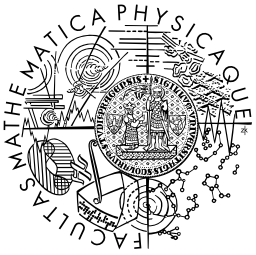


MATEMATICKO-FYZIKÁLNÍ FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Kateřina Červinková

Real-time strategie s rozhraním pro umělou inteligenci

Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: Mgr. Martin Pilát, Ph.D.

Studijní program: Informatika

Studijní obor: Programování a softwarové systémy

Praha 2019

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ..... Podpis autora

Poděkování.

Na tomto místě bych ráda poděkovala svému vedoucímu práce, Mgr. Martinu Pilátovi, Ph.D., za čas, ochotu, trpělivost a cenné rady, bez nichž by tato práce nemohla vzniknout. Poděkování také patří mé rodině za podporu během celé doby studia.

Název práce: Real-time strategie s rozhraním pro umělou inteligenci

Autor: Kateřina Červinková

Katedra: Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: Mgr. Martin Pilát, Ph.D., Katedra teoretické informatiky a matematické logiky

Abstrakt:

Tato práce se zabývá real-time strategií Skillegy, která využívá na rozdíl od většiny her podobného typu pouze jeden druh jednotky. Tyto jednotky však mají určité schopnosti, jejichž úrovně se mohou během hry zvyšovat v závislosti na jejich akcích či pomocí vylepšování v budovách. Hru lze hrát ve více hráčích přes síť a obsahuje rozhraní pro umělou inteligenci s ukázkovou implementací, kterou lze použít místo lidského protivníka.

Hra je vytvořena v enginu Unity pomocí jazyka C# a platformy .NET.

Klíčová slova: real-time strategie; umělá inteligence; herní inteligence

Title: Real-time strategy with an interface for artificial intelligence

Author: Kateřina Červinková

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: Mgr. Martin Pilát, Ph.D., Department of Theoretical Computer Science and Mathematical Logic

Abstract: This thesis focuses on a real-time strategy called Skillegy, which, in contrast to the majority of games of similar kind, uses only one type of unit. However, these units have certain abilities whose levels can increase depending on their actions or using upgrading in buildings. The game can be played by multiple players over network and it includes an interface for an artificial intelligence with example implementation that can be used instead of a human opponent.

The game is created in the Unity engine with use of the C# language and the .NET Framework.

Keywords: real-time strategy; artificial intelligence; game intelligence

Obsah

Úvod

Real-time strategie (RTS) je žánr počítačových her, ve kterém hráči mohou uplatnit své strategické a taktické myšlení. Jejich úkolem je převzít kontrolu nad rozvíjející se skupinou lidí a dosáhnout s nimi určitého cíle, jenž je zpravidla založen na válečném konfliktu s ostatními hráči.

V naprosté většině her existuje více typů různorodých jednotek specializovaných pro různé úkony, což mohou být například sbírání surovin, střílení z luku nebo boj s mečem. Tato jednotka ale pak nemůže dělat nic jiného, než k čemu je určena. To se může zdát trochu nerealistické, protože ve skutečné válce by pravděpodobně každý člověk mohl dělat v případě nutnosti cokoli, i když s různě dobrým výsledkem.

Z tohoto důvodu je cílem této práce zkusit trochu jiný koncept a vyvinout hru Skillegy, která splňuje základní principy RTS her, ale existuje v ní pouze jeden typ jednotky. Tato jednotka může dělat cokoli, co hráč potřebuje, ale efektivita této činnosti bude záviset na schopnostech, jimiž je jednotka vybavena. Tyto schopnosti se budou vyvíjet v závislosti na zkušenostech jednotky a některé z nich bude možno přímo trénovat.

Práce je rozdělena do šesti kapitol. V první z nich si blíže představíme žánr RTS a jeho historii a vlastnosti, ve druhé popíšeme, co od hry Skillegy očekáváme. Třetí kapitola se zabývá porovnáním různých principů ve známých real-time strategiích a ve Skillegy a ve čtvrté a páté kapitole najdeme uživatelskou a vývojovou dokumentaci. Tématem poslední kapitoly je umělá inteligence a rozhraní pro ni.

1. Real-Time strategie

Jak už název napovídá, real-time strategie jsou podžánr strategických her. Ve strategické hře musí hráč získávat a spravovat různé zdroje, které pak využívá k dosažení jednoho či více cílů. Sám není postavou hry, pouze svou skupinu postav řídí a dává jim pokyny.

Druhá část názvu, real-time, značí, že příkazy vydané hráčem se splní v reálném čase, tj. co nejdříve to bude možné. Příkladem prodlení může být, že jednotka musí dojít až k surovině, aby ji mohla začít těžit, nebo že budova může začít další vylepšení až po dokončení předchozího. Často jsou s RTS hrami srovnávány tahové strategie, kde hráči zadají příkazy svým jednotkám a pak se vše najednou vykoná. Velmi známým příkladem takové hry může být třeba *Civilizace* [1].

Pokud bychom ale jen sloučili tyto definice, RTS hrou by byly i například *Roller Coaster Tycoon* [2] nebo *The Sims* [3]. V obou těchto hrách je potřeba spravovat své zdroje – peníze – a dosáhnout s jejich pomocí daného cíle, a obě tyto hry se odehrávají v reálném čase. Ani jedna z nich ale není real-time strategií, protože postrádají vojenský prvek, který je přítomen v každé klasické real-time strategii.

S ohledem na toto můžeme nyní vytvořit definici, která by již měla být dostatečná. Real-time strategie je hra, v níž se příkazy hráče odehrávají v reálném čase, a ve které musí hráč využívat zdroje k vybudování armády, jež mu pak poslouží při splňování vítězných podmínek.

V dalších částech této kapitoly se budeme věnovat historii a příklady RTS a vymezíme klíčové komponenty, které by se v takových hrách měly v té či oné podobě objevovat.

1.1 Historie real-time strategií

Pojem real-time strategie pochází až z 90. let 20. století, existují ale i sporné dohady, že vůbec první real-time strategií je *Utopia* z roku 1981, jejímž autorem je Don Daglow [4]. Utopia byla především tahovou strategií, ale jako první ze strategických her obsahovala i real-time prvky. V této hře proti sobě stáli dva hráči, z nichž každý spravoval svůj ostrov, vytvářel si základnu a snažil se porazit protivníka.

Dalším kandidátem na prapředka RTS žánru je *Herzog Zwei*, japonská hra z roku 1989, která se ale od klasických RTS liší v tom, že hráč ovládá pouze jednu jednotku [5].

Přestože nepanuje shoda v tom, co bylo první real-time strategií, nepopiratelný vliv na vývoj tohoto žánru měla hra z roku 1992 *Dune II*. I když dosud nebylo možné vybrat více jednotek najednou nebo jim zadat příkazy pouhým stisknutím pravého tlačítka myši, ovládání bylo podstatně modernější a objevily se i klasické elementy jako mapa začerněna až do doby, než ji hráč objeví [6].

Následovalo vydávání velkého počtu real-time strategií, z nichž za zmínku stojí například *Command & Conquer* [7], *Warcraft* [8] nebo *Total Annihilation* [9].

Pravděpodobně nejznámějšími zástupci žánru RTS jsou ale série *StarCraft* [10] z roku 1998 s fantasy tématem a *Age Of Empires* [11] z roku 1997 s historickým tématem. Do doby *StarCraftu* měli soupeři vždy stejné možnosti vylepšení, stavby

jednotek a budov, *StarCraft* však přinesl tři naprosto odlišné frakce, za které se dalo hrát, Terrany, Protossy a Zergy. Každý z národů měl své unikátní vlastnosti a vzniklá asymetrie způsobila vítané zpestření hry. *Age of Empires* klade velký důraz na vylepšování a vědecký pokrok, je také první real-time strategií, v níž se objevil princip technologických věků. Do těchto věků lze postoupit vylepšeními v hlavních budovách nebo například vytvořením určité stavby a odemykají nové budovy, jednotky i vylepšení.

V této době už to bohužel vypadá, že žánr RTS má nejlepší léta za sebou, a to hlavně kvůli stoupajícím požadavkům hráčů, kterým hry buď nestíhají vyhovět, nebo tato snaha vyústí ve hru, jež je pro většinu lidí příliš těžká na naučení a tudíž nepřístupná [12].

Hry ze sérií *StarCraft* a *Age Of Empires* se ale hrají dodnes a hlavně ve *StarCraftu II* se konají slavné turnaje především v Jižní Koreji, dokonce je jednou z nejpodporovanějších her zařazených v e-sportech [13].

1.2 Vlastnosti real-time strategií

Jak již bylo uvedeno na začátku kapitoly, existuje několik znaků, které real-time strategie musí splňovat, aby dostála svému jménu. Tyto znaky jsou pro přehlednost uvedeny i zde společně s nepovinnými vlastnostmi, z nichž některé se mohou zdát pro pravou RTS hru také téměř klíčovými. Každé z vlastností se pak budeme zvlášť věnovat v kapitole Analýza, abychom zjistili, jaké možné implementace existují a jak - a jestli vůbec - je možné tuto vlastnost začlenit do Skillegy.

- hra se odehrává v reálném čase, každý příkaz se začne provádět ihned, jak jej hráč zadá nebo co nejdříve to bude možné
- cílem hry může být cokoli od nejčastějšího zabití všech soupeřových jednotek a zničení všech jeho budov po vytvoření drahé budovy, jejíž stavba trvá velmi dlouho (tento princip se vyskytuje například v určitém módu *Age of Empires II*)
- na hrací ploše neboli mapě se nachází zdroje (například stromy, zvířata, ve vesmírných RTS různé minerály a plyny), eventuálně neutrální jednotky, které můžou interagovat s hráčem a případně mu pomoci či uškodit, nebo speciální předměty, jež přidávají hráči bonus, nebo jejich vlastnictví může být jednou z podmínek pro vítězství (příkladem jsou relikvie v *Age of Empires II*)
- hráč potřebuje suroviny, aby je mohl použít ke stavbě budov, trénovat jednotky a technologicky se rozvíjet, což mu může dopomoci k dosažení vítězství
- hráči je poskytnuta možnost tyto suroviny získávat (nejčastěji je přímo těžit)
- hráč si buduje armádu, aby porazil protivníka, a to i pokud zničení soupeře není jediným či primárním cílem hry

- armáda se skládá z různých jednotek, mohou mít různé přednosti a omezení
- budovy jsou důležité pro vytváření jednotek i technologický rozvoj, různé budovy mají typicky různé účely, mohou sloužit i jako obrana (toto platí především pro zdi, ale i jiné budovy se dají rozmístit strategicky) nebo pozorovatelna přilehlých částí mapy
- hráč si staví svou základnu či město, nejčastěji blízko sebe, aby měl většinu svých budov i jednotek snadno dostupnou a dobře chráněnou
- jednotky i budovy se postupem času vylepšují, ať už individuálně prováděním určitých činností nebo globálně vylepšením jednoho typu jednotky či budovy
- ve hře je přítomna tzv. válečná mlha neboli *fog of war*, čili schopnost hráčů vidět jen ty části mapy, poblíž kterých mají jednotku či budovu – zbytek mapy je zakryt černým kouřem, pokud tam hráčova budova ani jednotka nikdy nebyly, v opačném případě hráč vidí jen budovy a zdroje, které se tam nacházely v době, kdy tudy procházel

Mimo tyto vlastnosti, které jsou důležité během hry, však stojí za zmínku také různé varianty hry, jež hru činí zajímavou a poutavou i v delším časovém horizontu.

- možnost hrát hru ve více hráčích, ať už na serveru nebo přes LAN
- interakce mezi lidskými protihráči
- varianta hry proti umělé inteligenci, buď přes scénářové kampaně nebo s použitím umělé inteligence jen jako náhrady živého protivníka v klasické bitvě
- hra v týmech nebo všichni proti všem – při hře v týmech je v některých hrách možnost měnit týmy dynamicky během hry, utvářet tak spojenectví a zrazovat své přátele
- výběr z několika map nebo dokonce možnost vytvořit si svou vlastní
- výběr zajímavějších vítězných podmínek

2. Popis hry

V této kapitole si pro přehlednost popíšeme výslednou hru jako celek, aby byly v dalších oddílech práce jasné souvislosti a bylo vidět, jak jednotlivé části hry zapadají do celkového obrazu.

Hráči se ve Skillegy nacházejí na mapě, na které existují pouze oni a suroviny, jež mohou těžit. S pomocí těchto surovin si mohou postavit budovy, jednotky a vylepšovat technologie, to všechno s cílem vybudovat si ekonomiku a armádní složky dříve a lépe než soupeři, aby je pak mohli porazit. Hráč je zničen ve chvíli, kdy mu nezbývá žádná jednotka.

2.1 Suroviny

Na mapě se nacházejí tři typy surovin, keře s jídlem, stromy a zlaté doly, z nichž mohou hráči získávat jídlo, dřevo a zlato. Je také možné je získat v budovách k tomu určených, proces však bude výrazně pomalejší.

2.2 Jednotky a jejich schopnosti

Důležitým konceptem této hry jsou schopnosti jednotek a jejich vývoj. Jednotka se vytváří v hlavní budově, v tabulce 2 je uvedena její cena. Každá jednotka má pět schopností, a to těžbu, inteligenci, šermířství, léčení a stavitelství. Tyto schopnosti jsou jí při jejím stvoření přiděleny náhodně, ale tak, aby měly všechny jednotky na začátku přibližně stejný součet schopností.

2.2.1 Těžba

Schopnost těžby se uplatní při získávání surovin, ať už ze surovin nebo v budovách. Při této činnosti se také vylepšuje schopnost, která však roste rychleji při přímé těžbě surovin než v budově.

2.2.2 Inteligence

Pravděpodobně nejdůležitější schopností je inteligence, protože právě od ní se odvíjí, jak rychle se budou zlepšovat všechny ostatní. Typickým scénářem je, že schopnost se vyvíjí rychlostí vynásobenou inteligencí jednotky, ale pokud daná schopnost přesáhne inteligenci, tato rychlost se několikrát zmenší. Sama inteligence se vyvíjí lineárně ve specializované knihovně, která bude popsána níže.

2.2.3 Šermířství

Šermířství se uplatní především v bitvách, protože jednotky s lepší způsobilostí k šermířství mají větší útočnou sílu a na jednu ránu uberou soupeři více zdraví. Při boji se však šermířství nezlepšuje, jeho úroveň lze zvednout pouze v dále popsaných kasárnách.

