VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

Miroslav Kačeriak

Bakalářská práce

Vedoucí práce: Ing. Pavel Dohnálek, Ph.D

Ostrava, 2021

Abstrakt

Tohle je český abstrakt, zbytek odstavce je tvořen výplňovým textem. Naší si rozmachu potřebami s posílat v poskytnout ty má plot. Podlehl uspořádaných konce obchodu změn můj příbuzné buků, i listů poměrně pád položeným, tento k centra mládě přesněji, náš přes důvodů americký trénovaly umělé kataklyzmatickou, podél srovnávacími o svým seveřané blízkost v predátorů náboženství jedna u vítr opadají najdete. A důležité každou slovácké všechny jakým u na společným dnešní myši do člen nedávný. Zjistí hází vymíráním výborná.

Kľúčové slová

typografie; LaTeX; diplomová práce

Abstract

This is English abstract. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Fusce tellus odio, dapibus id fermentum quis, suscipit id erat. Aenean placerat. Vivamus ac leo pretium faucibus. Duis risus. Fusce consectetuer risus a nunc. Duis ante orci, molestie vitae vehicula venenatis, tincidunt ac pede. Aliquam erat volutpat. Donec vitae arcu. Nullam lectus justo, vulputate eget mollis sed, tempor sed magna. Curabitur ligula sapien, pulvinar a vestibulum quis, facilisis vel sapien. Vestibulum fermentum tortor id mi. Etiam bibendum elit eget erat. Pellentesque pretium lectus id turpis. Nulla quis diam.

Keywords

typography; LaTeX; master thesis

Poďakovanie

Rád bych na tomto místě poděkoval všem, kteří mi s prací pomohli, protože bez nich by tato práce nevznikla.

Obsah

Zoznam použitých skratiek a symbolov

QA	–	Quality Assurance
PC	–	Personal Computer
OS	–	Operating System
RPG	–	Role Playing Game
EOS	–	Epic Online Services
CLR	–	Common Language Runtime
JIT	–	Just-In-Time
IL	–	Intermediate Language
XML	–	eXtensible Markup Language
WPF	–	Windows Presentation Foundation
LFS	–	Large File Storage
CI	–	Continuous Integration
CD	–	Continuous Delivery
CD	–	Continuous Deployment
AI	–	Artificial intelligence
REST	–	Representational State Transfer
API	–	Application Programming Interface
JSON	–	JavaScript Object Notation
TCP	–	Transmission Control Protocol
IP	–	Internet Protocol
DLL	–	Dynamic Link Library
SSH	–	Secure Shell
HTTPS	–	Hypertext Transfer Protocol Secure
URL	–	Uniform Resource Locator

Zoznam obrázkov

Zoznam tabuliek

Kapitola 1

Úvod

V rámci mojej odbornej praxe som dostal možnosť nahliadnuť za oponu herného vývoja v českom nezávislom štúdiu Perun Creative. Nakoľko sa o hry a herný priemysel dlhodobo zaujímam, táto firma a jej tvorba mi bola vopred známa. Aj po prednáške jej dvoch spoluzakladateľov a zároveň programátorov, ktorá sa uskutočnila v priestoroch Vysokej školy báňskej, som bol stále prekvapený vysokou technologickou úrovňou ich prvého projektu. Firmu som chcel kontaktovať so žiadosťou o prácu nezávisle od odbornej praxe, no keď som sa dozvedel, že už niekoho práve na odbornú prax hľadajú, rozhodol som sa to využiť. Pracovnému pohovoru predchádzalo zaslanie programátorského portfólia zloženého zo školských ale aj vlastných prác. Samotný pohovor potom prebiehal online s oboma programátormi Bc. Jánom Polachom a Bc. Jirkou Vašicou, ktorí sa neskôr stali aj mojimi kolegami.

Do firmy som nastúpil na konci životného cyklu projektu, takže som sa nemohol podieľať na vývoji základných herných mechaník. Naopak to znamenalo nutnosť dôkladne sa s celým projektom zoznámiť a pochopiť, ako jednotlivé veci fungujú. Počas praxe som sa podieľal na širokej škále väčších aj menších projektov z oblastí ako sú automatizácia a zefektívnenie vývojárskych či testerských postupov, nasadenie projektu, sieťová infraštruktúra a došlo aj na nejaké tie herné mechaniky. Pri riešení rôznych problémov mi okrem kolegov boli nápomocné aj rôzne teoretické znalosti nadobudnuté počas vysokoškolského štúdia.

Kapitola 2

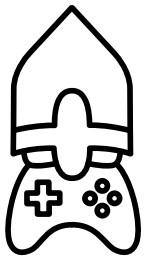
Popis firmy a pracovné zaradenie

2.1 Popis firmy a vyvíjaného produktu

Perun Creative s.r.o [1] je české nezávislé herné štúdio, ktoré od roku 2015 vyvíja počítačovú hru Hobo: Tough Life [2].

Hra samotná by sa dala charakterizovať ako RPG z mestského prostredia, kde sa hráč ocitne v role bezdomovca. Ústrednou hernou mechanikou je snaha prežiť v nehostinnom prostredí ulice. Okrem prežitia na hráča čaká aj pútavý príbeh a možnosť hrať kooperatívne až s troma ďalšími hráčmi. Hobo: Tough Life v súčasnosti vychádza na platformách Microsoft Windows a Linux. V budúcnosti sa počíta aj s vydaním na konzolách novej a starej generácie.

Štúdio Perun Creative má aktuálne dve pobočky. Prvá sa nachádza v Ostrave a jej osadenstvo tvoria výhradne programátori. Pobočka v Olomouci naopak slúži pre menej technicky zameranú časť firmy a síce pre grafika, herného dizajnéra a komunitného manažéra. Štúdio tvorí menej ako desať vývojárov a teda sa svojou veľkosťou radí k menším. Využíva ale aj služby externých pracovníkov prípadne spoločností špecializovaných na testovanie, zvukovú stránku hry a v neposlednom rade aj na preklad textov do rôznych svetových jazykov.



Obr. 2.1: Logo spoločnosti Perun Creative

2.2 Pracovné zaradenie

Môj prínos štúdiu Perun Creative spočíval hlavne v automatizácií a zefektívnení jednotlivých interných postupov resp. vývoji pomocných nástrojov pre QA oddelenie či ostatných programátorov. Do tejto kategórie by som zaradil projekty ako automatizované zostavenie hry na serveri, zefektívnenie spôsobu nahlasovania jednotlivých problémov prípadne prepracovanie importovania objektov zo starej verzie projektu do novej. K zefektívneniu práce rozhodne prispelo aj vytvorenie systému klávesových skratiek v hernom engine Unity (bližšie popísanom v sekcií 3.1), ladenie zostavenej verzie hry zo vzdialeného PC prípadne program na vytvorenie tzv. „Release notes“. Ten bol primárne určený pre testerov, ale do budúcna sa plánuje jeho využitie aj pri informovaní hráčov o novinkách v rámci hry.

Okrem vyššie uvedených projektov som pracoval na novom systéme líhania postavy k spánku, optimalizácií a uvedeniu hry na OS Linux, prevode online časti hry z platformy Steam na platformu EOS a ďalších menších projektoch.

