení reakce na vstupy, vykreslení nového snímku, a připočtení herního času. Interval timeru je stanoven na 25 milisekund, hra by tedy měla běžet rychlostí 40 snímků za vteřinu. Nakonec samotný formulář přijímá vstupy z klávesnice pro ovládání pohybu hráče.

4.2 Struktura programu

Program se skládá z pěti základních komponentů. Hlavní části kódu, objektu pro zpracování souborů, objektu pro zobrazování prvků uživatelského rozhraní, objektu hráče, a objektu prostředí. Každý z těchto objektů má na starosti určitý oddíl funkce programu, a v potřebné míře komunikuje s ostatními.

4.2.1 **Hlavní část kódu**

Hlavní část kódu je přímo provázaná s formulářem na kterém se hra odehrává, a spravuje jeho prvky, které jsou časovač zajišťující chod hry, a panely na které jsou vykreslovány vizuální části hry. Dále obsahuje ostatní objekty. Pro správnou hierarchickou strukturu programu by měly tyto objekty mezi sebou komunikovat pouze skrz hlavní část programu.

Program je řízen pomocí proměnné „gameState“ která nabývá pěti různých stavů. SizeSelect je výchozí stav ve kterém se hra nachází po spuštění, nabízí uživateli možnost zvolit si pomocí uživatelského rozhraní jednu ze tří velikostí herního okna. Po potvrzení přechází do stavu MainMenu. V tomto stavu program zobrazuje titul hry, a tlačítka pro spuštění nové hry a vypnutí aplikace. Také umožňuje pomocí klávesy mezerník přepínat debug mód který zobrazuje herní mapu navíc vedle běžného pohledu z první osoby. Po spuštění hry program přechází do stavu GameOn. Ten spouští časovač, skrývá kurzor, zprovozňuje ovládání klávesnicí, a vykresluje hráčův pohled. Hru je možné během chodu zastavit tlačítkem escape, tím program přechází do stavu Paused. Hráčův pohled je stále zobrazen, ale žádná část hry neběží. Navíc jsou k dispozici tlačítka pro návrat do hry, a ukončení aplikace. Při výhře přejde program do stavu GameEnd, který je podobný stavu MainMenu, jenom místo titulu ukazuje čas strávený v bludišti, a neumožňuje přepínání debug módu.

4.2.2 **Uživatelské rozhraní**

Objekt uživatelského rozhraní je využíván v podstatě celou dobu běhu programu. Spravuje veškerý text zobrazovaný uživateli, i tlačítka na obrazovce. Komunikuje s hlavní částí programu pomocí textových zpráv které jsou přiřazeny určitým tlačítkům. Hlavní část programu tyto zprávy přijímá pomocí metody Commander, a na jejich základě mění herní stavy, nebo velikost okna.

Tlačítka na obrazovce jsou v tomto programu řešena poněkud netradičním způsobem. Místo objektů tlačítek jsou používány barevné obrazce. Uživatelské rozhraní se skládá ze dvou vrstev, grafické, která obsahuje text který by uživatel měl na tlačítkách vidět, a funkční vrstvy, kterou uživatel nevidí. Ta na místech odpovídajících textu obsahuje obrazce

vyznačené určitou barvou. Tyto barvy jsou klíči ve slovníku příkazů, a každá odpovídá jedné zprávě pro Commander. Klikne-li tedy uživatel na tlačítko, program zjistí jaká barva se nachází na souřadnicích myši, a odešle odpovídající zprávu. Tento přístup dovoluje simulovat tlačítka bez nutnosti jednotlivých objektů, a také tvořit klikatelné oblasti libovolného tvaru.

Uživatelské rozhranní obsahuje svůj vlastní vestavěný font. Tento font má rozlišení 4x4 pixely, a je v programu uložený jako decimální číslo, které po konverzi do binární soustavy vyznačuje které pixely mají být černé a které bílé. Písmena jsou tedy vykreslována jako obrázky, což umožňuje snažší určování zarovnávání a vyplňování prostoru.

Poslední záležitost o kterou se uživatelské prostředí stará jsou chybové hlášky. Při spouštění programu provádí objekt pro zpracování souborů kontrolu správnosti úrovně, pokud při ní najde chybu, odešle jí do uživatelského prostředí, které jí zobrazí v messageboxu, a ukončí chod programu.

Obrázek 5 – Grafická vrstva hlavního menu

Obrázek 6 – Funkční vrstva hlavního menu

4.2.3 **File Handler**

Objekt pro zpracování souborů, neboli file handler provádí svou činnost hned na začátku programu, dříve než je vůbec uživateli cokoli zobrazeno. Tento objekt zajišťuje zpracování souboru úrovně, a jeho následné předání objektu prostředí.

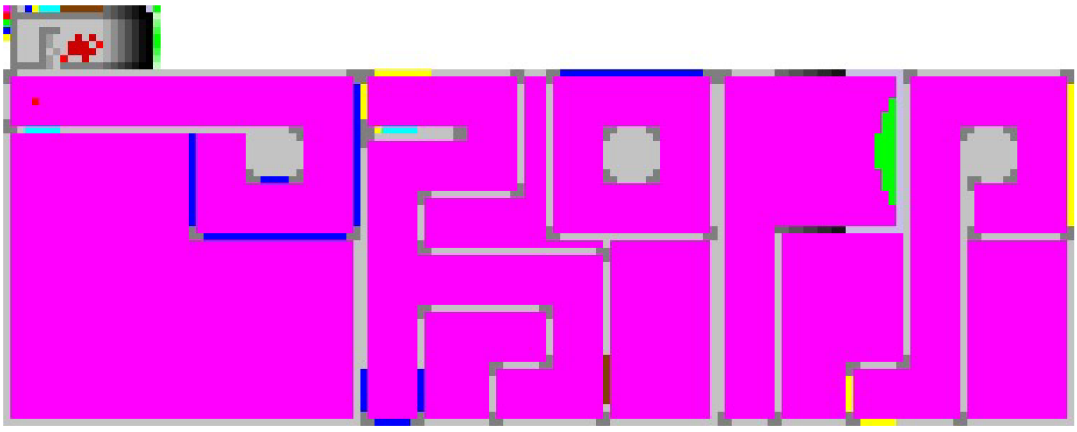
V případě rozšíření programu o editor úrovní by bylo nutné funkci file handleru upravit, aby byl schopen zpracovat soubor na vyžádání.

File handler provádí několik kroků, a v případě jejich selhání odesílá chybovou hlášku. Chyby jsou seřazeny podle návaznosti, a jsou předávány uživatelskému rozhraní které je zobrazuje uživateli.

