UNIVERZITA PARDUBICE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

BAKALÁŘSKÁ PRÁCE

2017 Jan Stránský

Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Implementace strategické hry

Jan Stránský

Bakalářská práce

2017

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2016/2017**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Jan Stránský**
Osobní číslo:	**I13226**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Implementace strategické hry**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

Práce je ohraničena tématy herního průmyslu, tvorby aplikací pro stolní počítače a metodami distribuce hotových řešených prostřednictvím oficiálních i alternativních tržních kanálů.

V práci je třeba vymezit prostředí jednotlivých operačních systémů z pohledu podpory Unity3D aplikací, a popsat distribuční kanály pro jednotlivá prostředí, přičemž důraz je kladen na časové, legislativní a finanční aspekty. Dále bude v práci zapracován rozbor postupů při vývoji počítačových her.

Realizovanou případovou studií bude aplikace, která bude odpovídat současným požadavkům na moderní hru. V rámci implementace bude využito 2D nebo 3D scény, práce s modely či texturami a lineárním příběhem. Případová studie musí být vymezena prostřednictvím korektních dokumentů (např. storyline, game design document, konceptuální schémata).

Student vypracuje strategickou hru zaměřenou na správu, hospodaření a rozšiřování virtuální osady. Student bude pracovat s kompletním 3D světem (implicitně bude používat raytraycing), přípustná práce s kamerou odpovídá mnimálně pohledu shora. Do hry bude implementován herní koncept evolučních schémat, student tento koncept může demonstrovat např. výzkumnými větvemi. Součástí práce bude práce s assety, přičemž kromě využívání open source zdrojů musí student prokázat schopnost vytvářet assety zcela původní.

Rozsah grafických prací:

Rozsah pracovní zprávy:

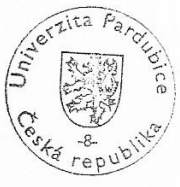
Forma zpracování bakalářské práce: **tištěná**

Seznam odborné literatury:

DILLE, Flint. The ultimate guide to video game writing and design. New York: Watson-Guptill Publications, 2007. ISBN 158065066X.
PECINOVSKÝ, Rudolf. OOP: Naučte se myslet a programovat objektově. Brno: Computer Press, a.s., 2010. ISBN 978-80-251-2126-9.
KNUTH, D. E.: Umění programování - Základní algoritmy, Brno, Computer Press 2008, ISBN: 978-80-251-2025-5.
WRÓBLEWSKI, Piotr. Algoritmy: datové struktury a programovací techniky. Vyd. 1. Překlad Marek Michalek, Bogdan Kiszka. Brno: Computer Press, 2004, 351 s. ISBN 80-251-0343-9.
KEOGH, Jim; DAVIDSON, Ken. Datové struktury bez předchozích znalostí : průvodce pro samouky. Vyd 1. Brno : Computer Press, 2006. 223 s. ISBN 80-251-0689-6.

Vedoucí bakalářské práce: **Ing. Josef Brožek**
Katedra informačních technologií

Datum zadání bakalářské práce: **31. října 2016**
Termín odevzdání bakalářské práce: **12. května 2017**



L.S.

Ing. Zdeněk Němec, Ph.D.
děkan

Ing. Zdeněk Šilar, Ph.D.
pověřený vedením katedry

V Pardubicích dne 31. března 2017

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

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

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 10. 5. 2017

Jan Stránský

PODĚKOVÁNÍ

Děkuji Ing. Josefu Brožkovi za pomoc při zpracování této bakalářské práce, hlavně četnými konzultacemi, pomocí kterých bylo možné vytvořit tuto práci do této výsledné podoby. Děkuji také své rodině, která mě při studiu a zpracování bakalářské práce podporovala a pomáhala.

ANOTACE

Tato bakalářská práce se zabývá tvorbou strategické počítačové hry v herním enginu Unity. V první části jsou popsány základní pojmy a nástroje použité při tvorbě hry. Následně jsou popsány obecné i praktické postupy, podle kterých byla hra vytvářena, a také popis vytvářené hry. Závěrečná část práce se zabývá praktickými problémy vzniklými při implementaci hry a popis funkcí, možností a podoby hry jako výsledného produktu.

KLÍČOVÁ SLOVA

engine, Unity, asset, program, programovací jazyk, skript, hra

TITLE

Strategic game implementation

ANNOTATION

This bachelor's thesis deals with the creation of computer strategy game in Unity game engine. First part describes the basic terms and tools used for its creation. Next part desribes the general and practical procedures used for the game creation and a description of the game itself are introduced. Final part of the work deals with the practical problems arising from the implementation of the game and the description of the features, options and appearance of the game as the final product.

KEYWORDS

engine, Unity, asset, program, programming language, script, game

OBSAH

SEZNAM ILUSTRACÍ

SEZNAM ZDROJOVÝCH KÓDŮ

ÚVOD

Cílem této práce je vytvoření počítačové hry za použití herního enginu Unity. Vytváření počítačové hry není snadné, protože je nutná znalost programování a je třeba mít zkušenosti s tvorbou a úpravou 3D modelů, textur, obrázků, zvuků a animací. Většina těchto souborů (assetů) se vytváří v nástrojích určených právě pro jejich tvorbu a editaci, a je tak nezbytné se v nich naučit pracovat.

Důvod zvolení této práce je, že tvorba počítačových her je zajímavá a je možné získat spoustu znalostí a zkušeností z různých oblastí informačních technologií, jako například z oblasti programování a modelování. Tyto znalosti a zkušenosti se později mohou hodit při uplatnění v zaměstnání nebo při tvorbě vlastních projektů.

Práce je strukturována tak, že nejdříve jsou vysvětleny základní pojmy z oblasti informačních technologií, jejichž znalost je nutná pro pochopení problematiky této práce. Následně jsou postupně představeny nástroje použité při tvorbě. Ke každému nástroji jsou napsány jeho základní vlastnosti, k čemu byl použit a proč byl vybrán. Po představení nástrojů následuje obecný popis paradigmat použitých při tvorbě. V paradigmatech jsou popsány postupy, podle kterých se vyvíjí a vytváří hra a jednotlivé části hry.

Další kapitola se věnuje problematice tvorby projektu z pohledu návrhu, podle kterého by se poté měla hra vytvářet. Je zde popsána základní myšlenka, principy a pravidla. Současně jsou zde popsány způsoby, jakými se hra ovládá, a také základní herní mechanismy a děj hry.

Následující kapitola popisuje praktické a stěžejní problémy vzniklé při tvorbě hry, jež jsou důležité, a je potřeba se jim vyvarovat při tvorbě dalších projektů.

V závěrečné kapitole je vytvářená hra popsána z pohledu výsledného produktu. Jsou zde popsány způsoby, jakými se hra hraje, jak se ovládá a co má uživatel k dispozici. Je zde také popsáno uživatelské rozhraní včetně jeho rozložení, co jednotlivé části zobrazují anebo co v nich může uživatel dělat.

Typografické konvence

Text psaný kurzívou označuje názvy z vlastní práce nebo proměnné a funkce knihoven Unity.
Text psaný kurzívou ohraničený uvozovkami označuje přímé citáty z cizích zdrojů.
Zdrojové kódy jsou psané fontem Consolas o velikosti 9,5 v barvách přidělených nástrojem Visual Studio.

1 ZÁKLADNÍ POJMY

V následující části jsou vysvětleny základní pojmy, z oblasti programování, používané při tvorbě této závěrečné práce. Základní pojmy jsou zde vysvětleny, aby bylo možné porozumět problematice této práce.

1.1 Program

Program, konkrétně počítačový program, je posloupnost instrukcí, které počítač vykonává. Počítačový program je vytvářen programátorem v libovolném programovacím jazyce, procesu vytváření programu se říká programování. Cílem programování je tedy nalezení takové posloupnosti příkazů, které počítači říkají jak má výsledný program vypadat a fungovat. K vytváření programů se používá vývojové prostředí, které usnadňuje práci programátorům lepší přehledností, nápovědou, snazším hledáním chyb ve zdrojovém kódu a mnoha dalšími funkcemi. Součástí vývojových prostředí je také grafické uživatelské rozhraní tzv. GUI, které umožňuje přímou manipulaci s grafickými prvky programu. Zdrojový kód, je soubor obsahující příkazy daného programovacího jazyka. Ze zdrojového kódu lze vytvořit program pomocí překladače nebo interpretu.

Zdroj [1].

1.2 Programovací jazyk

„Pod pojmem programovací jazyk rozumíme prostředek pro zápis programů, jež mohou být provedeny na počítači. V tomto smyslu je programovací jazyk komunikačním nástrojem mezi uživatelem počítače, který jeho jazykovými prostředky specifikuje algoritmus řešení daného problému, a počítačem, jenž svými technickými prostředky algoritmus interpretuje a realizuje tak transformaci vstupních údajů na výstupní.“ Zdroj [2].

Je mnoho různých programovacích jazyků, které se liší svými vlastnostmi. Programovací jazyky se nejčastěji dělí na vyšší a nižší, interpretované a kompilované (neinterpretované), lze je ale také dělit na procedurální a neprocedurální.

Nižší (nízkoúrovňové) programovací jazyky využívají příkazy procesoru. Kvůli tomu je programování v takovém jazyce obtížné, zdlouhavé a program v něm vytvořený není přenositelný na jiný druh procesoru. Mezi výhody patří lepší optimalizace a vyšší rychlost programu. Mezi nižší programovací jazyky patří Assembler.

Vyšší (vysokoúrovňové) programovací jazyky se vyznačují především lepší čitelností a logickým uspořádáním zdrojových kódů. Jsou tedy pro člověka lépe pochopitelné a

programování je tak rychlejší. Mezi tyto jazyky patří většina dnes používaných programovacích jazyků jako například C++, C#, Java, JavaScript a PHP.

„Interpretované jazyky potřebují ke svému spuštění tzv. interpret, kterým je kód překládaný za běhu samotného programu. Velkou výhodou je to, že v takovém programu není nutné deklarovat proměnné, velikost paměti, kterou mají proměnné zabírat, apod. Z čehož plyne jistá pohodlnost. Jako daň si takový program (přesněji skript) vybere na rychlosti programu, která je oproti kompilovaným jazykům o poznání nižší. Aby mohl být takový program na daném systému spuštěn, potřebuje, aby na takovém systému byl přítomen interpret příslušného programovacího (skriptovacího) jazyka. Mezi takovéto jazyky patří například - PHP, Perl, Python, ASP, JavaScript a Java.

Kompilované jazyky jsou naopak rychlejší, po zkompilování (přeložení) mohou být spouštěny samostatně. Je u nich ovšem důležitá správná implementace programového kódu, jinak bude docházet k chybám (např. přetečení zásobníku). Do této skupiny řadíme např. programovací jazyk Pascal (jeho modifikace jako např. Delphi), C (jeho modifikace jako C++) a mnoho dalších.“ Zdroj [3].

1.3 Skript

Skript je soubor se zdrojovým kódem, který představuje určitou sérii příkazů nebo algoritmus psaný ve skriptovacím programovacím jazyce. Je několik podob skriptů, můžou být v podobě dávkových souborů nebo psané vyšším programovacím jazykem.

Dávkový soubor nejčastěji obsahuje příkazy operačních systémů psané skriptovacím jazykem. Jedním z takových skriptovacích jazyků je například Batch, který je vyvinutý pro operační systémy od společnosti Microsoft, například pro Windows nebo MS-DOS. Batch pro běh používá příkazový řádek. V něm vytvořený skript obsahuje tedy sérii příkazů, které následně pracují s daným operačním systémem.

Skripty psané ve vyšším programovacím jazyce představují nejčastěji určitou část programu, například v podobě algoritmu nebo objektu. Takové skripty mohou být psané například v jazycích PHP, JavaScript nebo C#. Právě skripty psané ve vyšších programovacích jazycích se používají ve složitějších programech, jako například hrách, kdy pro jejich vytvoření mohou být potřeba až stovky skriptů. Díky rozčlenění na menší části je dosaženo lepší přehlednosti, znovupoužitelnosti a snazšího hledání případných chyb. V souvislostech s hrami mohou skripty pracovat se vstupy od uživatele (hráče), fyzikálním enginem (detekce kolizí), grafickým

prostředím, umělou inteligencí, uživatelským rozhraním, manipulaci s herními objekty a mnoha dalšími prvky hry.

Důležitým pojmem souvisejícím se skriptem je slovo this, které odkazuje na současnou instanci třídy (class) ve skriptu. Slovo this se používá například u parametrického konstruktoru. To je metoda, která se volá při vytvoření instance nějakého objektu a přijímá nějaké vstupní parametry, které mohou mít stejné označení jako atributy daného objektu. Pomocí slova this jednoznačně určíme, které označení je atribut objektu a které je parametr konstruktoru. This také lze použít pro předání současné instance jiné metodě nebo skriptu, které s ní budou dále pracovat.

Zdroj [4].

1.4 Asset

Assety jsou soubory různého typu, které jsou použity při tvorbě her. Nejčastěji se s nimi pracuje prostřednictvím editoru (herního enginu), který práci s nimi umožňuje a usnadňuje. Assety můžou vývojáři sami vytvářet nebo je lze získat již hotové, které mohou být placené i neplacené a pod různými licencemi. Některé assety jsou již součástí přímo herního enginu.

Nejčastějšími typy assetů jsou:

- 3D modely
- Textury
- Obrázky (Image)
- Prvky uživatelského rozhraní (Sprites)
- Zvukové efekty a hudba
- Animace
- Skripty
- Motion capture

Assety jsou většinou vytvářeny v aplikacích zaměřených na daný typ, například pro tvorbu 3D modelů lze použít Blender, pro tvorbu textur Gimp a pro psaní skriptů Visual Studio. Takto vytvořené assety jsou poté importovány do herního enginu, ve kterém se s nimi dále pracuje.

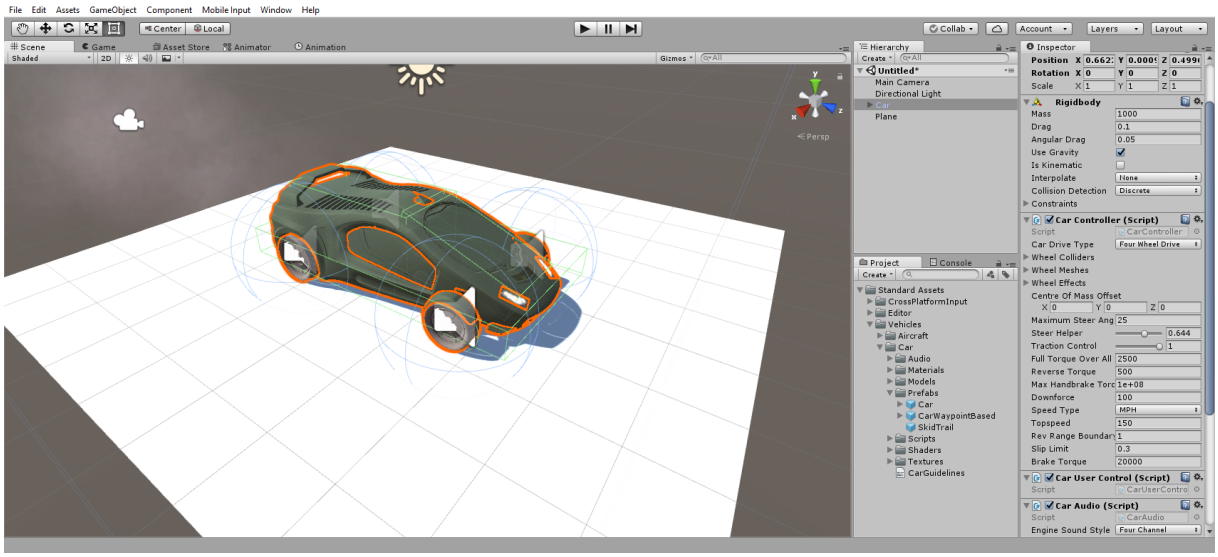
Zdroj [5].

2 POUŽITÉ NÁSTROJE A PARADIGMATA

V této kapitole budou představeny nástroje použité při tvorbě aplikace pro závěrečnou práci. Současně jsou zde vysvětleny paradigmata související s tvorbou této závěrečné práce.

