**STŘEDNÍ PRŮMYSLOVÁ ŠKOLA BRNO, PurkYŇOVA, příspěvková Organizace**



**HRA PRO PC**

MARTIN HRABOVSKÝ

V4D

**Profilová část maturitní zkoušky**

**MATURITNÍ PRÁCE**

**BRNO 2018**

# Prohlášení

Prohlašuji, že jsem maturitní práci hra pro PC vypracoval samostatně a použil jen zdroje uvedené v seznamu literatury.

Prohlašuji, že:

Beru na vědomí, že zpráva o řešení maturitní práce a základní dokumentace k aplikaci bude uložena v elektronické podobě na intranetu SPŠ Brno, Purkyňova, příspěvková organizace.

Beru na vědomí, že bude má maturitní práce včetně zdrojových kódů uložena v knihovně SPŠ Brno, Purkyňova, příspěvková organizace, dostupná k prezenčnímu nahlédnutí. Škola zajistí, že nebude pro nikoho možné pořizovat kopie jakékoliv části práce.

Beru na vědomí, že SPŠ Brno, Purkyňova, příspěvková organizace, má právo celou moji práci použít k výukovým účelům a po mém souhlasu nevýdělečně moji práci užít ke své vnitřní potřebě.

Beru na vědomí, že pokud je součástí mojí práce jakýkoliv softwarový produkt, považují   
se za součást práce i zdrojové kódy, které jsou předmětem maturitní práce, případně soubory, ze kterých se práce skládá. Součástí práce není cizí ani vlastní software, který je pouze využíván za přesně definovaných podmínek, a není podstatou maturitní práce.

Martin Hrabovský

Brněnská 1256

664 34 Kuřim

V Brně dne 17. 4. 2018 .........................................................

Vedoucí práce: RNDr. Lenka Hrušková

Oponent práce: Ing. Petr Pernes

# Poděkování

Děkuji RNDr. Lence Hruškové za spolupráci a odbornou pomoc při zpracovávání mého projektu. Také chci poděkovat mojí přítelkyni, Karolíně Hladké, která mi skrz vývoj hry pomáhala a dělala psychickou oporu. Dále bych chtěl poděkovat, jak jí, tak jejímu bratrovi, Janovi Hladkému, za testování různých verzí mojí hry. A jako poslednímu chci poděkovat Františkovi Krejčířovi, za úpravu modelu robota a jeho animování.

# Anotace

Mojí maturitní prací bylo vytvořit počítačovou hru na tématiky naší školy. Hra se odehrává v noční můře v hlavě studenta, který se stresuje z přicházející Maturitní zkoušky. Úkolem je posbírat všech 7 protokolů a dojít s nimi k východu z bludiště. Nesmí se však nechat chytit robotickým učitelem po tom totiž hráč prohrává. Tato dokumentace obsahuje postup vývoje uvedené hry a všechny problémy související s jejím vývojem.

Annotation

My maturita project is about student who has nightmares about upcoming maturita exams. He has to find 7 protocols and get to the exit. However, he has to be careful because there’s robotic teacher roaming the corridors of the maze. This document contains information about development of my game and everything that goes along with it.

# Obsah

[Prohlášení ii](#_Toc511603882)

[Poděkování iv](#_Toc511603883)

[Anotace v](#_Toc511603884)

[Obsah 1](#_Toc511603885)

[Teoretický úvod 3](#_Toc511603886)

[Seznam použitých zkratek 3](#_Toc511603887)

[1 Rozbor řešení 3](#_Toc511603888)

[1.1 Problémy spojené s vytváření hry 3](#_Toc511603889)

[1.1.1 Algoritmus na generování bludiště 3](#_Toc511603890)

[1.1.2 Vzhled hry 4](#_Toc511603891)

[1.2 Použité technologie 5](#_Toc511603892)

[1.2.1 Výběr softwaru 5](#_Toc511603893)

[1.2.2 Programovací jazyk 6](#_Toc511603894)

[2 Algoritmus bludiště 6](#_Toc511603895)

[2.1 Recursive Backtracking 6](#_Toc511603896)

[2.2 Moje verze algoritmu 6](#_Toc511603897)

[3 Ovládání 8](#_Toc511603898)

[3.1 Pohyb 8](#_Toc511603899)

[3.1.1 Pohyb hráče 8](#_Toc511603900)

[3.1.2 Pohyb kamery 9](#_Toc511603901)

[3.2 Script pro interakci 9](#_Toc511603902)

[4 UI 10](#_Toc511603903)

[5 Robot 10](#_Toc511603904)

[5.1 AI 10](#_Toc511603905)

[5.1.1 NavMesh 10](#_Toc511603906)

[5.1.2 Detekce hráče a reakce 10](#_Toc511603907)

[5.2 Model 12](#_Toc511603908)

[5.3 Animace 12](#_Toc511603909)

[6 Grafika 14](#_Toc511603910)

[6.1 Textury 14](#_Toc511603911)

[6.2 Materiály 14](#_Toc511603912)

[6.3 Efekty 15](#_Toc511603913)

[6.3.1 Mlha Světlo a stíny 15](#_Toc511603914)

[6.3.2 Bloom 16](#_Toc511603915)

[7 Zvuk 16](#_Toc511603916)

[8 Testy 17](#_Toc511603917)

[9 Uživatelská příručka 17](#_Toc511603918)

[10 Závěr 18](#_Toc511603919)

[11 Seznam ilustrací 19](#_Toc511603920)

[12 Zdroje 19](#_Toc511603921)

[13 Přílohy 19](#_Toc511603922)

# Teoretický úvod

Počítačová hra na téma naší školy, kde se studentovi zdá noční můra, že musí najít ve škole 7 protokolů k tomu, aby se dostal k maturitním zkouškám. Problém je ten, že chodby   
se proměnily v bludiště, které hráč musí zmapovat, zorientovat se v nich, najít všech   
7 protokolů a dostat se s nimi k východu, aniž by ho dostihl robotický učitel, který bloudí těmito chodbami a nachytání tímto učitelem je pro studenta/hráče konec.