První chybou která může nastat je absence souboru úrovně. Ten by se měl nacházet ve stejné složce jako EXE soubor, a být pojmenovaný „MAP.png.“ Pokud tomu tak není, není možné provést žádné další operace. Pokud je soubor k dispozici, další vlastnost kterou je nutné zkontrolovat jsou jeho rozměry. Obrázek by měl být minimálně 50 pixelů široký, aby se na něj vešla alespoň jedna místnost, a přesně 59 pixelů vysoký. Pokud tomu tak není, je vyhodnocen jako neplatný. Dále je pro správné fungování programu nutné aby byly řídící barvy specifikované v úrovni od sebe odlišné, aby bylo jasné jakou funkci mají představovat. Pokud by například barva označující volný prostor a cíl byla totožná, nebylo by možné hru

vyhrát, protože barva pro volný prostor je v kolizní mapě invertována a označuje oblasti kam nelze vstoupit, nebylo by tedy možné vsoupit ani do cíle. Další částí souboru úrovně je sada textur kterými bude bludiště vybarveno. Každá textura musí mít unikátní zástupnou barvu, aby bylo jasné jakým vzorem má být nahrazena. Zástupné barvy jsou navíc ukládány jako klíče do slovníku, tedy ani z programového hlediska není možné aby byly duplicitní. Předposlední kontrolovanou vlastností je přítomnost startovního bodu. Ten by měl být vyznačen určenou barvou někde na mapě, pokud není, program by neměl moci fungovat, avšak tato chybová hláška vlivem chyby programu nefunguje, místo toho pouze umístí hráče do levého horního rohu, kde je uvízlý ve stěně. Nakonec je zkontrolováno že je každá místnost ohraničená celistvou stěnou, to aby se předešlo možnosti vykročit z herního pole, nebo pohlédnout mimo vykreslovaný prostor, což by způsobilo spadnutí programu.

Během kontroly jsou místnosti zvětšeny na pětinásobek své původní velikosti pro zvýšení detailu pohybu hráče, a také jsou vygenerovány kolizní mapy.



Obrázek 7 - Výchozí úroveň

4.2.4 **Prostředí**

Objekt prostředí se stará o vykreslování pohledu, a o interakce hráče s prostředím. Obsahuje barevné mapy na základě kterých určuje textury které do pohledu vykreslovat, a také kolizní mapy pomocí kterých omezuje pohyb hráče při nárazu do překážky. Prostředí provádí raycast z pozice hráče v sedmdesáti stupňovém rozptylu. Každý raycast odpovídá jednomu sloupci na obrazovce. Vzdálenost kterou raycast urazí před dopadem na překážku určuje výšku a světlost sloupce.

Textury jsou uloženy ve slovníku, kde klíč textury je barva, a textura samotná je bitmapa o rozměrech 1x8 pixelů. Pokud raycast v barevné mapě narazí na překážku vyznačenou barvou která je obsažena ve slovníku, vykreslí odpovídající sloupec jako tuto texturu, pokud ne, vykreslí ho barvou pro volný prostor. Pokud raycast v určité vzdálenosti narazí na barvu která je ve funkčních barvách specifikována jako „další“ nebo „předchozí“ změní aktivní místnost. Tím je možné vytvořit iluzi měnícího se prostoru při pohledu na určitou spouštěcí oblast, nebo jednoduše vytvořit efekt otevírajících se dveří, nebo například výtahu.

Kolizní mapa místnosti v sobě obsahuje pouze stěny a cíl. Stěny jsou v kolizní mapě vyznačeny výrazně tučněji než v barevné mapě, a to pro simulování tloušťky hráče. Kdyby mohl hráč přistoupit až přímo ke stěně, mohlo by docházet k problémům s vykreslováním, proto má hráč stanovenou tloušťku která zajišťuje jeho odstup. Protože je ale hráč mobilní objekt, je těžké kontrolovat každý bod kterým by mohl do stěny narazit. Proto je tloušťka hráče řešena obráceně, hráč skutečně žádnou tloušťku nemá, ale stěny okolo něj jsou zesílené tak aby tento rozdíl kompenzovaly. Je tedy nutné kontrolovat kolizi pouze v jednom bodě.

Obrázek 8 - Výběr textur a stěna z nich poskládaná

Obrázek 9 - Pohled první osoby na stěnu z obrázku 8

4.2.5 **Hráč**

Objekt hráče v sobě uchovává pozici, orientaci, a hybnost postavy hráče. Přijímá vstupy z klávesnice a myši a na jejich základě způsobuje jeho pohyb. Hráč má zrychlení a zpomalení, takže chvíli trvá než dosáhne maximální rychlosti, a chvíli trvá než zastaví. aměr pohybu je závislý na směru natočení hráče.

Hráč obsahuje odkaz na objekt prostředí, aby mohl přímo z něj volat metodu která zjišťuje kolizi na základě jeho pozice a směru pohybu. Pokud je zjištěna kolize, hráč okamžitě zastaví. Pro lepší strukturovanost programu by měl být tento odkaz odstraněn, aby ke komunikaci mezi objekty docházelo pouze skrz nadřazený prvek.

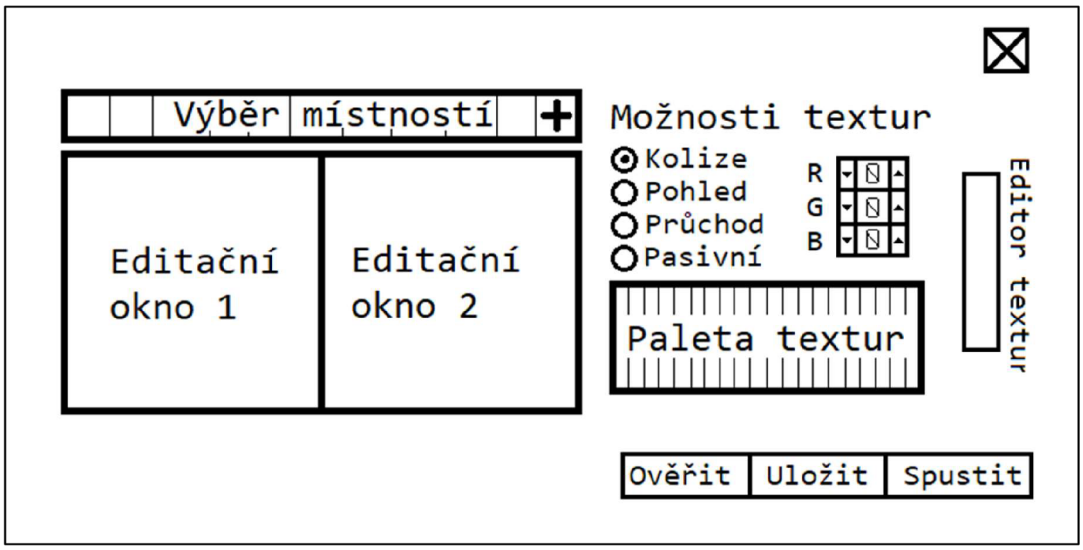
4.3 **Plánovaná rozšíření**

4.3.1 **Editor úrovní**

Implementace vestavěného editoru úrovní je zásadní krok pro vylepšení aplikace. Kromě usnadnění tvorby vlastních úrovní může navíc rozšířit možnosti hry.Prvním krokem je si představit rozložení uživatelského rozhraní a zvážit prvky které v něm budou potřebné.