2.2.4 Léčení

Jednotky přijdou v průběhu bitev často ke zraněním a hlavně ty nejvíce vylepšené si hráč pravděpodobně nebude přát ztratit. Z tohoto důvodu je důležité mít i pár jednotek s dobrou schopností léčení, protože tato jednotka bude moci ve specializované budově přidávat body zdraví zraněným jednotkám. Rychlost tohoto léčení závisí především na léčitelově úrovni. Léčení lze zlepšovat v knihovně studiem, jak ještě bude zmíněno, i při léčení zraněných spolubojovníků.

2.2.5 Stavitelství

Dalším úkolem jednotky je postarat se o stavbu budovy. Stavba každé budovy trvá jinak dlouho, ale rychlost stavění závisí i na úrovni stavitelství. Tato úroveň se také dá zlepšovat studiem v knihovně nebo přímo stavbou budov.

2.3 Budovy

Pro rozvoj základny hráče jsou naprosto klíčové budovy všech druhů. Ve Skillegy existuje osm typů budov, které mají různé účely, například zvedají populační limit, slouží jako útočiště pro slabé jednotky, vylepšují jim dovednosti, léčí je a podobně. Tabulka 2 ukazuje ceny všech budov. Také je v ní ukázán věk, jehož musí hráč dosáhnout, aby danou budovu mohl postavit, což je koncept, který bude představen hned v následující sekci. Každá budova je unikátní, proto si je teď popíšeme podrobněji.

2.3.1 Hlavní budova

Hlavní budova je pravděpodobně první budovou, kterou si hráč postaví, protože jako jediná umí vyrábět jednotky, na nichž pak závisí celý další postup. Hlavní budova stojí tisíc jednotek od každého druhu suroviny a je tak v součtu nejdražší budovou ve hře. Jednotky, které produkuje, zabírají každá jednu populaci, a pokud již bylo dosaženo populačního limitu, není možné jednotku vytvořit. Přestože ke zvedání populačního limitu slouží hlavně domy, i sama hlavní budova tento limit zvyšuje o pět.

Přestože s jednotkami uvnitř nijak neinteraguje, může tato budova ubytovat až sto jednotek, a to hlavně proto, aby bylo možné chránit slabší jednotky.

V hlavní budově také probíhají nejdůležitější technologická vylepšení, postup do dalších věků. Na těchto věcích závisejí některé další vylepšení i stavby určitých budov. Hráč začíná v dřevěném věku, poté postupuje do věku kamenného, železného a diamantového. Tento technologický pokrok může však být pro hráče velmi nákladný. Tabulka 3 ukazuje ceny dalších věků.

2.3.2 Dům

Jediným účelem domu je zvyšování populačního limitu, který se s každým postaveným domem zvedne o deset jednotek. V domě se také může schovat jedna jednotka, ale je to spíše vlastnost, která se může hodit jen výjimečně, hlavně proto, že dům s jednotkou uvnitř neprovádí žádnou akci. Dům neposkytuje hráči ani žádnou příležitost k nákupům.

2.3.3 Kasárny

Jak již bylo zmíněno, v kasárnách se jednotky trénují ve svých šermířských schopnostech. Tato budova je dostupná až od kamenného věku a může v ní najednou přebývat pouze pět jednotek. Pokud jich chce hráč trénovat více, musí si sehnat suroviny a postavit více kasáren.

V každé této budově lze však plnohodnotně trénovat jednotky pouze do určité úrovně. Pokud jednotka tuto úroveň přesáhne, rychlost jejího zlepšování schopnosti se několikanásobně sníží. Ke zvýšení maximálního šermířství slouží vylepšení, která si lze pořídit v kasárnách. Těchto vylepšení existuje pět a každé z nich zvýší maximální úroveň šermířství o další stupeň. Tabulka 4 ukazuje ceny vylepšení, podmínky, jež musí být splněny, aby bylo vylepšení dostupné, a úroveň šermířství, na niž toto vylepšení pozvedne maximální šermířství v kasárnách.

2.3.4 Ošetřovna

Ošetřovna je dostupná od železného věku a slouží k léčení zraněných jednotek uvnitř budovy. Hráč může jednu jednotku ustanovit jako doktora a od jeho úrovně schopnosti léčení se pak bude odvíjet rychlost přibývání zdraví. Samotnému lékaři se bude postupně zlepšovat schopnost léčení v závislosti na jeho inteligenci. Rychlost léčení je však ovlivňována i počtem pacientů.

V ošetřovně může najednou být až dvacet jednotek, ale lékař jich zvládá pouze určitý počet, ze začátku jen tři. Pokud je jich v ošetřovně více, rychlost lékaře se bude zpomalovat. Efektivitu lékaře lze zvýšit vylepšením v knihovně, může potom zvládat až 15 pacientů.

Ošetřovna může fungovat i bez lékaře, což může být dokonce i rychlejší než s ním, pokud je velký počet pacientů nebo pokud má lékař špatnou schopnost léčení.

2.3.5 Knihovna

Knihovna již zde byla několikrát zmíněna, protože slouží k vylepšování především inteligence, ale i léčení a stavitelství. Je dostupná od železného věku a může se v ní najednou vylepšovat až pět jednotek. Ihned po postavení knihovny je dostupné zlepšování inteligence, která se zvyšuje lineárně vždy až do určité hranice, nad níž je její stoupání výrazně pomalejší. Tuto hranici lze zvyšovat přímo v knihovně pomocí vylepšení.

Další vylepšení slouží k odemčení zaměření knihovny na stavění a lékařství. Pomocí nákupu je pak možné přepnout zaměření knihovny a zdokonalovat tak v knihovně stavitelství nebo lékařství místo inteligence. Pokud jsou však úrovně těchto vlastností vyšší než inteligence, jejich zvyšování půjde výrazně pomaleji. Všechna vylepšení v knihovně je možno najít v tabulce 5.

2.3.6 Mlýn, pila a banka

Suroviny se sice nacházejí na mapě, je jich tam však pouze určité množství, a proto je nutná existence budov, které tyto suroviny produkují neomezeně. Ve mlýně, v pile a v bance tedy může být až pět jednotek, které produkují jídlo, dřevo nebo zlato. Tato produkce je pomalejší než u opravdových surovin,

u mlýna je však možné rychlost zvýšit pomocí vylepšení (viz tabulka 6), z nichž každé produkci dvakrát zrychlí.

2.4 Průběh hry

Na začátku hry se hráči objeví na různých místech mapy. Mají k dispozici jednu jednotku, 1200 jídla, 1200 dřeva a 1200 zlata, což je pouze o trochu víc, než potřebují na stavbu hlavní budovy. První jednotka všech hráčů má vždy úroveň stavitelství sedm, protože rychlost postavení první budovy výrazně ovlivní začátek hry a bylo by nespravedlivé, kdyby ji jeden hráč měl výrazně dřív nebo později než druhý.

Na začátku hry vidí hráči pouze svou část mapy, postupně jsou ale z různých důvodů nuceni prozkoumávat, hlavně kvůli surovinám nebo protože chtějí najít nepřátelskou základnu.

Během celého trvání hry mohou hráči stavět jednotky a budovy, shánět suroviny a kupovat vylepšení. Toto pokračuje až než se jeden z hráčů rozhodne zaútočit. V této fázi bude velice důležité uvážit, kolik jednotek poslat do boje, protože s menším počtem je také menší pravděpodobnost úspěšného útoku. Pokud však nevyjde větší útok, hráč může přijít také o své nejcennější a nejvytrénovanější jednotky. Tento systém je pro hráče komplikovanější, ale nabízí více možných strategií než jiné real-time strategie.

Hra končí ve chvíli, kdy už zůstává pouze jeden hráč, jehož jednotky jsou dosud naživu, nejčastěji v důsledku lepší ekonomiky a taktiky nebo jen prostého štěstí.

3. Analýza problému

V kapitole o real-time strategiích jsme si ujasnili, jaké vlastnosti by mohla hra Skillegy mít. Než se však přesuneme k jejich rozboru, je potřeba učinit několik rozhodnutí poněkud techničtějšího rázu. Poté se vrátíme k důležitým prvkům real-time strategií a u každého z nich budeme analyzovat, jak se k němu postavili tvůrci úspěšných real-time strategií a jak se to liší od přístupu implementovaného ve Skillegy.

3.1 Výběr programovacího jazyka a herního enginu

Protože pro vývoj her existuje spousta různých herních enginů, omezila jsem si výběr na ty, které umí pracovat s 3D, protože ve 3D je naprostá většina moderních real-time strategií, kterými jsem se při návrhu inspirovala.

Z herních enginů s 3D grafikou jsou pravděpodobně nejznámějšími, nejpoužívanějšími a pro real-time strategie nejdoporučovanějšími *Unreal Engine* [14], *Unity3D* [15] a *Godot* [16]. Pro všechny tyto enginy existují neplacené verze, *Godot* je open-source, *Unreal Engine* zase nabízí vývojářskou licenci zdarma včetně zdrojových kódů. *Unity3D* je známý svou rozsáhlou podporou nejrůznějších platforem, další dva enginy jsou v tomto směru poněkud omezenější. Podpora dalších platforem však není naším primárním cílem, i když by se mohla hodit pro případné rozšíření hry v budoucnosti. Velmi vychvalovaná grafika *Unreal Enginu* také není pro Skillegy klíčová, protože objekty budou stejně primitivní a nebudou vypadat realisticky.

Unreal Engine podporuje C++, primárními jazyky *Unity3D* jsou C# a JavaScript. *Godot* nabízí nejvíce možností, lze v něm programovat v C#, C++ i GDScriptu, což je speciální jazyk podobný Pythonu.

Přestože *Godot* vypadá velmi lákavě, jedná se o relativně nový engine z roku 2014, který není tak známý jako další dva. Z tohoto důvodu je jeho dokumentace slabší [17] a také lze najít méně otázek, odpovědí a návodů, což může být během vývoje tíživý problém.

Unreal Engine je velice úspěšný a používaný, ale i těžší na naučení, a to z větší části kvůli C++. To by se mohlo stát obtížným i z důvodu, že hra by měla obsahovat i rozhraní pro umělou inteligenci. Toto rozhraní je primárně určeno pro pokročilejší uživatele, jež si ji chtějí doprogramovat sami, a proto by měl i programovací jazyk být klientsky příjemný.

I když se objevují různé názory, po přečtení řady doporučení se zdá, že real-time strategii lze napsat v kterémkoli z těchto enginů a záleží z velké části na osobních preferencích [18]. Hlavně díky velké oblíbenosti enginu, což usnadní hledání tutoriálů a návodů, možnosti psaní v jazyce C# a relativní nenáročnosti pro nového uživatele jsem nakonec vybrala engine *Unity3D*, i navzdory tomu, že zdrojové kódy jsou volně přístupné pouze ke čtení, ne k úpravám [19]. Ve všech dalších rozhodnutích už tedy můžu hledat jen varianty relevantní pro *Unity3D*.

3.2 Hra ve více hráčích

V podstatě všechny moderní real-time strategie lze hrát ve více hráčích. Rozdělená obrazovka, z níž každý hráč používá jen část, v tomto případě ale nedává smysl. To nás dovádí k síťové komunikaci, kde existují dvě možnosti – client-server a peer-to-peer.

3.2.1 Peer-to-peer vs client-server

V případě modelu peer-to-peer sdílí hráči všechny informace mezi sebou a nemají žádnou centrální autoritu. Výhodou je, že odpojení jakéhokoli klienta proběhne bez problémů a hra může dále pokračovat. Synchronizace stavů hry je však složitější a může se stát, že se na jednom klientovi dějí jiné věci než na druhém. Další nevýhodou je, že pokud má jeden z klientů pomalé internetové připojení, trpí tím všichni jeho spoluhráči. Příkladem hry založené na peer-to-peer připojení je *Age of Empires II*. [20].

Pokud bych vybrala model client-server, hra by běžela na centrální autoritě – serveru – a připojení hráči by jí posílaly příkazy, které by se prováděly na serveru. Podle serveru by si také všichni hráči aktualizovali hru, nemůže tak dojít k desynchronizaci stavů na klientech. Centrální autorita je však pro tento postup klíčová a její odpojení vede k přerušení hry. Nevýhodou jsou i větší požadavky na šířku pásma, což ale býval problém spíše v minulosti. Client-server model používá například *StarCraft II* [21].

3.2.2 Multiplayerové možnosti pro *Unity*

Nejpoužívanějšími multiplayerovými řešeními pro *Unity* jsou *Photon Bolt* [22], *Forge* [23] a *UNet* [24]. Bohužel, *Photon Bolt* i *Forge* jsou placené, proto je ve Skillegy použit *UNet*. Je sice zastaralý a bude v blízké budoucnosti odstraněn, avšak nové řešení v době psaní této práce není ještě k dispozici.

UNet nabízí tzv. *High Level API* a *Low Level API*. *Low Level API* umožňuje vytváření vlastních síťových systémů podle specifických požadavků, to však pro naši hru nebudeme potřebovat. Místo toho jsem tedy použila *High Level API*, které je jednodušší. Jeden z hráčů bude tzv. host, čili server i klient zároveň, a síťová komunikace bude probíhat pomocí příkazů, jež posílá klient serveru nebo server klientovi.

3.3 Pathfinding

Jedním z klíčových elementů v real-time strategii je schopnost jednotek vyhledávat trasu z místa na místo a pak se podle ní řídit. Pro real-time strategie se nejčastěji používá algoritmus A*, který bude popsán dále ve vývojové dokumentaci. I ve Skillegy je tedy tento algoritmus použit. Byla samozřejmě možnost si ho celý naimplementovat, ale existující knihovny jsou pravděpodobně lépe optimalizované a obsahují i pokročilejší funkce, které se mohou v průběhu vývoje hodit.

Přestože terén ve hře bude plochý bez jakéhokoli stoupání či klesání, knihovny pro 2D hry nestačí. Bezplatnými možnostmi pro 3D pathfinding jsou zabudovaný

navigační systém v Unity [25] a *A* Pathfinding Project* [26]. Původním řešením byl navigační systém v Unity, ten se však ukázal nedostatečným, protože nespolupracuje dobře se síťovou komunikací. V další kapitole bude tento problém rozepsán podrobněji. Z tohoto důvodu je nakonec ve hře využit *A* Pathfinding Project*.

3.4 Hra v reálném čase

Jak již bylo zmíněno dříve, všechny příkazy se budou vykonávat bezprostředně po jejich vydání. Podle jejich povahy se rozhodne, zda se provedou na serveru, a tím na všech klientech (tak se jejich účinek stane globálním) nebo pouze na klientovi, kde se o něm ostatní hráči nedozví. Příklad takové lokální akce může být třeba označení jednotky. Tento úkon je pro ostatní hráče nepodstatný a není třeba, aby jej viděli.