Cílem mojí práce bylo vytvořit hru, která se jednoduše ovládá a vzbudí v hráči nějaký pocit, ať už jde o strach nebo myšlenku. Chtěl bych, aby moje hra předávala myšlenku, kolik jsme toho schopni obětovat za určitou věc a zda to za to vůbec stojí. Jelikož v mé hře hrajete   
za studenta, který má právě noční můry kvůli stresu před maturitou a nechce zklamat   
a udělat vše na poprvé. Tudíž myšlenka hry je, jestli vlastně maturita v prvním termínu stojí za ten stres a psychické problémy, jestli není lepší se nestresovat naučit se na první termín a v nejhorším případě si dodělat maturitu v dalším termínu. Programování hry jsem   
si vybral, jelikož se s tímto tématem setkávám ve volném čase a již dříve jsem se zajímal   
a programoval počítačové hry.

# Seznam použitých zkratek

C# – C Sharp

2D – Prostředí co má dvě dimenze

3D – Prostředí co má tři dimenze

# Rozbor řešení

Moje počítačová hra je z prvního pohledu jednoduchá, 7 věcí k posbírání v bludišti kterým bloudí robot, po několika pokusech, jakmile hráč pochopí chod hry je hra jednoduchá a hráč ji zdolá, nicméně moje hra má za úkol vzbudit v hráči pocity, ať už strach z robota nebo jen temných chodeb, ale hlavně to, kolik jsme schopni obětovat pro to abychom něčeho dosáhly.

## Problémy spojené s vytváření hry

### Algoritmus na generování bludiště

Hlavní myšlenkou mého nápadu byla náhodná generace bludiště, kterým by právě hráč mohl procházet. Na internetu je spousta algoritmů sloužící ke generování bludiště, takže jsem si nejprve musel nějaký vybrat a zpracovat ho. Pro svoji generaci bludiště jsem si vybral tzv. Recursive Backtracking algoritmus.

O funkčnosti algoritmu bludiště a jeho funkčnost je vysvětleno v kapitole 2

#### Problémy s programováním algoritmu

Hlavní problém při programování algoritmu jsem měl s tím, aby jeden pixel byl buď zeď nebo cesta. Jelikož všechny návody na internetu a další články fungovaly na jiném principu než na tom, který jsem já chtěl pro svoji hru, takže jsem celý algoritmus musel naprogramovat podle sebe a mohl jsem se pouze inspirovat v popisu funkčnosti dříve zmíněného algoritmu (Recursive Backtracking). Nakonec se mi tento problém podařilo vyřešil tak, že se můj algoritmus neposunoval po 1 pixelu ale po 2 pixelech. Tím jsem dosáhl požadovaného výsledku, kde jsou zdi 1 pixel a cesta 1 pixel. Také jsem musel vyřešit náhodné vybírání startu a aby se bludiště nedostalo mimo rozmezí, které jsem mu zadal.

#### Implementace algoritmu do Unity

Po úspěšném vytvoření algoritmu, který dokáže vygenerovat bludiště a zakreslit jej do obrázku byl další krok, a to implementace samotného algoritmu do Unity. Sice se mi povedlo vygenerovat obrázek, ale teď mě čekalo vygenerování samotného bludiště   
za pomocí 3D objektů. Nejprve jsem chtěl vygenerovat celé bludiště včetně země, nakonec jsem zemi vložil zvlášť do Unity a generoval jsem pouze zdi bludiště jako „děti“ prázdného objektu „bludiště“.

#### Náhodné generování objektů

Dalším bodem zlomu byla generace objektů (v tomto případě protokolů) náhodně po mapě. Celý problém jsem nakonec vyřešil tak, že jsem velikost bludiště vydělil dvěma a za pomocí výsledku jsem vygeneroval náhodné číslo v rozmezí výsledku a nuly. Toto číslo jsem opět vynásobil dvěma a přičetl k němu jedničku a tím jsem docílil toho, že se nemůže stát, aby se objekt vygeneroval na místě, kde je zeď, protože v každém druhém místě v bludišti   
se nachází cesta.

### Vzhled hry

Před začátkem vývoje hry jsem přemýšlel nad několika variantami a kombinacemi, např. 2D a pixel art, 2D a realistické fotky, 3D a low polly, 3D a pixel art.

#### 3D nebo 2D

Celá moje hra se nejprve měla odehrávat v 2D prostředí. Nejprve byl můj návrh to, že celá hra bude v pohledu shora dolů, pak jsem přešel k nápadu, že by byla 2D z pohledu první osoby (hráče), ale nakonec jsem se ustálil na 3D. Problém s 3D je ten, že je nutné vytvořit modely, na které je pak nutné nastavit materiály a textury. Dále je potřeba postavy animovat a všechny tyto věci by byly jednodušší v 2D. Ale 3D se lépe hodí k žánru mojí hry (horor), také 3D bude ve finále vypadat lépe než jednodušší 2D grafika.

Více o vývoji grafiky je obsaženo v kapitole 6.

#### Výběr stylu grafiky

Pixel art je forma kreslení po jednotlivých obrázkových bodech (pixelech) ve většinou nízkém rozlišení např. 32x32 pixelů (standardní monitor má v dnešní době 1920x1080 pixelů), tento styl grafiky má sice nízkou kvalitu, ale ve správném provedení vypadá hezky a při tom je relativně jednoduchý na nakreslení. Low polly je něco podobného jako Pixel art, ale v 3D světě. Low polly využívá jednoduché 3D objekty a to tak, že používá co nejmenší počet polygonů (každý 3D objekt se skládá z polygonů, které mají podobu trojúhelníků)  
a převážně čistých barev bez jakýchkoliv textur k dosažení krásného, ale jednoduchého vzhledu.