Zcela nezbytnou částí musí být okno pro zobrazení a úpravu místnosti. Vzhledem k možnosti vytvoření návaznosti místností pomocí přechodů mezi nimi je ale potřeba aby mohl uživatel nahlížet na nejméně dvě místnosti současně, proto by toto okno mělo být použito dvakrát vedle sebe. Bude reagovat na vstup z myši a měnit označené pixely na základě výběru barvy z palety. Pro snadnou navigaci místnostmi by měl být k dispozici přehled jejich zmenšených verzí. Ten by mohl současně sloužit k otevření vybrané místnosti v editovacím okně pomocí levého nebo pravého tlačítka.

Dalším obsaženým nástrojem by měla být paleta textur, která by usnadňovala tvorbu a používání zástupných barev. Díky této paletě by také bylo možné do barev zakódovat informace pro rozšíření možností interaktivních prvků, například jako o kolik místností má aktivace prvku hráče posunout, nebo jestli má aktivační prvek reagovat na pohled nebo na kolizi.



Obrázek 10 - Návrh UI editoru úrovní

Po vytvoření úrovně de nutné ověřit její správnost pomocí systému dosud použitém ve třídě file handler. Další potřebnou vlastností úrovně by ale měla být její dokončitelnost kterou program nedokáže sám ověřit. Proto by měl tvůrce úrovně dokázat že je možné jeho úroveň dokončit. Během ověřování úrovně může program na základě posloupnosti jednotlivých pixelů vytvářet číslo ze kterého nakonec vytvoří unikátní barvu. Pokud tvůrce úspěšně předvede dokončitelnost svojí úrovně, program soubor opatří symbolem barvy

kterou na jeho základě vytvořil. Při opětovném načtení program ověří že je barva kterou na základě úrovně vytvořil shodná s tímto ověřovacím symbolem. Pokud ano, ví že úroveň je ověřená jako splnitelná. Pokud ne, ví že symbol je falešný, nebo že úroveň byla po ověření znovu upravena.

4.3.2 **Záznamník průchodu**

Pro další rozšíření možností této aplikace jako platformy je možnost sdílení výkonů. Vzhledem k jednoduchosti herních mechanik by bylo možné během každého snímku zaznamenat která směrová tlačítka byla stisknuta, a o kolik bylo pohnuto myší. Tyto údaje by mohly být postupně zapsány do souboru, a pomocí úpravy herního systému následně opět přečteny, a použity místo vstupů hráče. Tím pádem by bylo možné snadno uložit a znovu přehrát průchod bludištěm, čímž by bylo možné snadno ověřit validitu času průchodu v případě soutěžení o nejrychlejší průchod.

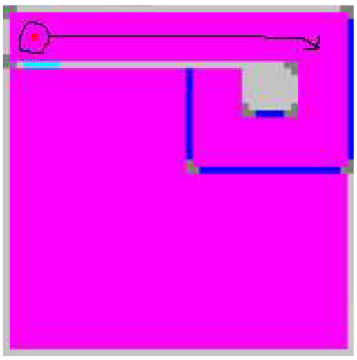
4.4 **Herní mapa**

Výchozí mapa přiložená ke hře je vytvořená tak, aby co nejlépe demonstrovala možnosti hry. To jsou v první řadě funkce umožňující změny prostředí ve kterém se hráč pohybuje. Mapa může obsahovat více místností, mezi kterými je možné přepínat pomocí zón vyznačených určitými funkčními barvami. V tomto případě se jedná o modrou a žlutou barvu, modrá přepíná na další místnost, žlutá na předchozí. Tyto barvy mají zároveň v knihovně textur přiřazenou stejnou texturu jako většina ostatních stěn, to proto, aby bylo možné je skrýt, a přepínání mezi místnostmi probíhalo bez povšimnutí. Stejně tak by bylo možné tyto prvky použít opačným způsobem, a to tak aby přepínání místností bylo patrné. V případě že by chtěl tvůrce mapy vytvořit dveře, mohl by vytvořit jednu místnost s patřičně natexturovanou stěnou, poblíž které by mohla být aktivační zóna natexturovaná jako tlačítko. Při pohledu na tlačítko by byl hráč přesunut do identické místnosti, která by ovšem „dveře“ neobsahovala, a tím by byl vytvořen efekt jejich otevření.

Způsob jakým je výchozí herní mapa má hráče především zmást. Místnosti jsou vytvořeny tak, aby části které hráč při přesunu mezi nimi vypadaly stejně, ale části které nevidí bývají výrazně odlišné.

4.4.1 **První místnost**

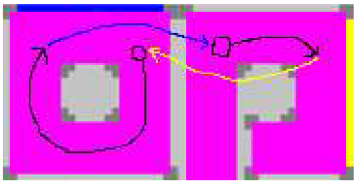
Hráč začíná v pravém horním rohu místnosti, který je označený červeným pixelem. Světle modrá část stěny vedle něj je kombinace několika textur které dohromady vytváří obraz dveří. Tyto dveře slouží jako orientační bod označující začátek mapy, a také jako prvek environmentálního vyprávění. Měly by odůvodňovat to, jak se hráč do bludiště vůbec dostal. Po průchodu rovnou chodbou která navazuje čekají hráče tři zatáčky doprava, které zdánlivě vedou do slepé uličky. K té by se ovšem hráč neměl dostat, stěny označené tmavě modrou barvou jsou aktivační zóny, které posunou hráče do další místnosti. Pokud hráč neví přesně co dělá, při odbočování téměř jistě některou z nich aktivuje.



Obrázek 11 - Cesta hráče první místností

4.4.2 **Druhá místnost**

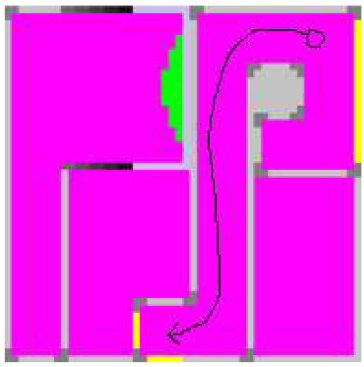
Potom co je hráč přenesen do druhé místnosti, jsou tři pravé zatáčky končící slepou uličkou nahrazeny smyčkou čtyř pravých zatáček. Toto je první část kde hráč zažívá nestandartní chování prostoru. Po čtyřech devadesátistupňových zatáčkách by měl hráč skončit tam kde začal. V tomto případě se ale hráč může točit dokola dle libosti, a svůj výchozí bod nenajde. Jedna ze stěn v této smyčce je označena modrou barvou, která při aktivaci přenese hráče do další místnosti, v té je ale stěna označena žlutou barvou, která ho hned přenese zpět pokud bude pokračovat dále stejným směrem. Řešením je se otočit a jít opačným směrem, tím hráč aktivuje modrou zónu která ho přenese do další místnosti, a místo aktivace žluté zóny v ní může pokračovat do nové navazující chodby.