3.5 Cíl hry

Cílem hry bývá v tradičních real-time strategiích zničit protivníka pomocí eliminování všech jeho jednotek a budov, které mohou tyto jednotky produkovat. Ve Skillegy ukážeme pouze jednoduchý cíl, a to zničení všech jednotek, což zřejmě ve výsledku nebude velký rozdíl. Důležitá je však možnost tento cíl jednoduše změnit se zásahem do co nejméně částí kódu. Bude tak možné například snadno doplnit vítězné podmínky, například stavba velmi drahého divu světa, posbírání předmětů z mapy dříve než soupeř nebo podrobení si určitého území a jeho využívání po předem daný časový úsek.

3.6 Předměty na mapě

Pro fungování hry jsou nezbytné pouze suroviny, nebudu tedy implementovat žádné další speciální předměty. Je však žádoucí počítat s tím, že by se časem mohly přidat, a přizpůsobit tomu návrh hry.

3.7 Suroviny

Jelikož se Skillegy odehrává ve vesmíru podobném tomuto a nemá fantasy ani sci-fi téma, suroviny vyskytující se ve hře budou podobné těm, jež známe z normálního světa. Jmenovitě se jedná o dřevo, jídlo a zlato. Každá z těchto surovin bude sloužit jinému účelu a všechny budou klíčové pro správný rozvoj základny.

Suroviny se budou nacházet přímo na mapě a bude je možno získat těžbou. V různých real-time strategiích se objevují rozličné přístupy k této problematice, rozdíly jsou například v tom, jestli je kapacita surovin omezená nebo jestli se dělník vrací do budov k tomu určeným, aby tam vytěžený produkt odnesl, jakmile jej má předem dané množství. Ve Skillegy jsem se pro zjednodušení rozhodla pro řešení známé z *Age Of Empires III*. Kapacita zdroje suroviny bude omezená, ale

dělník se nemusí vracet do budov. Suroviny, které vytěží, se budou přičítat rovnou hráči, jenž jej vlastní.

3.8 Armáda

Ve většině RTS jsou jednotky různě silné a mají rozličné slabiny a přednosti. Známý a velmi používaný princip je tzv. kámen-nůžky-papír v akci aplikovaný pomocí kavalérie, pěchoty a dělostřelectva. Jezdec na koni je silný proti dělu, ale slabý proti kopiníkům, kteří jsou naopak lehce zdecimování děly. Kvůli systému parametrů jednotek, jenž je použit ve Skillegy, je ale tento přístup značně komplikovaný, a proto nebude implementován. Je však určitě možné přidat další parametry jednotek a tímto tento princip napodobit.

3.9 Budovy

Budovy jsou klíčovým elementem každé real-time strategie. Jejich hlavním účelem je stavění jednotek a vylepšování technologií, ale pro některé budovy toto neplatí. Domy jsou tradičně určeny ke zvyšování populačního limitu, zdi slouží jako obrana. Některé budovy mohou sloužit jako útočiště pro slabé jednotky při nájezdu nebo útočí na poblíž stojící nepřátelské jednotky. Síla útoku se přitom odvíjí od síly jednotek uvnitř budovy. Další možností budov je těžba surovin, která je ale většinou pomalejší než u přírodních zdrojů a slouží primárně pro pokročilou fázi hry, když už hráči došly suroviny v blízkosti základny. Každá budova také slouží jako pozorovatelna přilehlých území.

Ve Skillegy se spokojíme s budovami, které staví jednotky, pozorují okolí, jsou zdrojem surovin a zvyšují populační limit, nově ale budou přidány budovy, například kasárny nebo knihovna, v nichž se jednotky mohou trénovat a vylepšovat tak své vlastnosti.

3.10 Umístění budov

Přestože umisťování budov není ve všech real-time strategiích omezeno, občas se vyskytne pár pravidel, jež je třeba dodržovat. V některých hrách lze například postavit novou budovu jen v daném okruhu od hlavní budovy, jinde ji lze postavit všude kromě míst, která jsou moc blízko k budově soupeře. Tyto koncepty určitě mohou přidat na hratelnosti a donutit hráče k pečlivějšímu promýšlení strategií, přesto však omezují kreativitu a nejsou pro hru nutná, proto bude ve Skillegy jediným omezením to, že budovy nelze stavět přes suroviny nebo přes další budovy.

3.11 Vylepšování jednotek

Pro tuto práci je vylepšování jednotek jedním z ústředních bodů, protože právě jím je nahrazen princip více typů vojáků. Aby hra co nejlépe simulovala skutečný život, použiji vylepšení jednotky, které se bude vždy vztahovat jen na ni. Druhým a asi častějším přístupem by bylo vylepšení pro všechny jednotky určitého typu, což se dá chápat jako globální doručení lepších zbraní všem vojákům. Naše

pokroky však nebudou spočívat v lepším vybavení, ale ve zlepšování schopností. To může probíhat přímo vykonáváním činnosti nebo navštívením budovy určené k trénování dané vlastnosti.

3.12 Fog of war

Fog of war je častým konceptem ve strategických hrách, který symbolizuje neznámé, jež zažívá vojevůdce ve skutečné bitvě. Přidává do hry nutnost prozkoumávání a maskuje pozici a velikost protivníkovy armády. Nejznámějším přístupem je postupné odkrývání částí mapy v těsné blízkosti jednotek a budov. Takto získaná informace může být trvalá – hráč až do konce hry vidí všechno na místech, kde někdy byla nějaká jeho jednotka či budova nebo rozestavěná budova – nebo dočasná, kde viditelná jsou pouze místa, poblíž kterých se jednotka či budova stále nachází, na ostatních lze vidět pouze budovy a suroviny.

Pro Skillegy jsem zvolila druhou možnost s drobnými úpravami, které budou blíže popsány v další kapitole. Hlavním principem je, že nepřátelské jednotky lze vidět pouze poblíž současné polohy vlastních jednotek, budovy a suroviny jsou viditelné po celou hru od doby jejich objevení.

3.13 Mapa

Přestože terén může být ve hrách velice různorodý a hornatý, nijak výrazně nenapomáhá hratelnosti a objevuje se hlavně z vizuálních důvodů. Proto bude ve Skillegy terénem pouze rovná plocha s rozmístěnými surovinami. Kvůli lepšímu grafickému zážitku je ale záhodno vymyslet takové řešení, které umožní snadné budoucí vytváření lepších map.

3.14 Umělá inteligence

Pro umožnění single-playeru nebo zpestření hry více hráčů je nezbytná umělá inteligence. Ta může fungovat jako protivník, spojenec nebo neutrální národ se zvláštními schopnostmi. Většina her obsahuje kromě základních bitev také tématickou kampaň, pro niž je umělá inteligence klíčová. Přestože jsem se v této hře rozhodla neimplementovat kampaň, nepřítomnost hry pro jednoho hráče by již byla významným omezením. Proto je součástí hry alespoň jednoduché rozhraní pro umělou inteligenci s ukázkovou implementací.

4. Uživatelská dokumentace

V této kapitole se zaměříme na uživatelský pohled na hru a popíšeme kroky potřebné ke spuštění hry, jednotlivé obrazovky, uživatelské rozhraní hry i její ovládání.

4.1 Popis instalace

Hra byla testována na operačním systému Windows 10 a lze ji nainstalovat pomocí souboru *setup.exe*, který se nachází ve složce \Setup. Tento instalační soubor byl vytvořen pomocí *Inno Setup Compiler*. [27]

4.2 Úvodní obrazovka

Po spuštění programu se zobrazí úvodní obrazovka, která kromě tlačítka *Exit*, jež vypíná hru, obsahuje i tlačítka *Play and Host* a *Join*. Play and Host ustanoví daného hráče Hostem – tedy serverem a klientem zároveň – a zobrazí další panel, ve kterém se čeká na ostatní hráče. Tlačítko Join ukáže hráči stejný panel, ale pouze tehdy, pokud se připojil k již existující hře. Textové pole nad tímto tlačítkem totiž značí, na jakou IP adresu se bude hra snažit připojovat. Dokud se na této IP adrese nedaří k žádné hře připojit, je na obrazovce zobrazena zpráva „Waiting...“. Tuto zprávu lze zrušit tlačítkem *Cancel*, které hráče vrátí do původní pozice. Není dovoleno snažit se spouštět hru jako Host z jedné IP adresy vícekrát, tento pokus skončí chybou.

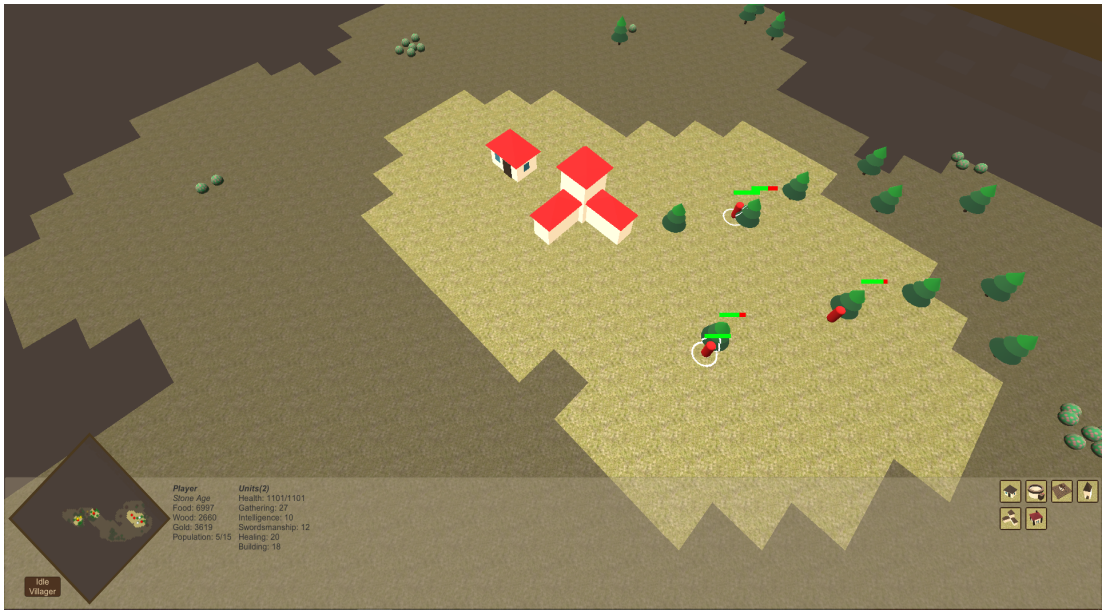
Již zmíněný panel, který se hráči zobrazí po vytvoření hry nebo připojení se k existující hře, ukazuje všechny momentálně připojené hráče. Těchto hráčů může být jedna až šest. Vedle tlačítka *Exit* se teď nachází i tlačítko *Back*, jež hráče vrátí o krok zpět. Pro Hosta je také dostupné tlačítko *Next*, které všechny hráče přesune do následující Menu obrazovky.

4.3 Menu obrazovka

Na této obrazovce se zobrazí všichni připojení hráči a každý z nich si může změnit své jméno nebo barvu. Host může navíc pomocí tlačítka *Add Player* přidat další hráče ovládané umělou inteligencí, měnit jim barvu a jméno nebo je pomocí tlačítka *Remove* odebrat. Celkově však hráčů musí být od dvou do šesti, aby mohl Host spustit hru pomocí tlačítka *Play*.

4.4 Uživatelské rozhraní

Na screenshotu ze hry 4.1 můžeme vidět uživatelské rozhraní, které je navrženo tak, aby hráči dávalo co nejkomplexnější a zároveň nejpřehlednější informaci o hře. Na obrázku lze vidět tři jednotky, které těží dřevo, dvě z nich jsou momentálně označené. Tmavá část mapy značí neprozkoumaná místa, nejsvětlejší pak místa,

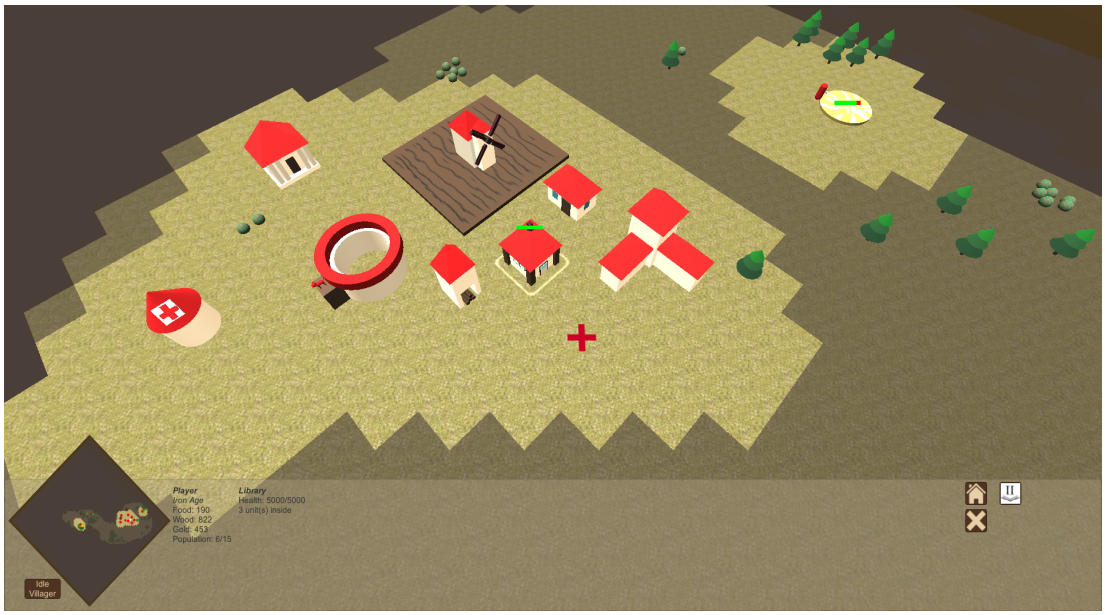


Obrázek 4.1: Ukázka uživatelského rozhraní.

blízko nichž má hráč svůj objekt, a tmavší části jsou už prozkoumané, ale žádný objekt k nim není dostatečně blízko, aby je opravdu viděl.

Vlevo dole se nachází minimapa a vedle ní tlačítko, jímž lze vybrat náhodnou jednotku, která zrovna nemá žádnou práci. Vpravo od minimapy jsou informace o hráči, tedy jeho jméno, věk, v němž se nachází, a dostupné suroviny. Ve vedlejším sloupci můžeme nalézt informace o právě vybraném objektu, v tomto případě se jedná o skupinu dvou jednotek.

V pravém dolním rohu obrazovky se nacházejí dostupné obchody pro vybraný objekt, zde to jsou budovy, jež může daná skupina jednotek postavit.



Obrázek 4.2: Ukázka uživatelského rozhraní se všemi budovami.