Výběr grafického stylu byl pro mě relativně obtížný, u 2D jsem nejdříve přemýšlel hlavně nad pixel artem a chvíli nad realistickými fotkami (fotky ze školy), avšak tento návrh jsem hned zavrhl. Nakonec jsem zavrhl celé 2D a zvolil jsem něco mezi low polly a skutečnou grafikou.

## Použité technologie

### Výběr softwaru

#### Herní engine

Herní engine je základ většiny moderních her, propojuje jednotlivé elementy a dělá z nich celek (hru), bez enginu je potřeba naprogramovat všechno, včetně toho, jak se objekty mají vykreslit nebo jak mají reagovat při kontaktu s ostatními objekty apod. Při výběru enginu jsem se rozhodoval mezi dvěma možnostmi, Unreal nebo Unity. Jak Unreal tak Unity jsou zadarmo, Unity v tom případě že jej využíváte pro studijní účely nebo jako firma s menším ročním výdělkem jak 100,000$. Oba enginy poskytují dobrou funkčnost   
a podporu různých stylů grafik. Můj výběr bylo Unity, je pro mě lepší, mám už rok zkušeností, hlavně jsem už zvyklý na ovládání Unity, na jeho funkčnost a v programu se mi lépe a rychleji pracuje.

#### Modelovací Software

Modelovací software mi umožní vytvořit jednotlivé 3D objekty pro moji hru. Jelikož jsem si vybral Unity jako herní engine, můj výběr modelovacího softwaru byl jednoduchý. Vybral jsem Blender, umožňuje dobrou funkčnost s Unity, např. načítání modelů v Unity v reálném čase (pokud vytvoříme v Blenderu model, který následně vložíme do Unity a pak tento model dále chceme upravovat, tak pokud jej upravíme a uložíme, tak se automaticky aktualizuje i model v Unity, takže nemusíme opětovně celý model znovu vkládat apod.). Blender je také zcela zdarma.

#### Další software

Další software, co jsem si vybral je Audacity, tento software slouží k úpravě zvukových stop a nahrávání zvuku. Já jsem si ho vybral převážně k úpravě a vytváření zvukových efektů do mojí hry. Audacity je taky volně dostupný ke stažení.

### Programovací jazyk

Další věc, co byla nutná je vybrat si scriptovací jazyk ve kterém bude celá hra napsána. Díky tomu, že jsem si zvolil Unity jako herní engine, tak byl můj výběr programovacího jazyka omezen na dva, jelikož Unity podporuje pouze dva programovací/scriptovací jazyky, a to Java Script, anebo C#. Jelikož dělám v C# už asi 3 roky, tak to pro mě byla také jasná volba.

# Algoritmus bludiště

## Recursive Backtracking

Recursive Backtracking funguje tak, že se vybere začáteční políčko, ze kterého se začne generovat bludiště tím, že se zkontrolují sousední políčka, jestli je možné se do nich přesunout (jestli vůbec tím směrem existuje políčko nebo jestli na tom místě již není cesta) ze všech možných políček se jedno náhodně vybere, algoritmus přidá stávající políčko do stacku a přesune se na další políčko. Pak se zase zavolá ta stejná funkce, takže se zjistí, kam je možné se pohnout, náhodně se vybere jeden směr, stávající políčko je přidané do stacku a algoritmus pokračuje na další políčko. Všechny místa, která jsme navštívili, se ukládají do Stacku (zásobník FIFO – První dovnitř, první ven) s tím, že pokud algoritmus dojde  
do slepého místa (odkud už se není kam přesunout), tak algoritmus zahodí políčko   
ze stacku, kde se právě nachází a přesune se na přechozí, tam se podívá, kam se může pohybovat. Pokud opět nemá kam se pohnout tak se proces opakuje (zahodí políčko   
na vrcholku stacku a dívá se na následující políčko, jestli se z něj má kam pohnout), pokud se ale má kam pohnout tak se pohne a pokračuje v generování bludiště. Pokud se celý stack vyprázdní tak je bludiště kompletně vygenerované.

## Moje verze algoritmu

Celé bludiště je objekt skládající se z dalších menších objektů, políček (buněk), podle velikosti bludiště, s tím že nesmí být velikost bludiště sudé číslo, naštěstí k tomu dojít nemůže, jelikož se velikost kontroluje při vytváření objektu bludiště. Každá buňka má uložené 4 hodnoty, a to svoji pozici x, y, jestli byla navštívena, a jestli se jedná o cestu či nikoliv.

class gridCell

{

public int x;

public int y;

public bool visited;

public bool path;

public gridCell(int x, int y)

{

this.x = x;

this.y = y;

this.visited = false;

this.path = false;

}

}

Moje verze algoritmu nekontroluje sousední políčka ale až 1 políčko za sousedním.   
A jak se vybere směr, kterým se algoritmus přesune, tak se nastaví i políčko mezi původním a tím kam jsme se přesunuli na cestu.

private void visitCell(gridCell visitedCell)

{

visitedCell.visited = true;

visitedCell.path = true;

touch((visitedCell.x+visitedCells.Peek().x)/2,(visitedCell.y+visitedCells.Peek().y)/2).visited = true;

touch((visitedCell.x+visitedCells.Peek().x)/2,(visitedCell.y+visitedCells.Peek().y)/2).path = true;

this.visitedCells.Push(visitedCell);