Obrázek 12 - Smyčka mezi druhou a třetí místností

4.4.3 Třetí místnost

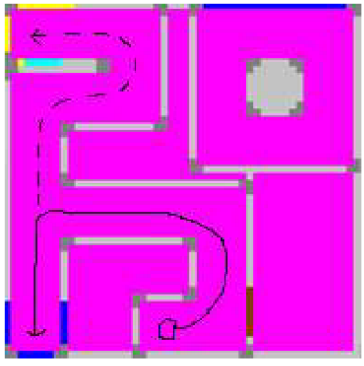
Po opuštění smyčky v druhé místnosti hráč pokračuje chodbou zakončenou slepou uličkou. Její konec je ovšem označen žlutou barvou, která ho přesune zpět do druhé místnosti, ovšem v jiném místě, takže může postoupit dále.



Obrázek 13 - Cesta ze smyčky do slepé uličky

4.4.4 Zpět ve druhé místnosti

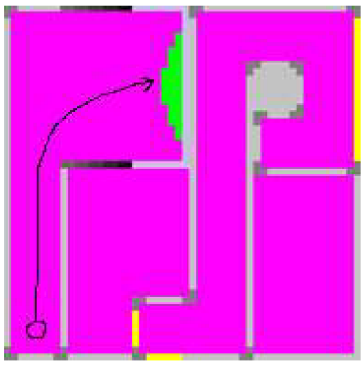
Poté co se hráč otočí zpět ze slepé uličky za sebou na stěně najde rudou skvrnu která zde před tím nebyla. Toto má u hráče vyvolat pocit překvapení, zvýšit napětí, a utvrdit hráče v tom že se okolo něj děje něco zvláštního. Když hráč obejde dvě zatáčky doleva které ho nyní čekají, dojde na rozcestí. Pokud se vydá cestou doprava, najde po několika zatáčkách povědomé místo. Na konci chodby jsou stejné dveře jako na samém začátku mapy obklopené žlutými zónami. Příjde-li tedy hráč dostatečně blízko, některou z nich pravděpodobně aktivuje, a po otočení zjistí že je opět na startu. Pokud se ale na rozcestí vydá doleva do slepé uličky, bude pomocí modrých zón přenesen do třetí místnosti.



Obrázek 14 - Rozcestí

4.4.5 Cíl

Když se hráč na konci slepé uličky otočí, zjistí že se prostor za ním opět změnil. Chodba nyní vede do místnosti která zdánlivě mizí do temnoty. Tohoto efektu je dosaženo postupným ztmavováním textur až do plné černé která se shoduje s barvou pozadí kterou je vybarven strop a podlaha. V místě kde je temnota nejhlubší se nachází zářící zelené těleso. Když do něj hráč vstoupí, vyhrává hru.



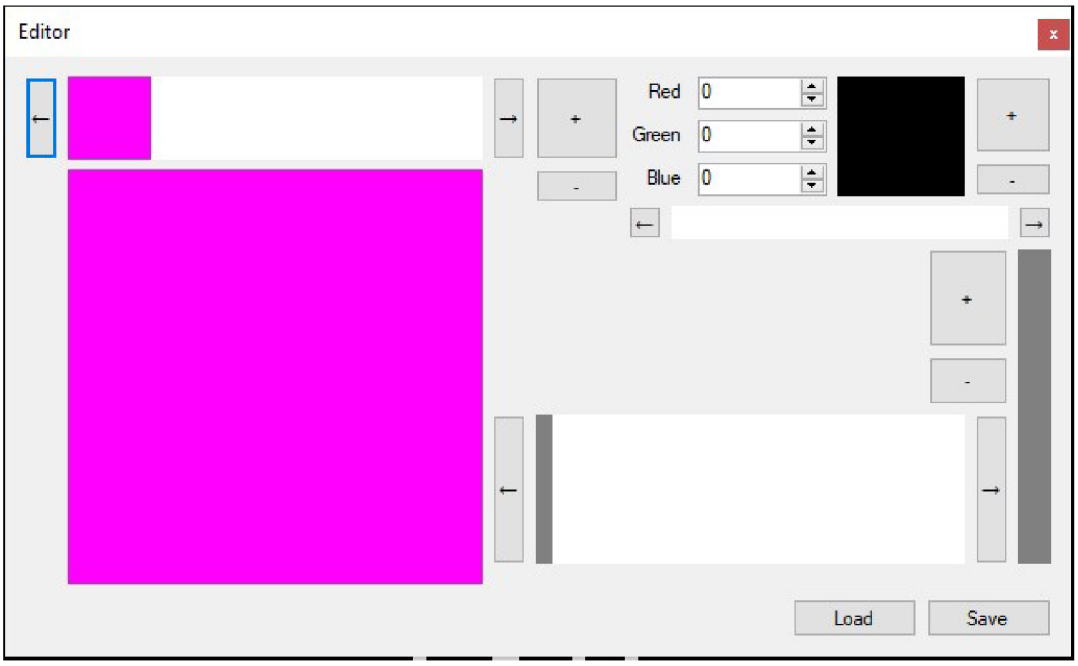
Obrázek 15 - Cesta do cíle

4.5 Implementace editoru úrovní

Vzhledem k potřebě velkého množství komplexních ovládacích prvků byl editor úrovní vytvořen jako zvláštní formulář používající místo rozhraní zbytku hry klasické formulářové prvky. Uživatelské rozhraní Spooky je vhodné pro vtažení hráče do atmosféry hry, vytvořit s jeho pomocí ale zadávací prvky potřebné k funkci editoru by bylo komplikované. Editor je v podstatě samostatný nástroj, a proto se použití herního rozhraní ani nehodí.

Pro otevření editoru slouží nově přidané tlačítko „Editor“ v hlavním menu hry. Editor se otevírá pomocí metody ShowDialog, a díky tomu není možné pracovat se základním formulářem dokud je editor otevřený. Tím se předchází otevírání více editorů současně, nebo přepisování úrovně při jejím hraní.

Editor je technicky vzato rozdělen do tří na sebe navazujících sekcí. První sekce spravuje místnosti, druhá barvy, a třetí textury.



Obrázek 16 - Okno editoru

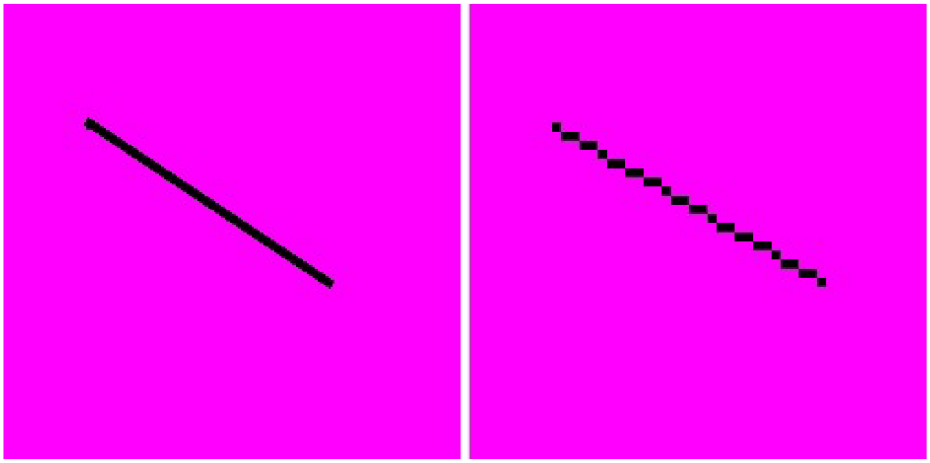
4.5.1 **Místnosti**

Sekce místností se nachází v levé části formuláře. Tvoří jí velký panel sloužící ke kreslení do místnosti, malý panel zobrazující výběr místností, tlačítka pro posouvání výběru, a tlačítka pro přidávání a odebírání místností.