2.1 Unity 3D

Unity je multiplatformní herní engine, tedy software sloužící k vytváření her. Umožňuje tvorbu her na mobilní zařízení, osobní počítače, herní konzole, pro virtuální realitu, chytré televize a také web. Unity také umožňuje převést hru vytvořenou pro určitou platformu na více jiných platforem. Unity je neustále udržováno a vylepšováno vydáváním aktualizací a nových verzí. Díky tomu narůstá počet podporovaných systémů a platforem, zlepšuje se optimalizace a přibývají nové funkce. Součástí herního enginu je i uživatelské rozhraní, které usnadňuje a urychluje práci, umožňuje práci s objekty přímo v okně editoru a zobrazuje výslednou podobu hry. Ukázka uživatelského rozhraní je vidět na obrázku 1.



Obrázek 1: Unity editor

Unity také obsahuje mnoho funkcí a knihoven umožňující a ulehčující tvorbu her. Mezi hlavní funkce enginu patří fyzikální engine, který se stará o dynamiku těles a detekci kolizí, renderování 2D a 3D grafiky. Dále také EventSystem, práce se zvukem, vytváření a osazování skriptů, animace, umělá inteligence a práce v síti. Pro tvorbu skriptů lze použít programovací jazyky C#, JavaScript nebo Boo.

Zdroj [6].

Tvorba v enginu Unity je snazší oproti konkurenčním enginům, a proto je vhodnější pro začátečníky. Toto je hlavní důvod zvolení Unity pro tvorbu této práce. Dalšími důvody jsou: bezplatná verze, mnoho podporovaných typů assetů a možnost psát skripty v jazyce C#.

Konkurenčními herní enginy jsou například Unreal Engine a CryEngine, oba tyto enginy mají také bezplatnou verzi. Oproti Unity jsou oba zmiňované výkonnější, což znamená, že je možné vytvářet složitější a propracovanější projekty s nižšími nároky na systém. Tvorba v nich je ale složitější a z toho důvodu nejsou moc vhodné pro začátečníky.

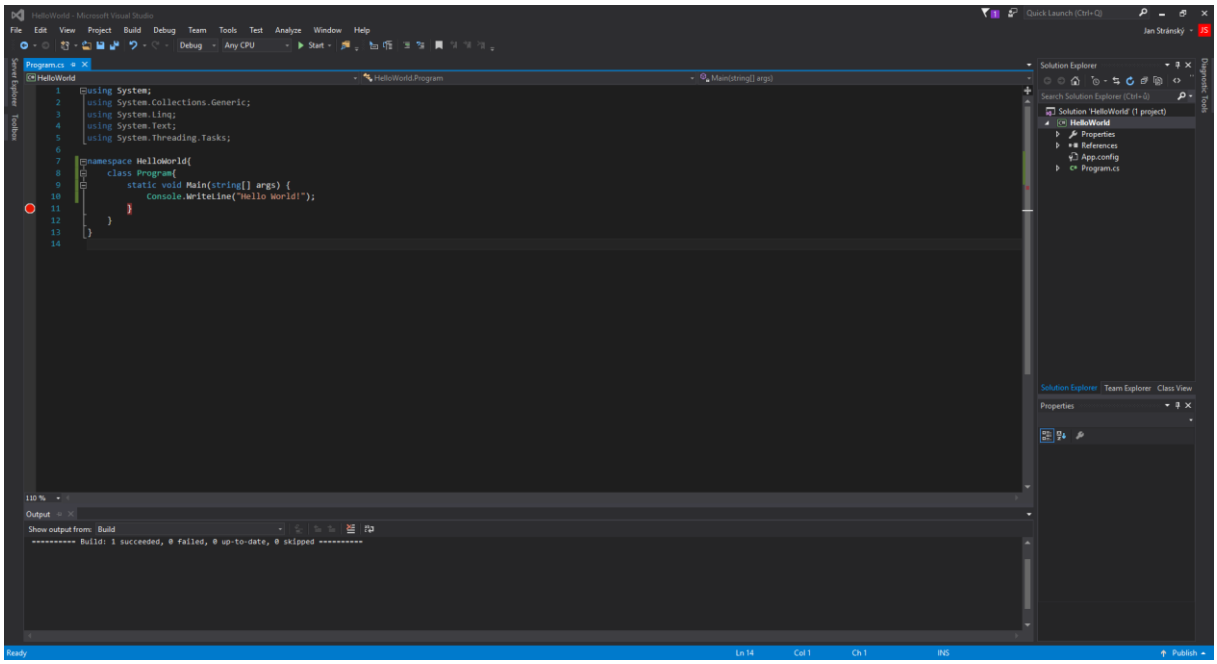
2.2 Visual Studio

Microsoft Visual Studio je sada nástrojů pro tvorbu softwaru, od fáze plánování až po tvorbu uživatelského rozhraní, psaní zdrojového kódu, testování, ladění, analýzu kvality a výkonu kódu, nasazení pro zákazníky, a shromažďování dat při použití. Tyto nástroje jsou navrženy tak, aby spolu spolupracovaly v co největší míře, a jsou dostupné přes integrované vývojové prostředí (IDE) zobrazené na obrázku 2.

Pomocí sady Visual Studio lze vyvíjet aplikace pro Android, iOS, Windows, web i cloud. Ve výchozím sestavení Visual Studia lze vytvářet aplikace v C#, C++, C, JavaScript, F# a Visual Basic. Visual Studio je možné rozšířit o vlastní nástroje, které si vývojář sám vytvořil, nebo o nástroje třetích stran, jako například nástroj pro propojení Visual Studia a Unity zvaný *Visual Studio Tools for Unity*. Při psaní kódu nabízí Visual Studio nápovědy bez ohledu na používaný programovací jazyk. Užitečná funkce je našeptávání, která automaticky nabízí doplňování a zvyšuje tak rychlost a přesnost psaní kódu.

Zdroj [7].

Nástroj Visual Studio byl použit při tvorbě této práce kvůli jeho propracovanosti, nástrojích a funkcích, které při vývoji usnadní práci. Je také dostupná bezplatná verze a je možné tento nástroj propojit s herním enginem Unity. Další nástroj, který lze použít pro práci se skripty, je MonoDevelop.



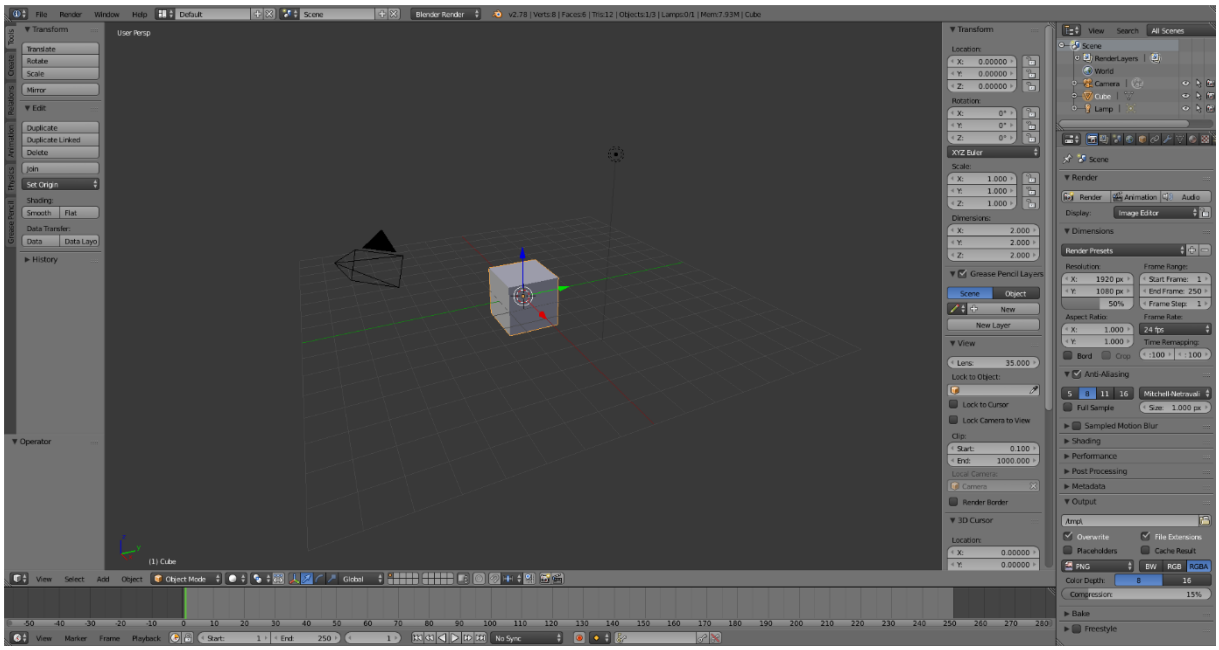
Obrázek 2: Vývojové prostředí Visual Studio

2.3 Blender

Blender je program s rozsáhlým obsahem nástrojů pro tvorbu 3D modelů, renderování, animaci, postprodukci, sledování pohybu, video editaci a vytváření her. Pokročilí uživatelé mohou použít rozhraní pro tvorbu skriptů v jazyce Python. Blender je multiplatformní a je možné ho provozovat na systémech Linux, Windows a Macintosh. Jeho uživatelské rozhraní a ovládání je složitější a začátečníci se tak musejí naučit jeho ovládání. Na obrázku 3 je zobrazeno uživatelské rozhraní Blenderu.

Zdroj [8].

V této práci byl použit pro vytváření 3D modelů a jejich následné texturování, samotné textury ale byly vytvářeny v jiném nástroji. Hlavním důvodem zvolení tohoto programu je dostupnost zdarma. Největšími konkurenty jsou Autodesk Maya a 3ds Max, oba tyto programu jsou ale placené s výjimkou zkušební 30 denní verze. Nabízejí však více funkcí a nástrojů.



Obrázek 3: Uživatelské rozhraní Blenderu

2.4 Gimp

„GIMP *je multiplatformní nástroj pro úpravu fotografií a rastrové grafiky. Název* GIMP *je zkratka z anglického* GNU *Image Manipulation Program (* GNU *program pro úpravu obrázků).*

GIMP *lze použít jako jednoduchý program pro malování, stejně jako profesionální aplikaci pro retušování fotografií, program pro dávkové zpracování obrázků, konvertor obrazových souborových formátů atd. Jeho schopnosti a možnosti jsou opravdu široké.*

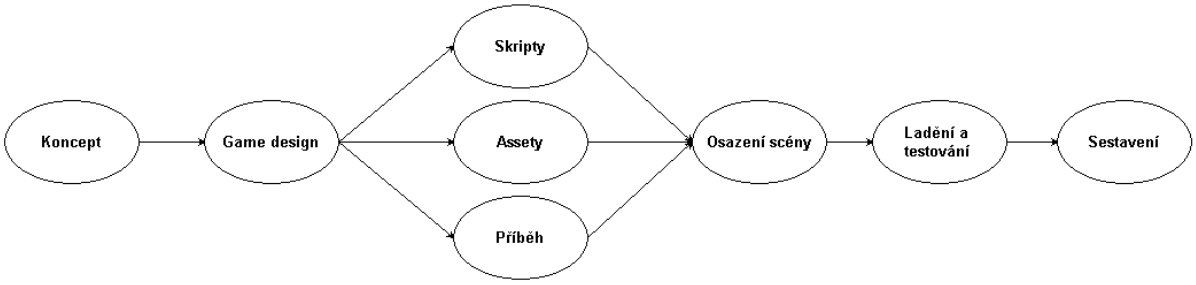
GIMP *je snadno rozšiřitelný. Je navržen tak, aby mohl být pomocí zásuvných modulů a skriptů snadno doplňován novými funkcemi. Pokročilé skriptovací rozhraní umožňuje automatizovat vše od nejjednodušších až po ty nejsložitější úlohy.*

Jednou z předností Gimpu je jeho volná dostupnost pro nejrůznější platformy a operační systémy. Většina distribucí GNU/Linuxu *obsahuje GIMP jako standardní součást systému.* GIMP *je k dispozici i pro další systémy, jako např.* Microsoft Windows™ *nebo Apple* Mac OS X™ *(*Darwin*).* GIMP *ovšem není freeware, jak někteří mylně tvrdí. Je to svobodná aplikace s otevřeným zdrojovým kódem chráněná Obecnou veřejnou licencí (*GPL*). Licence* GPL *zajišťuje uživatelům svobodný přístup ke zdrojovým kódům aplikace a svobodu tyto kódy měnit. To je mnohem více, než poskytují programy označované jako freeware.“* Zdroj [9].

Gimp byl použit, protože je to jednoduchý, přehledný nástroj pro práci s rastrovou grafikou. Díky svým vlastnostem je dobrý pro začátečníky, je v něm ale možné pracovat i na profesionální úrovni. Největším konkurentem je Adobe Photoshop, který je propracovanější a má více funkcí. Je ovšem placený. Gimp je Open Source. To znamená, že je dostupný zdarma včetně zdrojových kódů.

2.5 Postup při vývoji hry

Vývoj her probíhá podle kroků a postupů, které pomáhají zajistit co nejlepší způsob vytvoření zvolené hry. V počáteční fázi by se měl vytvořit Game design, popisující podobu plánované hry. Pokud je Game design hry hotový je možné vytvořit postup pro samotnou tvorbu hry. Pokud je hotový návrh hry, děje, herních prvků a mechanismů, tak je možné začít s tvorbou assetů. Vytvořené assety se poté v herním enginu vkládají do scén jako herní objekty, ke kterým se v případě potřeby vytváří skripty rozšiřující jejich vlastnosti. V průběhu tvorby a kompletace je hra testována, aby obsahovala co nejmenší množství chyb. V konečné fázi je hra sestavena, aby byla samostatně spustitelná. Na obrázku 4 je zobrazeno schéma postupu při vývoji hry.



Obrázek 4: Schéma postupu vývoje hry

2.5.1 Game design

Game design je popis plánované hry, který má pomoct při jejím vývoji. Popisuje se v něm základní myšlenka, principy a pravidla hry, ale také popis plynutí času ve hře, nástroje a funkce, které má hráč k dispozici, nebo dějová linie a další prvky a herní mechanismy. Součástí Game designu by také měl být popis herního žánru, světa, období a prostředí, ve kterém se hra odehrává.

Herní žánr rozděluje hry do skupin podle toho, jaké prostředky hra nabízí pro interakci s hráčem, způsobem ovládání, umístěním kamery, příběhem a také zda je to hra pro jednoho nebo více hráčů, anebo jestli je online. Umístění kamery znamená, jakým pohledem je hra zobrazována, například pohled ze shora, pohled třetí osoby nebo první osoby. Online hra se

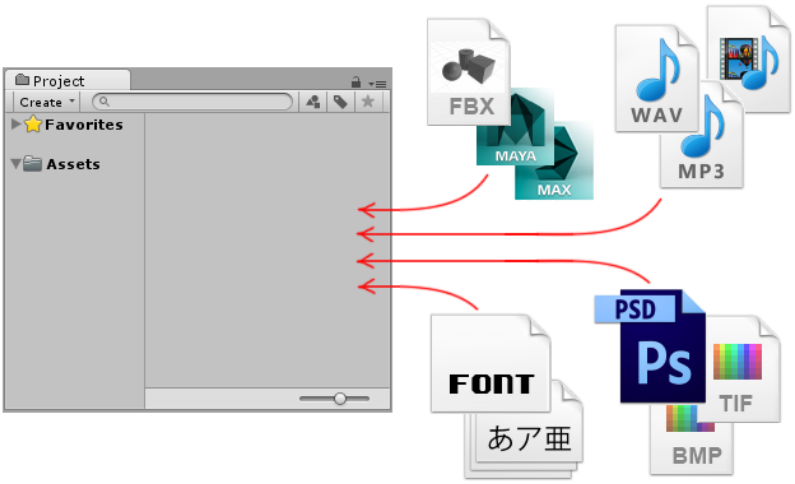
odehrává na vzdáleném serveru, ke kterému se hráč přípoji pomocí klienta, který zobrazuje veškeré dění na serveru. Pokud hra není online, tak se odehrává lokálně na systému hráče.

Svět ve hře může být smyšlený nebo vytvořený podle skutečného světa. Například hra o druhé světové válce bude zasazena do skutečného světa nebo světa podobného skutečnému. Obdobím se popisuje zvolená doba z lidské historie nebo budoucnosti, do kterého je děj hry zasazen. Zvolené období udává, jaké technologické možnosti bude herní svět nabízet. Pokud například bude hra vycházet z období středověku, budou v herním světě přítomny pouze takové technologie, které v té době mohly existovat. Popis prostředí je popis místa v herním světě, ve kterém se odehrává děj, například les, poušť, hory, město, moře, nebo vesmír.