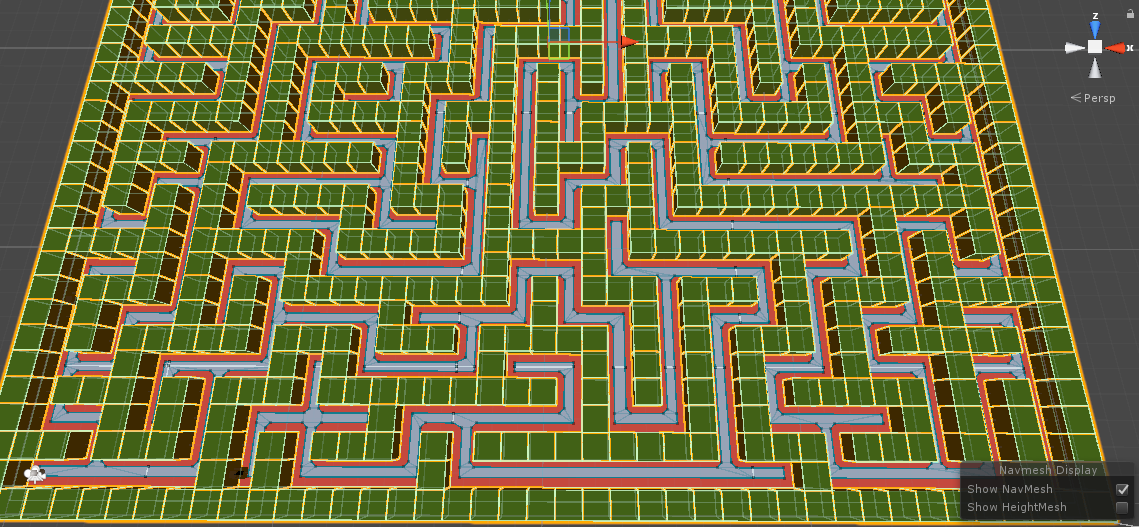
this.currCell = visitedCell;

moveTo();

}

V uvedené funkci visitCell(gridCell visitedCell), můžeme vidět na prvních   
2 řádcích funkce, které políčkům nastaví parametry, že už byly navštíveny, a že se jedná  
o cestu. Další 2 řádky se právě starají o to, aby políčko mezi políčkem, kde se právě nacházíme, a políčkem, kam se chceme přesunout, bylo taktéž navštíveno a nastaveno jako cesta.

Tím jsem docílil toho, že moje bludiště má stejně tlusté zdi jako je samotné bludiště.



Obrázek 1: Bludiště v unity

# Ovládání

Pro svoji hru jsem potřeboval udělat nějaký způsob k přesouvání se po bludišti a sbírání předmětů, jako jsou protokoly, anebo interakci s nimi např. východové dveře. Na ovládání hráčovy postavy slouží 2 scripty, první script slouží k pohybu hráče a kamery a druhý slouží právě k interakci s objekty.

## Pohyb

Pohyb je řešen jedním scriptem, který je rozdělen na 2 pomyslné části. Jedna na ovládání hráče jako takového, pohyb po ose x a y. Druhá slouží k otáčení pohledu (kamery).

### Pohyb hráče

Pohyb hráče je řešen jednoduchým scriptem, jelikož v mojí hře stačí pohyb po ose x, y a není třeba řešit jakékoliv skákání nebo gravitaci.

moveDir = new Vector3(Input.GetAxis("Horizontal"), 0, Input.GetAxis("Vertical"));

moveDir = transform.TransformDirection(moveDir);

if(Input.GetKey(KeyCode.LeftShift)) //Sprint

{

controller.Move(moveDir \* Time.deltaTime \* speed \* 2f);

}

else

{

controller.Move(moveDir \* Time.deltaTime \* speed);

}

V ukázce, na prvních dvou řádcích, se vytvoří Vector3 (vektor, co má 3 dimenze) do kterého jsou vloženy informace o pohybových osách x, y, které jsou snímány z klávesnice (w, s, a, d nebo šipek), tyto hodnoty se nachází v rozmezí 1 až -1. Dále následuje funkce, která kontroluje, jestli je nebo není držena klávesa sprintu (levý shift), pokud není, tak je aplikován pohyb na hráčovu postavu tím, že se vynásobí vektor hodnot z klávesnice časem který byl potřeba pro vytvoření minulého snímku a rychlostí, avšak je-li držena klávesa sprintu, tak je toto všechno ještě vynásobeno dvěma.

### Pohyb kamery

Pohyb kamery je o něco složitější, kdyby se totiž otáčela pouze kamera, tak by hráč šel jiným směrem než tím, kterým by chtěl. Pohyb kamery je řešen tak. že při otáčení doleva  
a doprava se otáčí celou postavou a při pohledu nahoru a dolů se otáčí pouze kamer, tímto způsobem je dosaženo vyžadovaného a přirozeného pohybu.

## Script pro interakci

Interakce s předměty je dělána pomocí tzv. SphereCast(), ten dělá to, že z určitého bodu „vystřelí“ neviditelnou kouli (zvolil jsem SphereCast(), jelikož při jeho použití se musí dívat hráč přímo na objekt, se kterým lze interagovat), tato funkce vrací objekt, do kterého narazila (pokud do nějakého narazila). Takže jsem využil této funkce, a to tak že „střílím“ tyto „koule“ z kamery směrem, kterým se dívá hráč, s tím, že jsem všem objektům nastavil „tag“ na „usable“, tudíž stačí kontrolovat, jestli jeden z objektů který vrací SphereCast() má „tag“ „usable“, jestliže má, tak je mu nastaven jiný materiál, který dělá, že se objekt rozsvítí žlutě, jestliže se hráč podívá jinam, je objektu nastaven původní materiál. Když je objekt zvýrazněn, tak může hráč zmáčknou interakční tlačítko „E“, které mu umožní s předmětem interagovat, (např. vzít).