Při kliknutí na velký panel se daný pixel zbarví barvou nastavenou v sekci barev. Protože je pro snazší používání místnost na kreslícím panelu pětinásobně zvětšena, je nutné souřadnice kliknutí vydělit pěti. Při kliknutí a tažení může uživatel kreslit vybranou barvou rovnou čáru mezi dvěma body. Pro správné vykreslování čáry je nutné aby uživatel viděl kudy přesně čára vede než je do místnosti přidána, proto se pohled při každém pohybu myši obnovuje, aby čára z výchozího bodu následovala myš až do puštění tlačítka. Toto ale způsobuje velké množství po sobě jdoucích překreslování panelu, a to může vést k problikávání. Kreslící panel má proto přiřazenou vlastnost Double Buffered která tomuto problému zabraňuje.

Při kreslení čáry se čára na samotnou bitmapu místnosti zapíše až ve chvíli kdy uživatel pustí tlačítko myši. Místnost je rasterový obrázek o velikosti padesát na padesát

pixelů. Čáru sloužící k náhledu jejího umístění je nutné vykreslovat zvlášť, protože kdyby byla vykreslována rovnou na bitmapu místnosti, nebylo by možné s ní hýbat. Pokud by byla vykreslována přímo na panel pomocí metody DrawLine, měla by pouze pětinu své skutečné velikosti, musíme pamatovat, že místnost je pětkrát zvětšená. Pokud vynásobíme souřadnice vykreslované čáry pěti, získáme výsledek znázorněný na obrázku. Čára se vyhladí, a není potom možné odhadnout které pixely přesně zasáhne.



Obrázek 17 - Náhledová čára vlevo, vpravo výsledná čára

Problémy vytyčené v minulém odstavci jsou řešeny pomocí přídavné bitmapy jménem overlay. Tato bitmapa má stejný skutečný rozměr jako místnost. Náhledová čára se vykresluje na ní, a poté je společně s místností zvětšena na velikost panelu. Mimo vykreslenou čáru je průhledná, a vykresluje se jako další vrstva přes náhled místnosti. Díky tomuto postupu zvětšení vypadá čára v náhledu stejně jako výsledná rasterizovaná čára.

Při použití pravého tlačítka funguje kreslení stejně jako při použití levého, s vyjímkou toho že použitá barva je barva magenta, tedy barva pozadí. Pravé tlačítko tedy funguje jako guma.

Každá dokončená kreslící operace je propsána na panel s výběrem místností.

Místnosti jsou uloženy v listu, to znamená že je možné je dynamicky přidávat a odstraňovat. Vybraná položka z listu se určuje pomocí číselného indexu.

Vybírací panel vykresluje místnosti vytvořené uživatelem. Místnosti je možné přidávat pomocí tlačítka plus. Při vykreslování rozhoduje pokud je místností méně než pět, pokud ano, vykreslí pouze jejich počet, pokud ne, vykreslí právě pět. Pokud je místností více než pět, je možné se k nim dostat pomocí tlačítka doleva a doprava.

Posun výběru je řízen proměnnou rOffset. Levé tlačítko kontroluje jestli je rOffset větší než nula, aby nebylo možné se posunout do mínusu. Pravé tlačítko kontroluje jestli je rOffset plus pět menší než počet místností, díky tomu není možné posunout výběr dále než k poslední místnosti. Tyto kontroly existují proto, aby v cyklu který místnosti vykresluje nedošlo k volání indexu v listu který je mimo jeho rozsah.

Při přidávání nové místnosti je rOffset nastaven na množství místností snížený o pět, pokud je počet místností větší než pět. Tím se výběr posune nakonec, aby zabíral nově přidanou místnost.

Při odstraňování místností pomocí tlačítka mínus se rOffset sníží pokud je větší než nula, a počet místností klesne pod rOffset plus pět. Tím se zajistí že se výběr posune dolů pokaždé když je smazána poslední místnost kterou ukazuje. Také se změní vybraná místnost, a to tak že pokud byla vybraná místnost která má být smazána, vybere se automaticky místnot před ní. Nakonec, pokud je smazána poslední místnost, je okamžitě nahrazena novou prázdnou místností.

Při kliknutí na vybírací panel se zjišťuje souřadnice x kliknutí na panelu. Tato souřadnice je vydělena padesáti, a poté je zjištěno jestli je výsledná hodnota menší než počet místností. Pokud ano, je výsledná hodnota použita jako index pro zvolení místnosti z listu.

Při tvorbě místností je nutné vyznačit po celém obvodu barvou označující stěnu s kolizí ohraničení. Dále na první místnosti musí být vyznačen alespoň jeden pixel čistě červenou barvou FF0000. Průměrná pozice těchto pixelů určuje startovní bod hráče. Pokud tyto podmínky nejsou splněny, hra na ně při pokusu o spuštění upozorní. Dále by měl být v úrovni vyznačen cíl čistě zelenou barvou 00FF00. Program by nedokázal ověřit zda je možné se ze startu do cíle dostat, proto absenci cíle nedetekuje, a nepovažuje za chybu. Úroveň ale nebude bez cíle možné dokončit.

4.5.2 **Barvy**

Sekce pro výběr barev funguje v podstatě analogicky k sekci místností. Barvy je možné přidávat do, a odebírat z palety. Tou je možné listovat pomocí tlačítek doleva a doprava prostřednictvím proměnné cOffset. Na rozdíl od palety místností má paleta barev ale kapacitu deseti položek než je nutné ní listovat.

Co má sekce barev oproti sekci místností ale navíc, jsou tři ovládací prvky pro zadávání čísel, známé jako NumericUpDown. Tyto prvky umožňují měnit svou číselnou hodnotu pomocí šipek, nebo jí přímo zadat pomocí klávesnice. Jejich rozsah je nastaven na celá čísla od nuly do dvou set padesáti pěti. Každý z těchto tří číselných zadávačů spravuje jednu RGB složku výsledné barvy která se zobrazuje na barevném panelu. Kdykoli se hodnota některého z nich změní, barva panelu se náležitě upraví.

Barva vyobrazená na barevném panelu je aktuální barva používaná ke kreslení. Pomocí tlačítka plus může uživatel přidat tuto barvu do palety. Kliknutím na barvu v paletě se barva načte do číselníků a tedy i do barevného panelu. Stejná barva může být v paletě zastoupena víckerát, při mazání pomocí tlačítka mínus je smazán první výskyt vybrané barvy v řadě.