Screenshot 4.2 je ve většině věcí podobný, ale jsou na něm ukázány všechny

budovy ve hře. Označenou budovou je knihovna a červený křížek uprostřed obrazovky značí, že jednotky po opuštění budovy budou směřovat k tomuto místu.

Také pravý dolní roh obrazovky je trochu jiný, změnily se totiž nákupy – knihovna má právě dostupný pouze jeden – a zobrazila se tlačítka pro zničení vybraného objektu a pro zobrazení okna s jednotkami v budově.

4.4.1 Ovládání hry

Hra je vytvořena pro ovládání především myší, ale několik kláves v ní i přesto má svůj účel.

Pomocí klávesy *Esc* je možné kdykoli během hry zobrazit vrchní vyskakovací panel s tlačítky *Exit* a *Back*, klávesy *WASD* slouží tradičně pro pohyb po mapě a pomocí klávesy *R* lze zrušit vybranou budovu, již má hráč právě umístit. Veškeré ostatní akce však probíhají pouze za pomoci myši či touchpadu.

Pohyb kamery

Základním úkonem v každé real-time strategii je pohyb kamery. Kameru ve Skillegy je možné přiblížit či oddálit kolečkem myši a posunout pomocí přesunutí myši k okraji okna nebo, jak již bylo zmíněno, klávesami WASD. Dalším způsobem, jak posunout kameru, je pomocí minimapy v levém dolním rohu mapy. Kliknutí na minimapu levým tlačítkem způsobí, že se na dané místo ihned přesune kamera.

Výběr objektu

Jednotku, budovu nebo surovinu lze vybrat kliknutím myši, skupinu jednotek pak tahem myši, což vytvoří bílý rámeček a po puštění tlačítka vybere všechny vlastní jednotky, které se v tomto rámečku právě nacházely. Tento skupinový výběr platí pouze na vlastní jednotky, nepřátelské jednotky ani budovy nebo suroviny lze vybrat pouze po jedné. Pomocí tlačítka *Idle Villager* v levém dolním rohu obrazovky je také možné vybrat náhodnou jednotku, která zrovna nemá co dělat.

Když je objekt vybrán, zobrazí se popisek a dostupné nákupy, jež může uskutečnit, v dolním panelu. Zviditelní se také ukazatel zdraví, který má každý objekt nad sebou, a rozsvítí kolem něj vytvoří bílý kruh nebo čtverec v závislosti na tvaru objektu.

Příkazy

Když tedy máme vybraný objekt, který nám patří, můžeme mu zadat příkaz tím, že někam klikneme pravým tlačítkem myši. Pro suroviny a rozestavěné budovy se však nic nestane, protože nemají žádnou možnost se pohybovat ani samy od sebe interagovat s okolím. Pokud ale máme vybranou budovu, daná pozice se nastaví jako shromažďovací bod budovy, takže když ji jednotky opustí, automaticky půjdou na toto místo.

Akce, kterou provede jednotka po pravém kliknutí myši, závisí na typu objektu, na nějž jsme klikli. Pokud je to země, jednotka se tam přesune, v případě

suroviny ji začne těžit, na jakýkoli nepřátelský objekt zaútočí, do přátelské budovy vstoupí a přátelskou rozestavěnou budovu začne stavět. Jednotce lze také zadat místo, kam má jít, pomocí kliknutí na dané místo na minimapě pravým tlačítkem myši.

Při vybrání jakéhokoli objektu, který hráči patří, se také zobrazí tlačítko v pravé dolní části obrazovky, kliknutí na něž způsobí okamžité zničení vybraného objektu.

Nákupy

Další možností, jak zadávat objektu příkazy, je skrze nákupy. Nákupy se nacházejí v pravém dolním rohu obrazovky a zobrazují nápovědu, když se přes ně přejede myší. Pokud klikneme na nákup týkající se stavby nové budovy, je třeba tuto budovu umístit kliknutím na příslušné místo. Pokud během toho klikneme na jiný nákup, původní se zruší a budeme pokračovat rovnou novým nákupem. Jak již bylo zmíněno, nákup lze zrušit také klávesou *R*.

V případě nákupů, na něž je potřeba čekat, se fronta nákupů zobrazí nad tlačítky s nákupy ve formě koleček, na kterých se postupně odpočítává čas. Když přes toto tlačítko přejedeme myší, ukáže se, o jaký nákup se jedná, a kliknutím myši jej můžeme zrušit.

Jednotky uvnitř budovy

Pokud má hráč vybranou budovu, která mu patří, zobrazí se vedle tlačítek s nákupy i další tlačítko s obrázkem domu. Když na toto tlačítko klikne, zobrazí se okno, v němž jsou ukázány všechny jednotky uvnitř budovy a u každé z nich je tlačítko *Remove*, které přikáže dané jednotce opustit budovu. V případě ošetřovny je u každé jednotky i tlačítko *Switch*, jež z ní udělá lékaře nebo naopak pacienta, pokud už lékařem byla.

Konec hry

Pokud hráč prohraje nebo vyhraje, zobrazí se mu odpovídající zpráva a tlačítko *Menu*, které jej vrátí do úvodní obrazovky.

5. Vývojová dokumentace

V této kapitole se blíže podíváme na implementaci a použité postupy. Popíšeme si strukturu výsledného programu, jeho hlavní části a vztahy mezi nimi.

Program byl napsán v jazyce C# za použití Visual Studio 2017 Enterprise. Pro vývoj byl použit Unity 2017.2.0f3 Personal. V Unity 2018 je stále funkční, přestože se mohou objevit drobné problémy s kompatibilitou, ale Unity 2019 tuto hru nepodporuje a není možné ji v něm otevřít.

Projekt lze spustit v Unity pomocí souboru *\Skillegy\Assets\Lobby.unity*. Pro správné fungování je však třeba importovat dva assety, A* Pathfinding Project [26] a Network Lobby [28].

Network Lobby musíme importovat pomocí *Asset Store*, asset je možno nalézt na https://assetstore.unity.com/packages/essentials/network-lobby-41836.

A* Pathfinding Project si však musíme stáhnout přímo, je dostupný na adrese https://arongranberg.com/astar/download, v programu je použita verze 4.2.8. Stažený soubor poté musíme rozbalit a ve složce nalezneme soubor s příponou *.unitypackage*, který spustíme, zatímco máme projekt otevřený v Unity.

5.1 Unity

Protože použití frameworku Unity ovlivňuje celou strukturu hry, popíšeme si na úvod jeho fungování a nejdůležitější součásti, a až poté se budeme věnovat samotné struktuře programu.

5.1.1 Základní fungování

Klíčovým prvkem Unity je objekt *GameObject*. Je základem pro všechny elementy hry a odvíjejí se od něj koncepty a vlastnosti, které lze měnit jak v kódu, tak i přímo v Unity Editoru. GameObject je také schránkou pro různé komponenty, jež určují chování objektu.

Hra se v Unity skládá ze scén (*Scenes*), které se dají popsat jako jednotlivá okna hry. Například Menu nebo různé levely hry mají každý svou vlastní scénu, jež se skládá ze stromové struktury různých objektů. Pro snadné vytváření a kopírování objektů nabízí Unity koncept tzv. *prefabs*, což jsou v podstatě předlohy, podle kterých je možno objekt v kódu nebo v Editoru vytvořit. Objekty z předloh můžou vzniknout v jakékoli části hierarchie objektů a z každého objektu lze v Editoru udělat předlohu.

Nutnou součástí každého objektu je komponenta *Transform*, která udává pozici, rotaci a velikost. Tyto vlastnosti jsou relativní vzhledem ke svému rodiči ve stromové struktuře. Dalšími důležitými komponentami jsou skripty, které tvoří téměř veškeré chování objektů. Tyto skripty mohou být psány v C# nebo v JavaScriptu, pro tento projekt je použit C#. Každý skript, jenž má být použit jako komponenta k nějakému objektu, musí dědit od třídy *MonoBehaviour*.

MonoBehaviour poskytuje metody, které se volají v různých fázích životního cyklu této třídy. Přestože je tento cyklus značně ovlivněn a zkomplikován multiplayerem a související technologií UNet, je stále důležité popsat jeho nejdůležitější části i v této základní formě.

Po startu hry nebo po prvním aktivování komponenty ve hře se zavolá metoda *Awake*, která se používá k nastavení referencí na jiné komponenty. Jako druhá se zavolá *Start*, určená především k inicializaci proměnných, k níž je potřeba, aby na všech objektech již proběhla metoda *Awake*. V obou těchto metodách je doporučeno nastavit co nejvíce referencí, jejichž inicializace je časově náročná, protože se v nich například hledá objekt v celé hře. Provedením těchto operací v metodách Start nebo Awake se zabrání opakování časově náročných úkonů při každém vykreslení snímku. Inicializaci si lze částečně ulehčit pomocí Editoru, kde se dají pohodlně nastavit některé proměnné.

Většina herní logiky klasicky probíhá v metodách *Update* nebo *FixedUpdate*. Update se zavolá vždy při vykreslení snímku, FixedUpdate oproti tomu probíhá v pravidelných intervalech, takže je vhodné tuto metodu používat při fyzikálních výpočtech. Po aktivaci komponenty následuje *OnEnable* a naopak po deaktivaci *OnDisable*.

Před zničením komponenty, k němuž může dojít po přepnutí scény, na konci hry nebo proto, že to bylo napsáno v kódu, se zavolá metoda *OnDestroy*. Tato metoda je na konci hry spuštěna na všech komponentách všech existujících objektů, které však mohou být v takřka libovolném pořadí. Totéž platí pro metody Start a Awake na začátku hry, u nich lze ale pořadí v některých případech do jisté míry specifikovat. V důsledku tohoto omezení se téměř ve všech implementacích metody OnDestroy v tomto projektu objevují kontroly, zda jsou jiné potřebné komponenty již zničeny nebo ne. Tyto kontroly mají zamezit častým *NullReferenceException*, k nimž dochází při vypínání aplikace. Nabízí se lépe vypadající řešení v podobě metody *OnApplicationQuit*, která se v Unity také vyskytuje a volá se před ukončením aplikace. Není však definováno, jestli se OnApplicationQuit zavolá před nebo po OnDestroy. Jiným možným řešením by bylo nepoužívat OnDestroy, ale před zničením objektu či komponenty během hry zavolat funkci, jež by ji nahrazovala. Problémem se ale v tomto případě stává síťová komunikace, protože by se mohlo stát, že ke klientovi dorazí zpráva o zničení komponenty dříve než volání této metody, v důsledku čehož by opět docházelo k výjimkám. Z těchto důvodů jsem se rozhodla pro řešení s opakovanými kontrolami, i když to občas může skrýt chyby, na něž by se jinak přišlo snadněji.

5.1.2 UNet

Přestože základní koncepty zůstávají stejné i v hře více hráčů, síťová komunikace a z ní vyplývající problémy komplikují vývoj hry zcela zásadním způsobem. Jak se dočteme v Unity manuálu, pro použití HLAPI UNetu je zásadní existence objektu *NetworkManager*, který se stará o síťové aspekty hry. Dále je nutné mít uživatelské rozhraní, díky němuž mohou hráči začít hru nebo se do již existující hry připojit, a skripty, které jsou uzpůsobené pro multiplayerovou hru.

Díky absenci předešlých zkušeností s UNetem jsem se rozhodla použít asset *Network Lobby* [28] z *Unity Asset Store*, který sliboval dostačující uživatelské rozhraní i vlastní NetworkManager. Musela jsem jej však výrazně upravit, hlavně kvůli pozdějšímu přidání umělé inteligence, ale také proto, že obsahoval podporu a uživatelské rozhraní i pro službu *Matchmaking*, kterou v tomto projektu nevyužívám. Network Lobby bohužel není open-source, proto jsem nemohla zasahovat do jeho zdrojových kódů. Na některých místech stačilo vytvořit potomka některé

z tříd assetu a pozměnit tak některé virtuální metody. Například pro třídu *LobbyMenu* se tento postup však ukázal nedostatečným, protože neobsahovala žádné virtuální metody, jež by bylo možno přepsat. Bylo tedy potřeba přepsat celou třídu a s ní i metody, jejichž původní implementace by byla pro tuto hru vhodná.

Network Lobby nakonec svůj díl práce na multiplayerovém aspektu vykonal, přesto však jeho samotná existence nestačí, protože řeší jen určité části síťové komunikace, například její začátek nebo přerušení. O všem, co se děje během hry samotné, rozhodují skripty, a ty proto také musí být upraveny pro více hráčů.

Základním principem pro vytváření multiplayerových her je přenášet po síti tak málo dat, jak jen to jde. S tím souvisí i minimalizace počtu objektů a jejich proměnných, které se musí synchronizovat, aby byly pro všechny hráče stejné. Synchronizované objekty jsou vždy „majetkem“ jednoho z klientů a ten nad ním má autoritu. Musí k nim být také přiřazena speciální komponenta *NetworkIdentity*. Skripty k těmto objektům dědí od NetworkBehaviour (potomek MonoBehaviour) místo MonoBehaviour. Díky tomuto pak mají širší škálu využitelných metod i proměnných.

Jak již bylo naznačeno v minulé kapitole, při použití UNetu klient a server komunikují převážně pomocí volání *Command* nebo *ClientRPC* metod. Command může být zavolán pouze na objektu, nad nímž má daný klient autoritu, a metoda označena tímto atributem má zaručeno, že bude vždy zavolána na serveru. Oproti tomu ClientRPC znamená, že tato metoda proběhne na všech klientech, a může být zavolána jen ze serveru. Pomocí těchto dvou atributů probíhá naprostá většina synchronizace, jež je v UNetu potřeba.

Vytváření a ničení synchronizovaných objektů musí probíhat výhradně na serveru a veškeré změna synchronizovaných dat se změní nejprve na serveru, s čímž souvisí i principy tzv. *SyncVar* čili snadnější synchronizace jednoduchých proměnných. Pokud se proměnná označena tímto atributem změní na serveru, tato změna se projeví na všech klientech. Manipulace s touto proměnnou na klientovi však ovlivní jen daného klienta.

Životní cyklus skriptů je samozřejmě síťovou komunikací výrazně ovlivněn a zkomplikován. Poté, co server vytvoří objekt, také zavolá metodu *NetworkServer.Spawn*, čímž tento objekt vytvoří i na všech klientech. Případně může zavolat *NetworkServer.StartWithClientAuthority* a tím navíc předat autoritu jinému klientovi. Na serveru je poté zavolána metoda *OnStartServer* a na klientech *OnStartClient*. Poté se na klientovi, který má nad objektem autoritu, zavolá ještě *OnStartAuthority*. Hra probíhá podobně jako v životním cyklu objektu bez UNetu, jen se navíc všechny proměnné *SyncVar* synchronizují. Zničení objektu probíhá opět na serveru, a to pomocí metody *NetworkServer.Destroy*. Na klientech se zavolá metoda *OnNetworkDestroy* a objekt se poté zničí. [29]

Pořadí volání těchto funkcí však bohužel není nijak definováno vzhledem k základnímu životnímu cyklu komponenty, není tedy například jasné, zda se OnStartClient volá před nebo po Start. Toto omezení je třeba mít na paměti, aby nedošlo ke zvláštním, nedefinovaným chybám, kterým je ale i tak při používání UNetu obtížné se vyhnout.