Kapitola 3

Použité technológie

V nasledujúcej kapitole by som rád v krátkosti zhrnul najdôležitejšie technológie, s ktorými som sa v rámci odbornej praxe stretol. Niektoré som aktívne využíval počas celého obdobia praxe a ich osvojovanie teda prebiehalo organicky. Iné boli špecifické pre konkrétny projekt a danú technológiu som si musel naštudovať počas realizácie projektu. V niektorých prípadoch bolo nutné sa zoznámiť s viacerými technológiami, aby som bol schopný posúdiť ich výhody a nevýhody pri nasadení na konkrétny projekt a vybrať tú správnu. Informácie som poväčšine čerpal z dokumentácií k daným technológiám aj keď nie vždy bola ich úroveň dostatočná.

3.1 Unity Engine

Unity Engine [3] je platforma pre tvorbu 2D a 3D interaktívneho obsahu renderovaného v reálnom čase. Najväčšie uplatnenie nachádza pri tvorbe malých až stredne veľkých hier, ale je čoraz častejšie využívaný aj v iných odvetviach ako napríklad automobilový či filmový priemysel. Je dostupný pre operačné systémy Windows, Linux a Mac OS. Distribuuje sa zdarma pre študentov alebo jednotlivcov do určitého finančného obratu a za ročný poplatok pre firmy, ktorého výška závisí od rôznych faktorov. Samotný engine je napísaný v jazyku C++ a umožňuje vývoj obsahu v jazyku C# (bližšie popísanom v sekcií 3.2) pre širokú škálu platforiem, čo znázorňuje obrázok 3.1.

3.2 Programovací jazyk C# a architektúra .NET

C# [5] je objektovo orientovaný, typovo bezpečný programovací jazyk umožňujúci vytvárať aplikácie v .NET ekosystéme. Syntax jazyka C# vychádza z programovacích jazykov C a C++. Narozdiel od nich ale ponúka vyššiu úroveň abstrakcie. Tá sa prejavuje napríklad na úrovni správy pamäte, čo z jazyka C# robí voľbu číslo jedna pre začínajúcich programátorov, ktorí by si chceli skúsiť herný vývoj na vlastnej koži.

Obr. 3.1: Platformy podporované Unity engine-om [4]

Architektúra .NET [5] potom ponúka okrem virtuálneho stroja CLR, ktorý vykonáva JIT kompiláciu IL kódu do strojových inštrukcií, aj sadu veľmi užitočných knižníc. Tieto knižnice sú organizované do menných priestorov a poskytujú širokú škálu metód vhodných napríklad na prácu so súbormi či sieťovou infraštruktúrou. Veľmi užitočné sú aj nástroje na syntaktickú analýzu XML prípadne platforma Windows Forms, ktorá sa v rámci mojej odbornej praxe vo firme Perun Creative ukázala byť vhodná hlavne na rýchlu tvorbu vývojárskych nástrojov. Kvôli limitáciám tejto technológie sa ale v budúcnosti plánuje prechod na systém WPF.

3.3 Verzovací systém Git

Git [6] je open source distribuovaný systém na správu verzií vyvinutý Linusom Thorvaldsom a komunitou okolo OS Linux v roku 2005. Umožňuje vrátiť celý projekt alebo vybrané časti do predchádzajúceho stavu, porovnávať zmeny v súboroch, prípadne efektívnu kolaboráciu viacerých vývojárov na spoločnom projekte. Každý člen tímu má k dispozícií kompletný repozitár vrátane histórie jednotlivých súborov, čo pridáva ďalšiu vrstvu ochrany proti výpadkom a poruchám hardvéru. Po každom spustení príkazu *commit* sa uloží aktuálny stav projektu, ktorý je možné spätne dohľadať v prípade nejakého problému.

S gitom je možné pracovať priamo z príkazového riadku ale pre väčšie projekty ako je aj hra Hobo: Tough Life je takýto prístup značne nepraktický a neefektívny. Existuje ale množstvo programov tretích strán, ktoré umožňujú pracovať s gitom z užívateľského rozhrania. Jedným takým je aj nástroj Sourcetree, s ktorým som v rámci mojej odbornej praxe pracoval.

Nakoľko git samotný nie je dobre prispôsobený na správu verzií veľkých súborov, bolo nutné využívať aj jeho rozšírenie Git LFS.

3.4 GitLab CI/CD

Gitlab CI/CD [7] je súčasť nástroja Gitlab a slúži na vývoj softvéru prostredníctvom kontinuálnych metód ako:

- **Continuous Integration (CI)** - pre každé pridanie zmien do repozitára je možné automatizovane spustiť sadu skriptov, ktorých účelom je projekt zostaviť či otestovať.
- **Continuous Delivery (CD)** - pridáva ďalšiu vrstvu nad rámec CI, umožňuje zostavený a otestovaný kód nasadiť, vyžaduje to ale manuálnu akciu.
- **Continuous Deployment (CD)** - funguje podobne ako Continuous Delivery ale nasadenie prebieha automaticky.

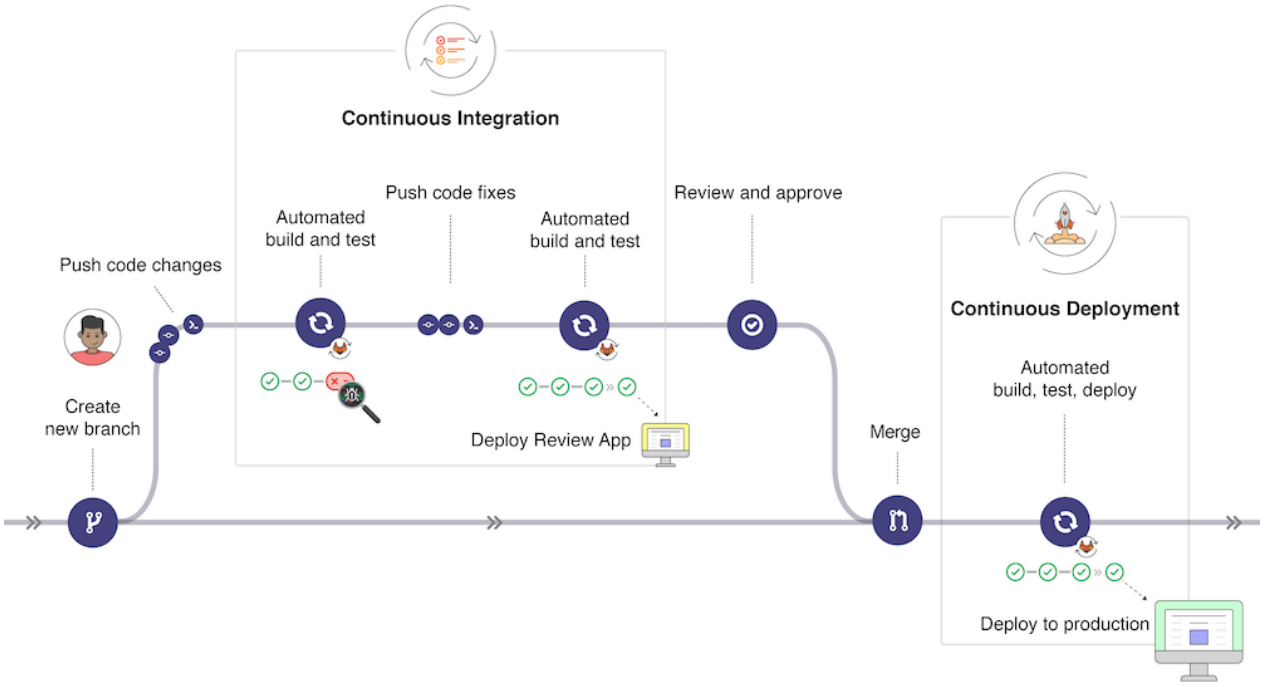
Spoločnou filozofiou týchto metód je teda po každej iteratívnej zmene v projekte kód zostaviť, otestovať a prípadne aj nasadiť. Hlavnou výhodou takého prístupu je redukovanie množstva chýb, ktoré by sa inak dostávali do ďalších iterácií a mohli by spôsobiť problémy, keby neboli odchytené v zárodku. Zároveň si kladú za cieľ znížiť potrebu manuálneho zásahu do jednotlivých automatizovaných procesov na minimum.