Řízení proměnné cOffset která zajišťuje listování paletou probíhá stejně jako v případě rOffset u sekce místností.

Na rozdíl od ostatních palet, paleta barev může být prázdná. Není pro funkci úrovně potřebná, a slouží v podstatě pouze jako pomůcka pro urychlení a usnadnění používání.

4.5.3 **Textury**

Sekce textur funguje obdobně jako ostatní sekce. Obsahuje kreslící pole, paletu, tlačítka pro listování, přidávání a odebírání. O listování paletou textur se stará proměnná tOffset, která je řízena stejně jako rOffset a cOffset.

Kliknutím na kreslící pole textur může uživatel nastavit barvu danného pixelu na barvu zvolenou v sekci barev. Pixel se určuje pomocí dělení souřadnice y dvaceti, což je faktor kterým je skutečná velikost textury zvětšena pro zobrazení na kreslícím panelu textur.

Kliknutí pravým tlačítkem na kreslící pole textur načte barvu daného pixelu do číselníků a panelu pro zobrazení barev pro usnadnění měnění barev.

Při vybírání textur z palety textur je zjištěna barva prvního pixelu textury, a ta je načtena do číselníků a panelu pro zobrazení barev. Každá textura je bitmapa o šířce jednoho pixelu, a výšce devíti pixelů. Samotnou texturu která bude vidět ve hře ale tvoří pouze spodních osm pixelů. První pixel je klíčem textury, který označuje její umístění v místnosti. Je nahráván do barvy aby mohl uživatel vybírat textury z palety, a rovnou s nimi osazovat místnosti.

Klíče textur by neměly být stejné, protože jsou v rámci hry uchovávány v Dictionary, který stejné klíče neumožňuje. Pokud uživatel vytvoří úroveň se shodnými klíči textur, bude na tuto chybu upozorněn při pokusu o spuštění.

4.5.4 Ukládání

V pravém dolním rohu se nachází tlačítko Save. Tímto tlačítkem je možné vyexportovat vytvořenou herní úroveň do formátu spustitelného hrou.

```
SaveFileDialog sfd = new SaveFileDialog();
sfd.Filter = "png|*.png";
if(sfd.ShowDialog() == DialogResult.OK)
```

Obrázek 18 - Inicializace a použití SaveFileDialog

Proces začíná vyvoláním instance třídy SaveFileDialog. Tato třída představuje standartní okno pro ukládání souborů v systému Windows. Této instanci je nastavena vlastnost Filter, a to na hodnotu „png|*.png“. Tento zápis specifikuje v jakém formátu může SaveFileDialog soubor uložit. Část před svislou čárou je označení formátu pro uživatele. Část za svislou čárou je zápis který počítači říká že výsledný název souboru má být libovolné

množství libovolných znaků přijatelných pro název souboru, následované příponou .png. Po nastavení tohoto filtru je okno otevřeno pomocí metody ShowDialog. Tato metoda pozastaví kód až do chvíle kdy bude okno zavřeno. SaveFileDialog má výhodu toho, že za nás vyřeší problém shodnosti jmen souborů. Pokud se pokusíme uložit soubor pod jménem které je již ve vybraném adresáři použito, upozorní nás na to, a dá nám možnost soubor přepsat nebo zvolit jiné jméno. Potom co je vybrána platná cesta pro uložení souboru, se zjistí zda dialog skončil úspěšně, a pokud ano, odehraje se zbytek kódu.

```
Bitmap bmp = new Bitmap(50 * rooms.Count > textures.Count + 1 ? 50 * rooms.Count : textures.Count + 1, 59);
```

Obrázek 19 - Rozhodnutí velikosti výstupní bitmapy

Výstupem editoru je obrázkový soubor. Ten se v kódu vytvoří jako bitmapa, její velikost ale není jednoznačná. Výška bitmapy je vždy stanovena na padesát devět pixelů, to aby se zde vešla řada textur o výšce devíti pixelů, a řada místností po padesáti. Šířka se ale musí rozhodnout dynamicky podle počtu textur a místností. Pomocí ternárního operátoru se zjistí, která hodnota je více. Buď počet místností vynásobený padesáti, to proto že každá místnost je padesát pixelů široká, nebo počet textur zvednutý o jednu, to proto že první sloupec je vyhrazený pro funkční barvy. Vyšší hodnota se potom použije jako výsledná šířka bitmapy.

```
using (Graphics g = Graphics.FromImage(bmp))
{
    g.Clear(Color.White);
    bmp.SetPixel(0, 0, Color.Magenta);
    bmp.SetPixel(0, 1, Color.Red);
    bmp.SetPixel(0, 2, Color.Lime);
    bmp.SetPixel(0, 3, Color.Blue);
    bmp.SetPixel(0, 4, Color.Yellow);
```

Obrázek 20 - Vyčištění a nastavení funkčních barev

Pomocí klíčového slova using se vytvoří grafický objekt který umožňuje do bitmapy kreslit. Jeho prostřednictvím se celá bitmapa nastaví na bílou barvu. Bílá barva by měla být pro bitmapu výchozí, tento krok je prováděn jako pojistka proti uložení s průhledností ve formátu bmp.

Pět pixelů nad sebou v prvím sloupci jsou poté nastaveny na jednotlivé výchozí funkční barvy, magenta pro volný prostor, červená pro start, zelená pro cíl, modrá pro aktivační zónu přenášející do následující místnosti, a žlutá pro přenos do předchozí. Následující šestý pixel v řadě zůstává bílý, ale i přes to plní funkci jako barva označující konec seznamu textur.

```
for (int i = 0; i < rooms.Count; i++)
{
    g.DrawImage(rooms[i], new Point(50 * i, 9));
}
```

Obrázek 21 - Cyklus vykreslení místností

Pomocí cyklu for se postupně vykreslí všechny místnosti uložené v listu. Pro opakování kódu pro každou položku v seznamu by byl vhodný cyklus foreach který umožňuje snažší vyvolání momentálně probíraného objektu, kvůli určování pozice vykreslování by ale bylo nutné přidávat iterační proměnnou kterou má cyklus for vestavěnou přímo v sobě, proto je tento zápis preferovaný.

```
for (int i = 0; i < textures.Count; i++)
{
    g.DrawImage(textures[i], new Point(1 + i, 0));
}
```

Obrázek 22 - Cyklus vykreslení textur

Cyklus for je použit podruhé, tentokrát pro vykreslení všech textur na jejich patřičné souřadnice.

```
bmp.Save(sfd.FileName);
```

Obrázek 23 - Uložení bitmapy

Nakonec se pomocí metody save ze třídy Bitmap výsledek uloží na adresu která byla určena v SaveFileDialog. Tímto je proces ukládání dokončen.