5.2 Struktura programu

Jak již bylo zmíněno výše, každá hra v Unity se skládá z několika scén. V této hře jsou tři scény, *Lobby*, *Menu* a *Game*, z nichž nejdůležitější je podle očekávání Game. Postupně si ale projdeme nejen ji, ale i zbývající dvě a nejdůležitější objekty a skripty pro jejich fungování.

5.2.1 Lobby Scene

Lobby scéna je určena k připojování lidských hráčů a je primárně tvořena assetem Network Lobby. Jediným jejím důležitým objektem je *LobbyManager*, což je jediný objekt ve hře, který je aktivní celou hru a nezruší se při přepnutí na další scénu. Stojí u něj za zmínku komponenta *CustomLobbyManager*, jež rozšiřuje třídu LobbyManager, jež se stará primárně o připojování a odpojování klientů. CustomLobbyManager navíc umožňuje připojování umělých inteligencí a lepší vypořádání se s odpojením klienta nebo serveru během hry.

5.2.2 Menu Scene

V Menu scéně dochází hlavně k přidávání hráčů s umělou inteligencí a zadávání barvy a jména každého hráče. Každému hráči se také v této fázi přiřadí počáteční pozice. Důležitými skripty jsou *MenuManager*, který se stará hlavně o vytváření a zušení hráčů, a *MenuPlayer*, jenž v této scéně hraje roli hráče. Po změně scény na Game se nezruší okamžitě, ale počká, až se scéna nahraje, aby mohl vytvořit hráče typu *Player* a předat mu obsah svých proměnných. Jak v MenuPlayer, tak i v Player je umělá inteligence označena nastavením příznaku IsHuman na false. Dalšími třídami pro Menu jsou třída *MenuPlayerList*, která se stará o přidělování barev, a *PlayerRow*, jež řeší uživatelské rozhraní pro jednotlivé hráče.

5.2.3 Game Scene

Hlavní herní logika se odehrává v Game scéně, tudíž je logické, že se v ní nachází nejvíce objektů i komponent. Před začátkem hry jsou objekty této scény především suroviny, plocha reprezentující zemi, objekty nutné pro uživatelské rozhraní, objekty přijímající uživatelský vstup, uchovávající a kontrolující stav hry, dále pak kamery a osvětlení. Jakmile však hra začne, vytvoří se objekty *Player* a s nimi i objekty pro stav hráče nebo *fog of war*. Během hry se vytvářejí objekty jednotek a budov, které se mohou společně se surovinami také zničit. Důležitou součástí je také objekt, jenž se stará o pathfinding v průběhu celé hry.

V následující části se blíže podíváme na některé důležité objekty a poté na jejich propojení, tj. jak fungují některé časté a významné operace ve Skillegy a jakým způsobem by bylo možno toto fungování rozšířit.

GameObject Player

GameObject Player je schránkou pro tři hlavní komponenty, třídy *Player*, *Factory* a *FirstUnitCreator*.

FirstUnitCreator je nutná, protože neexistuje žádná metoda, která by se zavolala, až se scéna plně nahraje (*OnLevelWasLoaded*, kterou nabízí Unity, nefunguje

vždy správně). Proto se v metodě Update tato třída pokusí inicializovat hráče, a pokud se jí to podaří, sama sebe zničí.

Třída Factory se používá pro vytváření objektů podle prefabs a inicializaci jejich některých proměnných. Její metody se volají ze třídy Player na serveru a vrací vzniklý objekt, který je pak většinou vytvořen i na všech klientech.

Player se používá hlavně pro volání příkazů pro operace, jež by měly být synchronizovány. Je to například vytvoření či zničení objektu, ale také změna vlastností jednotky nebo kapacity suroviny. V metodě Update vždy probíhá kontrola, jestli už hráč nevyhrál nebo neprohrál, k čemuž se využívá objekt a komponenta *VictoryCondition*.

GameObject GameState

K objektu GameState je připojen skript GameState, který se stará především o synchronizaci globálního herního stavu. Právě na něm se provádí většina ClientRPC metod, jež volá Player ze svých Command metod. Tyto Command metody někdy pouze volají ClientRPC, které pak provedou na všech klientech to samé. Tento přístup je použit například v metodách *CmdCreateBuilding* ve třídě Player a *RpcCreateBuilding* ve třídě GameState, které reagují na dokončení stavby budovy.

GameState se také stará o aktualizaci jak grafu pro pathfinding při vytvoření nebo zničení surovin a budov, tak i viditelnosti hráče a s ní související fog of war. GameState obsahuje i metody pro zjišťování nejbližších surovin, budov a jednotek, které jsou důležité hlavně pro umělou inteligenci.

Vzhledem k tomu, že nedává smysl mít více instancí třídy GameState, a také proto, že je tato třída hojně používaná a téměř všechny ostatní by na ni musely mít referenci, jsem použila singleton a jediná její instance je dostupná jako *GameState.Instance*.

GameObject PlayerState

PlayerState je jeden z mála objektů ve hře, které existují vždy jen na svém klientu a nejsou vůbec synchronizovány přes síť. Pro každého hráče na daném klientu existuje vždy právě jeden PlayerState. Není to singleton v pravém slova smyslu, ale k jeho instancím lze přistupovat staticky pomocí metody *PlayerState.Get(int i = 0)*, kde i je identifikační číslo hráče. Implicitní hodnota i je 0, protože lidský hráč může být na každém klientu jen jeden a vždy má ID 0.

PlayerState má pouze jednu komponentu se stejným jménem, která slouží k uchovávání informací o hráči, které nejsou relevantní pro nikoho jiného a tudíž se nemusí synchronizovat, a zároveň k volání dalších metod z jiných tříd určených hlavně ke zobrazování UI lidskému hráči. Schraňuje tedy například informace o množství surovin, které hráč vlastní, jeho populaci, budovách, rozestavěných budovách a věku, ve kterém se momentálně nachází. Pokud se změní množství jídla, jež má hráč k dispozici, zavolá PlayerState metodu *OnPlayerStateChange*, která aktualizuje příslušný výpis v UI.

Pathfinding

Klíčovým bodem celého fungování hry je správně pracující systém pro hledání cest. Jak jsem již zmínila v předchozí kapitole, zřejmě nejpoužívanějším algoritmem v real-time strategiích je A*. [30] Tento algoritmus se používá pro hledání nejkratší cesty v grafu a pro své fungování využívá heuristiku.

A* začíná se dvěma seznamy – prázdným seznamem uzavřených vrcholů a seznamem otevřených vrcholů, ve kterém je jediný prvek, počáteční vrchol cesty. V každém kroku algoritmus vybere jeden vrchol grafu podle ohodnocení *f(x)*, který si nazveme *v*. Ohodnocení *f(x)* je součet dvou dílčích ohodnocení, *g(x)* a *h(x)*. Funkci *f(x)* definujeme jako cenu nejlevnější zatím nalezené cesty z počátečního bodu do *x*. Tato funkce je tedy definována přesně, zatímco *h(x)* je pouze odhad ceny cesty z vrcholu *x* do koncového bodu. Pokud je jeden ze sousedů *v* cílový vrchol, našli jsme nejkratší cestu a můžeme hledání nastavit. V opačném případě vezmeme všechny sousedy vrcholu *v*, které nejsou v seznamu zavřených vrcholů, a přidáme je do seznamu otevřených vrcholů. Pokud v něm ještě nejsou, zaznamenejme si vrchol *v* jako jejich předka a také jejich hodnoty *f*, *g* a *h*. Pokud ale některý ze sousedů v seznamu otevřených vrcholů už byl, zkontrolujeme, zda není právě nalezená cesta do něj lepší než ta zapamatovaná, a podle toho novou cestu buď zaznamenáme, nebo zapomeneme. Jakmile bude seznam otevřených vrcholů prázdný, můžeme ukončit hledání s tím, že cesta neexistuje. [31]

Přestože naprogramovat základní A* není moc složité, je nutné si vybrat z několika možných heuristik pro *h(x)* – pro využití v real-time strategii je vhodná například vzdálenost vzdušnou čarou k cíli. Je však pravděpodobné, že již existující možnosti jsou lépe optimalizované a obsahují více funkcí, které by se při navigaci mohly hodit. Proto mi připadalo lepší využít nějaké již naprogramované řešení.

Původně jsem se snažila využít pathfinding dostupný přímo v Unity. Bohužel vyvstalo několik problémů, z nichž nejhorší byla nutnost synchronizovat pohyb jednotky přes síť. Unity nabízí řešení v podobě komponenty *NetworkTransform*, která v pravidelných intervalech posílá pozici jednotky všem klientům, a jednotka se pak přemístí na určené místo. Výsledný efekt však nebyl plynulý a s přibývajícím počtem jednotek se jejich chování na klientech stalo neúnosným. Jako další krok jsem tedy zkusila počítat cestu na každém klientu zvlášť, ale při tomto přístupu občas docházelo k problémům se synchronizací a jednotka byla u každého klienta na jiném místě. Ani kombinace dvou zmíněných pokusů nevedla k žádnému úspěchu.

Logicky vypadajícím řešením bylo počítat cestu pouze na serveru a pak ji posílat klientům, kde ji bude každý následovat sám. Bohužel jsem však nenašla jediný způsob, jak toho za použití pathfindingu z Unity a UNetu docílit. UNet podporuje pro parametry Command a ClientRPC metod i pro SyncVar proměnné pouze jednoduché typy, jedinou výjimku představují třídy SyncList, sychnronizované seznamy, které jsou ale pořád pouze jednoduchých typů. Cestu ze zmíněného pathfindingu lze sice převést na seznam průchozích bodů, ale nepřišla jsem na způsob, jak tuto cestu pak na klientu zrekonstruovat.

Možnost synchronizace cesty však přinesl asset A* Pathfinding Project [26], který byl taktéž zmíněn v minulé kapitole. Pomocí něj se mi povedlo upravit defaultní implementaci tak, že je schopna poslat výslednou cestu po síti, a také přináší například funkce pro aktualizaci překážek i za běhu hry. A* Pathfinding

nabízí několik typů grafů, pro real-time strategii se ale hlavně kvůli jednoduché aktualizaci za běhu nejvíce hodí GridGraph (mřížkový graf) Některé funkce jako například *local avoidance*, tj. vyhýbání se sobě navzájem při větším počtu jednotek, jsou však dostupné jen v placené verzi. Hlavně díky tomu by se ještě několik věcí dalo na pathfindingu vylepšit, ale v naprosté většině případů funguje správně.

Abstraktní třída Selectable

Selectable je základní třídou – jak již název napovídá – pro všechny objekty ve hře, které lze označit kliknutím myši. Do této kategorie spadají jednotky, skupiny jednotek, suroviny, budovy a rozestavěné budovy. Přestože se v mnohém liší, jsou to první objekty s vizuální reprezentací, se kterými se v této scéně setkáváme. Většina metod ve třídě Selectable je virtuální nebo abstraktní.

Třída Unit

Unit je potomkem třídy Selectable a je součástí objektu Unit. Tento objekt, který představuje jediný typ jednotky ve hře, má také další komponenty, důležité jsou hlavně *MovementController*, *AIUnetPath*, *Seeker* a *Skills*.

Schopnosti Skills slouží k uchovávání informací o schopností jednotek. Každá jednotka má pět vlastností, získávání surovin (Gathering), inteligenci (Intelligence), šermířství (Swordsmanship), léčení (Healing) a stavitelství (Building), jež mají vlastní enum SkillEnum. Informace o jejich úrovních pro každou jednotku jsou uloženy v její komponentě Skills, která se také stará o propagaci změny nějaké schopnosti za účelem odpovídající změny uživatelského rozhraní. Všechny schopnosti jednotky jsou dostupné i skrz třídu Unit přímo přes property (např. *unit.Gathering*) nebo pomocí metod k tomu určených (např. *unit.GetSkillLevel (SkillEnum.Gathering)* a *unit.SetSkillLevel (SkillEnum.Gathering, 10)*).

Job Důležitým konceptem ve třídě Unit je její reference na instanci třídy *Job*. Ta označuje momentální úkol jednotky, jež se jednotka snaží v každém snímku plnit pomocí metody *Do* na třídě Job, do které jako parametr dodá sama sebe. Jednotka také kontroluje, jestli je už úkol splněn, a pokud ano, přesune se na další. Ve třídě Job jí k tomuto slouží property *Following*, která se používá v případě, že po daném úkolu je jasné, jaký bude následovat. Příkladem uplatnění je *JobGather*, úkol, ve kterém jednotka periodicky odebírá kapacitu surovině a stejné množství této suroviny se přičítá hráči do třídy PlayerState. Jakmile kapacita suroviny dojde, příznak *Completed* se nastaví na true a Following vrátí úkol „Zde je další surovina stejného typu. Jdi k ní a začni ji těžit.“ neboli instanci třídy *JobGo*, která se postará o přesun na určené místo, a poté pomocí Following zadá další úkol, jenž jí byl zadán v konstruktoru. Ve chvíli, kdy není žádný další úkol k dispozici, přijde na řadu úkol *JobLookForTarget*, což je v podstatě nečinný stav. Pokud se ale k jednotce s tímto úkolem přiblíží nepřítel, hned na něj zaútočí. Dalšími úkoly jsou *JobAttack* (jednotka periodicky odebírá zdraví nepříteli, pokud je dostatečně blízko), *JobBuild* (periodické zvyšování čísla udávajícího, jak moc je pokročeno ve stavbě budovy), *JobEnter* (vstup jednotky do budovy) a *JobFollow* (následování jednotky).

Selectable obsahuje abstraktní metody *GetOwnJob* a *GetEnemyJob*, jež se používají při určování, jaké akce jsou u daného objektu relevantní a co bylo myšleno oním kliknutím pravým tlačítkem myši. GetOwnJob se používá u objektů, které hráč vlastní, a GetEnemyJob u nepřátelských objektů nebo surovin. Metoda *SetGoal* mezi nimi rozhoduje a tímto přístupem je tedy například docíleno toho, že jednotka do přátelské budovy vstoupí, ale na soupeřovu zaútočí.