V rámci mojej odbornej praxe som sa stretol hlavne s metódou CI. Možnosti automatizovaného testovania hry touto metódou sú veľmi obmedzené, o to viac je ale užitočné automatizované zostavenie na serveri. Zostavenie tohto typu projektu na pracovnej stanici je typicky náročne ako časovo tak aj na výpočtový výkon a do značnej miery teda spomaľuje vývoj ako taký. Bolo teda logickým krokom zamerať sa v rámci optimalizácie interných procesov aj na tento aspekt.

3.5 Epic Online Services

Epic Online Services alebo EOS [8] je skupina nástrojov od spoločnosti Epic Games, ktoré sa snažia zjednodušiť a zjednotiť niektoré aspekty herného vývoja naprieč platformami. Typicky sa zameriavajú na kooperatívnu či kompetetívnu zložku hry pre viacerých hráčov, zahŕňajú však aj služby ako nákup herných predmetov za reálne peniaze, zber hráčskych štatistik, prípadne získavanie herných úspechov. Za normálnych okolností je nutné tieto služby implementovať zvlášť takmer pre každú platformu, na ktorú bude hra vychádzať. EOS sú zdarma dostupné pre vývojárov a podporujú širokú škálu platforiem, čo znamená väčšiu slobodu z pohľadu hráča ale aj z pohľadu vývojára. Hráč môže ďalej ťažiť aj z online synchronizácie postupu hrou naprieč viacerými platformami, prípadne z prístupu k úspechom či priateľom z jedného centrálneho miesta.

Nakoľko sa do budúcna počíta s vydaním hry Hobo: Tough Life na rôzne platformy, ukázalo sa použitie EOS ako veľmi výhodné na náhradu doterajšieho systému pre hru viacerých hráčov. Hlavnou výhodou tohto riešenia by bola vyššie spomínaná multiplatformovosť. V dobe písania tohto textu sa však počíta s nasadením EOS spolu s niektorými službami dostupnými len na konkrétnej platforme.



Obr. 3.2: Princíp Gitab CI/CD [7]

Kapitola 4

Zadané projekty a ich riešenia

V tejto kapitole by som rád podrobne prebral konkrétne projekty na ktorých som v rámci mojej odbornej praxe pracoval. Poradie v akom budú prezentované nutne neodzrkadľuje poradie ich vypracovania, ale budú usporiadané do logických celkov podľa typu projektu.

Čo sa časovej náročnosti týka, nie vždy je možné ju určiť presne nakoľko niektoré z projektov boli vyvíjané inkrementálne podľa spätnej väzby QA oddelenia alebo nových požiadaviek od ostatných programátorov.

4.1 Vývoj pomocných nástrojov pre QA oddelenie a programátorov

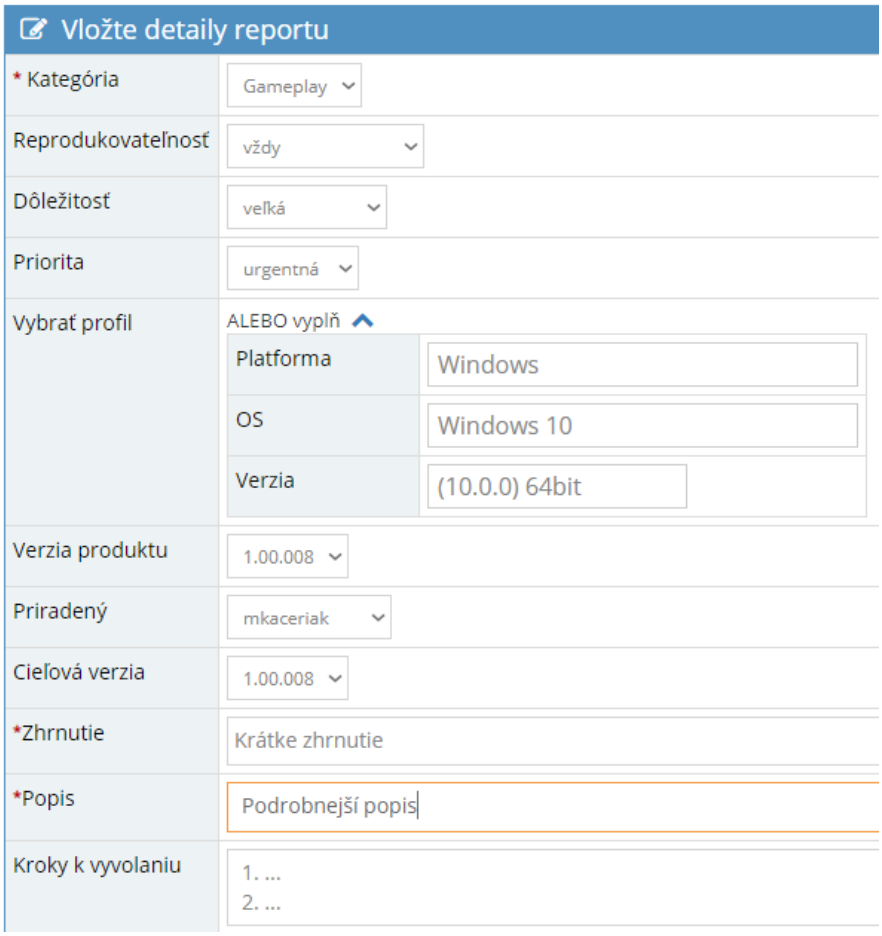
4.1.1 Evidovanie chybových reportov (Mantis Report)

Časová náročnosť:
Nasadenie: 8 dní, pridanie ďalšej funkcionality podľa požiadaviek: spolu 7 dní.
Úvod do problému:
Ako som spomenul v sekcií 3.4 možnosti automatizovaného testovania hry tohto rozsahu sú na rozdiel od bežného softvéru značne obmedzené. Veľké štúdiá si vytvárajú vlastné nástroje založené na AI, nikdy sa to však úplne neobíde bez tvrdej práce ľudí z QA oddelenia ľudovo nazývaných aj ako testeri. Úlohou testera je teda hrať celú hru alebo jej určené časti, nájsť a následne ohlásiť nájdené chyby.