Součástí Game designu je i popis děje hry, který popisuje příběh a cíle hry. Cíle hry určují, čeho musí hráč dosáhnout, aby hru úspěšně dokončil. Příběh představuje na sebe navazující úkoly, pomocí kterých se příběhem postupuje. Cílem příběhu je dokončení všech zadaných úkolů. Součástí popisu děje jsou i způsoby, kterými lze dosáhnout cíle hry.

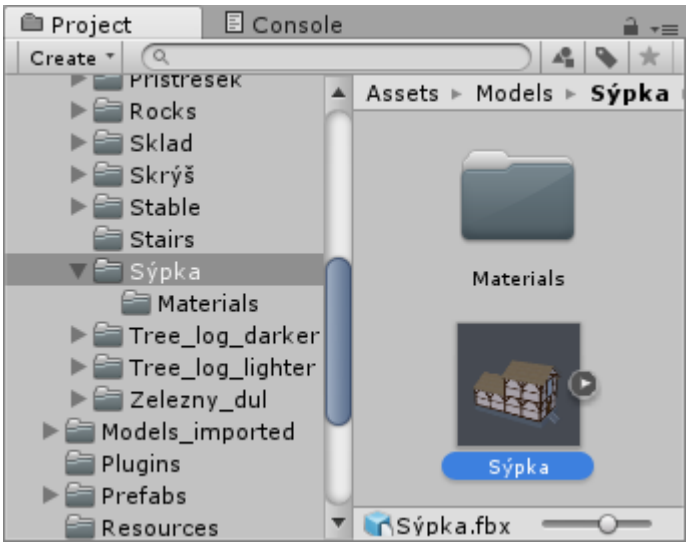
2.5.2 Tvorba assetů

Většina assetů je vytvářena v programech k tomu určených, ale některé lze vytvářet přímo v Unity. Vytvoření vlastních assetů si žádá čas a zkušenosti a čím víc zkušeností vývojář má, tím rychleji mu tvorba jde. Ovšem proč vytvářet něco, co už vytvořil někdo jiný. Na internetu je mnoho webů obsahujících velké množství již hotových assetů, které je možné legálně použít za předpokladu, že budou dodrženy licenční podmínky. Získat assety lze například z Asset Storu Unity, kde je jich k dispozici ohromné množství už připravených a odzkoušených. Jsou zde k dispozici assety buď placené, nebo zdarma. Rozdíl je většinou v rozsahu nebo kvalitě. Assety je doporučeno ukládat do složky s názvem Assets ve složce s umístěním projektu, jak je znázorněno na obrázku 5. Pokud dojde ke změně assetu, Unity automaticky zaznamená jeho změnu a aktualizuje ho.



Obrázek 5: Ukázka několika podporovaných typů assetů[1]

3D modely jsou vytvořeny v nástroji Blender, který nejen umožňuje jejich tvorbu, ale i následné otexturování, animování nebo motion tracking (snímání pohybu). Textury pro modely jsou vytvořeny v nástroji Gimp. Hotové textury se poté promítnou na model a tak vznikne výsledný 3D model, který je poté potřeba uložit ve formátu podporovaném herním enginem, například v hodně používaném FBX. Součástí souboru FBX nejsou textury, pouze materiál odkazující na texturu a rozložení textury na modelu. Samotné textury je proto nutné uložit v samostatném souboru, nejčastěji ve formátu PNG nebo JPEG. Na obrázku 6 je zobrazena ukázka hotového modelu již v prostředí Unity.



Obrázek 6: Ukázka modelu sýpky v panelu Project

[1] Obrázek převzat z [5].

Obrázky a grafické prvky uživatelského rozhraní jsou vytvářeny v editorech rastrové grafiky, například v Gimpu. Prvky uživatelského rozhraní spadají v Unity do skupiny označované jako Sprites a patří mezi ně například pozadí oken, okraje oken, vzhled tlačítek a mnoho dalších.

Skripty se vytváří ve vývojových prostředích podporující programovací jazyk, ve kterém je skript vytvářen. Pro tvorbu v jazyce C# lze použít Visual Studio, které nabízí spoustu nástrojů a funkcí usnadňující tvorbu.

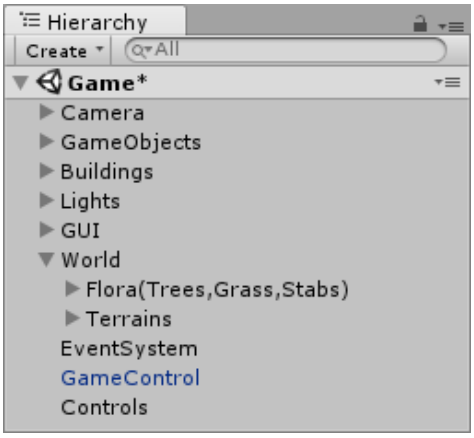
Když jsou assety hotové, je možné z nich vytvořit jeden funkční celek v prostředí Unity do takzvaného prefabu. Prefab je kompletní herní objekt, který je možné znovupoužít. Příkladem prefabu může být například auto, tedy herní objekt tvořený 3D modelem potaženým texturami, se skriptem zajištujícím jeho funkčnost, animacemi a zvukovými efekty.

Zdroje [5], [10], [11].

2.5.3 Tvorba scén a osazení assetů

Scéna je základním prvkem při tvorbě hry v Unity, bez které by hru nebylo možné vytvořit. Vytváří se přímo v prostředí Unity. Scéna představuje plátno, na kterém se odehrává veškeré dění hry. Scén v jedné hře může být několik. Pokud má například hra několik úrovní, pak každá úroveň má svou vlastní scénu.

Ve scéně jsou osazeny objekty, které dohromady tvoří kompletní herní prostředí, příkladem takových objektů jsou stromy, budovy, lidé, zvířata, ale také uživatelské rozhraní, zvuky a další objekty, které jsou vytvořené z připravených assetů a poskládaných do finální podoby a funkčnosti. Veškeré objekty osazené ve scéně jsou zobrazeny v kartě Hierarchy, ve které se s nimi pracuje. Na obrázku 7 je zobrazena kartu Hierarchy se scénu *Game*. Objekty umístěné ve scéně tvoří stromovou strukturu, jejímž kořenem je scéna.



Obrázek 7: Karta Hierarchy s aktivní scénou a všechny objekty v ní osazené

Osazení objektů do scény je možné buď přímo přes prostředí Unity, nebo za běhu programu ze skriptu. Přes prostředí se objekt může buď přetáhnou do scény, nebo do karty Hierarchy. Pro vytvoření objektu pomocí skriptu se použije funkce Instantiate. Vstupním parametrem funkce je objekt, ze kterého je vytvořen klon a ten je poté umístěn do scény. Nově vytvořený objekt je ve struktuře scény umístěn pod objektem, ve kterém se skript nachází. Tento objekt je poté takzvaným rodičem (parrent) nově vytvořeného objektu. Nově vytvořenému objektu je možné změnit jeho vlastnosti nebo umístění ve struktuře scény změnou rodiče. Ukázka vytvoření objektu s názvem *Building_info_window* ze skriptu je zobrazena ve zdrojovém kódu 1.

```
public void OpenInfoWindow() {
    openedWindow = Instantiate(Resources.Load<GameObject>("Building_info_window"))
    as GameObject;
}
```

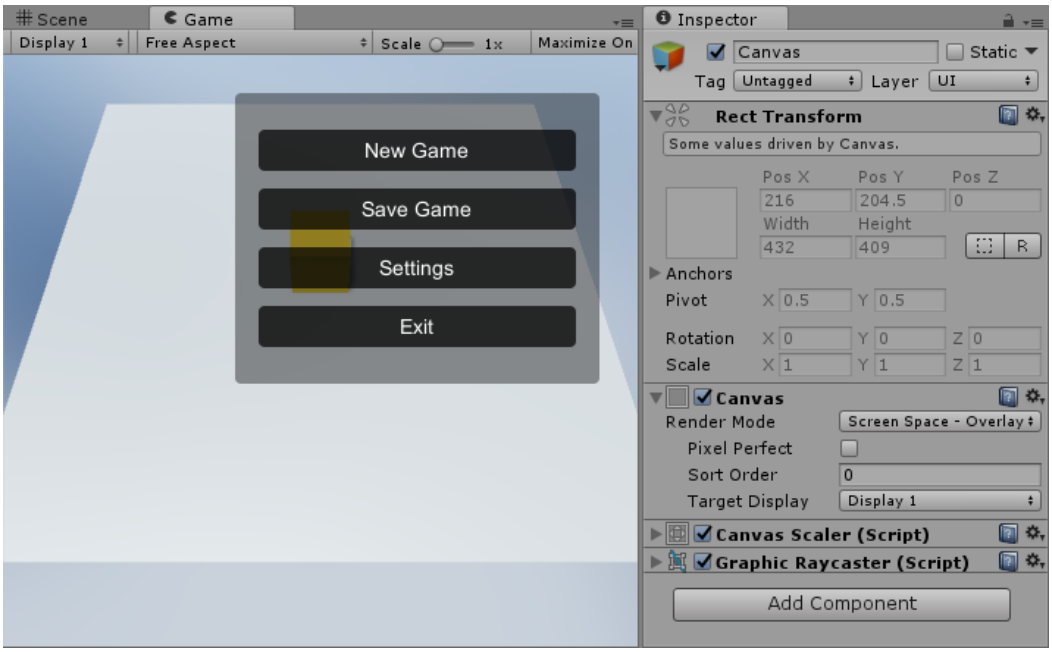
Zdrojový kód 1: Ukázka vytvoření objektu ze skriptu

Zdroje [12], [13].

2.5.4 Tvorba uživatelského rozhraní

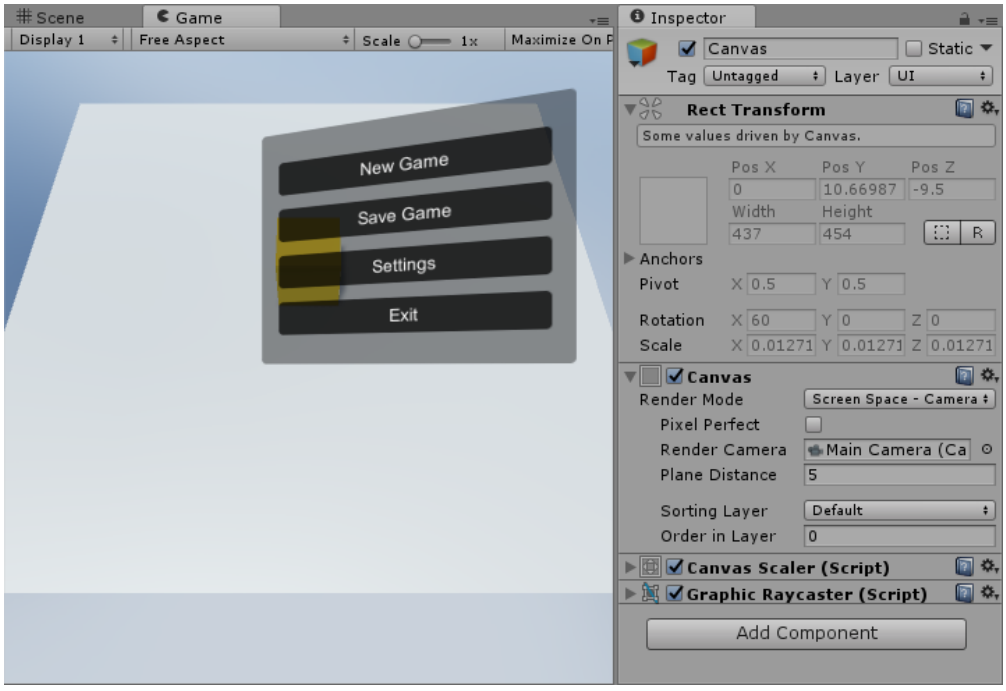
Pro tvorbu uživatelského rozhraní se v Unity používá objekt Canvas, který používá stejnojmennou komponentu. Canvas představuje plochu, na které jsou umístěny prvky uživatelského rozhraní (UI), jako například tlačítka, panely, obrázky, okna a další. Všechny prvky UI umístěné v Canvasu musí být jeho potomky v hierarchii, aby je bylo možné zobrazit. Canvas se zobrazuje podle zvoleného módu vykreslení v nastavení zvaném Render Mode. Módy vykreslení jsou Screen Space – Overlay, Screen Space – Camera a World Space.

Mód Screen Space – Overlay vykreslí uživatelské rozhraní nad všechny ostatní objekty scény přímo před aktuálně používanou kameru. Při změně kamery je stále zobrazeno stejné uživatelské prostředí. Velikost Canvasu v tomto režimu je daná velikostí obrazovky a nelze tak jeho velikost měnit. Při změně obrazovky nebo rozlišení se Canvas automaticky přizpůsobí. Ukázka vykreslení uživatelského rozhraní pomocí tohoto módu je na obrázku 8.



Obrázek 8: Vykreslení uživatelského rozhraní pomocí Screen Space – Overlay

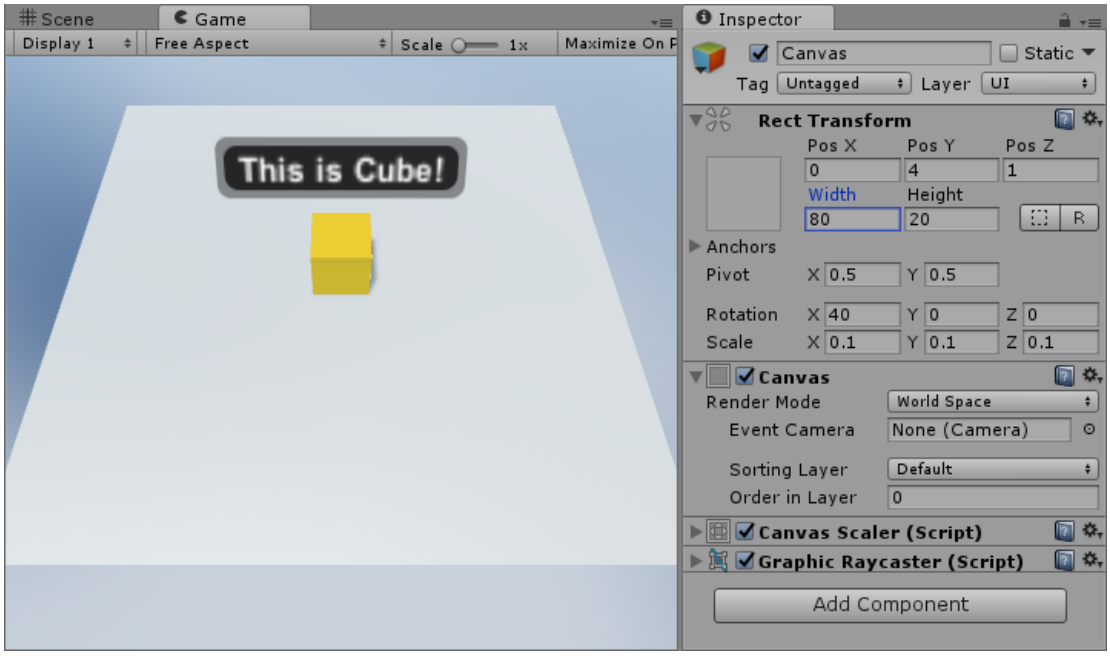
Při použití Screen Space – Camera režimu je uživatelské rozhraní vykresleno před přidělenou kameru v nastavené vzdálenosti. Na ostatních kamerách není toto uživatelské rozhraní zobrazeno. I zde je velikost Canvasu dána velikostí obrazovky a nelze velikost upravit. Oproti Screen Space – Overlay je v tomto režimu možné vytvořit rozhraní s prostorovým efektem, například natočením panelu. Na obrázku 9 je uživatelské rozhraní vykresleno za pomoci tohoto módu.



Obrázek 9: Vykreslení uživatelského rozhraní pomocí Screen Space – Camera

Ve World Space režimu je uživatelské rozhraní vykresleno do scény podobně jako ostatní objekty. Je tak možné nastavit Canvasu polohu a velikost. Tento režim je vhodný pro zobrazení rozhraní, které má být součástí světa. Canvas s tímto režimem je zobrazen kamerou, která snímá oblast, ve které se nachází. Na obrázku 10 je zobrazeno vykreslení Canvasu při zvolení režimu World Space.

Zdroj [14].