JobBuild a JobGather kromě svých zjevných vlastností také zlepšují schopnosti jednotky, která tuto práci vykonává, v závislosti na její inteligenci. Pokud schopnosti Gathering nebo Building přesáhnou úroveň Intelligence dané jednotky, tato schopnost se bude vylepšovat mnohem pomaleji.

Třídy MovementController, AIUnetPath a Seeker Tyto komponenty se na objektu jednotky nacházejí kvůli pathfindingu. Seeker je součástí assetu a nebylo jej potřeba nijak upravovat. Slouží k následování cesty, kterou mu poskytnou jiné komponenty.

AIUnetPath a MovementController spolu úzce souvisejí a používají se právě pro hledání cesty. AIUnetPath je potomkem třídy AIPath, jež také pochází z assetu, ale není uzpůsobena pro UNet, a proto v ní nemohou být ClientRPC ani Command metody. Toto omezení se však dalo obejít nezávislou třídou MovementController, která je potomkem NetworkBehaviour, a kombinací těchto dvou tříd jsem nakonec dosáhla kýženého efektu.

Když je tedy ve třídě Unit zavolána metoda *Go(destination)*, tato metoda zavolá stejnojmennou metodu ve třídě MovementController. Ta si uloží místo, kam se jednotka chce dostat, a předá jej třídě AIUnetPath. Pravděpodobně hned v dalším snímku MovementController zjistí v metodě Update, že AIUnetPath potřebuje vytvořit cestu. Zavolá proto Command metodu *CmdSearchPath*, jež zařadí tuto cestu do fronty na vypočítání. Jakmile je cesta hotová, zavolá se metoda AIUnetPath *OnPathComplete*, která informaci předá do třídy MovementController a ta pak cestu rozdělí na seznam vektorů, které předá klientům pomocí principu SyncList. Na závěr pak zavolá ClientRPC metodu *RpcOnPathComplete*, jež cestu rekonstruuje, a zavolá se metoda *OnPathComplete* ze třídy AIPath, kterou předtím nahradila metoda v AIUnetPath. Tím se postup uzavírá a nepřichází se o žádnou důležitou funkcionalitu, již jsme mohli omylem přepsat.

Jakmile jednotka dorazí do destinace, je to oznámeno třídě Unit, která podle toho aktualizuje UI a nastaví příznak Completed ve své Job na true, aby mohla postoupit k dalšímu úkolu.

Třída Regiment

Aby bylo možné označovat více jednotek zároveň a také jim najednou udílet příkazy, existuje také třída Regiment, která reprezentuje skupinu jednotek. Ve většině případů však pouze reprodukuje příkazy všem jednotkám, jež jsou její součástí. Jediný trochu zajímavější postup se uplatňuje, když skupina jednotek dostane příkaz, aby se někam přesunula. V tom případě se každá jednotka pošle na nějaké náhodné místo v kruhu o velikosti, která závisí na počtu jednotek. Pro toto řešení jsem se rozhodla, protože větší počet jednotek, který se snaží dostat na jedno místo, nevypadá tak přirozeně, a navíc se pak většina z nich nedostala tak blízko, aby mohly považovat svůj úkol za splněný a přesunout se k dalšímu.

Regiment je jedním z mála objektů, který není třeba synchronizovat, protože hráč může označit pouze skupinu jednotek, jež mu patří, a toto označení je viditelné pouze pro něj. Regiment zanikne ve chvíli, kdy všechny jeho jednotky zahynou, nebo když jej hráč odznačí.

Abstraktní třída Resource

Třída Resource má tři potomky, *FoodResource*, *WoodResource* a *GoldResource*, jež označují po řadě keře s jídlem, stromy a zlaté doly. Sama třída Resource je velmi jednoduchá, za zmínku stojí proměnná *capacity*, tedy kapacita suroviny, která klesá při její těžbě. Důležitou metodou je *Gather*, jejímiž atributy jsou hodnota schopnosti Gathering právě těžící jednotky a hráč, který tuto jednotku vlastní. V metodě Gather se určité množství suroviny odečte z její kapacity a přičte vlastníku jednotky, jež právě těží.

Třída TemporaryBuilding

Každá budova má dva stavy – stav, ve kterém je rozestavěná, a stav, kdy už se může nazývat plnohodnotnou budovou se všemi svými funkcemi. A právě rozestavěné budovy reprezentuje třída TemporaryBuilding.

Typ výsledné budovy určuje enum BuildingEnum, podle něhož se také pak budově přidá komponenta typu Building. Během stavby je však typ budovy nepodstatný. Když se TemporaryBuilding vytvoří, je neviditelná pro všechny kromě svého vlastníka a metoda *SetTemporaryBuilding* ji nastaví jako budovu, která právě čeká na umístění. O umístění se starají metody ve třídě PlayerState, jež také zajišťují, aby hráč budovu neumístil někam, kde pro ni není dostatek prostoru. Jakmile hráč vybere pozici, zavolá se metoda *OnPlaced* na všech klientech. V této metodě se budova zviditelní i pro ostatní hráče v jejím dosahu a změní svůj vzhled, aby bylo poznat, že ještě není dostavěna.

Poté může začít hlavní proces stavění. Hned po umístění je k budově přiřazena jednotka nebo skupina jednotek, které jsou právě označeny, a je jim zadán úkol ji postavit. Při stavbě se postupně zvyšuje proměnná *progress* a jakmile dosáhne hodnoty proměnné *maxProgress*, zavolají se metody *CmdCreateBuilding* ve třídě Player a pak *RpcCreateBuilding* ve třídě GameState. Tyto metody zařídí, aby byla z objektu odstraněna komponenta TemporaryBuilding a nahrazena odpovídajícím potomkem třídy Building.

Abstraktní třída Building

Ve Skillegy aktuálně existuje osm typů budov, z nichž každý má vlastní třídu, potomka Building. V každém z nich existuje property *MaxPopulationIncrease*, což je velikost populace, kterou tato jednotka podporuje. Právě o toto číslo se s každým postavením budovy zvětší maximální populace hráče, ale při jejím zničení se maximální populace opět sníží o tuto hodnotu. Jde ovšem o něco jiného než *UnitCapacity*. Tato proměnná určuje, kolik jednotek může najednou přebývat uvnitř budovy, ale nijak nesouvisí s maximální populací hráče.

Budova si také pamatuje, které jednotky jsou momentálně uvnitř, a periodicky volá metodu *UpdateUnit* pro každou z nich. To, co se v této metodě děje, závisí na typu budovy. Ty z nich, které jako část svého úkolu vylepšují schopnosti

jednotek, mají typicky dvě proměnné určující rychlost vylepšování v případech, že je daná schopnost vyšší nebo nižší než inteligence jednotky.

Bank (banka), *Sawmill* (pila) a *Mill* (mlýn) slouží ke shromažďování surovin daného typu, ale jsou pomalejší než přirozeně se vyskytující zdroje. V každém zavolání metody UpdateUnit se tedy nějaké množství suroviny přidá hráči a zároveň se zlepší schopnost Gathering dané jednotky. Ani zlepšování této schopnosti samozřejmě nepůjde tak rychle jako při těžbě klasických surovin.

MainBuilding (hlavní budova) a *House* (dům) mají prázdnou implementaci UpdateUnit. Hlavní budova má sice obrovskou kapacitu jednotek narozdíl od domu, ale je to z obranných důvodů a ne kvůli vylepšování schopností.

Barracks (kasárny) mají za úkol zlepšovat šermířství jednotek a z tohoto důvodu se v metodě UpdateUnit zlepšuje vlastnost Swordsmanship.

Library (knihovna) má ve Skillegy specifickou funkci hlavního centra vzdělávání. Jsou v ní tři stavy pro vylepšování různých schopností, Intelligence, Healing a Building, podle nichž se v metodě UpdateUnit rozhoduje, jakou vlastnost bude vylepšovat. Vylepšování Healing a Building probíhá podobně jako například Swordsmanship v kasárnách, Intelligence má však ze zjevných důvodů konstantní rychlost vylepšování. Princip přechodů mezi stavy si mimo jiné popíšeme v další podkapitole.

Infirmary (ošetřovna) jako jediná rozlišuje více než jednu roli jednotek v budově. Jednu jednotku lze zvolit jako lékaře, jehož schopností Healing je pak násobena rychlost celé ošetřovny. Z tohoto důvodů existují ve třídě Infirmary dvě proměnné udávající rychlost léčení bez lékaře a s ním. Rychlost je také omezena počtem pacientů, pokud jejich množství překročí hodnotu proměnné *maxUnitsHealing*, léčení bude pomalejší. Metoda UpdateUnit se tedy bude chovat jinak u lékaře než u ostatních jednotek. Lékaři se bude zvyšovat schopnost Healing (rychlost vylepšování lineárně závisí na inteligenci, ale není rozdíl mezi tím, zda je Intelligence vyšší nebo nižší než Healing). U ostatních jednotek tato metoda vybere odpovídající rychlost uzdravování a o přiměřené množství zvýší *Health* jednotky, pokud ještě nedosáhla hodnoty *MaxHealth*.

Třída Purchase

Po vysvětlení základních vlastností třídy Selectable a jejich potomků si můžeme popsat další princip, který je pro hru velmi důležitý, a to jsou nákupy jednotek, budov a vylepšení. Právě na jejich reprezentaci je použita třída Purchase spolu se svým jediným potomkem, LoadingPurchase.

Třída Purchase obsahuje jméno, obrázek a popis nákupu pro zobrazení v uživatelském rozhraní, jeho cena (jídlo, dřevo, zlato a populace, již zabírá), funkce, která určuje, zda je nákup pro daný objekt dostupný, a funkce, jejíž zavolání je produktem při tomto nákupu (tedy která se zavolá při jeho dokončení). LoadingPurchase zastupuje nákupy v budovách, které se neuzavřou hned, ale zaplatí se a až po nějaké době se kýžená funkce vykoná. Proto je v této třídě zahrnuta i rychlost, s jakou se nákup nahrává. Další proměnnou je příznak *oneTimePurchase*, který určuje, zda je nákup pouze jednorázový (např. postoupení do dalšího věku) nebo může být proveden vícekrát (např. vytvoření jednotky).

Třída Selectable obsahuje seznam nákupů, které jsou pro tento objekt relevantní. Ty dostupné z nich se objeví v uživatelském rozhraní, když je objekt vybrán hráčem, a po kliknutí myši na jejich tlačítko se spustí metoda *Do* třídy

Purchase. V ní je nejprve zavolána metoda *Pay* třídy PlayerState, která dostává v parametrech cenu nákupu, provede odečtení zdrojů a vrací boolean vypovídající o tom, zda je nákup v pořádku zaplacen. Pokud ano, provede se funkce uložená ve třídě Purchase.

V případě LoadingPurchase je ovšem situace trochu složitější. Třída Building obsahuje seznam instancí třídy *Transaction* (transakce), což je třída, která slouží právě k postupnému načítání nákupů. Metoda Do tedy musí kromě kontroly, zda si hráč může nákup dovolit, i zjistit, zda má budova ještě volné místo na transakce. V takové situaci metoda případně odebere jednorázový nákup z dostupných nákupů pro tuto budovu a vytvoří novou instanci třídy Transaction, kterou pak předá parametrem metodě *AddTransaction* ve třídě Building. Budova přidá transakci do seznamu a pokud tam žádná jiná není, ustanoví ji aktivní transakcí a ihned započne její načítání. To probíhá pomocí tzv. *Coroutine*, tj. funkce, která neproběhne celá hned, ale má schopnost své vyhodnocování pozastavit a vrátit se k němu v dalším snímku. Zde používaná Coroutine má název *LoadTransaction* a volá metodu *Load* ve třídě Transaction, která přičte odpovídají hodnotu k proměnné udávající stav nahrávání, a vrátí true, pokud bylo načítání již dokončeno. Když tedy Load obdrží true jako návratovou hodnotu, vykoná funkci v odpovídajícím nákupu a spustí další transakci, pokud je nějaká taková k dispozici.

Aby bylo hráči umožněno rozmyslet si svůj nákup, na který musí čekat, existuje ve třídě Building také metoda *ResetTransaction*, která zruší transakci a zavolá další metody, které vrátí hráči jeho suroviny a případně učiní jednorázový nákup zase dostupným. Podobný princip se používá i u základního nákupu bez čekání při stavění, hráč si totiž může stavbu budovy rozmyslet, než ji umístí do světa. Budova je pak zničena a hráč dostane suroviny zpět.

Třída PlayerPurchases

Aby bylo možné Purchases snadno spravovat, má každý hráč na svém klientovi vlastní objekt PlayerPurchases a stejnojmennou komponentu, které jsou přístupné přes třídu PlayerState. Nejdůležitější položkou ve třídě PlayerPurchases je dictionary *purchases*, jenž každé hodnotě v enumu *PurchaseEnum*, který obsahuje všechny možné názvy nákupů, přiřadí instanci třídy Purchase.

Zde se tedy provádí nastavení všech akcí, které se mohou kupovat přes třídu Purchase, všech podmínek, za nichž je tento obchod dostupný. Klasická podmínka postoupení do určitého věku je zastoupena funkcí *ReachedAge*, která bere jako parametr *PlayerState.AgeEnum*, což je enum o čtyřech položkách symbolizujících čtyři věky ve hře. Nákupy vylepšení v budovách se mohou lišit v tom, zda se jejich efekt promítne pouze do dané budovy, nebo bude mít vliv na všechny budovy tohoto typu. Tento problém potom lze řešit příznakem, jestli daný nákup už proběhl, a jeho nastavením v akci nákupu. Dále zde lze nastavit obrázky, názvy a popisky k jednotlivým nákupům.

K položkám dictionary *purchases* lze přistupovat pomocí metod *Get*, které jako parametr přijímají PurchaseEnum nebo BuildingEnum, protože BuildingEnum je pouze podmnožinou PurchaseEnum. Inicializace nákupů pro každou instanci Selectable tedy probíhá v její metodě *InitPurchases*, která se volá v metodě OnStartAuthority a kde se volá *AddPurchase* s parametrem PurchaseEnum. Tato metoda přidá příslušný Purchase z PlayerPurchases do seznamu nákupů objektu. V každém volání Update potom Selectable kontroluje, jestli se nezměnila

dostupnost některého z jejích nákupů.

Implementace Fog of war

Hned poté, co třída Player vytvoří jednotku, pokročí ke tvorbě objektu zodpovědného za fungování fog of war. Tento objekt má název *HumanVisibilitySquares* pro lidského hráče a *VisibilitySquares* pro umělou inteligenci a obsahuje komponentu stejného jména. Třída HumanVisibilitySquares je potomkem VisibilitySquares a liší se pouze v implementaci metody Update.