4.5.5 Načítání

Vedle tlačítka Save se nachází tlačítko Load. To umožňuje načíst do editoru již existující úroveň za účelem jejího upravování.

```
OpenFileDialog ofd = new OpenFileDialog();
ofd.Filter = "png|*.png";
if(ofd.ShowDialog() == DialogResult.OK)
```

Obrázek 24 - Inicializace a použití OpenFileDialog

Proces načítání začíná podobně jako u ukládání, ovšem místo třídy SaveFileDialog je použita třída OpenFileDialog, která jak jméno napovídá slouží k otevírání souborů. Je jí nastaven filtr aby přijímala pouze soubory formátu png, a po ověření kladného výsledku formuláře se provádí následující kód.

```
rooms.Clear();
textures.Clear();
```

Obrázek 25 - Vynulování listů

V prvním kroce se vymažou veškeré místnosti a textury které mohly být v editoru uložené do této chvíle. Bez toho by byly nově načtené místnosti a textury přidány k předchozím, tento výsledek při otevírání souborů není obvykle žádoucí.

```
Bitmap bmp = new Bitmap(ofd.FileName);
if (bmp.Width < 50 || bmp.Height != 59) MessageBox.Show("Map too small", "Error");
```

Obrázek 26 - Kontrola formátu

Je vytvořena nová bitmapa která vzniká přímo načtením souboru podle výběru v OpenFileDialog do paměti programu. Následně se zkontroluje, zda má bitmapa padesát devět pixelů na výšku, a nejméně padesát na šířku. Kdyby tomu tak nebylo, nebylo by možné načíst žádné místnosti, a program by při pokusu o vykreslení neexistujícího indexu selhal. Pokud je bitmapa v pořádku, program pokračuje následujícím kódem.

```
for (int o = 0; o < bmp.Width / 50; o++)
{
    rooms.Add(bmp.Clone(new Rectangle(o * 50, 9, 50, 50)
}
```

Obrázek 27 - Cyklus přečtení místností

Jednotlivé místnosti jsou načtěny do listu pomocí cyklu for. Množství jeho opakování je vypočítáno podílem šířky úrovně padesáti. Zde může docházet k chybě. Pokud je v souboru takové množství textur, že je šířka úrovně o padesát či více pixelů širší než prostor který vyžadují místnosti, bude volný prostor pod texturami brán jako čistě bílá místnost. Pokud by se uživatel pokusil o spuštění takovéto úrovně, obdrží chybovou hlášku o tom, že všechny místnosti musí mít solidní ohraničení. Kvůli tomu by mohlo být výhodné vyplňovat veškerý volný prostor pod texturami nějakou barvou bez zvláštních vlastností.

```
int i = 1;
while (bmp.GetPixel(i, 0) != bmp.GetPixel(0, 5) && i < bmp.Width)
{
    textures.Add(bmp.Clone(new Rectangle(1 + i, 0, 1, 9), System.I
    i++;
}
```

Obrázek 28 - Cyklus přečtení textur

V následujícím kroku se z úrovně načtou textury. Předpokládá se, že textur je takové množství že nevyplní celou šířku úrovně. Proto se načítání odehrává v cyklu while který má dvě podmínky. První hlídá, jestli je první pixel sloupce který se chystáme načíst odlišný od pixelu na souřadnicích 0, 5. Jak bylo výše zmíněno, tento pixel zůstává bílý, pokud tedy při načítání narazíme na bílý pixel, víme že jsme došli na konec textur. Druhá podmínka hlídá šířku úrovně za pomocí iterační proměnné i. Proměnná i začíná na hodnotě jedna, protože první sloupec úrovně náleží funkčním barvám, a proto ho přeskakujeme. Pokud by hodnota i přesáhla šířku úrovně, víme že nemůžeme dále načítat, a cyklus končí.

```
roomView.Refresh();
roomPalette.Refresh();
textureView.Refresh();
texturePalette.Refresh();
```

Obrázek 29 - Obnovení vykreslovacích panelů

Posledním krokem je obnovení panelů zobrazujících místnost, texturu, a jejich palety, vzhledem k tomu že do nich byl načten nový obsah. Paleta barev zůstává při načítání nezměněná.

4.6 **Záznamník průchodu**

Pro účely uchovávání a sdílení záznamů z průběhu hry byl přidán systém Replay jako nová třída v kódu. Tato třída umožňuje nahrát, a opět reprodukovat hráčovy činy při hraní. Je aktivní po celou dobu hry, a při úspěšném dokončení umožňuje záznam uložit do textového souboru. Pro funkci této třídy bylo nutné upravit a doplnit kód o několik částí.

4.6.1 **Princip**

Třída obsahuje tři proměnné které existují v celém jejím kontextu. První z nich je list typu string jménem inputs. Ten umožňuje uchovávat dynamické množství textových proměnných. Druhou je celé číslo rider. To slouží k určení vybranného záznamu v listu inputs. Poslední je booleová proměnná End. Jedná se o veřejnou vlastnost podle které zbytek programu pozná, jestli přehrávání záznamu došlo do konce.

```
List<string> inputs = new List<string>();
int rider = 0;
public bool End { get; private set; } = false;
```

Obrázek 30 - Globální proměnné třídy Replay

Pro opětovné používání třídy jsou k dispozici metody Reset a Clear, které mažou obsah listu inputs, resetují rider na nulu a End na hodnotu false.

Při každém herním taktu je volána metoda Record která zapisuje do listu inputs informace o tlačítkách které hráč používá, a úhlu ve kterém je natočen. Při přehrávání záznamu je potom místo toho volána metoda Playback, která vrací postupně po sobě jdoucí záznamy z listu inputs.

4.6.2 **Ukládání**

Když hráč úspěšně dokončí úroveň, je mu nabídnuta možnost uložit si záznam svého průchodu. Pokud se rozhodne tak učinit, je ze třídy Replay zavolána metoda Save. Tento proces začíná vyvoláním instance třídy SaveFileDialog, která hráčovi umožní vybrat si cestu pro uložení souboru, a případně vyřeší problémy s duplicitou jmen. Filtr tohoto dialogu je nastaven na soubory s příponou .txt, záznam se tedy ukládá jako textový soubor.

Po úspěšném vybrání cesty pro uložení souboru vytvoří program instanci třídy FileStream a StreamWriter. Třída FileStream umožňuje programu přístup do souboru, jako parametry jsou jí podány cesta k souboru, a FileMode který určuje co má se souborem dělat, v tomto případě vytvořit nový. Třída StreamWriter umožňuje samotný zápis do souboru otevřeného třídou FileStream.

Kvůli nové možnosti umožňující snazší přepínání herní úrovně je nutné asociovat záznam ke konkrétní úrovni na které se má konat, proto se na první řádek souboru zapíše název souboru úrovně na které právě proběhla hra. Poté se cyklicky řádek po řádku do souboru zapíší všechny záznamy z listu inputs. Nakonec program zavře StreamWriter a FileStream aby byl soubor uvolněn z používání programem.

4.6.3 **Načítání**

Načtení záznamu je možné z hlavního menu hry. Obdobně jako u ukládání, tento proces začíná vyvoláním třídy OpenFileDialog nastavené na otevírání textových souborů. Po vybrání souboru se pomocí třídy FileStream nastavené na otevření a čtení souboru a třídy StreamReader začne číst soubor.