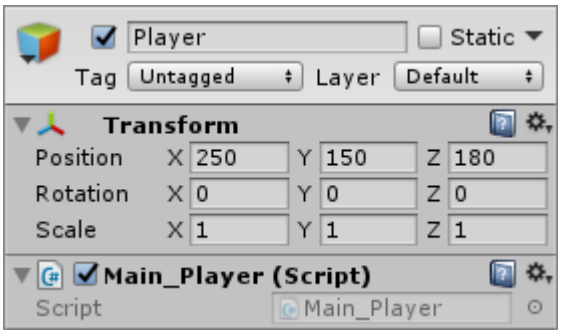


Obrázek 10: Ukázka použití Canvasu s World Space režimem

2.5.5 Komponenty objektů

Chování herních objektů je řízeno podle připojených komponent. Unity nabízí mnoho vestavěných komponent, které jsou univerzální, ale omezené ve své funkčnosti. Pokud je potřeba funkčnost překračující možnosti vestavěných komponent, nabízí Unity možnost připojení skriptů. Komponenty objektů jsou v Unity dostupné v panelu Inspector.

Jednou z hlavních komponent je Transform, který je součástí všech objektů osazených ve scéně. Transform určuje polohu, natočení a měřítko objektu ve 3D prostoru scény. 3D prostor je reprezentovaný osami X, Y a Z. Každá z os má své barevné označení pro jednodušší rozlišení. Na obrázku 11 je ukázka komponenty Transform připojené k objektu Player.



Obrázek 11: Ukázka komponent

Každý objekt, který má přidělený skript, obsahuje po spuštění jeho instanci. Pokud tedy například jeden skript používá více objektů, každý objekt obsahuje svoji vlastní instanci přiděleného skriptu. Na obrázku 11 je ukázka objektu s připojeným skriptem s názvem *Main_Player*.

Zdroj [15].

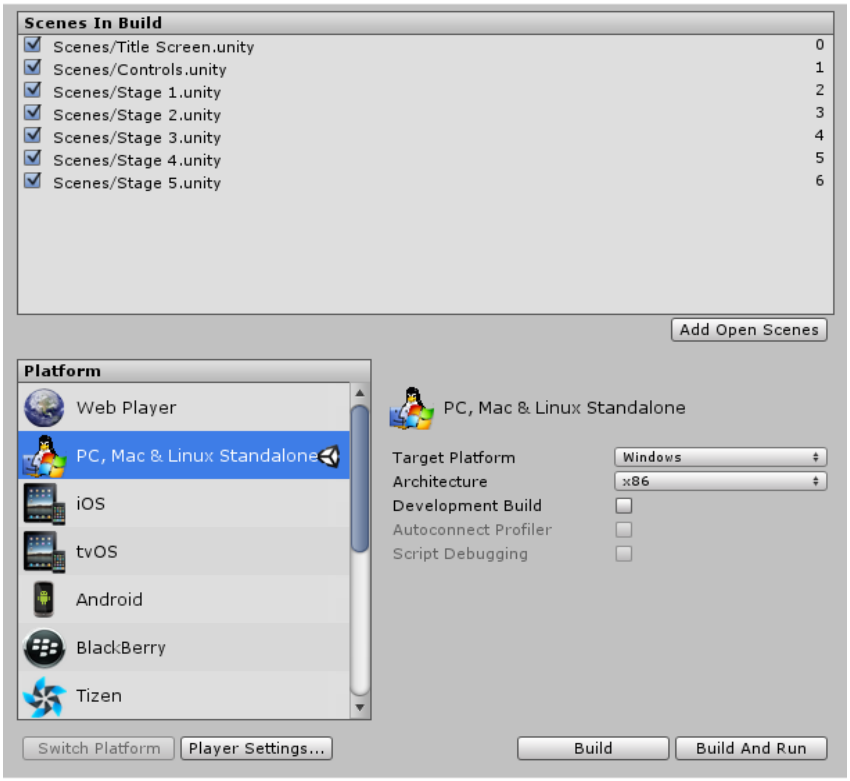
2.5.6 Proces sestavování hry

Při vývoji je hra (projekt) spouštěna pouze pomocí herního enginu. Pro samostatné spuštění je nezbytné hru sestavit (build). Sestavení znamená převést projekt do formátu, se kterým dokáže cílená platforma pracovat. Sestavení projektu v Unity se provádí z okna zvaného Build Settings, ve kterém se nastaví několik potřebných parametrů a poté lze zahájit proces sestavení. Na obrázku 12 je zobrazeno okno Build Settings. Po dokončení sestavení vznikne program, který by měl být spustitelný na všech cílových platformách, například pro systémy Windows vznikne spustitelný soubor s příponou *.exe.

Parametry pro sestavení mohou být:

- zvolení scén, které mají být součástí výsledného sestavení.
- cílová platforma, například PC, Mac & Linux Standalone, nebo Android.
- upřesňující možnosti cílové platformy, například architektura nebo operační systém.

Scény zvolené v parametrech sestavení mají své pořadí. Pokud se má například po spuštění hry otevřít hlavní menu, musí být nastavena jako první ta scéna, která obsahuje hlavní menu.



Obrázek 12: Build Settings[2]

Správu a práci se scénami zajišťuje Scene Manager. To je knihovna Unity, která mimo jiné umožňuje přechod mezi jednotlivými scénami. Pro načtení nové scény je možné použít metodu SceneManager.LoadScene s parametrem volané scény. Při přechodu na novou scénu jsou všechny objekty ze staré scény zničeny, jedinou výjimkou jsou chráněné objekty.

Ve zdrojovém kódu 2 je zobrazena ukázka načtení scény s názvem *Main Menu*. Pro načtení scény je nezbytné zadat identifikační číslo nebo název scény, která má být načtena. Volitelnou možností je zvolení módu jakým má být nová scéna načtena. Jsou k dispozici dva módy a to *Single* a *Additive*. Při zvolení *Single* se všechny aktuálně načtené scény ukončí a načte se jen zadaná scéna. Při zvolení *Additive* se nová scéna otevře k aktuálním scénám. Pokud nemá být nějaký herní objekt zničen při přechodu na novou scénu, je možné použít funkci *DontDestroyOnLoad*. Vstupním parametrem je objekt, který má být zachován.

```
public void BackToMainMenu() {
    SceneManager.LoadScene("Main Menu", LoadSceneMode.Single);
}
```

Zdrojový kód 2: Ukázka načtení scény s názvem Main Menu

Zdroj [16].

[2] Obrázek převzat z [16].

3 GAME DESING

Výsledkem této práce je strategická hra odehrávající se ve 3D prostředí, které je zasazeno do fiktivního světa inspirovaného obdobím středověku. Prostředí hry je zobrazeno skrze kameru z ptačí perspektivy, to znamená, že se na svět ve hře hráč dívá ze shora dolů.

Čas ve hře plyne stejně rychle jako reálný čas, při pozastavení hry se herní čas a tím i všechny herní prvky, jako například produkce surovin nebo pohyb vojsk, také pozastaví. Po uložení a následném načtení hry se bude pokračovat od okamžiku, ve kterém byla hra uložena.

Základním cílem hry je výstavba vesnice a obrana před neustálými útoky nepřátel. Výstavba a obrana vesnice jsou dva prvky hry, které jsou na sobě navzájem závislé, protože bez dostatečné infrastruktury se nelze efektivně bránit a bez obraných jednotek jsou budovy snadným cílem k vyloupení nebo zničení.

Útoky nepřátel jsou náhodně generovány podle podmínek, které mají za cíl dosažení takové obtížnosti, kterou hráč zvládne, ale zároveň nebude příliš jednoduchá a nezačne hráče hra nudit. Podmínky se budou v průběhu hry přizpůsobovat pokroku ve hře a síle hráčovy vesnice.

Nepřátelé mají za cíl vyloupit a oslabit vesnici. Za tímto účelem mohou zničit nebo poničit některé budovy, hlavně hradby a vojenské budovy. Pokud se nebude hráč bránit nebo prohraje v souboji, nepřátelé ukradnou všechny suroviny uložené ve skladišti a sýpce. Část surovin lze schovat do úkrytu a uchránit tak suroviny před uloupením. S uschovanými surovinami nelze pracovat a proces uschování a navrácení surovin do skladu nebo sýpky trvá určitou dobu.

Souboj nepřátelské a hráčovy armády je řešen pouze formou upozornění a oznámení, které statisticky zobrazuje výsledek souboje. V oznámení je zobrazena statistická tabulka rozdělená na dvě části, a to na obránce a útočníka. V každé této části je poté zobrazen počáteční stav vojska a ztráty v boji, případně další informace o účastnících souboje. Upozornění informuje o blížícím se nebezpečí jako je například blížící se nepřátelská armáda.

K dosažení cílů hry hráč může stavět nové budovy nebo vylepšovat současné, může trénovat vojenské jednotky a zkoumat nové technologie. K uskutečnění všech těchto úkonů jsou potřeba suroviny.

Budovy lze stavět nové nebo vylepšovat současné a to z hlavní budovy. Zahájení stavby nebo vylepšení je možné až po splnění požadavků, které mohou například být: přítomnost jiné budovy, nebo úroveň hlavní budovy. Po výstavbě začne budova poskytovat účinek a při vylepšení budovy je daný účinek dále zesilován. Každá budova má předem určenou maximální

úroveň, po jejímž dosažení není možné budovu dále vylepšovat. Několik budov poskytuje kromě účinků také další herní prvky, jako například trénink jednotek, stavbu budov nebo výzkum technologií.

Hráč má pro obranu vesnice k dispozici několik typů vojenských jednotek a budov. Jednotky jsou trénovány v kasárnách nebo stájích, v kasárnách se trénují pěší jednotky a ve stájích jezdci na koních. Pro získání jednotek hráč zvolí požadovaný počet dané jednotky, a pokud jsou splněny požadavky, tak započne výcvik, který trvá určitý čas. Po úspěšném dokončení výcviku se stane daná jednotka součástí vojska vesnice.

Jednotky ve vesnici jsou v případě napadení vesnice automaticky poslány bránit vesnici. Malá část vojska hlídkuje a v případě odhalení útoku varuje vesnici. Ta se může na blížící se útok připravit a získat tak krátkodobý bonus k obraně. Hradby kolem vesnice pomáhají při její obraně poskytováním obranné výhody jednotkám a zvyšují tak šanci na odražení útoku a snižují ztráty obránce.

Výzkum technologií slouží k posilování účinků a odemykání nových budov. Než může být technologie vyzkoumána, musejí být splněny požadavky na úroveň budovy, ve které probíhá výzkum a dokončený výzkum požadovaných technologií v případě že jsou nějaké požadované.

Hra je primárně určena pro PC platformu, po úpravě uživatelského rozhraní by mohla být vydána i na některých mobilních platformách.

3.1 Suroviny

Suroviny jsou základní částí hry, bez nich nejde stavět budovy, trénovat jednotky nebo zkoumat technologie. Suroviny ve vesnici jsou produkovány již od jejího založení základní produkcí, která je neměnná po celou dobu hry. Základní produkci však lze navýšit pomocí účinků budov a technologií na několikanásobek původní hodnoty.

Dřevo

Dřevo se používá ke stavbě a vylepšování budov, výzkumu a tvorbě vojenských jednotek. Hlavním producentem dřeva je budova dřevorubec. Produkce dřeva může být dále navýšena výstavbou budovy pila nebo vyzkoumáním nových technologií.

Kámen

Kámen je nepostradatelnou surovinou při stavbě a vylepšování budov. Je také používán při výzkumu a v malém množství při tvorbě vojenských jednotek. Kámen je produkován kamenolomem, ve kterém dělníci těží kámen z masivních skalních bloků a následně ho

zpracovávají pro běžné použití. Produkci kamene lze zvýšit výzkumem nových technologií nebo rozšířením kamenolomu.

Železo

Železo je nezbytné pro rozvoj, výzkum a vojsko vesnice. Z nitra hory je dolována železná ruda, která je následně přetavena na železo. Ze železa může kovář vykovat zbraně, nástroje pro dělníky, podkovy pro koně a mnoho dalších potřebných věcí pro chod a rozvoj vesnice.

Jídlo

Každý člověk potřebuje jídlo k životu. Na farmě se z polí získá obilí, ze kterého se umele mouka a z mouky se upeče chléb. Jídlo produkuje také lovec, který loví po lesích a pláních divokou zvěř. Produkci jídla lze dále navýšit stavbou budovy mlýn, výzkumem nových technologií nebo zvýšením úrovně farmy.

3.2 Jednotky

Ve hře je k dispozici několik různých vojenských jednotek. Jednotky se od sebe odlišují silou při útoku a obraně, náklady a časem potřebnými pro jejich vycvičení. Každá jednotka přítomná ve vesnici spotřebovává určité množství jídla potřebné pro její přežití.

Kopiník

Kopiník je jedna ze základních jednotek. Její výhody jsou nízké náklady a krátká doba verbování. Nevýhodami jsou nižší útočné a obranné číslo oproti ostatním jednotkám. Tuto jednotku je vhodné verbovat na začátku hry, při nedostatku surovin nebo při potřebě rychlého navýšení obranných jednotek vesnice před útokem nepřátel.

Šermíř

Šermíř je další ze základních jednotek. Je to vcelku univerzální jednotka, která se hodí na všechny typy vojenských akcí. Šermíř má vyšší útočné a obrané číslo oproti kopiníkům, ale za cenu delší doby výcviku a vyšších nákladů na výcvik.

Lučištník

Lučištník společně se šermířem a kopiníkem tvoří trojici základních jednotek dostupných hned po dostavbě budovy kasárna. Lučištník vyniká svoji silou při boji na dálku a díky tomu může vesnici bránit, aniž by přišel do přímého kontaktu s nepřátelskými jednotkami. Díky tomu má lučištník vyšší obrané číslo než šermíř a kopiník.

Jízda

Jízda je typ jednotky dostupný po výstavbě budovy stáje. Je to silná jednotka, která se hodí spíše do útoku. Při obraně už nebude tak účinná jako ostatní jednotky. Jízdu je vhodné použít například při protiútoku, kdy hráčovi jednotky využijí momentu překvapení a přepadnou nic netušící nepřátelské vojsko, které se chystá zaútočit na vesnici.

Rytíř

Rytíř je nová jednotka dostupná po dosažení 4. úrovně u budov kasárna a akademie. Rytíř vyniká poměrem síly a počtu obyvatel potřebných na tuto jednotku, který je nejlepší ze všech jednotek. Tvorba této jednotky je ale velmi nákladná a zdlouhavá. Rytíř se hodí hlavně na obranu, protože používá těžkou zbroj, kvůli které se pohybuje velice pomalu.

3.3 Obyvatelstvo

Počet obyvatel je daný počtem obyvatel přiřazených k budovám, počtem vojáků ve vesnici a počtem právě verbovaných vojáků. Ve vesnici může být pouze omezený počet obyvatel daný kapacitou domů a kasáren. Domů lze postavit několik a dále je pak vylepšovat pro navýšení kapacity. Kasárna jsou pouze jedna, ale maximální počet obyvatel je zvyšován s rostoucí úrovní budovy. Všechny ostatní budovy vyžadují pro svůj chod určitý počet obyvatel, který je nutné rezervovat již při zahájení stavby budovy nebo vylepšení. Pokud nebude dostatek obyvatel, nebude možné stavbu nebo vylepšení zahájit. Každý voják je zároveň i obyvatel a při verbování nových vojáků musí být dostatek obyvatel již při zahájení verbování. Každý obyvatel spotřebovává jídlo a je vhodné mít vyšší produkci, než spotřebu. Nedostatek jídla ovlivní pouze vojáky, kteří mohou začít umírat a hráč tak může přijít o svou armádu nezbytnou pro obranu vesnice.

3.4 Účinek

Účinky zlepšují jednotlivé základní hodnoty vesnice, například produkci surovin, rychlost výcviku jednotek, útočné a obrané čísla jednotek, obranné schopnosti hradeb, náklady na stavbu budov a další. Účinek může být poskytován pouze budovami a technologiemi. Účinek začne působit až po úspěšném dokončení výzkumu technologie nebo výstavby budovy. Síla účinku u vyzkoumaných technologií se dále nedá zvyšovat. U budov se síla účinku odvíjí od úrovně budovy, čím vyšší je její úroveň tím silnější účinek poskytuje.