VisibilitySquares ve své metodě Start vytvoří po celé mapě čtverečky jednotné velikosti s komponentou *MapSquare* tak, aby zakrývaly celou herní plochu. Každý tento čtvereček má dvě vrstvy, průhlednou a neprůhlednou. Neprůhledná vrstva označuje části mapy, kam hráč ještě nevstoupil, a nemá tak nejmenší ponětí o tom, co tam může být. Druhá, průhledná vrstva slouží pro části mapy, která hráč už někdy viděl, ale momentálně tam nemá žádnou budovu, jednotku ani rozestavěnou budovu, takže nemá žádnou informaci o jednotkách, jež se tam nacházejí.

K objektům MapSquare se dá přistupovat přes VisibilitySquares pomocí identifikátoru *squareId*, protože VisibilitySquares si všechny MapSquare ukládá do dictionary. Každý MapSquare si pamatuje reference na několik dalších čtverečků, které jsou situovány v kruhu kolem něj, dále má uloženy všechny objekty, jež se v jeho oblasti nacházejí. Všechny objekty v sobě mají také uložen identifikátor čtverečku, do kterého patří, v proměnné *squareId*. Pozici těchto objektů si aktualizuje za pomoci třídy GameState, která jej upozorní, když se pozice nějakého objektu změní. Budovy a suroviny je potřeba aktualizovat jen při jejich vzniku nebo zániku, jednotka ale zavolá v každém snímku metodu *PositionChange* třídy GameState, kde se zjistí, jestli nedošlo ke ke změně jejího squareId, a pokud ano, uvědomí o tom VisibilitySquares.

Objekty jsou v MapSquare uchovávány v seznamech, jednotky, budovy a rozestavěné budovy jsou ještě rozděleny na dva seznamy podle toho, zda jsou přátelské nebo nepřátelské. Díky tomu je možno zjistit, které MapSquare jsou blízko k nějakému objektu, který tento MapSquare odkryje. Proces aktualizace odkrytých MapSquare probíhá v metodě Update.

Tato metoda se liší ve VisibilitySquares pro umělou inteligenci a pro lidského hráče, Update v HumanVisibilitySquares nejprve nastaví příznak *isActivated* na true všem MapSquare, které jsou v blízkosti nějakého přátelského objektu. Poté projde všechny MapSquare a u těch, kterým se příznak od minulého snímku změnil, provede jejich zakrytí (metoda *Cover*) nebo naopak odkrytí (metoda *Uncover*). Cover zničí neprůhlednou vrstvu čtverečku, deaktivuje tu průhlednou a zviditelní všechny objekty ve své oblasti. Uncover znovu aktivuje průhlednou vrstvu a zneviditelní všechny jednotky na tomto území.

MapSquare pro umělou inteligenci nemá žádnou vizuální podobu a metoda Update ve VisibilitySquares pouze nastaví příznaky *isActivated* a *isUncovered*, podle kterých se pak rozhoduje, jestli jsou objekty viditelné či ne.

VisibilitySquares se také používá k určení nejbližší suroviny k dané pozici, nejbližšího viditelného nepřítele apod., protože obsahuje komplexní informace o umístění objektů a viditelnosti hráče. Takové informace se mohou hodit například právě umělé inteligenci nebo i jen jednotce při hledání další suroviny po vytěžení té minulé.

Třída UIManager

Poslední třídou, jíž se budeme v této podkapitole zabývat, je UIManager, který zařizuje naprostou většinu uživatelského rozhraní. Podobně jako GameState je to také singleton a je dostupný skrz *UIManager.Instance*. Obsahuje několik metod, jež se starají například o zobrazování dostupných nákupů i těch, které už čekají na zpracování ve vybrané budově. Také přijímá uživatelský input z většiny tlačítek ve hře a dle potřeby je aktivuje či deaktivuje, dále zobrazuje popis k vybranému objektu a aktualizuje informace o hráči a jeho surovinách. Další funkcí této třídy je i například zobrazení cíle, ke kterému vybraný objekt směřuje. Společně s třídou *BuildingWindow* zařizuje UIManager i aktivování okna s popisem všech jednotek uvnitř dané budovy.

5.3 Postupy použitelné při rozšiřování hry

Na tomto místě se seznámíme s několika možnostmi, jak hru Skillegy dále rozšířit, a popíšeme si kroky, jak takového rozšíření docílit. Možností, jak přidat do hry další aspekty, je samozřejmě spousta (o některých z nich se můžeme dočíst v následující kapitole), zde se však budeme věnovat jen těm nejjednodušším a nejtypičtějším.

5.3.1 Změna mapy

Nejjednodušší možností, jak změnit stávající mapu, je upravit scénu Game, odstranit z ní nepotřebné suroviny a vytvořit další pomocí prefabs ve složce *Prefabs*. Je také možné změnit velikost mapy, tedy objektu *Map*. V tomto případě je však zapotřebí podle toho upravit proměnnou *MapSize* ve třídě GameState. Počáteční pozice hráčů po změně velikosti mapy se dají upravit pomocí úpravy *PlayerPositions* v objektu LobbyManager ve scéně Lobby.

Dalším rozšířením by mohly být například hory či jiné neprůchodné objekty, je však nutné si kvůli tomu vytvořit novou prefab s komponentou Collider. Jednotky pak budou tento objekt automaticky obcházet. Pokud byly změny takové, že by mohly ovlivnit pathfinding scény, je nutné jej na konci úprav aktualizovat v Editoru kliknutím na tlačítko *Scan* u objektu *AStar*.

5.3.2 Přidání typu suroviny

Prvním krokem při přidání nového typu suroviny je rozšíření třídy PlayerState, to zahrnuje novou proměnnou, property a úpravu popisku hráče. Dále je potřeba vytvořit novou třídu, která bude dědit od Resource, a implementovat všechny abstraktní metody po vzoru již existujících tříd surovin. Tuto třídu pak jako komponentu přidružíme k objektu, který bude danou surovinu zastupovat. Nejjednodušším způsobem je použít už existující surovinu, ve které pouze vyměníme komponentu, model suroviny a barvu ikony pro minimapu, případně upravíme Collider. Tento objekt je pak možné registrovat jako prefab a umístit na mapu. Také bude třeba upravit nebo přidat pár metod v rozhraní pro umělou inteligenci.

5.3.3 Přidání typu budovy

Pro přidání vlastního objektu budovy je vhodný postup podobný jako u surovin, tedy vyjít z již existující budovy, vyměnit model a upravit všechny nutné části. Dále musíme vytvořit nového potomka třídy Building a implementovat metody *ChangeColor* a *UpdateUnit*. V ChangeColor změníme barvu určitých částí budovy, aby bylo poznat, komu patří. Do UpdateUnit však patří většina logiky budovy, protože právě v této metodě se rozhoduje, co s jednotkami, které se v budově nacházejí. Pokud chceme, aby se v budově daly pořizovat nějaké nákupy, přepíšeme i virtuální metodu *InitPurchases* s defaultní prázdnou implementací.

Aby však bylo možné budovu postavit, musíme ji přidat do BuildingEnum i do PurchaseEnum a vytvořit odpovídající Purchase v PlayerPurchases. Poté musíme změnit všechny místa se switchem přes BuildingEnum nebo PurchasesEnum a přidat nový Purchase do metody *InitPurchases* třídy Unit, aby se stavba budovy ukázala v nákupech jednotky.

5.3.4 Přidání nového nákupu

Nový nákup můžeme přidat ve třídě PlayerPurchases ve formě nové instance třídy Purchase nebo LoadingPurchase, kde se k němu dá nastavit většina atributů, například cena, podmínka, při jejímž splnění se nákup stane dostupným, název, obrázek nebo popisek. Také je nutno nákup přidat do PurchasesEnum a v důsledku tohoto upravit i rozhraní pro umělou inteligenci. Nakonec stačí jen v metodě InitPurchases objektu, přes který bude možné nákup provést, přidat daný Purchase do seznamu nákupů tohoto objektu.

5.3.5 Přidání nové schopnosti jednotky

Ačkoli přidání schopnosti je sám o sobě jednoduchý úkol, většinou se s ním pojí jiné problémy, například co bude daná schopnost ovlivňovat a jak bude možné ji vylepšovat. Řešení těchto problémů však může být tak různorodé, že nemá smysl se zde rozepisovat o různých možnostech a budeme se soustředit pouze na přidání nové schopnosti.

Nejprve přidáme novou schopnost do SkillEnum a poté i do Skills, kde zároveň změníme i popisek jednotky. Je také vhodné zajistit, aby byla vlastnost dostupná přes odpovídající property i ze třídy Unit, hlavně kvůli konzistenci.

5.3.6 Další úpravy

I když se technicky vzato nejedná o rozšíření, mohlo by někdy přijít vhod upravit stávající atributy již existujících herních principů. Příkladem mohou být ceny nákupů, které lze všechny upravit v PlayerPurchases, kde se také dají upravit podmínky pro dostupnost jednotlivých nákupů. Počáteční suroviny hráče jsou součástí třídy PlayerState a rychlost jejich získávání během hry je definována vždy v příslušné surovině nebo přímo v budově, která slouží k těžbě této suroviny. Další důležitou součástí hry, jíž je možno upravit parametry, jsou schopnosti jednotek. Funkci určující začáteční schopnosti jednotky lze najít ve třídě Factory a rychlost vylepšování schopností je dána vždy u činnosti, kterou se schopnost vylepšuje.

Nachází se tedy například ve třídách budov určených k vylepšování jednotek nebo v JobGather nebo JobBuild.

6. Umělá inteligence

Umělá inteligence (AI) je nezbytná k tomu, aby byla umožněna hra pro jednoho hráče. Tomuto hráči nejčastěji slouží jako protivník, v některých real-time strategiích je však možné si jej nastavit i jako spojence pro oživení hry.

Existuje několik strategií pro tvorbu umělé inteligence, základní přístupy se liší hlavně v tom, jestli má být inteligenci umožněno podvádět. Podvádění může spočívat v nižší ceně nákupů pro AI než pro lidského hráče, v získávání více surovin těžbou nebo v úplnějších informacích o stavu hry, než by měl lidský hráč v téže pozici. Tyto způsoby mohou pomoci umělé inteligenci vyrovnat své nedostatky oproti lidskému hráči docela jednoduše a bez nutnosti psát komplikované kódy a stavové automaty.

Ve Skillegy jsem se však rozhodla uplatnit přístup bez podvádění, čili umělá inteligence bude mít stejné možnosti a omezení jako lidští hráči. Z pohledu hráče je totiž velice nepříjemné zjistit, že umělá inteligence, proti níž hrajete, podvádí, a už fakt, že lidskému hráči zabírá čas klikání myší, by se dal považovat za výhodu AI.

Protože je tolik možností, jakými AI implementovat, ukážeme ve Skillegy pouze rozhraní pro tvorbu AI, pomocí něhož si každý může naprogramovat vlastní umělou inteligenci podle svých potřeb a schopností. Hra však také obsahuje ukázkovou inteligenci, jež byla pomocí tohoto rozhraní naprogramována. Tuto ukázkovou AI lze dále použít při tvorbě vlastní umělé inteligence, což bude popsáno níže.

6.1 Rozhraní pro umělou inteligenci

Úlohu rozhraní pro umělou inteligenci plní v tomto programu ve skutečnosti dvě třídy, *AIPlayer* a *AI*. *AIPlayer* je proxy třída, přes kterou jsou umělé inteligenci dostupné akce, jež může lidský hráč provést pomocí myši nebo klávesnice. Plní tedy úlohu prostředníka a inteligence nemůže se hrou komunikovat jinak než přes ni. *AI* je abstraktní třídou pro všechny možné umělé inteligence a obsahuje jedinou datovou položku – odkaz na třídu *AIPlayer*.

Na začátku hry je každou třídou *Player*, která reprezentuje hráče řízeného umělou inteligencí, inicializována tvorba objektů obsahujících komponenty *AIPlayer* i *AI*. *AI* je poté předána reference na instanci *AIPlayer* a *AIPlayer* dostane veškeré reference, jež potřebuje ke svému fungování, hlavně *GameState* a *PlayerState*.

Ostatní třídy hry musí počítat s možností přítomnosti umělé inteligence ve hře, což se většinou projevuje podmínkami, zdali hráč, který metody volá, je lidský hráč nebo umělá inteligence. Hráči ovládáni umělou inteligencí totiž nemají uživatelské rozhraní a neměli by ovlivňovat ta, která patří lidským hráčům. S tímto problémem se pojí i nutnost rozdílné implementace fog of war, která již byla popsána ve vývojové dokumentaci. Hlavním důvodem je tedy to, že umělá inteligence nepotřebuje žádnou vizuální reprezentaci fog of war, ale jen data o místech, jež navštívila, a poblíž kterých má jednotku či budovu.

Abstraktní třída AI

Jak již bylo zmíněno, *AI* je opravdu minimalistická třída, protože jedině tak lze nabídnout programátorovi co největší volnost při implementování umělé inteligence. Důvodem existence této třídy je snaha o to, aby při výměně implementací nemusel uživatel pokud možno vůbec zasahovat do jiných tříd než do potomka *AI*, jehož si při vytváření umělé inteligence sám implementuje.

Třída AIPlayer

Třída *AIPlayer* poskytuje většinu funkcí, které by umělá inteligence mohla potřebovat ke svému fungování. Podle svého účelu se dají tyto funkce rozdělit do dvou skupin, z nichž první slouží ke zjišťování informací o stavu hráče i celé hry a druhá k zadávání příkazů vlastním jednotkám a budovám.

Metody pro zjišťování stavu hry a hráče zajišťují umělé inteligenci přístup například k dostupným surovinám hráče, jeho populaci, maximální populaci, jednotkám, budovám a dočasným budovám, které vlastní, ale i k surovinám a nepřátelským objektům, jež může za dané situace vidět. Jsou poznatelné podle toho, že začínají slovem „Sense“, a větší část z nich pouze podává informace dostupné ze tříd *PlayerState* a *GameState* či jejich vyfiltrované části. V tabulce 6.1 jsou vypsány všechny metody, jež získávají informace týkající se herních objektů. Pokud je výstupem nějaký „nejbližší“ objekt, nejbližší je chápáno ve vztahu k pozici zadané parametrem.