Nejprve se načte první řádek, který by měl obsahovat název souboru úrovně na které by se měl záznam odehrávat. Metoda zprostředkovávající tento proces je metoda s návratem typu string, název úrovně je hodnota kterou vrací. Pokud program nenajde požadovanou úroveň, nebo na ní zjistí chybu, vrátí patřičnou chybovou hlášku.

Následně se pomocí cyklu while řádek po řádku přečte každý záznam v souboru, a uloží je do listu inputs, který byl před tím vyčištěn pomocí metody Clear. Když soubor dojde nakonec, cyklus skončí, a nezbývá než uzavřít StreamReader a FileStream.

4.6.4 **Záznam vstupů**

Aby mohla třída Replay ukládat záznamy, musí je nejprve obdržet. O to se stará nově přidaná metoda jménem Package ve třídě hráče. Tato metoda zakóduje aktivní vstupy do textového výstupu. Pro jednoduchost dešifrování byl zvolen systém kde je stisknutá klávesa označena 1 a nestisknutá 0. Pořadí kláves je „nahoru, dolu, doleva, doprava“, tedy pokud by hráč například držel klávesy nahoru a doprava, metoda Package by vrátila „1001“.

Druhý údaj který musí být uložen je informace o směru kterým je hráč natočen. Tato informace je ve třídě hráče uchovávána v proměnné angle, a mění se každý herní takt podle pohybu myši. Pro získání tohoto údaje byla metoda zpracovávající vstup myši upravena na metodu s návratem, která po upravení proměnné angle vrací její hodnotu.

4.6.5 **Přehrávání vstupů**

Do třídy hráč byla přidána nová metoda Replay, která je volána během přehrávání záznamu namísto obvyklých metod které jí umožňují přijímat a zpracovávat vstupy z myši a klávesnice. Mimo to ale vše probíhá stejně jako při hraní hry. Dalo by se říct, že během přehrávání záznamu je hráč v podstatě řízen autopilotem.

Záznamy vytvořené tímto způsobem jsou výrazně méně paměťově náročné než kdyby se jednalo například o video záznam, můžou ale způsobit některé jiné problémy. Pokud je pozměněna úroveň na které záznam probíhá, nebo je přepsán samotný záznam, může dojít k tomu že hráč do cíle netrefí. Pokud záznam skončí dříve než hráč dorazí do cíle, hra se vrátí do hlavního menu, a zobrazí hlášku informující uživatele o selhání záznamu.

Díky formátu záznamu jako textového souboru je možné ho také přepsat úmyslně, za účelem vytvoření teoreticky nejlepšího možného průchodu úrovní.

5 Zhodnocení výsledků

V rámci práce byla aplikace 3D bludiště úspěšně rozšířena o vestavěný editor úrovní který umožňuje snáze vytvářet a sdílet herní úrovně. Uživatel se díky němu nemusí zaobírat formátem souboru, a palety textur a místností umožňují snazší a rychlejší pracovní postup než při používání běžného editoru rasterové grafiky. V souvislosti s tímto přírůstkem byl také upraven postup programu při otevírání souborů. Místo toho aby program načítal předem daný soubor při spuštění nyní program umožňuje před startem hry zvolit soubor ze složky, a teprve poté ho načítá.

Systém záznamů byl rovněž úspěšně implementován. Díky němu je nyní možné vytvářet snadno šiřitelné záznamy sloužící jako průkazný materiál při soutěžení o nejlepší výsledky, nebo jako návody pro řešení úrovní.

Pro rozšíření možností ve hře byly také přidány nové možnosti tvorby úrovní. Pomocí specifických barevných kódů je možné vytvořit více různých triků pro zvýšení zajímavosti hry. Stěny označené barvou jejíž zelená složka má hodnotu 71 nyní reagují stejně jako modré aktivační zóny, a stěny s hodnotou 71 v modré složce stejně jako žluté aktivační zóny. Hodnota 71 v červené složce způsobí, že se stěna nezapíše do kolizní mapy, a je tedy viditelná, ale ne hmotná, hráč může projít skrz ni. Nakonec hodnota 73 v červené složce vytvoří stěnu která není ani hmotná, ani viditelná. Při kombinaci těchto vlastností je možné vytvořit aktivační zóny které nereagují na hráčův pohled, ale na jeho průchod. Čísla 71 a 73 byla zvolena proto, že jejich umístění v kombinaci se skutečností že se jedná o prvočísla by měly vést k menší frekvenci používání, a tedy k menšímu riziku neúmyslné záměny.

V rámci zavádění nových částí programu byla upravena některá uživatelská rozhranní. Hlavní menu nyní obsahuje tlačítka pro spuštění editoru a záznamu, vítězné menu místo opětovného spuštění hry a ukončení programu nyní obsahuje tlačítko pro návrat do hlavního

menu a uložení záznamu. Program byl také rozšířen o nové metody a herní stavy pro správný chod záznamů, a zejména pro možnost jejich pozastavení.

Poslední důležitou změnou která v programu proběhla byla oprava některých známých problémů. V původní verzi program kvůli chybě porovnávání prázdných objektů nedokázal správně rozpoznat že v herní úrovni chybí vyznačení startovního bodu, tento problém je nyní vyřešen. Dále kvůli nesprávnému pořadí operací při startu hry mohl hráč na základě pozice myši při kliknutí na tlačítko start začínat v lehce odlišném směru. Po opravě je směr hráče při startu vždy stejný.

6 Závěr

Aplikace je nyní ve stavu kde má za pomoci několika jednoduchých triků potenciál pro tvorbu a hraní komplexních a neobvyklých úrovní s velkým množstvím zvláštních prvků, které by byly v mnoha známějších hrách podobného žánru složité nebo nemožné vytvořit, a to vše v relativně nenáročném a snadno šiřitelném balení. Zároveň je díky možnosti zaznamenávání času průchodu též otevřena možnost kompetetivního hraní, ke kterému díky použití běžných formátů souboru není potřeba žádných specializovaných přídavných nástrojů.

7 Seznam obrázků, tabulek, grafů a zkratek

7.1 Seznam obrázků

Všechny použité obrázky jsou vytvořeny autorem práce

7.2 Seznam použitých zdrojů

1. Buonanno, E. Functional Programming in C# : How to Write Better C# Code
2. Buonanno, E. Functional Programming in C#, Second Edition
3. Inc. BarCharts, Trigonometry
4. Sturm, O. Functional programming in C#-classic programming techniques for modern projects.
5. Raycasting. Lode Vandevenne [online]. Dostupné z: https://lodev.org/cgtutor/raycasting.html
6. Purdum, J 2008, Beginning C# 3. 0 : An Introduction to Object Oriented Programming, John Wiley & Sons, Incorporated, Hoboken. Available from: ProQuest Eboo-k Central. [2 March 2023].
7. Doom rendering engine - The Doom Wiki at DoomWiki.org. [online]. Dostupné z: https://doomwiki.org/wiki/Doom_rendering_engine