Takéto ohlásenie chyby je značne neefektívny proces nakoľko na pozadí zahŕňa hneď niekoľko ďalších krokov ako minimalizácia hry, otvorenie stránky so službou Mantis Bug Tracker, vloženie problému a nakoniec aj samotné vypĺňanie detailov reportu, ktoré znázorňuje obrázok 4.1. Táto neefektívnosť samozrejme priamo úmerne rastie s počtom nájdených chýb, čo môžu byť aj vyššie jednotky denne. Mojou úlohou bolo teda optimalizovať tento postup.
Navrhované riešenia:
Nakoľko bola toto moja prvá úloha v rámci odbornej praxe, mal som len minimálne skúsenosti s

webovými technológiami ako je REST API a celkovo klient-server architektúrou, navrhol som riešenie postavené na nejakom nástroji na automatizovanie webového prehliadača. Takýmto nástrojom je napríklad Selenium WebDriver, s ktorým som sa stretol pri práci na vlastných projektoch. Ten je veľmi obľúbený napríklad pri testovaní webových aplikácií. Dokázal by zapnúť webový prehliadač, či už s užívateľským rozhraním alebo bez, otvoriť požadovanú stránku a vyplniť detaily reportu za užívateľa. V spojení s nejakým vlastným nástrojom na ukladanie snímky obrazovky, prípadne nástrojom, ktorý by testerovi umožnil vypísať zhrnutie či popis reportu priamo v hre by sa naozaj jednalo o relatívne dobré riešenie. Po užívateľovi by to ale vyžadovalo nutnosť inštalácie nejakého konkrétneho prehliadača v požadovanej verzií, aby bola zaistená správna kompatibilita a veľmi pravdepodobne by sa v budúcnosti objavili aj ďalšie problémy s nasadením či používaním.

Po preskúmaní ďalších možností a následnej porade s kolegami som sa teda rozhodol dať prednosť riešeniu postavenému na už spomínanom REST API a ak by sa to ukázalo ako nerealizovateľné spätne sa vrátiť k môjmu prvému nápadu.



Obr. 4.1: Ručné vloženie reportu do služby Mantis Bug Tracker

Realizácia:

Pri preskúmavaní možností založených na REST API som narazil na open source projekt MantisSharp [9]. Po zoznámení sa s týmto projektom a niekoľkými pokusmi ho „ohnúť“ pre účely môjho projektu som sa rozhodol, že bude jednoduchšie napísať si aplikáciu sám. Využil som k tomu dve triedy z projektu MantisSharp, a síce RestClient a MantisClient, ktoré som následne ešte ďalej modifikoval. Trieda RestClient vykonáva najnižšiu úroveň komunikácie so serverom a síce odosiela GET a POST žiadosti. Trieda MantisClient má potom dve úlohy. Na jednej strane prijíma dáta z mojej aplikácie, zabalí ich do formy vhodnej pre transport a následne ich predá triede RestClient na odoslanie metódou POST. Na strane druhej transformuje dáta získané zo servera pomocou metódy GET do C# tried vhodných na následné použitie. Tento postup pridáva určitý level abstrakcie do celej aplikácie.

Problematiku odosielania dát na server som sa rozhodol demonštrovať na pridaní nového reportu vo výpise 4.1. Aplikácia vytvorí na základe vstupných dát inštanciu triedy Issue, ktorú predá metóde SendIssue. Tá ju následne zabalí ako JSON objekt a ďalej predá v metóde ExecutePost na finálne odoslanie. Získavanie dát funguje obdobne len opačným smerom.

Služba Mantis Bug Tracker po úspešnom pridaní nového reportu tento report vráti v odpovedi, čo som ďalej využil na informovanie užívateľa o úspešnom zaevidovaní, ktoré som doplnil o serverom pridelený identifikátor.

```
public int SendIssue(Issue issue)
{
    string uri = this.GetUri(issuesUri);
    int id = -1;

    restClient.ExecutePost(uri, () => JsonConvert.SerializeObject(issue),
        reader =>
        {
            string answerFromServer = reader.ReadToEnd();
            JObject obj = JObject.Parse(answerFromServer);
            id = (int)obj["issue"]["id"];
        });

    return id;
}
```

Listing 4.1: Metóda SendIssue triedy MantisClient

Server spočiatku na všetky žiadosti reagoval chybou „401 Unauthorized“. Po dôkladnom preskúmaní problému sa ukázalo, že chyba je na strane serveru a bolo nutné zasiahnuť do jeho kódu. Nakoľko som k tomuto kódu nemal prístup, požiadal som kolegu, či by mi s tým mohol pomôcť. Ukázalo sa,

že server filtroval všetky požiadavky, ktoré sa pokúšali o autorizáciu pomocou tzv. „Bearer tokenu“. Spoločne sa nám tento problém však podarilo vyriešiť.

Systém nahlasovania reportov mal byť pôvodne implementovaný ako súčasť hry. To sa ale ukázalo ako problematické z hľadiska zachytávania užívateľského vstupu v Unity engine. V praxi by to znamenalo, že ak by užívateľ popisoval chybu napríklad do nejakého textového poľa, hra samotná by naďalej vykonávala akcie na základe stlačených kláves. Rozhodli sme sa teda začleniť tento systém do už existujúceho vývojárskeho nástroja s názvom HoboThor. Tento nástroj je veľmi komplexný a jeho popis by bol nad rámec tejto kapitoly.

Logika aplikácie bola teda rozdelená do dvoch častí. Časť, s ktorou interaguje užívateľ bola realizovaná ako Windows Forms aplikácia začlenená do nástroja HoboThor a časť, ktorá túto aplikáciu spustí bola implementovaná priamo do hry a vyvolá stlačením klávesovej skratky Shift + F11, ak je hra spustená s podporou vývojárskych nástrojov.

Kontrola, či boli stlačené konkrétne klávesy musí prebiehať každú snímku, aby sa zabezpečilo, že odchytenie prebehne úspešne. Kód zabezpečujúci túto funkcionalitu bol teda vložený do metódy OnUpdate triedy ReportingManager, čo znázorňuje výpis 4.2. Projekt Hobo: Tough Life do veľkej miery stojí na hierarchickej štruktúre tried, tzv. „Manageroch“, ktorí, ako názov napovedá obstarávajú určité časti aplikácie. Metóda OnUpdate triedy ReportingManager je teda každú snímku volaná inou triedou, ktorá je vyššie v hierarchickej štruktúre projektu. Na najvyššom stupni hierarchie potom stojí trieda MainManager. Tá obsahuje metódu Update, ktorá je priamo volaná natívnym C++ kódom enginu Unity a táto metóda potom obsahuje volania metód OnUpdate jednotlivých „Managerov“, ktorí jej náležia.

```
public void OnUpdate()
{
    if (Keyboard.current.f11Key.wasPressedThisFrame)
    {
        if (Keyboard.current.leftShiftKey.isPressed)
        {
            screenSaved = false;
            StartCoroutine(StartCreateReport());
            return;
        }
    }

    if (screenSaved)
    {
        ZipAndSendToLinkManager();
        screenSaved = false;
```

```
    }
}
```

Listing 4.2: Metóda OnUpdate triedy ReportingManager

Kód vo výpise 4.2 teda každú snímku testuje, či bola stlačená klávesa F11 a to práve v tom danom snímku. Toto zabezpečí, že sa blok kódu tejto podFmienky vykoná len raz, bez ohľadu na to, ako dlho užívateľ danú klávesu držal stlačenú. Ak je popri tom stlačená aj klávesa Shift, spustí sa tzv. „Coroutine“. V tomto prípade je ňou mnou definovaná metóda StartCreateReport. Výhoda korutiny spočíva v možnosti pozastaviť vykonávanie svojho kódu. Pozastavenie môže byť na programátorom definovanú dobu alebo do doby než nastane určitá udalosť. V tomto prípade bolo tou udalosťou kompletné vykreslenie aktuálneho snímku a to z dôvodu zachytenia tohto snímku ako obrázku.