# UI

UI neboli User Interface (uživatelské rozhraní) je tvořeno pouze dvěma důležitými prvky, a to jednoduše vysvětlené ovládání a počítadlo které informuje kolik z kolika protokolů hráč posbíral.

# Robot

Robot není moje vlastní tvorba, vybral jsem ho a stáhl jsem ho ze stránek kde jsou různé modely a další grafické prvky volně k stažení zcela zdarma viz. zdroje.

## AI

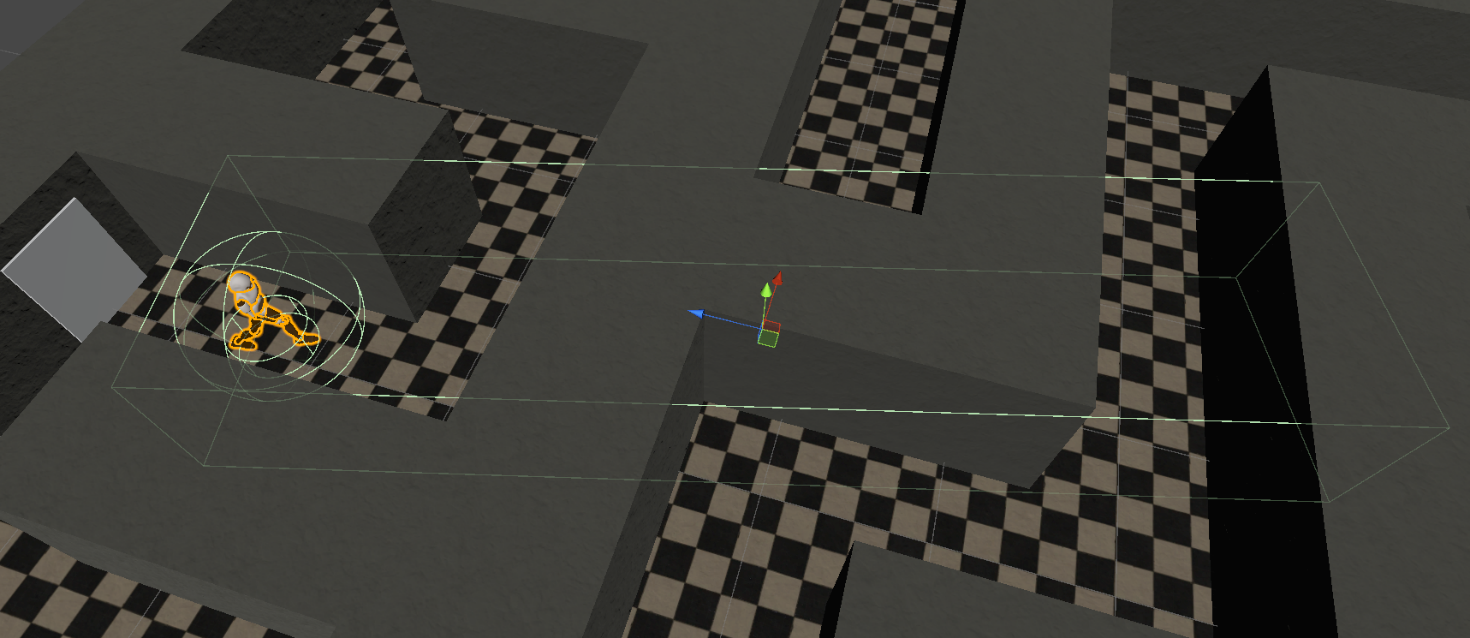
AI neboli artificial inteligence (umělá inteligence) musí splňovat 3 prvky, musí se umět pohybovat po bludišti, musí umět detekovat, že se před ním nachází hráč a musí na tuto situaci reagovat, a když se hráči nepodaří uniknout, tak ho vylekat pomocí jumpscaru. Celý script se skládá z 2 částí, a to čekání na místě, kde měl dojít, a další, kde se vygeneruje nový cíl, k němu cesta a robot se po ní vydá k cíli. Jumpscare je řešen scriptem, který je zvlášť.

### NavMesh

Využil jsem tzv. NavMesh v Unity, NavMesh mi umožnil, vygenerovat plochu po které  
se bude moct pohybovat tzv. NavMeshAgent. Vygenerovaný NavMesh můžeme vidět na obrázku „Bludiště v Unity 1“. NavMesh se generuje po vygenerování bludiště. Robot  
se pohybuje po modře vyznačených částech, stačí mu dát vektor pozice, kam chceme, aby  
se vydal, a on se tímto směrem vydá, pokud najde k tomuto místu cestu. Já jsem udělal, aby robot začínal pokaždé přede dveřmi, kde je východ z bludiště a náhodně si vybral jeden protokol ke kterému půjde, když dojde na místo chvíli tam stojí rozhlédne se a vybere další náhodný objekt ke kterému se pohybuje.

### Detekce hráče a reakce

K detekci hráče využívám dvou prvků, první je neviditelný kvádr, který je před robotem, tento objekt funguje jako tzv. Trigger, to znamená že detekuje, když se do tohoto objektu dostane nějaký jiný objekt.