3.5 Budovy

Ve vesnici je možné postavit několik typů budov. Budovy poskytují účinek nebo je možné v nich zadat určitý úkol. Všechny budovy kromě domů vyžadují pro svou funkci obyvatele, kteří jsou přiděleni již při stavbě budovy.

Hlavní budova

Centrem vesnice je hlavní budova, ze které je vesnice spravována, probíhá odsud správa výstavby a vylepšování budov ve vesnici. Se zvyšováním úrovně hlavní budovy klesá doba výstavby a vylepšení budov. V jeden okamžik mohou probíhat pouze dvě výstavby nebo vylepšení. Probíhající stavba nebo vylepšení budov jsou zobrazovány na informačním panelu vesnice.

Kasárna

V budově kasárna se cvičí nové pěší jednotky a jsou zde ubytovány současné. Součástí budovy je i nádvoří s pomůckami pro výcvik. Se zvyšováním úrovně budovy se zkracuje doba výcviku jednotek a zvyšuje se maximální počet obyvatel ve vesnici.

Dřevorubec

Dřevo je nejzákladnější stavební surovinou a o jeho produkci se stará budova dřevorubce. Dřevorubci musejí nejdříve pokácet strom, který potom zpracují na poptávaný dřevěný produkt. Jak poroste úroveň budovy, poroste i počet dřevorubců a produkce dřeva.

Kamenolom

V kamenolomu kameníci těží kámen z masivních skalních bloků, který následně upravují pro použití při stavbě budov. Těžba kamene je pracná, z toho důvodu v kamenolomu pracuje více dělníků v porovnání s ostatními budovami. Zvyšováním úrovně budovy se zvýší i produkce kamene.

Železný důl

V železném dole horníci dolují železnou rudu nezbytnou pro každou rozvíjející se civilizaci, která je následně přetavena na železo. S každým navýšením úrovně budovy vzroste i produkce železa.

Farma

Farma je hlavní budovou produkující jídlo. Jídlo se vyrábí z pšenice sklizené na blízkých polích. Zvýšením úrovně budovy se zvýší produkce jídla.

Sklad

Ve skladu jsou uskladněny všechny vyprodukované a aktuálně nepotřebné suroviny kromě jídla. Kapacita skladu je omezená a je možné ji navýšit zvýšením úrovně budovy nebo výzkumem nových technologií.

Sýpka

V sýpce jsou uklidněny veškeré zásoby jídla ve vesnici. Vedlejší vlastností sýpky je, že jídlo v ní uskladněné je chráněno před živočichy a přírodními vlivy. Zvýšení maximální skladovací kapacity je možné dosáhnout zvýšením úrovně budovy nebo výzkumem technologie.

Palisáda

Palisáda je dřevěná hradba, která chrání vesnici před útoky nepřátel. Také poskytuje obraným jednotkám výhodu, díky níž jsou silnější. Při silném nepřátelském útoku může být palisáda poškozena nebo dokonce zničena. Zvýšením úrovně budovy se zvyšuje i odolnost budovy a je také posílena obranná výhoda jednotek.

Kovář

Kovář vyrábí a vylepšuje zbraně, nástroje a brnění. Jednotlivá vylepšení je možné provést, až když je splněn požadavek na úroveň budovy. Zvyšování úrovně budovy se tak odemykají jednotlivá vylepšení.

Skrýš

Ve skrýši je možné schovat část surovin ze skladu a sýpky před útokem nepřátel. Se surovinami uloženými ve skrýši však není možné pracovat. Pokud je po útoku, je možné suroviny vrátit zpět. Uschování a vrácení surovin trvá nějakou dobu. Při zvýšení úrovně budovy se zvýší kapacita skrýše.

Tržiště

Na tržišti je možné si nechat vyměnit jeden typ surovin za jiný. Například dřevo za železo. Výměna surovin bude probíhat v poměru 1,25 : 1 nabízené ku přijímaným a v omezeném množství. Zvýšením úrovně budovy se zvýší množství vyměněných surovin.

Stáje

Ve stájích jsou ustájení koně a cvičí se zde jezdci na koních. Zvýšením úrovně budovy se zrychlí rychlost výcviku.

Akademie

Akademie slouží pro technologický rozvoj vesnice a to pomocí výzkumu nových technologií. Nové technologie jsou odemykány zvyšování úrovně budovy a splněním požadavků.

Pila

S pomocí nových technologií byl navržen a postaven mechanismus zvaný pila, který zrychlí a zjednoduší řezání dřeva na trámy, prkna, latě a další dřevěné produkty potřebné ke stavbě a vylepšování budov. Díky tomuto mechanismu se zvýší celková produkce dřeva ve vesnici o určité procento odvozené od úrovně pily.

Mlýn

Budova mlýn přinesla obrovský pokrok v získávání mouky. Dosud museli farmáři ručně vytloukat mouku ze zrnek pšenice, nyní tuto práci odvede mlýn hnaný silou větru. Mlýn tak urychluje produkci jídla ve vesnici o procento odvozené od úrovně budovy.

Bodce

Bodce jsou rozšířením hradeb a jsou to dřevěné kůly s hroty na koncích. Bodce ztěžují přístup k hradbám a tak zvyšují obranou výhodu jednotek ve vesnici a také odolnost samotných hradeb. Bodce mají pouze jednu úroveň a po výstavbě je není možné dále rozšiřovat.

Lovec

Lovec je dodatečná budova produkující jídlo. Jídlo je získáváno z lovu divoké zvěře z okolí vesnice. Zvýšením úrovně budovy se zvýší i produkce jídla.

Obytné domy

Obytné domy poskytují ubytování vesničanům a vojákům. Se zvyšováním úrovně budou ve vesnici přibývat domy a tím se bude zvyšovat maximální možný počet obyvatel ve vesnici.

3.6 Technologický strom

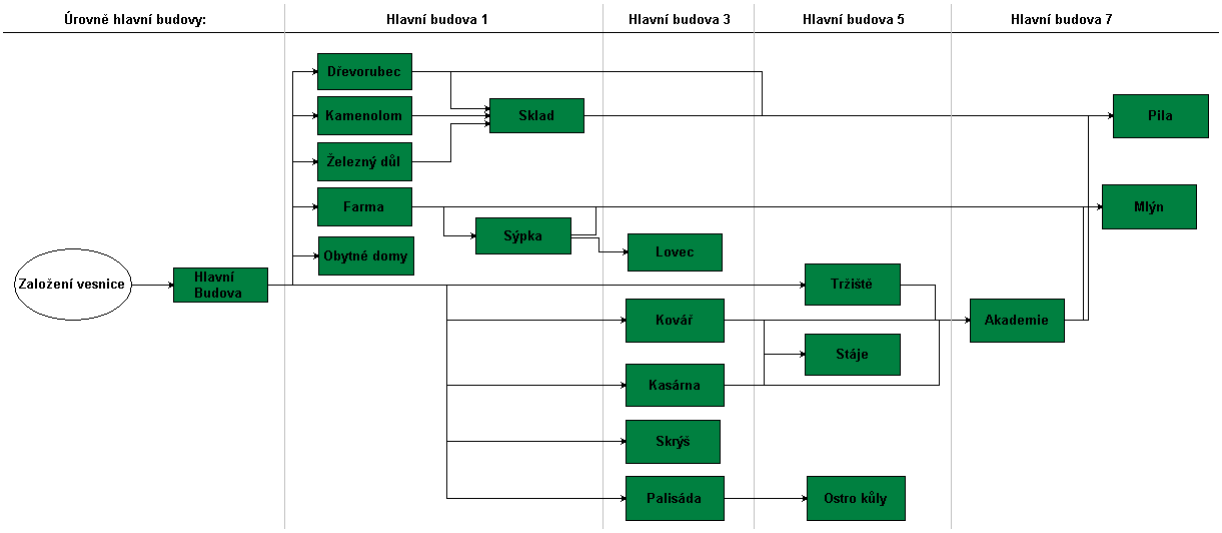
Technologický strom je přehled požadavků stavby jednotlivých budov nebo výzkumu technologií. Jsou celkem tři technologické stromy:

1. Budovy
2. Jednotky a jejich vylepšení
3. Výzkum nových technologií

3.6.1 Technologický strom budov

Budovy nelze stavět libovolně, pro stavbu musí být splněny požadavky dané budovy. Všechny budovy požadují pro zahájení stavby určitou úroveň hlavní budovy, která je jako jediná

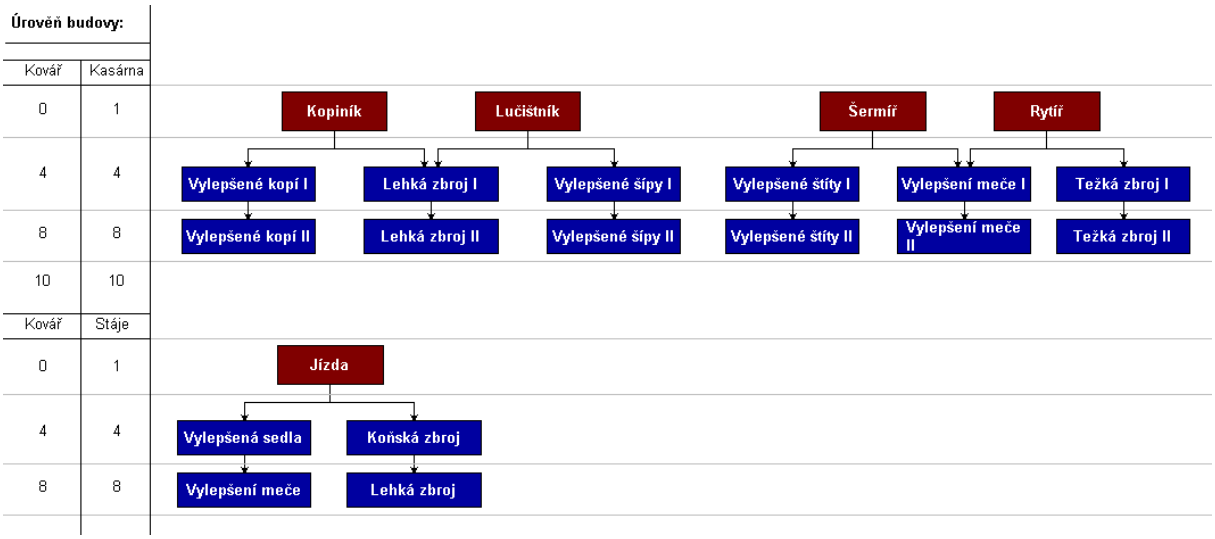
postavena automaticky po založení vesnice. Několik budov požaduje ještě konkrétní další budovu, ze které vychází nebo kterou rozšiřuje. Například pila může být postavena až poté co je ve vesnici postaven dřevorubec a sklad. Strom budov je zobrazen na obrázku 13.



Obrázek 13: Strom budov

3.6.2 Jednotky a jejich vylepšení

Než mohou být jednotky rekrutovány, musí být nejdříve splněny požadavky. Pokud jsou požadavky splněny je možné jednotky verbovat. Každá jednotka může být dále vylepšena výzkumem vylepšení za předpokladu splnění požadavků výzkumu. Vylepšení je aplikováno na všechny jednotky patřící hráči v době dokončení výzkumu a na všechny nově naverbované jednotky. Technologický strom vojenských jednotek je zobrazen na obrázku 14. Jak je vidět na uvedeném obrázku každá vojenská jednotka jde vylepšit čtyřmi vylepšeními, z toho dvě vylepšují útok a dvě obranu. Některé vylepšení jsou ale sdílená a v tom případě stačí provést vylepšení pouze jednou a bude aplikováno na všechny jednotky, co dané vylepšení sdílejí.

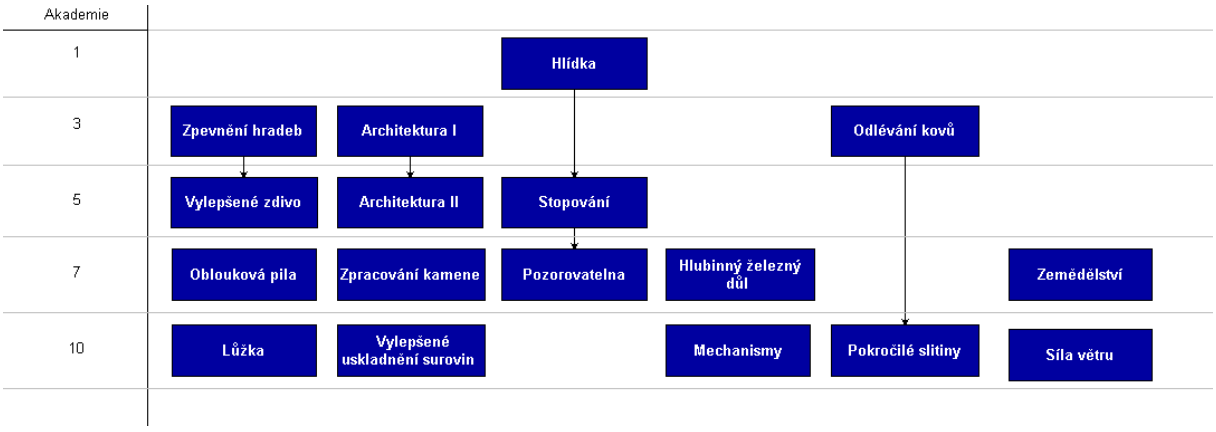


Obrázek 14: Technologický strom vojenských jednotek

Podrobný seznam vylepšení vojenských jednotek obsahující název, popis a účinek daného vylepšení, je uveden v příloze A.

3.6.3 Výzkum nových technologií

Každá civilizace se potřebuje rozvíjet, držet krok s nepřáteli nebo dokonce nepřátele technologicky předstihnout. Nové technologie nemohou být vyzkoumány libovolně, před započetím výzkumu musejí být splněny požadavky pro výzkum dané technologie. Technologický strom výzkumu technologií je zobrazen na obrázku 15.



Obrázek 15: Diagram technologií

Seznam všech dostupných technologií zkoumaných v budově akademie jsou včetně popisu a účinku uvedeny v příloze B.

4 PRAKTICKÉ ŘEŠENÉ PROBLÉMY

Tato část se zabývá problémy vzniklými při tvorbě této práce. Zejména se jedná o vytvoření některých skriptů, které se starají o určitou funkci hry.

4.1 Třída Player

Veškerý herní postup, který hráč vykoná, bylo potřeba uchovat, a právě pro tyto účely bylo nutné vytvořit třídu *Player.* Celý zdrojový kód této třídy je uveden v Příloze C. Třída *Player* uchovává veškeré informace o budovách, surovinách, jednotkách, výzkumu technologií, přijatých oznámeních, upozorněních, a další důležité informace.

Informace o budovách jsou uloženy v seznamu typu List a každá položka seznamu je typu *Building*, která uchovává data o dané budově. Každá budova obsahuje informaci o aktuální a maximální úrovni, nákladech na výstavbu a vylepšeních, účinku, který poskytuje, požadavcích na stavbu, názvu a popisu.

Informace o surovinách jsou jednotlivě uloženy ve třídě *Resource*. Ta obsahuje hodnoty aktuální produkce, skladovaného množství, maximální skladovací kapacity, velikost kapacity skrýše a názvu suroviny. Počet jednotek je uložen ve třídě *Player*. Ovšem informace o jednotkách jsou uloženy ve třídě *UnitsControl,* kde je každý typ jednotky uložen ve třídě *Unit*, která obsahuje informace, vlastnosti a náklady na výcvik.

Ve třídě *Player* jsou také uloženy technologie rozdělené do dvou seznamů, kde každá technologie je uložena ve třídě *Technology.* V prvním seznamu jsou uloženy technologie vylepšující hodnoty jednotek. Ve druhém seznamu jsou uloženy technologie vylepšující vlastnosti vesnice.

Současně jsou ve třídě *Player* uloženy v seznamu přijatá oznámení a upozornění, dále aktuálně stavěné budovy a cvičené jednotky. Je zde také uložen seznam osahující jednotlivé efekty, které poskytují budovy a technologie. Jednotlivá oznámení a upozornění jsou uloženy ve třídě *Report* a jednotlivé efekty jsou uloženy ve třídě *Effect*.

4.2 Uložení a načtení stavu hry

Aby bylo možné ve hře pokračovat i po jejím ukončení, je nutné ukládat stav hry. Postup hrou je uložen v binárních souborech. Tento způsob byl zvolen kvůli jednoduchosti použití, nízkému zatížení systému při ukládání a načítání, velikosti takto uloženého souboru a také proto, že takto uložený soubor je hůře čitelný samotným uživatelem.