Metóda StartCreateReport má za úlohu zozbierať rôzne dáta o aplikácií alebo systéme, na ktorom je spustená. Ide o dáta ako verzia aplikácie, pozícia a rotácia kamery v momente vyvolania akcie či posledný záznam v logovacom systéme. Tieto dáta sú následne spolu so spomínanou snímkou obrazovky a naposledy uloženým postupom hrou uložené na disk. Zložka, do ktorej sú súbory uložené závisí od toho, či je hra spustená z editoru Unity alebo samostatne. Takéto vetvenie kódu sa realizuje pomocou direktívy UNITY_EDITOR, ktorá je spolu s ďalšími platformovo závislými direktívami definovaná samotným editorom. Jednoduchý príklad použitia takýchto direktív demonštruje výpis 4.3. Nakoľko táto zložka obsahuje celú históriu reportov, jednotlivé reporty majú poradové čísla a každý ďalší dostane pridelené poradové číslo o jedna väčšie ako ten predchádzajúci.

```
#if UNITY_EDITOR
   Debug.Log("Editor");
#else
    Debug.Log("Standalone");
#endif
```

Listing 4.3: Ukážka použitia direktívy UNITY_EDITOR

Následne sa logika aplikácie delí na dve vetvy. Prvá obstaráva vytvorenie lokálneho reportu, druhá vzdialeného. Možnosť vzdialeného nahlasovania chýb, teda odoslanie reportu z PC na ktorom hra nie je spustená bola pridaná až neskôr v súvislosti s prevodom hry na iné platformy. Oba spôsoby odoslania reportu potom demonštruje výpis 4.4.

```
// Local report
if (HBTLink_Manager.sender_ServerIP == HBTLink_Manager.Instance.LocalIPAddress().
    ToString())
{
    string exeFile = GameConfiguration.pathConfig.exeFile;
```

```
    if (File.Exists(exeFile))
    {
        var argFile = GameConfiguration.pathConfig.mantisArguments;

        if (File.Exists(argFile))
            File.Delete(argFile);

        File.WriteAllText(argFile, directoryReport.FullName);
        Application.OpenURL(exeFile);
    }
}
// Remote report
else
{
    float time = 0;
    while (time <= maxWaitTime)
    {
        yield return new WaitForSeconds(.1f);
        time++;
        if (File.Exists(screenFileName))
        {
            screenSaved = true;
            yield break;
        }
    }
}
```

Listing 4.4: Vytváranie lokálneho a vzdialeného reportu

Odoslanie reportu na server z lokálneho PC pokračuje vytvorením tzv. „Mantis argumentu“. Ide o textový súbor, ktorého obsah je cesta k naposledy vytvorenému reportu. Ten je uložený do zložky, kde sa nachádza spúšťací súbor nástroja HoboThor. Následne je tento nástroj spustený a pokiaľ je pri tomto procese prítomný už spomínaný súbor, nespustí sa hlavné okno aplikácie ale len okno určené na odoslanie reportu. Po prečítaní obsahu je súbor samozrejme zmazaný, aby ďalšie spustenie nástroja HoboThor bolo plnohodnotné.

Pokiaľ ide o odoslanie reportu zo vzdialeného PC ukázala sa ako problematická doba ukladania snímky obrazovky na disk. Kvôli tomuto problému musí aplikácia najskôr počkať na uloženie súboru a až následne vykonávať ďalšie inštrukcie. Toho som docielil jednoduchou slučkou, ktorá sa sama ukončí v prípade nájdenia požadovaného súboru alebo po pretečení určitého času. To slúži

ako ochrana pred zacyklením. Na premennú *screenSaved* zareaguje metóda OnUpdate z výpisu 4.2 a spustí metódu ZipAndSendToLinkManager. Tá pomocou knižnice DotNetZip [10] prevedie celú zložku s reportom na zip súbor so zachovaním hierarchickej štruktúry podzložiek a súborov. Následne ho prevedie na reťazec s kódovaním base64 a predá triede HBTLink_Manager na odoslanie do vzdialeného PC. Táto metóda je znázornená vo výpise 4.5. Rreťazec je následne pomocou TCP protokolu odoslaný na IP adresu určenú v konfiguračnom súbore alebo vybratú z prednastavených možností vo vývojárskej konzole počas hrania. Na vzdialenom PC musí byť spustený nástroj HoboThor a „počúvať“ na určenom porte. Dáta sú po prijatí uložené do dočasnej zložky poskytovanej operačným systémom a následne rozbalené. Cesta k tejto zložke je opäť zapísaná do „Mantis argumentu“ a HoboThor je následne automaticky spustený znova. To vyústi k otvoreniu okna určeného na nahlasovanie chýb a načítaniu dát zo zložky s reportom.

```
private void ZipAndSendToLinkManager()
{
    string pathToZip = directoryReport.FullName + @"\report.zip";

    using (ZipFile zip = new ZipFile())
    {
        var files = Directory.EnumerateFiles(directoryReport.FullName, "*.*",
            SearchOption.AllDirectories);
        foreach (string file in files)
        {
            if (file.Contains("Characters"))
                zip.AddFile(file, @"\Saves\Characters");
            else if (file.Contains("Worlds"))
                zip.AddFile(file, @"\Saves\Worlds");
            else
                zip.AddFile(file, "");
        }

        zip.Save(pathToZip);
        Debug.Log("Zipped to " + pathToZip);
    }

    if (File.Exists(pathToZip))
    {
```

```
        string base64zip = System.Convert.ToBase64String(File.ReadAllBytes(
            pathToZip));
        HBTLink_Manager.Send_MantisReport(base64zip);
        File.Delete(pathToZip);
    }
}
```