Obrázek 2: Neviditelné objekty

void OnTriggerStay(Collider other)

{

if(other.tag == "Player")

{

if(Physics.Raycast(transform.position, (other.gameObject.transform.position - transform.position), out hit, maxRange))

{

if(hit.transform.tag == "Player")

{

agent.speed = runningSpeed;

if(robotAnim.GetFloat("Speed") < 2f)

{

robotAnim.SetFloat("Speed",2f);

walking.mute = true;

running.mute = false;

robotFlashLight.color = color;

robotFlashLight.spotAngle = 40f;

robotFlashLight.intensity = 0.75f;

}

Debug.Log(robotAnim.GetFloat("Speed"));

agent.destination = hit.transform.position;

}

}

}

}

Pokud se do tohoto kvádru dostane jiný objekt, tak se zavolá funkce OnTriggerEnter()  
a následně OnTriggerStay(). Pro mé účely jsem využil hlavně OnTriggerStay(), který se volá pokaždé, co se nachází nějaký objekt v právě zmiňovaném neviditelném kvádru, nejdřív  
je nutné zkontrolovat, jestli se opravu jedná o hráče (opět kontroluji podle tagu). Další částí scriptu je kontrola toho, jestli je hráč opravdu viditelný pro robota (jestli se nenachází za zdí nebo jiným objektem), toho dosáhnu pomocí RayCast(), který funguje podobně jako SphereCast(), jen je tentokrát vystřelován paprsek. Tento paprsek narazí do objektu a vrátí tento objekt, tudíž pokud je hráč za zdí, tak vrátí pouze zeď (objekt co nemá tag „player“ neboli hráč). Pokud je vrácený objekt hráč, tak se nastaví pozice hráče jako cíl, kam se má robot dostat a zvětší se jeho rychlost, změní se jeho animace, změní se zvuk a také barva světla, kterým robot svítí před sebe. Jestliže se dostane robot ke hráči dostatečně blízko, tak se hráč dostane do dalšího neviditelného objektu, tentokrát kouli, která když detekuje hráče tak zpustí jumpscare script, který se stará o chod celého jumpscaru, tento script zapne neviditelnou kameru, na které je animace jumpscaru, také se stará o to že, zapne zvuk jumpscaru.

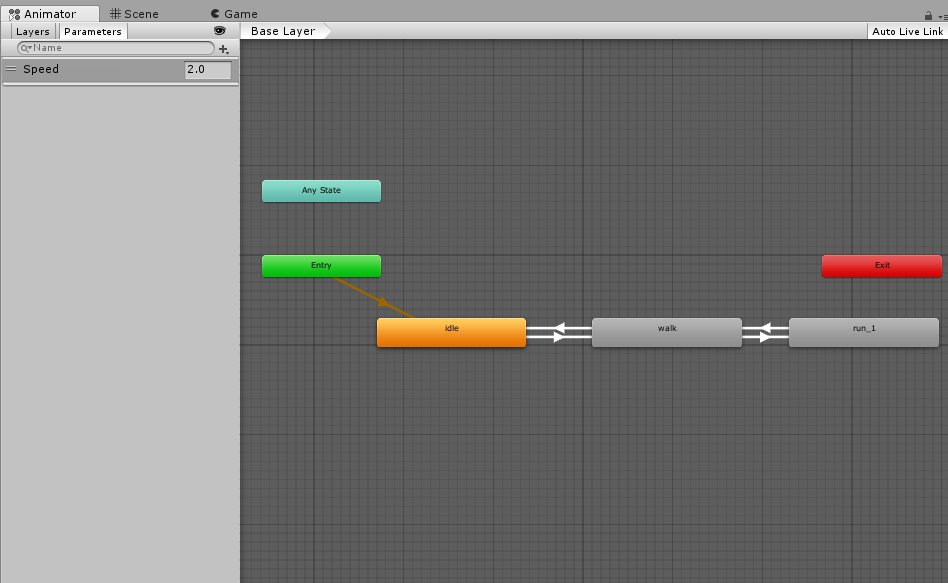
Více o zvuku naleznete v kapitole 7.

## Model

Celý model je vymodelovaný v programu Blender, uživatelem novak. Z webu [cgtrader](https://www.cgtrader.com/free-3d-models/character/other/fully-rigged-doll), viz. zdroje. K modelu jsem v unity přidal pouze zmiňované neviditelné objekty pro funkčnost  
a také 2 světla, jedno jako světlo z očí, kterým si robot svítí na cestu a druhé jako světlo, které má simulovat reálné světlo z baterky, které osvěcuje robota, toto světlo má simulovat rozptýlení světla, kterému by došlo ve skutečném životě. Unity umí vypočítávat odražené světlo, ale tato varianta by byla příliš nákladná na výkon.

## Animace

Veškeré animace byly vytvořeny Františkem Krejčířem ze školy SŠUM. František vytvořil animaci pro chození, utíkání, stání a také jumpscare. Jelikož jsem měl model s již vytvořenými animacemi, tak mi stačilo tyto animace pouze po propojovat v tzv. Animation Controlleru (ovladač animace), který mi umožňuje lehce nastavit přechody mezi jednotlivými animacemi a podmínky za kterých k přechodům dojde.



Obrázek 3: Animation controller

Mojí podmínkou byla rychlost, kterou se robot pohyboval. Nastavil jsem tedy do ovladače animace proměnou pro rychlost. Tuto proměnou jsem pak měnil ve scriptu pro ovládání robota. Když robot stojí je rychlost rovna nule, když jde, tak je rychlost rovna jedné, a když utíká je rychlost rovna dvěma. Přechody mezi animacemi si Unity dokáže samo vypočítat.

Obrázek 4: Textura podlahy

Obrázek 5: Textura zdi

# Grafika

U grafiky jsem vybíral z mnoha různých stylů. Nakonec jsem využil realisticky vypadající textury pro podlahu a zdi z webu [textures](https://www.textures.com). Můj cíl byl, aby hra byla temná a tím nabudit strašidelnou atmosféru. K nabuzení atmosféry jsem využil různých efektů a stylů světel  
co nabízí Unity. Dále jsem využil balíčku od Unity s post-processing efekty.