Pro ukládání souborů, které mají na systému přetrvat až do dalšího spuštění programu, nabízí Unity možnost použít složku určenou pro tento typ souborů. Cestu ke složce lze získat z proměnné *persistentDataPath* uložené ve třídě Application. Postup hráče je uložen do podsložky *Saves,* v souboru s názvem *JmenoHrace.sav.* Uložení postupu hráče je řešeno serializací třídy *Player* do binárního proudu, který je následně uložen do zvoleného souboru. Jak je vidět ve zdrojovém kódu 3, je použita pro serializaci třída BinaryFormatter a její metoda *Serialize*, která přijímá jako parametr datový proud a objekt pro serializaci, v tomto případě *Player*. Datový proud je typu FileStream, který je vytvořen pro práci se souborem *JmenoHrace.sav* v módu *Create*. To znamená, že při každém ukládání je vytvořen nový soubor.

```
public void SavePlayerData() {
    string fileName = player.name + ".sav";
    string playerDataFile = Path.Combine(Application.persistentDataPath,
        "Saves/" + fileName);
    if (!Directory.Exists(Path.Combine(Application.persistentDataPath, "Saves"))) {
        Directory.CreateDirectory(Path.Combine(Application.persistentDataPath,
            "Saves"));
    }
    BinaryFormatter bf = new BinaryFormatter();
    FileStream stream = new FileStream(playerDataFile, FileMode.Create);
    bf.Serialize(stream, player);
    stream.Close();
}
```

Zdrojový kód 3: Uložení dat hráče

Načítání je podobné ukládání, akorát s několika rozdíly. Za prvé je nutné ověřit, zda soubor existuje, aby se předešlo chybám, pokud by neexistoval. Za druhé je soubor otevřen v módu Open a za třetí je místo funkce *Serialize* použita funkce *Deserialize,* která uložený objekt vytvoří do stavu, ve kterém byl před uložením.

4.3 Pohyb a zoom kamery

Hlavní kamera kvůli svému umístění nedokáže zobrazit celé herní prostředí a z toho důvodu bylo potřeba zajistit možnost jejího pohybu. Pohyb kamery ovládá hráč a pohybu může dosáhnout buď pomocí kláves klávesnice, nebo přiblížením ukazatele myši k okraji obrazovky. Pohyb pomocí klávesnice lze uskutečnit stiskem šipek (nahoru, dolů, doprava, doleva) nebo kláves W, S, A, D. V Unity tyto klávesy představují vertikální a horizontální osu s kladnými a zápornými hodnotami, například klávesa W je kladná vertikální a S je záporná vertikální osa.

Detekce stisku kláves nebo polohy ukazatele myši probíhá ve funkci Update, která je funkcí Unity a je volána každý snímek. Jak je vidět ve zdrojovém kódu 4, je použita pro zjištění stisku klávesy funkce *GetAxis,* ze třídy Input, s názvem osy jako vstupním parametrem. Pro zjednodušení ověření je hodnota, kterou vrací funkce *GetAxis*, převedena na absolutní hodnotu.

Pokud funkce vrátí 0, tak není stisknuta žádná klávesa, pokud nenulové číslo, tak je nějaká klávesa zadané osy stisknuta. Pokud je klávesa osy stisknuta, je možné pokračovat na výpočet a následný pohyb kamery. Při držení klávesy se pohyb kamery postupně urychluje až do maximální hodnoty (*maxAcceleration*), hodnota urychlení (*acceleration*) je dále použita při výpočtech posunu. Následně dojde k výpočtu horizontálního (*horizontalMove*) a vertikálního posunu (*verticalMove*). Nakonec jsou tyto vypočítané hodnoty ověřeny pomocí funkce *CheckNewPosition* a pokud jsou správné, jsou použity pro pohyb kamery funkcí *Translate* s parametry posunu v osách X, Y a Z. Funkce *Translate* posune objekt o zadané hodnoty oproti původní poloze objektu.

```
if (Math.Abs(Input.GetAxis("Horizontal")) > 0 ||
Math.Abs(Input.GetAxis("Vertical")) > 0) {
    if (acceleration <= maxAcceleration)
        acceleration += AccelerationStep;
    horizontalMove = Input.GetAxis("Horizontal") * moveSpeed * acceleration *
        Time.deltaTime;
    verticalMove = Input.GetAxis("Vertical") * moveSpeed * acceleration *
        Time.deltaTime;
    transform.Translate(CheckNewPosition(horizontalMove, 0, verticalMove));
}
```

Zdrojový kód 4: Detekce stisku kláves a pohyb kamery

Pro pohyb kamery pomocí myši je nutné přesunout ukazatel k okraji obrazovky. Ukazatel myši je detekován ve vzdálenosti od okraje rovnající se jednomu procentu pixelů z rozlišení obrazovky. Pokud je například rozlišení 1920×1080, tak u levého okraje je myš detekována ve vzdálenosti 19 pixelů. Ve zdrojovém kódu 5 je vidět část skriptu zabývající se pohybem kamery pomocí myši.

```
if (enabledMouseCameraMove) {
    if (Input.mousePosition.x >= Screen.width * (1 - MouseDetectDistance)) {
        horizontalMove = 2 * moveSpeed * acceleration * Time.deltaTime;
        transform.Translate(horizontalMove, 0, 0);
    } else if (Input.mousePosition.y >= Screen.height * (1 - MouseDetectDistance)) {
        verticalMove = 2 * moveSpeed * acceleration * Time.deltaTime;
        transform.Translate(0, 0, verticalMove);
    } else if (Input.mousePosition.x <= Screen.width * MouseDetectDistance) {
        horizontalMove = 2 * moveSpeed * acceleration * Time.deltaTime;
        transform.Translate(-horizontalMove, 0, 0);
    }else if (Input.mousePosition.y <= Screen.height * MouseDetectDistance) {
        verticalMove = 2 * moveSpeed * acceleration * Time.deltaTime;
        transform.Translate(0, 0, -verticalMove);
    }
}
```

Zdrojový kód 5: Pohyb kamery pomocí myši

Pro lepší přehled vesnice má kamera 3 výškové polohy, mezi kterými je možné přecházet kolečkem myši. Plynulý přechod mezi jednotlivými polohami je řešený animací. Pro detekci

použití kolečka je opět použita funkce *GetAxis,* která vrací hodnotu větší než nula při pohybu kolečkem nahoru a menší než nula při pohybu kolečkem dolů. Dále je podle současné polohy spuštěna animace objektu pomocí funkce *SetTrigger* z komponenty Animator s parametrem názvem spouště (Trigger), který určuje, jaká animace má být spuštěna. Ve zdrojovém kódu 6 je zobrazena část skriptu zabývající se přibližováním a oddalováním kamery.

```
if (Input.GetAxis("Mouse ScrollWheel") > 0) {
    if (currentZoom == cameraZoom3) {
        cameraAnimator.SetTrigger("GoMedium");
        currentZoom = cameraZoom2;
    } else if (currentZoom == cameraZoom2) {
        cameraAnimator.SetTrigger("GoLow");
        currentZoom = cameraZoom1;
    }
} else if (Input.GetAxis("Mouse ScrollWheel") < 0) {
    if (currentZoom == cameraZoom1) {
        cameraAnimator.SetTrigger("GoMedium");
        currentZoom = cameraZoom2;
    } else if (currentZoom == cameraZoom2) {
        cameraAnimator.SetTrigger("GoHigh");
        currentZoom = cameraZoom3;
    }
}
```

Zdrojový kód 6: Přiblížení a oddálení kamery

4.4 Uživatelská interakce

Uživatel může s některými herními objekty nebo uživatelským rozhraním provést interakci, například pro zjištění informací nebo zadaní úkolů. Unity pro takovýto typ interakce nabízí několik způsobů jak interakci vyřešit. Je možné použít funkce třídy MonoBehaviour, EventSystem nebo Raycastery. Je ovšem možné vytvořit a použít vlastní řešení.

Ve hře bylo potřeba vyřešit dva druhy interakce. První je možnost zobrazení okna s důležitými informacemi při najetí myší na některé části uživatelského rozhraní nebo na budovy. Druhým typem je otevření okna pro zadání úkolu nebo získání podrobnějších informací. Na obrázku 16 je ukázka okna s informacemi o označené surovině.

Obrázek 16: Okno s informacemi o surovině dřevo

Při implementaci zobrazení informací o surovinách se vyskytl problém. Pro zobrazení informací byly nejdříve použity funkce třídy EventSystem, jenže docházelo k blikání okna a tento problém se nepodařilo vyřešit. Z toho důvodu bylo zvoleno vytvoření vlastních funkcí zajišťujících tuto interakci. Vytvořené funkce fungují tak, že pokud souřadnice, na kterých se nachází myš, jsou uvnitř ukazatele surovin, tak se otevře okno s informacemi.

Jak je vidět ve zdrojovém kódu 7, je pro detekci ukazatele myši použita třída Rect, která představuje obdélník s totožnými rozměry a pozicí, jako ukazatel dané suroviny. Pozice a rozměry jsou vypočítány ve funkci *CalculateRect*. Součástí třídy Rect je i funkce *Contains*, která ověří, zda souřadnice vložené jako vstupní parametr jsou uvnitř obdélníku. Pokud je myš uvnitř obdélníku a není již okno otevřeno, dojde k zobrazení informací. Pokud je myš mimo obdélník a je otevřeno okno, dojde k jeho zavření.

```
private void FixedUpdate() {
    rect = calculateRect();
    if (rect.Contains(Input.mousePosition) && ! isOpened){
        guiManager.CreateInfoResourceDialog(name, Input.mousePosition, gameObject);
        isOpened = true;
    }
    if (!rect.Contains(Input.mousePosition) && isOpened) {
        guiManager.HideInfoResourceDialog();
        isOpened = false;
    }
}
private Rect calculateRect() {
    var scale = GetComponent<RectTransform>().lossyScale;
    float x = GetComponent<RectTransform>().sizeDelta.x * scale.x;
    float y = GetComponent<RectTransform>().sizeDelta.y * (scale.y + 0.1f);
    return new Rect(transform.position.x, transform.position.y - y, x, y);
}
```

Zdrojový kód 7: Funkce detekující ukazatel myši

Pro interakci s budovami byly použity funkce *OnMouseDown*, *OnMouseEnter*, *OnMouseExit* a *OnMouseOver* ze třídy MonoBehaviour. Tyto funkce mohou být použity pouze na prvky grafického uživatelského rozhraní (GUI) nebo objekty s komponentou Collider. Každá budovy tak musí mít připojenou komponentu Collider.

Z počátku vše fungovalo jak má, ovšem při testování bylo zjištěno, že pokud je budova překrytá prvkem uživatelského rozhraní, tak i přesto zachytává interakce, což je nežádoucí. Tento problém byl u funkcí *OnMouseEnter*, *OnMouseOver* a *OnMouseDown* vyřešen použitím komponenty Graphics Raycaster připojené k objektu Canvas a následně použitím funkce *Raycast* ze třídy Graphics Raycaster. Funkce *Raycast* vrací seznam prvků uživatelského rozhraní nacházejících se na zadaných souřadnicích. Jako vstupní parametry přijímá souřadnice a seznam, do kterého budou uloženy výsledky. Pokud bude v seznamu výsledků nula objektů,

tak se nad objektem nenachází žádný prvek uživatelského rozhraní. Je také možné udělat výjimku, pokud je nad objektem detekováno pouze informační okno. Získání a zpracování výsledků z funkce *Raycast* probíhá ve funkci *OverThisObject*, která je zobrazena ve zdrojovém kódu 8. Funkce vrátí hodnotu *true*, pokud je otevřeno okno s informacemi pod názvem *Building_info(Clone)* nebo není nad objektem žádný prvek uživatelského rozhraní. V ostatních případech vrací funkce *false*.

```
public bool OverThisObject() {
    PointerEventData ped = new PointerEventData(null);
    ped.position = Input.mousePosition;
    List<RaycastResult> results = new List<RaycastResult>();
    gr.Raycast(ped, results);
    if (results.Count == 0) {
        return true;
    } else {
        foreach (RaycastResult rr in results) {
            if (rr.gameObject.name == "Building_info(Clone)") {
                return true;
            }
        }
    }
    return false;
}
```

Zdrojový kód 8: Funkce OverThisObject

Ve zdrojovém kódu 9 jsou zobrazeny funkce *OnMouseEnter* a *OnMouseExit*. První jmenovaná funkce je volána při vstupu ukazatele myši na budovu. Pokud nad budovou není prvek uživatelského rozhraní, tak se zobrazí okno se základními informacemi o budově. *OnMouseExit* je volána, pokud ukazatel myši opustí budovu. V tom případě je okno s informacemi zničeno a proměnná uchovávající odkaz na okno je vymazána.

```
private void OnMouseEnter() {
    if (OverThisObject()) {
            OpenInfo();
    }
}

private void OnMouseExit() {
    Destroy(openedInfo);
    openedInfo = null;
}
```

Zdrojový kód 9: Funkce OnMouseEnter a OnMouseExit ze skriptu Building_script

Ve zdrojovém kódu 10 je uvedena funkce *OnMouseOver*, která je volána každý snímek, dokud se ukazatel myši nachází nad objektem s touto funkcí. Hlavním úkolem této funkce je zajistit pohyb okna s informacemi podle pohybu myši. Okno je tak vždy zobrazeno poblíž ukazatele myši. Při vstupu do funkce dojde nejdříve k ověření, zda není nad objektem prvek uživatelského

rozhraní pomocí funkce *OverThisObject*. Pokud není, ověří se, zda je již otevřeno okno s informacemi a pokud ano tak v případě pohybu myši se vypočítá nová poloha okna pomocí funkce *SetPositionOfWindow*, která je uvedena ve zdrojovém kódu 10. Pokud okno s informacemi není otevřeno tak dojde k jeho otevření poblíž aktuální polohy ukazatele myši.

```
private void OnMouseOver() {
    if (OverThisObject()) {
            if (openedInfo != null) {
                if (Input.mousePosition != openedInfo.transform.position)
                    SetPositionOfInfoWindow();
            } else {
                OpenInfo();
            }
    } else {
        if (openedInfo != null) {
            Destroy(openedInfo);
            openedInfo = null;
        }
    }
}

public void SetPositionOfInfoWindow() {
    var windowScale = openedInfo.GetComponent<RectTransform>().lossyScale;
    var position = Input.mousePosition;
    var windowSize = openedInfo.GetComponent<RectTransform>().sizeDelta.x *
        windowScale.x;
    if ((Screen.width - windowSize) > position.x) {
        openedInfo.transform.position = position;
    } else {
        openedInfo.transform.position = new Vector3(position.x - windowSize,
            position.y, position.z);
    }
}
```

Zdrojový kód 10: Funkce OnMouseOver a SetPositionOfInfoWindow

Funkce *OnMouseDown*, uvedená ve zdrojovém kódu 11, je volána při stisknutí tlačítka myši. Při zavolání této funkce je uzavřeno okno s informacemi a následně dojde k ověření, zda není nad objektem prvek uživatelského rozhraní pomocí funkce *OverThisObject.* Pokud není okno budovy již otevřeno, dojde k jeho otevření pomocí funkce *OpenBuildingWindow*.

```
private void OnMouseDown() {
    Destroy(openedInfo);
    if (OverThisObject()) {
        if (openedWindow == null) {
            OpenBuildingWindow();
        }
    }
}
```

Zdrojový kód 11: Funkce OnMouseDown

Celý skript s názvem Building_script, obstarávající všechny interakce související s budovami, je uveden v příloze D.

4.5 Práce se surovinami

Jedním z herních prvků jsou suroviny, které jsou nezbytné pro rozvoj vesnice. Suroviny jsou získávány pravidelnou produkcí z budov a vylepšení. Produkce surovin ve hře je realizována přírůstkem určitého množství každou jednu vteřinu.

Produkce surovin se provádí z funkce *Update*, která je volána každý snímek. Aby bylo možné produkci zajistit, bylo nezbytné zjistit, kdy uplyne jedna vteřina. Pro vyřešení tohoto problému je možné použít proměnou *time* ze třídy Time, která poskytuje čas od spuštění hry. Tento čas je možné pozastavit, což je vhodné, pokud by například chtěl uživatel hru pozastavit. Také je tento čas pozastaven, když je hra minimalizována nebo přesunuta na pozadí.