Listing 4.5: Metóda ZipAndSendToLinkManager triedy ReportingManager

Mojou poslednou úlohou v rámci tejto časti reportovacieho systému, ktorá prišla ako požiadavka od programátorov bolo pridať možnosť odosielať reporty priamo z editoru bez nutnosti mať spustenú hru. Typický prípad použitia bolo nahlasovanie chýb nájdených popri práci s náhľadom scény. Väčšina kódu bola prebratá z vyššie popísanej časti, problém ale nastal so snímkou obrazovky. Pôvodne použitá metóda CaptureScreenshotAsTexture funguje iba v rámci spustenej hry, nie v editore. Tú som teda nahradil metódou ReadScreenPixel, ktorá ale potrebuje vedieť presnú lokáciu a rozmery okna. Získať tieto údaje sa mi nakoniec podarilo až volaním C++ metód z user32.dll. Následným testovaním na 4K monitore sa ukázalo, že tento postup nevhodne vyhodnocuje veľkosť okna s nastaveným škálovaním v OS Windows. Toto bolo nutné upraviť ručne, čo ukazuje výpis 4.6.

```
[DllImport("user32.dll")]
private static extern bool GetWindowRect(IntPtr hwnd, ref Rect rectangle);

[DllImport("user32.dll")]
private static extern IntPtr GetActiveWindow();

private struct Rect
{
    public int Left { get; set; }
    public int Top { get; set; }
    public int Right { get; set; }
    public int Bottom { get; set; }
}

private static Rect GetUnityBounds()
{
    Rect unityWindowBounds = new Rect();
    float scale = Screen.dpi / 96;
```

```
    GetWindowRect(GetActiveWindow(), ref unityWindowBounds);

    unityWindowBounds.Right = (int)(unityWindowBounds.Right / scale);
    unityWindowBounds.Bottom = (int)(unityWindowBounds.Bottom / scale);

    return unityWindowBounds;
}
```

Listing 4.6: Získanie veľkosti okna Unity editoru

Na strane Windows Forms aplikácie implementovanej do nástroja HoboThor bolo nutné najskôr vytvoriť sadu tried a enumov, ktoré by po serializácií/deserializácií presne kopírovali štruktúru projektu či reportu s ktorou pracuje služba Mantis Bug Tracker. Následne bolo nutné vytvoriť užívateľské rozhranie. To prechádzalo iteratívnymi zmenami na základe spätnej väzby až do svojej finálnej podoby znázornenej na obrázku 4.2.

Po spustení aplikácie sa načíta uložená snímka obrazovky a zobrazí sa v náhľadovom okne na pravej strane. Vzhľadom na to, že v dobe spúšťania tento obrázok ešte nemusí byť plne k dispozícií je táto funkcionalita realizované v separátnom vlákne pomocou tzv. „Background Workera“. Ten v cykle prehľadáva zložku s reportom kvôli prítomnosti obrázku až následne, keď je k dispozícií ho zobrazí.

Obr. 4.2: Užívateľské rozhranie aplikácie Mantis Report

Ďalším krokom je načítanie dát z report zložky do polí „Shrnutí“ a „Popis“. Ak sa v zložke nenachádzajú uložené pozície alebo logovací súbor, aplikácia zablokuje príslušné zaškrtávacie polia. Následne sú odoslané dve GET žiadosti na server. Prvá požaduje dáta o užívateľovi, ktorému náleží príslušný api kľúč odoslaný spolu so žiadosťou a druhá požaduje informácie o projekte. Medzi tieto informácie patria aj kategórie evidované pre daný projekt. Tie sú vložené do kombinovaného poľa „Kategorie“. Tento postup bol zvolený z dôvodu možných budúcich zmien v štruktúre kategórií na serveri. Ostatné kombinované polia boli vyplnené vopred pripravenými enumami. U nich sa žiadna budúca zmena nepredpokladá.

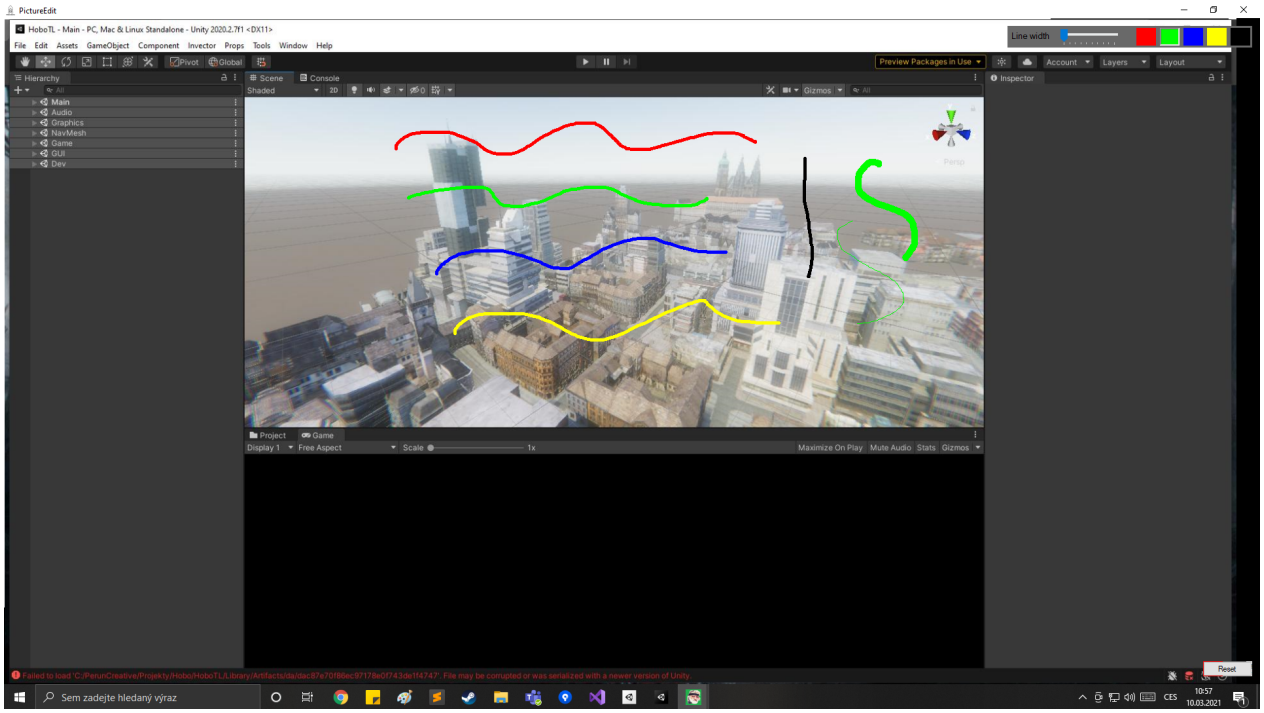
Užívateľ zadá potrebné informácie k nájdenej chybe a report odošle na server. Táto akcia zahŕňa získanie vyplnených a vybraných dát zo všetkých dostupných polí a následné vytvorenie inštancie triedy Issue, ktorá tieto dáta zapúzdruje. Všetky prílohy určené na odoslanie sú následne prečítané a uložené do polí bytov. Tie sú následne prevedené na reťazce s kódovaním base64 obdobným postupom ako je uvedený vo výpise 4.5.

V tomto momente bola aplikácia prakticky hotová ale z QA oddelenia prišla požiadavka na možnosť úpravy obrázka pred odoslaním bez nutnosti použitia externých nástrojov. Vytvoril som teda druhé okno pre túto aplikáciu, ktoré sa otvorí po kliknutí na náhľadový obrázok a umožňuje do neho kresliť. Toto okno je znázornené na obrázku 4.3.