## Textury

Pro textury jsem si vybral, textury zdarma z internetu (viz. zdroje). Pro podlahu jsem vybral černobílé dlaždičky, které umožní jednoduchou orientaci při prvním zapnutí hry,  
kdy si hráč podvědomě spojí tento vzhled k podlaze. Pro zdi jsem vybral texturu bílé omítky, opět pro co nejjednodušší rozlišení zdí a podlahy. Původní vzhled byl totiž, červená podlaha a zelené zdi, což většinu hráčů zmátlo a trvalo jim delší dobu, než se zorientovali.

## Materiály

Jelikož web textures má podporu i pro různé více propracované textury, tak jsem toho využil a použil jsem height mapy a normal mapy, které dodají výslednému materiálu hloubku, a tím výsledný vzhled je oživen o třetí rozměr, díky tomuto zdi nevypadají tak hole a podlaha vypadá, jako by byla opravdu vyskládána z jednotlivých dlaždiček.

Obrázek 6: Robot dívající se na hráče

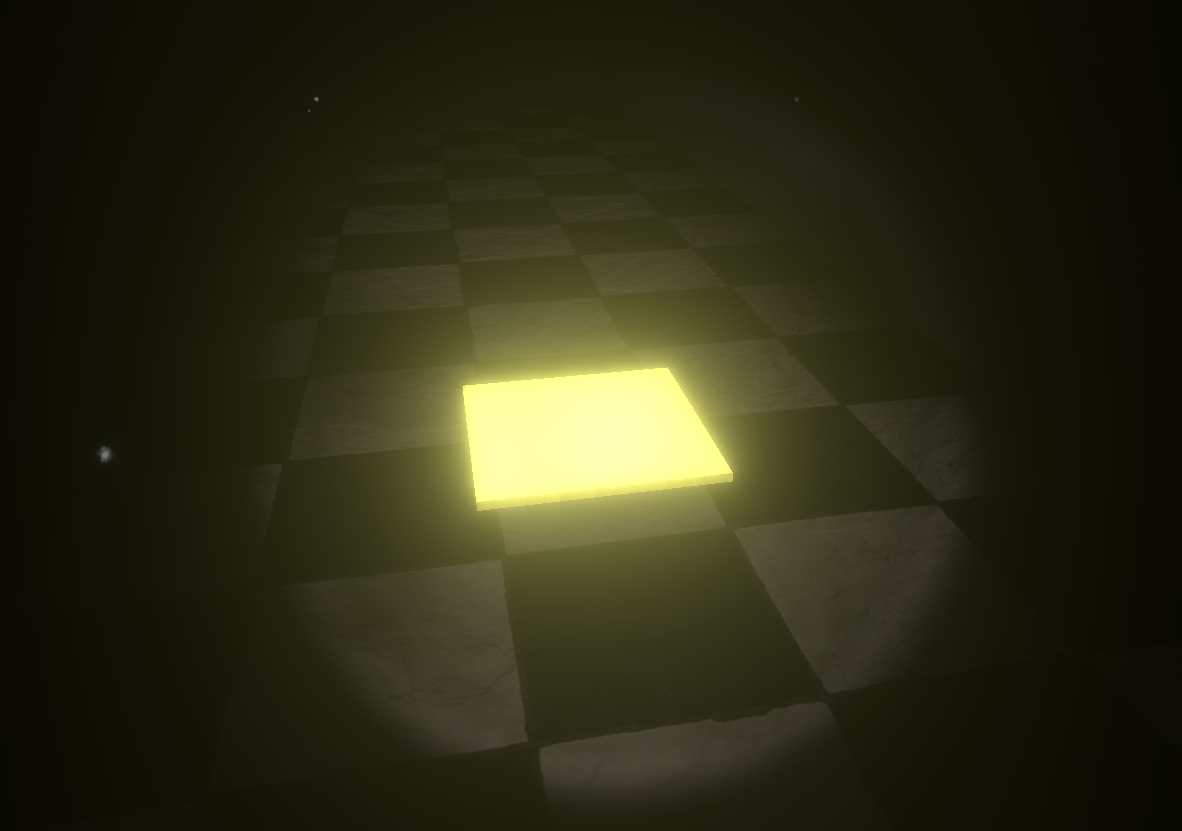
## Efekty

Pro efekty jsem využil balíčku Unity, který umožňuje na kameru přidat různou škálu efektů, ale hlavní efekty, které jsem využil já byly mlha, bloom následně i oprava různých barev a světel.

### Mlha Světlo a stíny

S tímto efektem byl asi největší problém, jelikož unity podporuje mlhu pouze na materiály, které jsou renderování určitým stylem. Hlavní problém bylo to, že moje bludiště funguje tím stylem, že se vezme vzorový objekt pro zdi a ten se nakopíruje několikrát do hry, aby tvořil zdi bludiště, avšak tyto objekty neumí pracovat se světli a stíny, takže jsem musel vytvořit krátký script, který prošel tyto objekty a nastavil jim správný materiál, problém je že tento materiál neumí pracovat s mlhou, takže jsem musel nastavit tzv. Globální mlhu. Která funguje na všechny objekty, které se ve hře nachází. Takže se mi povedlo dosáhnout toho, že jsem měl mlhu a už nesvítili světla skrze objekty. Dále jsem potřeboval dosáhnout efektu, že je robot vidět na konci chodby, jelikož na něj působila globální mlha tak bylo nejdříve vidět světlo z baterky robota a pak až robot. Tohoto efektu jsem dosáhl tak že jsem přidal hráč 2 kamery s tím, že na jedné je efekt mlhy a na druhé ne, druhá kamera se stará pouze o vykreslování robota na obrazovku.

### Bloom

Bloom efekt nám umožňuje simulovat oslnění světlem a dále nám pomáhá rozšířit simulaci realistického šíření světla prostorem.