Jak je vidět ve zdrojovém kódu 12, tak pro zjištění uplynutí jedné vteřiny je použita proměnná *secondAccumulator*. Tato proměnná je porovnávána v podmínce s proměnnou *time* a pokud je menší, tak je do podmínky vstoupeno. Uvnitř podmínky je tato proměnná inkrementována o jednu vteřinu a dále jsou postupně volány funkce zajišťující produkci surovin, stavbu budovy, výcvik jednotek a další. Produkce surovin je uskutečněna funkcí *UpdateResource*, která přidá ke každé surovině množství vyprodukované za jednu vteřinu.

```
void Update() {
     if (Time.time > secondAccumulator) {
        secondAccumulator++;
        timeFromStart = timeFromStart.AddSeconds(1);
        playersControl.UpdateResources();
        villigeInfo.UpdateTimers();
        clock.text = string.Format("{0:HH\\:mm\\:ss}", timeFromStart);
    }
}
```

Zdrojový kód 12: Funkce Update zajišťující přírůstek surovin

4.6 Získání a využívání assetů

Jak již bylo popsáno výše, pro tvorbu hry se používají assety. Ty je možné si vytvořit svépomocí nebo je získat již hotové. Hotové lze získat například z Asset Storu Unity [17].

Z Asset Storu je použito několik balíků assetů. Hojně používány jsou assety z balíku Fancy Villige [18]. Z tohoto balíku pochází většina stromů, keřů a další různé dekorační objekty. Také z něho pochází několik typů budov umístěných ve vesnici a textury, které byly použity při tvorbě terénu a vesnice.

Z několika dalších balíků jsou použity také textury. Těmito balíky jsou: Stone and Brick Texture Pack [19], Yughues Free Stone Materials [20], YugHues Free Cobble Materials [21], a Roof textures [22].

Veškeré zvukové efekty a hudba je použita z balíků UI Sfx [23], Fantasy Music Lite [24] a Authentic Early Medieval Ages Pack (Free) [25].

Současně bylo použito několik assetů ze Standard Assets. Tyto assety jsou součástí Unity a je možné je používat ve vlastních projektech. Ze Standard Assets jsou ve hře použity textury povrchů.

Svépomocí bylo vytvořeno několik assetů budov, textur, celé uživatelské rozhraní a všechny skripty.

5 VLASTNÍ HRA

Kapitola vlastní hra popisuje hru jako výsledný produkt. Je zde popsáno, jak hra vypadá, jak se ovládá, jaké funkce má a jaké prostředky nabízí uživateli pro interakci. Na obrázku 17 je zobrazen záběr z rozehrané hry je tak možné vidět všechny objety, prvky hry a uživatelské rozhraní.

Obrázek 17: Ukázka ze hry

5.1 Základní herní principy

Mezi základní principy hry patří stavba a rozvoj vesnice, stavba vojska a obrana před útoky nepřátel. Pro všechny tyto účely jsou nezbytné suroviny. Suroviny jsou produkovány budovami a vylepšeními. Množství surovin je zvyšováno produkcí každou vteřinu o hodnotu aktuální produkce v sekundách.

Dalším principem jsou náhodné útoky nepřátelských vojsk na vesnici. Cílem nepřátel je vyloupit a oslabit hráčovu vesnici. Útok nepřátel je možné odhalit. Šanci na odhalení lze zvýšit výzkumem technologií. Pokud na vesnici útočí nepřítel a je jeho útok odhalen, obdrží hráč upozornění. Pokud proběhne boj, hráč obdrží oznámení s výsledkem boje. Při odhalení útoku na vesnici je odemknuta možnost provést protiútok a překvapit tak útočící jednotky nepřítele.

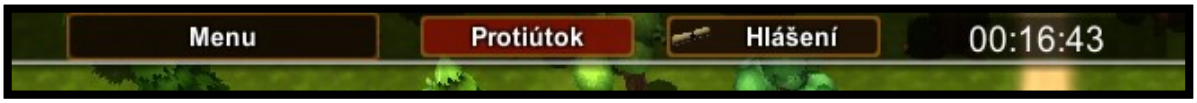
Protiútok lze provést pomocí tlačítka *Protiútok*, po stisknutí se otevře okno, ve kterém se vybere počet jednotek pro útok a pro odeslání se stiskne tlačítko *Odeslat.*

5.2 Uživatelské rozhraní

Většina informací týkajících se hry je zobrazována hráči prostřednictvím uživatelského rozhraní. To je tvořeno stálými a dynamickými částmi. Stálé (neměnné) části uživatelského rozhraní jsou ty části, které jsou vždy vidět a nejde je nijak vypnout. Mezi tyto části patří horní, pravý a spodní panel. Dynamické části jsou v průběhu hry zobrazovány a zavírány, příkladem takových částí jsou okna budov, která se otevřou až po kliknutí na budovu. Když už není okno budovy potřeba, je zavřeno pomocí symbolu křížku.

Horní panel

Jak je vidět na obrázku 18, tak v levé části horního panelu se nachází tlačítka *Menu* a *Hlášení*, a počítadlo od spuštění hry. Při stisknutí tlačítka *Menu* je zobrazeno menu hry. Při stisknutí tlačítka *Hlášení* se otevře okno se všemi obdrženými oznámeními. V levé části horního panelu se také může zobrazit tlačítko *Protiútok*.



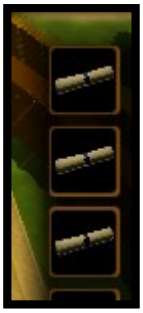
Obrázek 18: Levá část horního panelu

Pravá část horního panelu, zobrazená na obrázku 19, obsahuje informace o stavu surovin, které jsou aktuálně uloženy ve skladu, a také o aktuálním a maximálním počtu obyvatel ve vesnici. Informace o surovinách i obyvatelích jsou získány ze třídy *Player*, ve které jsou uloženy a pravidelně aktualizovány.

Obrázek 19: Pravá část horního panelu

Pravý panel

Pravý panel slouží pro zobrazení nově příchozích hlášení, která jsou dále rozdělena na upozornění a oznámení. Při najetí myší na hlášení se zobrazí důležité informace, typ hlášení, a pokud jde o upozornění tak i odpočítávání události, na kterou upozorňuje. Na obrázku 20 je zobrazen pravý panel.



Obrázek 20: Pravý panel uživatelského rozhraní

Spodní panel

Ve spodním panelu jsou zobrazeny některé další informace o vesnici. Je zde uveden aktuální počet jednotek ve vesnici, právě trénované jednotky, a právě stavěné nebo vylepšované budovy. Spodní panel je zobrazen na obrázku 21.

Obrázek 21: Spodní panel

5.3 Stavba budov

Stavba nebo vylepšení jakékoliv budovy se uskutečňuje v okně hlavní budovy, které je zobrazeno na obrázku 22. Okno se otevře po kliknutí na hlavní budovu. V seznamu budov v okně hlavní budovy lze vybrat budovu, po vybrání se zobrazí panel s popisem, informacemi a možností stavby nebo vylepšení budovy. Vylepšení je možné provést, pokud již byla budova postavena. Stavbu nebo vylepšení lze provést, pouze pokud jsou splněny požadavky a jsou dostupné zdroje.

Obrázek 22: Okno hlavní budovy

5.4 Výcvik jednotek

Výcvik jednotek je možný buď z budovy kasárna, nebo ze stájí. Po kliknutí na tyto budovy se otevře okno, ve kterém se zadá požadovaný počet daného typu jednotky a pokud jsou dostupné zdroje, lze zahájit výcvik stisknutím tlačítka *Rekrutovat*. Pokud by nebyl dostatek zdrojů, tak nepůjde trénink zahájit a hráč bude o nedostatku zdrojů informován prostřednictví textového upozornění. Okno budovy kasárna je zobrazeno na obrázku 23.

Obrázek 23: Okno budovy kasárna

5.5 Výzkum technologií

Výzkum technologií se provádí z okna budov akademie nebo kováře. Při spuštění nové hry je každá technologie nejdříve uzamčená. Při každém otevření okna akademie nebo kováře

proběhne u všech technologií ověření, zda jsou splněny požadavky pro výzkum. Technologie splňující podmínky jsou odemčeny a je možné je vyzkoumat. Zahájit výzkum odemčené technologie je možné, pokud je dostatek zdrojů a není aktuálně zkoumána jiná technologie. Každý výzkum trvá určitou dobu a aktuální výzkum je zobrazován v okně akademie nebo kováře. Po dokončení výzkumu je účinek technologie aktivován.

ZÁVĚR

Vytvořil jsem strategickou hru pro jednoho hráče v herním enginu Unity, odehrávající se ve 3D prostředí a primárně určenou pro PC platformu. Při tvorbě hry si bylo možné zkusit, jak vypadá takový vývoj a vytvoření počítačové hry. Největším problém bylo, že toto byla moje první větší tvorba v jakémkoliv herním enginu. Z toho vyplývá, že spoustu věcí jsem se musel naučit při samotném vytváření hry a tvorba tak trvala déle, než kdybych již měl zkušenosti s tvorbou her. Díky tomu jsem ale měl možnost získat spoustu nových znalostí a zkušeností, které se hodí při tvorbě dalších projektů nebo při uplatnění v zaměstnání.

Stěžejní byla tvorba v herním enginu Unity, ve kterém byla hra seskládána z jednotlivých assetů a skriptů do výsledné podoby. Unity je propracovaný herní engine, který nabízí spoustu nástrojů a funkcí pro tvorbu různých typů her pro mnoho podporovaných platforem. Je uživatelsky přívětivý a je tak vhodný pro začátečníky. Je v něm ale možné vytvářet komplexní a propracované projekty. Jednou z dalších výhod je podpora skriptů v programovacím jazyce C#, který je uživatelsky přívětivější a programování je tak rychlejší a snazší než například v C++.

Skripty v jazyce C# jsem vytvářel ve Visual Studiu a při jejich tvorbě jsem mohl využít základních znalostí a dovedností získaných při studiu na škole avšak pro tvorbu hry bylo nezbytné si osvojit spoustu nových znalostí. Především se jednalo o práci s knihovnami a funkcemi samotného herního enginu nebo o tvorbu různých algoritmů zajišťujících funkčnost jednotlivých částí hry.

Vytvořená hra se oproti mému původnímu návrhu liší, protože kvůli časovým nárokům nebylo možné vytvořit a implementovat hru do navrhované podoby, funkčnosti a obsahu. Hra je tak vytvořena s pouze základními funkcemi a mechanismy umožňující fungování tohoto typu hry.

Pokud bych ve vývoji hry pokračoval, tak bych do hry přidal prvky z původního návrhu. Například objekty představující obyvatele, možnost spravovat více vesnic, generování mapy, vojsko v podobě herních objektů, implementaci hráčů řízených počítačem a mnoho dalších.

POUŽITÁ LITERATURA

[1] Co je programování? *Jak se naučit programovat.py.cz* [online]. 2005-01-09 [cit. 2017-03-21]. Dostupné z: http://jaksenaucitprogramovat.py.cz/cztutwhat.html

[2] KRUBOVÁ, Kateřina. Historie programovacích jazyků. *Fi.muni.cz* [online]. 2000 [cit. 2017-03-21]. Dostupné z: http://www.fi.muni.cz/usr/jkucera/pv109/2000/xkrubova.htm

[3] KADLEC, Josef. Obecné pojednání o programovacích jazycích. *Linuxsoft.cz* [online]. 2004-07-16 [cit. 2017-03-21]. ISSN 1801-3805. Dostupné z: http://www.linuxsoft.cz/article.php?id_article=268

[4] MORAVEC, Zdeněk a Martin ŠIMEČEK. Klíčové slovo this v C#. *Programujte.com* [online]. 2009-06-08 [cit. 2017-03-28]. Dostupné z: http://programujte.com/clanek/2009060701-klicove-slovo-this-v-c/

[5] Asset Workflow. In: *Unity - Manual* [online]. 2017-03-29 [cit. 2017-04-05]. Dostupné z: https://docs.unity3d.com/Manual/AssetWorkflow.html

[6] Unity - Game engine, tools and multiplatform. *Unity3d.com* [online]. 2017 [cit. 2017-04-05]. Dostupné z: https://unity3d.com/unity

[7] Visual Studio IDE. *MSDN Library* [online]. 2017 [cit. 2017-04-15]. Dostupné z: https://msdn.microsoft.com/en-gb/library/dn762121(v=vs.140).aspx

[8] About. *Blender.org* [online]. 2017 [cit. 2017-04-15]. Dostupné z: https://www.blender.org/about/

[9] Kapitola 1. Úvod: 1. Vítá vás GIMP. *GIMP Documentation* [online]. 2007 [cit. 2017-04-15]. Dostupné z: https://docs.gimp.org/2.2/cs/introduction.html

[10] Importing Assets. In: *Unity - Manual* [online]. 2017-03-29 [cit. 2017-04-15]. Dostupné z: https://docs.unity3d.com/Manual/ImportingAssets.html

[11] Prefabs. In: *Unity - Manual* [online]. 2017-03-29 [cit. 2017-04-15]. Dostupné z: https://docs.unity3d.com/Manual/Prefabs.html

[12] Scenes. In: *Unity - Manual* [online]. 2017-03-29 [cit. 2017-04-15]. Dostupné z: https://docs.unity3d.com/Manual/CreatingScenes.html

[13] Instantiating Prefabs at runtime. In: *Unity - Manual* [online]. 2017-03-29 [cit. 2017-04-15]. Dostupné z: https://docs.unity3d.com/Manual/InstantiatingPrefabs.html

[14] Canvas. In: *Unity - Manual* [online]. 2017-03-29 [cit. 2017-04-11]. Dostupné z: https://docs.unity3d.com/Manual/UICanvas.html

[15] Introduction to components. In: *Unity - Manual* [online]. 2017-03-29 [cit. 2017-04-15]. Dostupné z: https://docs.unity3d.com/Manual/Components.html

[16] Publishing Builds. In: *Unity - Manual* [online]. 2017-03-29 [cit. 2017-04-15]. Dostupné z: https://docs.unity3d.com/Manual/PublishingBuilds.html

[17] Asset Store. *Unity3d.com* [online]. 2017 [cit. 2017-04-26]. Dostupné z: https://www.assetstore.unity3d.com/en/

[18] NIX SOLUTIONS. Fancy Village. In: *Asset Store* [online]. 2016-03-29 [cit. 2017-04-20]. Dostupné z: https://www.assetstore.unity3d.com/en/#!/content/58614

[19] LYLEK GAMES. Stone and Brick Texture Pack. In: *Asset Store* [online]. 2015-09-02 [cit. 2017-04-22]. Dostupné z: https://www.assetstore.unity3d.com/en/#!/content/25764

[20] NOBIAX / YUGHUES. Yughues Free Stone Materials. In: *Asset Store* [online]. 2015 [cit. 2017-04-22]. Dostupné z: https://www.assetstore.unity3d.com/en/#!/content/12962

[21] NOBIAX / YUGHUES. Yughues Free Cobble Materials. In: *Asset Store* [online]. 2015 [cit. 2017-04-22]. Dostupné z: https://www.assetstore.unity3d.com/en/#!/content/12957

[22] LANGVV. Roof textures. In: *Asset Store* [online]. 2012-12-28 [cit. 2017-04-24]. Dostupné z: https://www.assetstore.unity3d.com/en/#!/content/5844

[23] LITTLE ROBOT SOUND FACTORY. UI Sfx. In: *Asset Store* [online]. 2015-07-16 [cit. 2017-04-22]. Dostupné z: https://www.assetstore.unity3d.com/en/#!/content/36989

[24] GROSSMANN, Vasco. Fantasy Music Lite. In: *Asset Store* [online]. 2017-01-23 [cit. 2017-04-22]. Dostupné z: https://www.assetstore.unity3d.com/en/#!/content/72931

[25] MARMA. Authentic Early Medieval Ages Pack. In: *Asset Store* [online]. 2016-05-09 [cit. 2017-04-22]. Dostupné z: https://www.assetstore.unity3d.com/en/#!/content/34407

PŘÍLOHY

Příloha A: Seznam vylepšení vojenských jednotek.