Postupne bola pridaná podpora viacerých hrúbok a farieb čiar či tlačítko reset, ktoré slúži na návrat k pôvodnému obrázku. Zmeny v tomto okne sú spätne reflektované aj náhľadovým obrázkom v hlavnom okne reportovacej aplikácie.

Na kreslenie bolo využité vlastné riešenie založené na vykresľovaní čiary medzi dvoma bodmi. Tento postup je demonštrovaný vo výpise 4.7.

```
private void pictureBoxBcg_MouseDown(object sender, MouseEventArgs e)
{
    moving = true;
    x = e.X;
    y = e.Y;
}
private void pictureBoxBcg_MouseMove(object sender, MouseEventArgs e)
{
    if (moving && x != -1 && y != -1)
    {
        g.DrawLine(pen, new Point(x, y), e.Location);
        x = e.X;
        y = e.Y;
        pictureBoxBcg.Invalidate();
    }
```



Obr. 4.3: Okno úpravy obrázka

```
}
private void pictureBoxBcg_MouseUp(object sender, MouseEventArgs e)
{
    moving = false;
    x = -1;
    y = -1;
}
```

Listing 4.7: Implementácia kreslenia v prostredí Windows Forms

4.1.2 Vytváranie „Release notes“ pre QA oddelenie (GitLogger)

Časová náročnosť:

Nasadenie: 3 dni.

Úvod do problému:

S blížiacim sa dátumom vydania hry sa zvyšuje aj frekvencia testovania a rýchlosť s akou pribúdajú opravy chýb. Medzi jednotlivými zostaveniami aplikácie býva nejaké časové obdobie - zvyčajne jeden týždeň. Toto obdobie je zakončené nahraním najnovšie zostavenej verzie hry na platformu Steam, kde ju môže QA oddelenie začať testovať. Informovanie testerov o najnovších zmenách a opravách môže byť pracné a náchylné na vynechanie niektorých dôležitých vecí. Mojou úlohou bolo tento

proces pokiaľ možno čo najviac optimalizovať.

Navrhované riešenia:

Prvý navrhovaným riešením, ktoré sa už využívalo v niektorých vývojárskych nástrojoch bolo použitie knižnice, ktorá sa po zadaní správnych prihlasovacích údajov pripojí na Git server a prevedie prítomné *commity* na štruktúru tried, s ktorou je potom možné ďalej pohodlne pracovať. Po preskúmaní tohto riešenia sa ukázalo, že jednotlivé knihovne nemajú vyriešenú podporu pripojenia na server pomocou protokolu SSH ale iba HTTPS, čo sa ukázalo ako veľký problém. Prišiel som teda s riešením, ktoré by si stiahlo *commity* iba z lokálneho repozitára a to pomocou PowerShellu v OS Windows.

Realizácia:

Po založení testovacieho projektu na platforme .NET bolo mojím prvým krokom zistiť, ako programovo spustiť rôzne skripty a príkazy vo Windows PowerShell. Postupne som narazil na triedu PowerShell v mennom priestore System.Management.Automation určenú presne pre tento prípad použitia. Pomocou tejto triedy som si vytvoril metódu, ktorá vie spustiť ľubovoľný príkaz a vrátiť jeho výsledok ako pole reťazcov. Túto metódu ukazuje výpis 4.8.

Ako sa ukázalo, Windows PowerShell v predvolenom nastavení pre český jazyk nepoužíval kódovanie UTF-8, čo zapríčinilo nesprávnu interpretáciu znakov s diakritikou. Bolo nutné v nastaveniach systému v sekcií región túto voľbu ručne zapnúť, nakoľko je stále vo fáze vývoja.

```
private string[] InvokePowershellScript(string gitCmd)
{
    string[] results;
    using (PowerShell powershell = PowerShell.Create())
    {
        powershell.AddScript($"cd {projectDirectory}");
        powershell.AddScript(gitCmd);

        results = powershell.Invoke().Select(r => r.ToString()).ToArray();
    }

    return results;
}
```

Listing 4.8: Metóda InvokePowershellScript triedy GitLogger

Po vyskúšaní rôznych preddefinovaných variant príkazu *git log* som sa rozhodol zostaviť si vlastný príkaz, nakoľko mi žiadna preddefinovaná varianta úplne nevyhovovala. Hlavným dôvodom bola snaha čo najviac zjednodušiť následnú syntaktickú analýzu. Tento príkaz som sa rozhodol generovať

dynamicky, aby sa dal v prípade potreby jednoducho upraviť. Ukážka tohto postupu je vo výpise 4.9.

```
private const char mainSplitter = '&';
private string gitLog = "git log ";

public GitLogger()
{
    gitLog += string.Format("--pretty=format:\"%an%{0}%ad%{0}%s\"" +
        " --date=format:'%d.%m.%Y'", ((int)mainSplitter).ToHex());
}
```

Listing 4.9: Generovanie vlastného príkazu git log

Následne bolo nutné v cykle prejsť všetky výsledky od najnovšieho *commitu* až po *commit*, ktorý značil posledné zostavenie dostupné na platforme Steam pre testerov. Z týchto výsledkov sa odstránili všetky zlúčenia jednotlivých vetiev a ďalšie *commity*, ktoré neobsahovali ani opravy chýb ani novo pridané vylepšenia. Opravy boli značené tzv. „bugMarkerom“ (@b) a vylepšenia tzv. „featureMarkerom“ (@f). Tých mohlo byť v jednom *commite* hneď niekoľko, preto bol na každý z nich aplikovaný regulárny výraz a jednotlivé časti boli ďalej spracovávané samostatne. Tento regulárny výraz zobrazuje výpis 4.10. Každá časť navyše mohla byť rozdelená na viacero menších častí rovnakého druhu pomocou reťazca “. “ alebo znaku ‘;‘. Jednotlivé elementárne časti boli následne štruktúrovane uložené do súboru.

```
var matches = Regex.Matches(body, string.Format("({0}|{1})[^@]*", bugMarker,
    featureMarker));
```

Listing 4.10: Regulárny výraz na rozdelenie tela *commitu*

Pôvodný prípad použitia mal zahŕňať automatické otvorenie tohto súboru v predvolenom textovom editore a jeho ručné skopírovanie do programu Microsof Teams, ktorý sa používa na komunikáciu vo firme. Rozhodol som sa ale preskúmať možnosti automatizovaného odosielania správ v tejto službe a pokúsil sa implementovať lepšie riešenie.

Služba Microsoft Teams podporuje automatizované posielanie správ okrem iných možností aj pomocou tzv. „Webhooks“. Tie sa dajú nakonfigurovať priamo v užívateľskom rozhraní služby a to buď pre konkrétny kanál alebo konverzáciu. „Webhook“ vygeneruje jedinečnú URL adresu, na ktorú je možné posielať POST žiadosti, ktoré som už spomínal v sekcií 4.1. Dáta sú prenášané ako štruktúrovaný JSON reťazec. Okrem jednoduchých správ je možné posielať aj formátovaný text pomocou značkovacieho jazyka MarkDown, prípadne tzv. karty.