Obrázek 7: Bloom

# Zvuk

Zvuky jsem stáhnul zdarma z webu [freesound](https://freesound.org/) (viz. zdroje). Odtud jsem stáhl zvuk pro jumpscare a také jsem tu stáhl zvuk dupnutí, tento zvuk jsem následně upravil v programu Audacity, kde jsem zvuk zkopíroval, změnil jsem jeho hloubku, abych vytvořil efekt chození dvou nohou a aby každý krok nebyl identický. V unity jsem následně ještě zrychlil zvukovou stopu, jak pro chodící animaci, tak pro utíkající.

# Testy

Moje hra byla testována asi třemi lidmi, s tím že mi každý z nich dali zpětnou vazbu, co   
a jak bych měl zlepšit. Já jsem uvážil, jestli jde o rozumnou změnu k lepšímu nebo jestli by tato změna ovlivnila nějaké prvky mé hry.

Bohužel k aplikacím ve formě her se nedají dělat jednotkové testy.

# Uživatelská příručka

Veškeré informace o ovládání jsou ve hře, úkolem je posbírat 7 předmětů, nenechat se chytit robotem a dostat se ven z bludiště za pomocí dveří.

# Závěr

V mojí maturtiní práci „Hra pro pc“ jsem splnil všechny požadavky a jednotlivé body uvedené v zadání, které jsem doplnil o další funkce. Po různých testech jsem došel k závěru, že hra dokáže zabavit, navodit strašidelnou atmosféru, udržet si pozornost hráče a nabudit adrenalin v hráči. Ve hře se neobjevují žádné zásadní chyby, které by zabraňovaly k jejímu dokončení.

Doplnil jsem si své znalosti herního designu, co funguje na papíře a co v reálném světě. Zjistil jsem, že je důležité u her znát pohled někoho jiného a sledovat, jak reaguje na různé elementy hry. Také jsem se naučil využívat psychologických triků k tomu, aby má hra byla lehčeji pochopená a hratelná i méně znalými hráči. Hra sama o sobě nenaučí hráče všechny její prvky, ale pouze základní ovládání, s tím že hráče nenápadně navádí ke správnému řešení.

Také jsem si zlepšil své dovednosti s jazykem C# a herním enginem Unity. Naučil jsem  
se lépe pracovat, jak se samotnými C# knihovnami, tak i knihovnami Unity jako je   
např. Navmesh nebo UI apod.

Jak jsem již zmínil, tak jsem si musel rozšířit své vědomosti o různé psychologické prvky, abych dokázal navodit správnou atmosféru a hráče skutečně vylekal děsivým, ale i přesto neodrazujícím způsobem.

Jsem velice rád za svoji spolupráci s Františkem Krejčířem, který mi pomohl s modelem  
a jeho animacemi, kdybych si měl tyto věci dělat sám, tak bych si příště musel vymezit více času na naučení práce s grafickým programem Blender a na vlastní návrhy, modelování postavy a její animování.

Výše jste se již mohli dočíst, že se zajímám o hry, jak z ohledu hraní, tak i po stránce jejich funkčnosti a jejich vytváření. Pro svoji maturitní práci jsem se inspiroval ve hře  
Joy of Creation, která je inspirována sérií her Five Nights at Freddy's.

# Seznam ilustrací

[Obrázek 1: Bludiště v unity 8](file:///C:\Users\hrabo\Desktop\obhajobaHra\Documentation\Martin-Hrabovsky_Obhajoba-Hra.docx#_Toc511597993)

[Obrázek 2: Neviditelné objekty 11](#_Toc511597994)

[Obrázek 3: Animation controller 13](#_Toc511597995)

[Obrázek 4: Textura podlahy 13](file:///C:\Users\hrabo\Desktop\obhajobaHra\Documentation\Martin-Hrabovsky_Obhajoba-Hra.docx#_Toc511597996)

[Obrázek 5: Textura zdi 13](file:///C:\Users\hrabo\Desktop\obhajobaHra\Documentation\Martin-Hrabovsky_Obhajoba-Hra.docx#_Toc511597997)

[Obrázek 6: Robot dívající se na hráče 14](file:///C:\Users\hrabo\Desktop\obhajobaHra\Documentation\Martin-Hrabovsky_Obhajoba-Hra.docx#_Toc511597998)

[Obrázek 7: Bloom 16](#_Toc511597999)

# Zdroje

1. *Microsoft Docs [online]. Bill Wagner, 2018 [cit. 2018-04-15]. Dostupné z:* [*https://docs.microsoft.com/cs-cz/dotnet/csharp/*](https://docs.microsoft.com/cs-cz/dotnet/csharp/)
2. *Unity Documentation [online]. Unity Technologies, 2018 [cit. 2018-04-15]. Dostupné z:* [*https://docs.unity3d.com/Manual/index.html*](https://docs.unity3d.com/Manual/index.html)
3. *Textures.com [online]. Nizozemí: Textures.com, 2018 [cit. 2018-04-15]. Dostupné z:* [*https://www.textures.com/*](https://www.textures.com/)
4. *Cgtrader-model [online]. novak, 2016 [cit. 2018-04-15]. Dostupné z:* [*https://www.cgtrader.com/free-3d-models/character/other/fully-rigged-doll*](https://www.cgtrader.com/free-3d-models/character/other/fully-rigged-doll)
5. *Blender.org [online]. Nizozemí: Blender Foundation, 2018 [cit. 2018-04-15]. Dostupné z:* [*https://www.blender.org/*](https://www.blender.org/)
6. *Audacity [online]. WordPress, 2018 [cit. 2018-04-15]. Dostupné z:* [*https://www.audacityteam.org/*](https://www.audacityteam.org/)
7. *Freesound [online]. Španělsko: Freesound, 2018 [cit. 2018-04-15]. Dostupné z:* [*https://freesound.org/*](https://freesound.org/)

# Přílohy

Příloha 1: Zdrojové kódy a projekt v Unity

Příloha 2: Výsledná sestavená hra