- Kopiník:
  - Vylepšené kopí I:
    Zvyšuje kvalitu zbraně kopí a zvyšuje tak útočnou sílu jednotky o 10%.
  - Vylepšené kopí II:
    Další úroveň vylepšení. Útočná síla jednotky je navýšena o dalších 10%.
  - Lehká zbroj I:
    Získání zbroje, která bude odolná, ale dost lehká na to, aby nezhoršovala pohyblivost jednotky. Obrana jednotky navýšena o 10%.
  - Lehká zbroj II:
    Vylepšení lehké zbroje pro ještě lepší vlastnosti. Obrana jednotky navýšena o dalších 10%.
- Lučištník
  - Vylepšené šípy I:
    Vylepšuje kvalitu šípů a zvyšuje tak útočnou sílu lučištníků o 10%.
  - Vylepšené šípy II:
    Další úroveň vylepšení šípů. Útočná síla lučištníků je navýšena o dalších 10%.
  - Lehká zbroj I:
    Získání zbroje, která bude odolná, ale dost lehká na to, aby nezhoršovala pohyblivost jednotky. Obrana jednotky navýšena o 10%.
  - Lehká zbroj II:
    Vylepšení lehké zbroje pro ještě lepší vlastnosti. Obrana jednotky navýšena o dalších 10%.
- Šermíř
  - Vylepšené meče I:
    Vylepší kvalitu meče a zvyšuje útočnou sílu o 10%.
  - Vylepšené meče II:
    Další úroveň vylepšení meče. Útok je zvýšen o dalších 10%.
  - Vylepšené štíty I:
    Vylepšení štítů, které zvýší obranu o 10%.
  - Vylepšené štíty II:
    Další úroveň vylepšení štítů, které zvýší obranu o dalších 10%.
- Jezdec
  - Vylepšená sedla:
    Zlepšuje boj ze sedel jejich vylepšením. Útok je zvýšen o 10%.

- Vylepšené meče:
  Vylepší kvalitu meče a zvyšuje útočnou sílu o dalších 10%.

- Koňská zbroj:
  Získání zbroje pro koně, díky které bude kůň lépe chráněn před nepřáteli. Obrana jízdy zvýšena o 10%.

- Lehká zbroj:
  Získání zbroje, která bude odolná, ale dost lehká na to, aby nezhoršovala pohyblivost jednotky. Obrana jednotky navýšena o 10%.

- Rytíř
  - Vylepšené meče I:
    Vylepší kvalitu meče a zvyšuje útočnou sílu o 10%.

  - Vylepšené meče II:
    Další úroveň vylepšení meče. Útok je zvýšen o dalších 10%.

  - Těžká zbroj I:
    Vyzkoumání vysoce odolné těžké zbroje. Obrana rytíře zvýšena o 10%.

  - Těžká zbroj II:
    Další vylepšení těžké zbroje zlepšující odolnost. Obrana zvýšena o dalších 10%.

Příloha B: Seznam technologií.

- **Hlídka**
  Ve vesnici a kolem vesnice budou hlídkovat jednotky. Hlídky mohou objevit špeha, snažícího se zjistit důležité informace o vesnici, nebo mohou objevit blížící se útok a varovat tak vesnici, která se může na útok připravit. Šance na odhalení špeha se zvýší o 20% a šance na objevení blížícího se útoku se zvýší o 20%.

- **Stopování**
  Naučí hlídky číst ve stopách a zvýší tak šanci na odhalení nepřátel o dalších 15%.

- **Pozorovatelna**
  V okolí vesnice budou vybudovány ukryté pozorovatelny, ze kterých budou mít hlídky lepší přehled nad okolím a zvýší se tak šance na odhalení nepřátel o dalších 15%.

- **Architektura I**
  Nalezení nových postupů stavby a zrychlení tak výstavby a vylepšení budou o 10%.

- **Architektura II**
  Další vylepšení stavebních postupů a materiálů, za účelem zrychlení výstavby a vylepšení budou o dalších 10%.

- **Zpevnění hradeb**
  Zvýšení odolnosti hradeb před útoky nepřátel o 10% a také zvyšuje obranu jednotek uvnitř vesnice o dalších 5%.

- **Vylepšené zdivo**
  Zvýšení odolnosti všech budov ve vesnici a kolem vesnice o 10%.

- **Odlévání kovů**
  Snižuje dobu verbování jednotek zrychlením výroby zbraní.

- **Pokročilé slitiny**
  Díky pokročilejším slitinám jsou zbraně ostřejší a lehčí a zbroje jsou pevnější, odolnější a lehčí. Zvyšuje útok a obranu všech typů jednotek o dalších 10%.

- **Oblouková pila**
  Zrychluje těžbu dřeva a tím je zvýšena produkce o 10%.

- **Zpracování kamene**
  Kameníci těží kámen rychleji a zvyšují tak produkci kamene o 10%.

- **Hlubinný železný důl**
  Železné doly jsou prohloubeny za účelem nalezení dalších žil železné rudy a zvýšit tak produkci železa. Produkce železa se zvýší o 15%.

- **Zemědělství**
  Pomocí nových metod a technologií dokážou farmáři produkovat více jídla a produkce jídla je tak zvýšena o 10 %.

- **Lůžka**
  Úpravou a uspořádáním ložných prostor je navýšen maximální počet obyvatel o 15%.

- **Vylepšené uskladnění surovin**
  Uspořádáním a systémem ve skladování je zvýšena skladovací kapacita skladu a sýpky o 15%.

- **Mechanismy**
  Umožňuje stavbu složitější mechanismů a odemyká budovu pila, ve které jsou nové technické vědomosti využity.

- **Síla větru**
  Umožňuje využívat síly přírody ve prospěch obyvatel výstavbou mlýnu hnaného větrem, který umele z pšenice mouku jen s minimem manuální práce.

Příloha C: Skript se třídou Player

```
using System.Collections.Generic;
using System;

[Serializable()]
public class Player{

    public string player_name;
    public List<Building> buildings;
    public List<Unit> units_list;

    public Resource player_wood;
    public Resource player_stone;
    public Resource player_iron;
    public Resource player_food;
    public Resource player_population;

    public List<Technology> improvement_techs;
    public List<Technology> military_techs;

    public int player_spears;
    public int player_swordsmans;
    public int player_archers;
    public int player_cavalrys;
    public int player_knights;
    public List<Attack> counterAttacks;
    public List<Attack> inCommingAttacks;

    public List<Report> player_reports;
    public List<Effect> effects;
    public List<BuildingConstruction> constuctionQueue;
    public List<Unit> unitsTrainingQueue;

    public DateTime timeFromStart;

    public Player() { }
    public Player(string name, Resource wood, Resource stone, Resource iron,
        Resource food, Resource population, int spears, int swordsman, int archers,
        int cavalry, int knights, List<Report> reports) {
        player_name = name;
        player_wood = wood;
        player_stone = stone;
        player_iron = iron;
        player_food = food;
        player_population = population;
        player_spears = spears;
        player_swordsmans = swordsman;
        player_archers = archers;
        player_cavalrys = cavalry;
        player_knights = knights;
        player_reports = reports;
    }

    public bool EnoughSpaceForPeople(int pop) {
        int diff = player_population.storageCapacity -
            (int)player_population.currentAmount;
        if (diff > pop)
            return true;
        return false;
    }
```

```
    public bool IsEnoughOfResource(int wood, int stone, int iron, int food, int
population) {
        if (wood <= player_wood.currentAmount && stone <= player_stone.currentAmount
            && iron <= player_iron.currentAmount && food <= player_food.currentAmount
            && EnoughSpaceForPeople(population)) {
            return true;
        }
        return false;
    }

    public Effect FindEffect(int id) {
        if (effects != null) {
            foreach (Effect eff in effects) {
                if (eff.id == id)
                    return eff;
            }
        }
        return null;
    }

    public Technology FindTechnology(int id) {
        if (improvement_techs != null && military_techs != null) {
            foreach (Technology tech in improvement_techs) {
                if (tech.tech_ID == id)
                    return tech;
            }
            foreach (Technology tech in military_techs) {
                if (tech.tech_ID == id)
                    return tech;
            }
        }
        return null;
    }

    public Building FindBuilding(int id) {
        foreach (Building building in buildings) {
            if (building.id == id) {
                return building;
            }
        }
        return null;
    }

    public bool isConstuctionQueueFull() {
        if (constuctionQueue.Count >= 2) {
            return true;
        }
        return false;
    }
}
```

Příloha D: Skript zajišťující pohyb a zoom kamery

```
using UnityEngine;
using System;
using UnityEngine.EventSystems;
using System.Collections.Generic;
using UnityEngine.UI;

public class CameraMotion : MonoBehaviour{

    public float moveSpeed = 20;
    public float maxAcceleration = 5;
    public float AccelerationStep = 0.1f;
    private float horizontalMove = 0;
    private float verticalMove = 0;
    private float acceleration = 1;
    private short cameraZoom1 = 120;
    private short cameraZoom2 = 150;
    private short cameraZoom3 = 200;
    public short currentZoom = 150;
    public bool ZommEnabled = false;
    public bool enabledMouseCameraMove = false;
    public float MouseDetectDistance = 0.01f;
    Bounds bounds;
    private GraphicRaycaster gr;
    private Animator cameraAnimator;

    private void Start() {
        cameraAnimator = GetComponentInChildren<Animator>();
        bounds = new Bounds(new Vector3(250, 150, 250), new Vector3(500, 1, 500));
        gr = GameObject.Find("Canvas_ingame").GetComponent<GraphicRaycaster>();
    }

    public Vector3 CheckNewPosition(float x, float y, float z) {
        Vector3 newPosition = transform.position + new Vector3(x, y, z);
        if (bounds.Contains(newPosition)) {
            return new Vector3(x,y,z);
        }else if(bounds.Contains(new Vector3(transform.position.x +
x,transform.position.y,transform.position.z))){
            return new Vector3(x, 0, 0);
        }else if(bounds.Contains(new Vector3(transform.position.x,
transform.position.y, transform.position.z + z))) {
            return new Vector3(0, 0, z);
        }
        return new Vector3();
    }

    public bool NoUIOnPointerPosition() {
        PointerEventData ped = new PointerEventData(null);
        ped.position = Input.mousePosition;
        List<RaycastResult> results = new List<RaycastResult>();
        gr.Raycast(ped, results);
        if (results.Count == 0) {
            return true;
        } else {
            foreach (RaycastResult rr in results) {
                if (rr.gameObject.name == "Building_info(Clone)") {
                    return true;
                }
            }
        }
        return false;
```

```
    }

    void Update() {
        if (Math.Abs(Input.GetAxis("Horizontal")) > 0 ||
        Math.Abs(Input.GetAxis("Vertical")) > 0) {
            if (acceleration <= maxAcceleration)
                acceleration += AccelerationStep;
            horizontalMove = Input.GetAxis("Horizontal") * moveSpeed * acceleration *
                Time.deltaTime;
            verticalMove = Input.GetAxis("Vertical") * moveSpeed * acceleration *
                Time.deltaTime;
            transform.Translate(CheckNewPosition(horizontalMove, 0, verticalMove));
        } else {
            if (acceleration != 1)
                acceleration = 1;
        }

        if (enabledMouseCameraMove) {
            if (Input.mousePosition.x >= Screen.width * (1 - MouseDetectDistance)) {
                horizontalMove = 2 * moveSpeed * acceleration * Time.deltaTime;
                transform.Translate(CheckNewPosition(horizontalMove, 0, 0));
            } else if (Input.mousePosition.y >= Screen.height *
                (1 - MouseDetectDistance)) {
                verticalMove = 2 * moveSpeed * acceleration * Time.deltaTime;
                transform.Translate(CheckNewPosition(0, 0, verticalMove));
            } else if (Input.mousePosition.x <= Screen.width * MouseDetectDistance) {
                horizontalMove = 2 * moveSpeed * acceleration * Time.deltaTime;
                transform.Translate(CheckNewPosition(-horizontalMove, 0, 0));
            } else if (Input.mousePosition.y <= Screen.height * MouseDetectDistance) {
                verticalMove = 2 * moveSpeed * acceleration * Time.deltaTime;
                transform.Translate(CheckNewPosition(0, 0, -verticalMove));
            }
        }

        if (ZommEnabled) {
            if (NoUIOnPointerPosition()) {
                if (Input.GetAxis("Mouse ScrollWheel") > 0) {
                    if (currentZoom == cameraZoom3) {
                        cameraAnimator.SetTrigger("GoMedium");
                        currentZoom = cameraZoom2;
                    } else if (currentZoom == cameraZoom2) {
                        cameraAnimator.SetTrigger("GoLow");
                        currentZoom = cameraZoom1;
                    }
                } else if (Input.GetAxis("Mouse ScrollWheel") < 0) {
                    if (currentZoom == cameraZoom1) {
                        cameraAnimator.SetTrigger("GoMedium");
                        currentZoom = cameraZoom2;
                    } else if (currentZoom == cameraZoom2) {
                        cameraAnimator.SetTrigger("GoHigh");
                        currentZoom = cameraZoom3;
                    }
                }
            }
        }
    }
}
```

Příloha E: Skript s názvem Building_script

```
using UnityEngine;
using UnityEngine.EventSystems;
using UnityEngine.UI;
using System.Collections.Generic;

public class Building_script : MonoBehaviour {

    public GameObject buildingWindowPrefab;
    private GameObject buildingInfoPrefab;
    private GameObject openedWindow = null;
    private GameObject openedInfo = null;
    private Transform WindowParrent;

    public int buildingId;
    private Building building;
    private GraphicRaycaster gr;

    private void Start() {
        buildingInfoPrefab = Resources.Load<GameObject>("Building_info");
        WindowParrent = GameObject.Find("Dynamics").transform;
        building = GameObject.Find("Manager").GetComponent<PlayersControl>().
            player.FindBuilding(buildingId);
        gr = GameObject.Find("Canvas_ingame").GetComponent<GraphicRaycaster>();
        if (buildingWindowPrefab == null) {
            buildingWindowPrefab =
                Resources.Load<GameObject>("BuildingWindow_default");
        }
    }

    public bool OverThisObject() {
        PointerEventData ped = new PointerEventData(null);
        ped.position = Input.mousePosition;
        List<RaycastResult> results = new List<RaycastResult>();
        gr.Raycast(ped, results);
        if (results.Count == 0) {
            return true;
        } else {
            foreach (RaycastResult rr in results) {
                if (rr.gameObject.name == "Building_info(Clone)") {
                    return true;
                }
            }
        }
        return false;
    }

    private void OnMouseEnter() {
        if (OverThisObject()) {
            OpenInfo();
        }
    }

    private void OnMouseOver() {
        if (OverThisObject()) {
                if (openedInfo != null) {
                    if (Input.mousePosition != openedInfo.transform.position)
                        SetPositionOfInfoWindow();
                } else {
                    OpenInfo();
                }
```

```
        } else {
            if (openedInfo != null) {
                Destroy(openedInfo);
                openedInfo = null;
            }
        }
    }

    private void OnMouseExit() {
        Destroy(openedInfo);
        openedInfo = null;
    }

    private void OnMouseDown() {
        Destroy(openedInfo);
        if (OverThisObject()) {
            if (openedWindow == null) {
                OpenBuildingWindow();
            }
        }
    }

    public void OpenInfo() {
        if (openedInfo != null)
            Destroy(openedInfo);
        openedInfo = Instantiate(buildingInfoPrefab) as GameObject;
        openedInfo.transform.SetParent(WindowParrent, false);
        openedInfo.GetComponent<BuildingInfoWindow>().FillTexts(building.name,
            building.currentLevel.ToString());
        SetPositionOfInfoWindow();
    }

    public void SetPositionOfInfoWindow() {
        var windowScale = openedInfo.GetComponent<RectTransform>().lossyScale;
        var position = Input.mousePosition;
        var windowSize = openedInfo.GetComponent<RectTransform>().sizeDelta.x *
            windowScale.x;
        if ((Screen.width - windowSize) > position.x) {
            openedInfo.transform.position = position;
        } else {
            openedInfo.transform.position = new Vector3(position.x - windowSize,
                position.y, position.z);
        }
    }

    public void OpenBuildingWindow() {
        openedWindow = (Instantiate(buildingWindowPrefab) as GameObject);
        openedWindow.transform.SetParent(WindowParrent, false);
        openedWindow.GetComponent<BuildingWindowScript>().SetBuilding(this,building);
    }

    public void CloseBuildingWindow() {
        Destroy(openedWindow);
        openedWindow = null;
    }
}
```

Příloha F: CD disk

Nedílnou součástí práce je disk cd, který obsahuje:

- vlastní text
- assety použité při tvorbě práce ve složce Assets.
- výslednou sestavenou hru pro systémy Windows, která se nachází ve složce Release se spustitelným souborem s koncovkou exe.