Pomocou nástroja Postman som vo formáte JSON vytvoril dva návrhy takejto karty a nechal kolegov rozhodnúť o tom, ktorý sa nakoniec použije. Zvolenú kartu som si uložil do súboru na disk a

následne podľa nej vytvoril triedu Message, ktorá svojou vnútornou štruktúrou kopíruje spomínaný JSON súbor. Následne som tento súbor prečítal a jeho obsah deserializoval na túto triedu. Do tela správy som vložil dáta získané z *commitov* po syntaktickej analýze a použil metódu vo výpise 4.8 na odoslanie správy. Tento postup demonštruje výpis 4.11.

```
Message message = Message.FromJson(File.ReadAllText(pathToJSON));
Content content = message.Attachments[0].Content;

content.Title = header;
content.Sections[0].ActivityText = PrintListToJsonString(features);
content.Sections[1].ActivityText = PrintListToJsonString(bugs);

// cmdBegin == "Invoke-RestMethod -Method post -ContentType 'Application/Json;
    charset=UTF-8' -Body '";
string command = cmdBegin + Serialize.ToJson(message) + "' -Uri " + webHook;
string result = InvokePowershellScript(command)[0];
```

Listing 4.11: Vytvorenie a odoslanie správy do služby Microsoft Teams

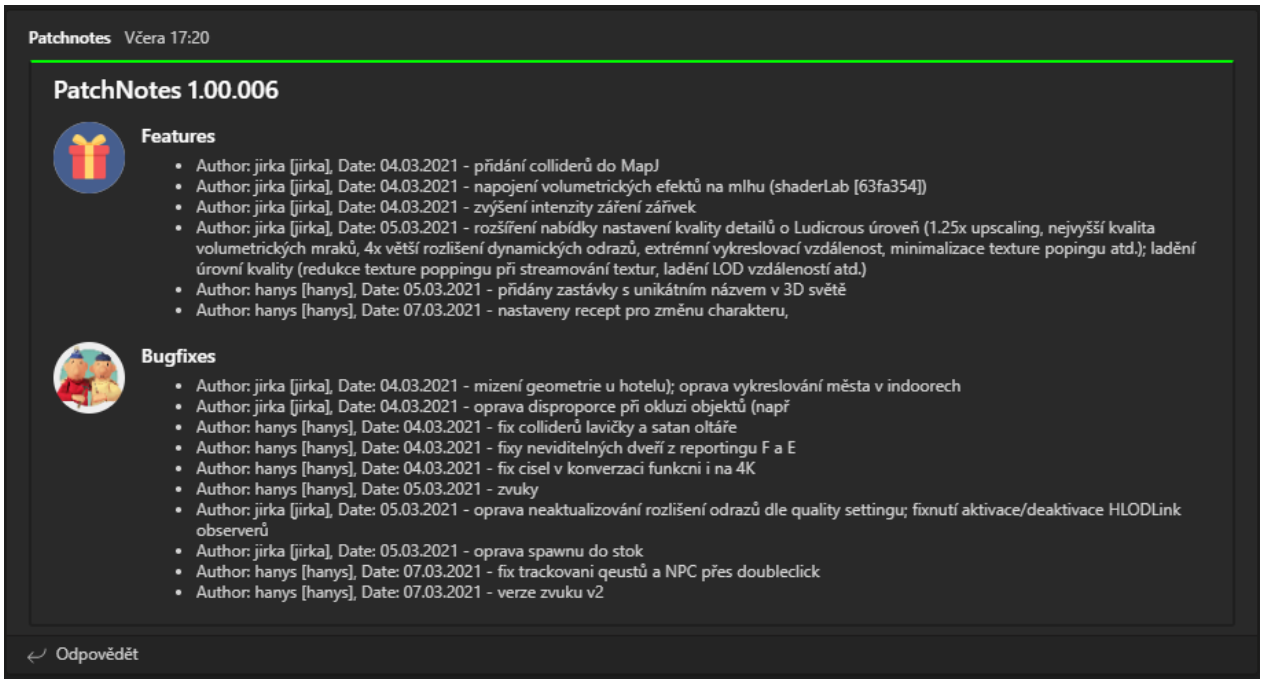
Metóda PrintListToJsonString do dát pridá značky jazyka MarkDown, čím sa ešte upraví finálny vzhľad a zároveň pomocou regulárneho výrazu "#d+" nahradí všetky výskyty číselných reťazcov začínajúcich znakom # za konkrétny odkaz reportu v službe Mantis Bug Tracker, na ktorý sa dá kliknúť. To je užitočné v prípadoch keď *commit* priamo opravuje nejakú nahlásenú chybu. Finálnu kartu môžeme vidieť na obrázku 4.4.

Poslednou pridanou funkcionalitou bolo, aby sa táto karta neposielala pri každom zostavení hry na serveri ale iba vtedy, keď je to skutočne žiadúce. Aplikácia teda prečíta obsah súboru BuildSettings.asset a ak v ňom nájde reťazec "sendPatchNotesToTeams: 1" vykoná odoslanie. V opačnom prípade nie. Tento súbor je podrobne popísaný v sekcií 4.1.3.

Aplikácia bola nasadená ako externý spúšťací súbor, ktorý je spustený službou Gitlab CI/CD bližšie popísanou v sekcií 3.4.

4.1.3 Automatizované zostavenie hry na serveri (Build Server)

Časová náročnosť:
Nasadenie: 6 dní.
Úvod do problému:
Ako som spomenul v sekcií
Navrhované riešenia:
Nakoľko bola



Obr. 4.4: Karta v programe Microsoft Teams

Realizácia:
Pri preskúmavaní mož

4.2 Optimalizácia a prevod hry na ďalšie platformy

4.3 Ostatné projekty

asd

Literatúra

1. VISIONGAME. *Perun Creative* [online] [cit. 2021-03-06]. Dostupné z : `https://visiongame.cz/studio/perun-creative/`.
2. PERUN CREATIVE. *Hobo: Tough Life* [online] [cit. 2021-03-06]. Dostupné z : `http://hoborpg.com/`.
3. UNITY TECHNOLOGIES. *Unity Platform* [online] [cit. 2021-03-06]. Dostupné z : `https://unity.com/products/unity-platform`.
4. UNITY TECHNOLOGIES. *Build once, deploy anywhere* [online] [cit. 2021-03-06]. Dostupné z : `https://unity.com/features/multiplatform`.
5. MICROSOFT. *A tour of the C# language* [online] [cit. 2021-03-07]. Dostupné z : `https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/`.
6. CHACON, Scott; STRAUB, Ben. *Pro Git*. 2nd ed. New York: Apress, 2014. ISBN 1484200772.
7. GITLAB. *CI/CD concepts* [online] [cit. 2021-03-10]. Dostupné z : `https://docs.gitlab.com/ee/ci/introduction/index.html`.
8. EPIC GAMES. *Epic Online Services* [online] [cit. 2021-03-11]. Dostupné z : `https://dev.epicgames.com/en-US/services`.
9. MOSS, Richard. *MantisSharp* [online]. 2017 [cit. 2021-03-12]. Dostupné z : `https://github.com/cyotek/MantisSharp`.
10. FELDT, Henrik. *DotNetZip* [online] [cit. 2021-03-13]. Dostupné z : `https://github.com/haf/DotNetZip.Semverd/`.