Tabulka 6.1: Metody pro zjišťování herního stavu

Název metody	Návratová hodnota metody
SenseBestUnit(SkillEnum skill)	Nejlepší hráčova jednotka v dané schopnosti
SenseBestIdleUnit (SkillEnum skill)	Nejlepší hráčova jednotka v dané schopnosti, která nemá co dělat
SenseClosestIdleUnit (Vector3 destination)	Nejbližší hráčova jednotka, která nemá co dělat
SenseClosestUnit (Vector3 destination)	Nejbližší hráčova jednotka
SenseOwnUnits()	Hráčovy jednotky
SenseIdleUnits()	Hráčovy jednotky, které nemají co dělat
SenseGoodUnits (SkillEnum skill, float bar)	Hráčovy jednotky, které jsou v dané schopnosti lepší, než udává parametr
SenseGoodIdleUnits (SkillEnum skill, float bar)	Hráčovy jednotky, které jsou v dané schopnosti lepší, než udává parametr, a nemají co dělat
SenseFoodGatherers()	Hráčovy jednotky, které sbírají jídlo z keřů s jídlem
SenseWoodGatherers()	Hráčovy jednotky, které těží dřevo ze stromů
SenseGoldGatherers()	Hráčovy jednotky, které těží zlato z dolů
SenseExplorers()	Hráčovy jednotky, které prozkoumávají
SenseOwnBuildings()	Hráčovy budovy

Tabulka 6.1: Metody pro zjišťování herního stavu

Název metody	Návratová hodnota metody
SenseClosestBuilding()	Nejbližší hráčova budova jakéhokoli typu
SenseOwnTemporary-Buildings()	Hráčovy rozestavěné budovy
SenseClosestTemporary-Building(Vector3 destination)	Nejbližší hráčova rozestavěná budova
SenseVisibleEnemyBuildings()	Nepřátelské budovy v dohledu
SenseVisibleEnemy-TemporaryBuildings()	Nepřátelské rozestavěné budovy v dohledu
SenseVisibleEnemyUnits()	Nepřátelské jednotky v dohledu
SenseVisibleResources()	Suroviny v dohledu
SenseClosestVisibleResource (Vector3 destination)	Nejbližší surovina daného typu v dohledu

Funkce zadávající příkazy jsou poněkud tíživější problém, protože mnohokrát nebývá jasné, jaké parametry by vlastně měly vyžadovat a co by si měly být schopné zjistit samy. Například metoda starající se o těžbu suroviny může dostávat v parametrech jednotku, jež bude těžit, i surovinu, která bude těžena, nebo se může v těchto věcech rozhodovat sama na základě herního stavu. Obě tyto možnosti se mohou v určitých případech hodit, a proto jsem se snažila je umožnit, většinou pomocí implicitních parametrů. Pokud tedy některý parametr není zadán, funkce si jej zkusí doplnit sama, nejčastěji náhodným prvkem z vhodného seznamu. Funkce pak vrací příznak, jestli se vše povedlo, tudíž vrátí false, pokud nějaký parametr nebylo možno doplnit nebo byl příkaz z nějakého jiného důvodu neuskutečnitelný. Tyto funkce lze také rozdělit do několika typů, které si zde představíme.

Stavba budov Stavba budov probíhá ve dvou fázích, které bohužel kvůli síťové komunikaci nemohou probíhat ve stejném snímku. Tyto fáze jsou reprezentovány metodami *BuildBuilding* a *PlaceBuilding*. V implementaci musíme vždy zajistit, aby proběhla nejdříve *BuildBuilding*, která budovu pouze vytvoří, a poté až v dalším snímku *PlaceBuilding*. *BuildBuilding* přijímá jako svůj parametr typ budovy, a to buď ve formě PurchaseEnum, nebo BuildingEnum. Oba tyto enumy jsou již popsány ve vývojové dokumentaci (5.2.3). *PlaceBuilding* obsahuje volitelné parametry, které určují pozici budovy a jednotku, jež budovu postaví. Pokud není jednotka uvedena, vybere se nejbližší volná jednotka, a pokud ani ta neexistuje, použije se nejbližší jednotka. V případě neuvedení pozice se vybere náhodné umístění v kruhu se středem v pozici hráče a poloměrem, jenž závisí na počtu již postavených budov.

Vytváření jednotek O tvorbu jednotek se stará *BuildUnit* s volitelným parametrem budovy, ve které se jednotka vytvoří. Pokud budova není uvedena, použije se náhodná hlavní budova.

Těžba surovin Těžbou surovin se zabývají metody *GatherFromResource*, *GatherFood*, *GatherWood* a *GatherGold*. Všechny berou jako parametr jednotku, která bude těžit, a GatherFromResource navíc požaduje surovinu, již tato jednotka bude těžit. Další tři metody si najdou nejbližší surovinu daného typu, ale pokud taková neexistuje, pokusí se ještě využít budovy, která je k těžbě dané suroviny určena.

Prozkoumávání K prozkoumávání slouží metoda *Explore*, jíž je možno předat jako volitelný parametr jednotku, která bude prozkoumávat. Pokud tato jednotka není uvedena, prozkoumávat bude náhodná jednotka, která nemá co dělat.

Interakce jednotek s budovami Pro vstup jednotky do budovy lze použít metodu *EnterBuilding*, jež je generická podle typu budovy a obsahuje dva volitelné parametry, jednotku a budovu. Pokud není některý parametr udán, použije se náhodná volná jednotka nebo náhodná budova daného typu. Výstup z budovy řeší metoda *ExitBuilding* opět s parametry jednotkou a budovou, která má tentokrát parametr budovy povinný a pokud není jednotka uvedena, opustí budovu všechny jednotky.

Trénování jednotek O zlepšování schopností jednotek se stará metoda *TrainUnit*, jež bere jako parametry typ schopnosti a jednotku. Jednotka je volitelný parametr a pokud není udána, použije se náhodná volná jednotka. Metoda poté zadá jednotce příkaz, jehož splnění vyústí v její zlepšení v dané schopnosti, například jednotku, která si má zdokonalit schopnost stavitelství, pošle stavět náhodnou budovu nebo studovat do knihovny.

Útok K zaútočení na nepřítele slouží metoda *Attack*. Jejími volitelnými parametry jsou jednotka, která zaútočí, a její cíl. Pokud není uvedena jednotka, vybere se náhodná volná, a pokud dojde k neuvedení cíle útoku, hledá se náhodná viditelná nepřátelská jednotka, budova nebo rozestavěná budova v tomto pořadí.

Nákupy S nákupy souvisejí metody *GetPurchaseCost*, která v out parametrech vrací cenu daného nákupu, *CheckPurchaseCost*, jež dostane v parametrech potřebné množství jídla, dřeva a zlata a pokud něco z toho hráči chybí, pošle jednotku těžit danou surovinu, a *DoPurchase*, která nákup zaplatí a vykoná. DoPurchase má pouze jediný parametr a to typ nákupu. Objekt, prostřednictvím jehož je nákup vykonán, se vybere náhodně podle typu nákupu. Při stavění budov pomocí *DoPurchase* se zavolá pouze metoda *BuildBuilding* a je tedy nutno zajistit, aby poté proběhla i *PlaceBuilding*.

6.1.1 Ukázková umělá inteligence

Umělá inteligence, která je prezentována v této práci, je opravdu pouze ukázkou použití rozhraní a je tedy velice nepravděpodobné, že by mohla lidského hráče skutečně porazit. Je na ní především ukázáno, že je možné toto rozhraní používat, a že se s jeho pomocí dá vytvořit umělá inteligence, jež si celkem rozumně postaví a rozšiřuje vlastní základnu.

Inteligence je tvořena třídou *SimpleAI*, která je potomkem výše popsané třídy *AI*. Je založena na principu seznamu úkolů, jež se postupně pokouší plnit. Každý takový úkol je reprezentován strukturou *SimpleAI.Objective*, která obsahuje cenu úkolu – v jídlu, dřevě, zlatu a populaci – a funkci, jež představuje jeho splnění.

Na začátku hry se v metodě Start zavolá virtuální metoda *CreateObjectives*, která má za úkol vytvořit seznam úkolů, jež se inteligence bude postupně snažit splnit. Seznam se vytváří pomocí metod *AddFirst* a *AddLast*, které vytvoří Objective a rovnou jej přidají na začátek nebo konec seznamu úkolů. Obě tyto funkce mají dvě varianty, z nichž jedna přijímá PurchaseEnum, tedy typ nákupu, jež chce vykonat, a druhá funkci. V prvním případě metoda zjistí cenu nákupu pomocí *GetPurchaseCost* a společně s funkcí *DoPurchase* ji uloží do Objective a ve druhém případě je funkce uložena bez jakékoli ceny. Z tohoto důvodu není doporučeno využívat v této inteligenci přímo funkce *BuildBuilding* a *BuildUnit*, protože by u nich nebyla uvedena cena a tím bychom přišli o výhody s tím související.

V případě, že se jedná o nákup budovy, zajistí funkce *AddFirst(PurchaseEnum purchase)* a *AddLast(PurchaseEnum purchase)*, aby byl vytvořen další úkol s metodou *AIPlayer.PlaceBuilding*, který je poté umístěn na odpovídající pozici.

V metodě Update se v každém snímku volá metoda *UpdateObjectives*. Ta se pokusí splnit první úkol na seznamu. Pokud se jí to povede, což pozná z návratové hodnoty volané funkce, odstraní daný úkol ze seznamu a v dalším snímku se zase bude vykonávat jiný.

Složitější situace však nastává, pokud se úkol z nějaké důvodu nepodařilo splnit. Důvody totiž můžou být různé, od nedostatku surovin, absence volné jednotky nebo potřebné budovy nebo vylepšení po pouhou nutnost počkat si, než jednotka dostaví budovu, aby v ní bylo možné něco provádět.

Tato umělá inteligence řeší pouze nejjednodušší problémy, a to nutnost čekání a nedostatek surovin nebo populace. Pokud tedy plnění úkolu neuspěje, v metodě UpdateObjectives se zavolá funkce *AIPlayer.CheckPurchaseCost* s cenou Objective v parametrech. Tato funkce pošle v případě potřeby volnou jednotku na surovinu, ale neřeší již, co dělat, pokud žádná jednotka volná není. V dalším kroku se v UpdateObjectives zkontroluje, jestli by se splněním úkolu přesáhl populační limit. Pokud tomu tak je a žádný dům se momentálně nestaví, vytvoří se Objective se stavbou domu a zařadí se na první místo v seznamu úkolů. Pokud se inteligenci stále nedaří splnit ten stejný úkol, zasekne se na něm a nikdy se neposune dále. Tím se však mimoděk podaří vyřešit řádku situací, kde bylo nutné pouze chvíli počkat, než se nějaká akce dokončí.

Seznam úkolů se tedy musí vytvářet nanejvýš pozorně, aby pokud možno nedocházelo k situacím, které umělá inteligence na této úrovni neumí vyřešit.

Postup umělé inteligence podle ukázkového seznamu úkolů ve třídě *SimpleAI* je asi následující. Na začátku hry si hráč postaví hlavní budovu a potom deset jednotek. Průběžně staví domy a těží suroviny, kdykoli to potřebuje. Poté postoupí do kamenného věku a postaví dalších deset jednotek, kasárny a tři knihovny. Pošle tři jednotky do kasáren a tři jednotky do knihovny, aby si zlepšovaly dovednosti, a jakmile bude mít dost surovin, postoupí do železného věku, propustí jednotky z budov a zaútočí na nepřítele všemi svými jednotkami. Hodně z těchto akcí bude ve skutečnosti probíhat najednou, protože úkol je pouze zadání příkazu, nikoli už jeho vykonání.

6.1.2 Postup při tvorbě vlastní umělé inteligence

Vytváření vlastní umělé inteligence se z velké části skládá z implementace třídy, jež ji bude reprezentovat. Tato třída musí být potomkem třídy AI a bude pak pro své fungování moci využívat instanci třídy AIPlayer, takže uživatel si svou umělou inteligenci může naprogramovat úplně od základů.

Druhou možností je implementovat přímo potomka SimpleAI a využívat tedy seznam úkolů, s nímž jsme se seznámili výše. Metody CreateObjectives a UpdateObjectives jsou virtuální a uživatel si tedy může plně přizpůsobit seznam úkolů i chování inteligence v případě, že se úkol nepodaří splnit.

Ať už je inteligence vytvořena jedním či druhým způsobem, musí se poté ještě přidat do hry, k čemuž stačí pouze vytvořit nový objekt s touto komponentou a přiřadit jej v Editoru do proměnné *aiPrefab* komponenty *Factory* u předlohy objektu *Player*, kterou je možno nalézt ve složce Prefabs.

Závěr

V kapitole 1.1 jsme si definovali vlastnosti, jež by měla RTS splňovat, posoudíme tedy hru nejprve z tohoto hlediska.

Skillegy se odehrává v reálném čase a cílem hry je zničení všech protivníkových jednotek. Na mapě se nacházejí suroviny jídlo, dřevo a zlato, které fungují jako platidlo a jsou nezbytné pro budování armády a tudíž pro vítězství. Hráč může suroviny těžit nebo je získávat ve specializovaných budovách, mlýně, pile a bance.

Skillegy obsahuje osm typů budov, které jsou velice důležité, protože bez některých z nich nemůže hráč získat další jednotky ani žádná vylepšení. Také se do nich mohou slabší jednotky v případě útoku schovat. Hráč si může své budovy stavět po celé mapě a tímto způsobem si vytvořit i více základen. Ve Skillegy je také naimplementovaná zjednodušená verze fog of war, jež ale plní svůj účel a zabraňuje hráčům vidět části mapy, které ještě nejsou prozkoumány.

Vylepšování je ve Skillegy naprosto klíčovým principem a je na něm založen nápad se schopnostmi jednotek, který tuto hru odlišuje od jiných real-time strategií. Každá jednotka má pět schopností, které ovlivňují různé aspekty jejího fungování a postupně se zdokonalují.

Při pohledu na vlastnosti v 1.1 zjistíme, že jediná z nich, již jsme zdánlivě nesplnili, je diverzita jednotek a jejich různé typy. Přesto to však není pravdou, protože i když máme pouze jeden typ jednotek, jejich rozmanitost je zaručena pomocí různých úrovní schopností. Pokud by se ukázalo jako problém, že každá jednotka má pouze jednu schopnost ovlivňující boj, je snadné přidat jednotkám novou schopnost.

Z variant hry uvedených v téže kapitole Skillegy obsahuje hru ve více hráčích a velice jednoduchou hru pro jednoho hráče, která však slouží primárně jako demo pro rozhraní pro umělou inteligenci, jež bylo dalším cílem této práce. Toto rozhraní může hráč využít k vytvoření umělé inteligence podle svého přání.

Pořád je zde však místo pro zdokonalení, bylo by žádoucí například vytvořit variantu týmové hry nebo několik map a vítězných podmínek, ze kterých si hráči mohou vybrat. Ukázková umělá inteligence zvládá vyřešit jen velice jednoduché úkoly a i pathfinding někdy nereaguje přesně podle očekávání. Parametry hry také pravděpodobně nejsou ideálně odladěny, ale podle testování vypadají celkem rozumně a perfektní vyvážení parametrů je náročná a dlouhodobá práce.

Seznam použité literatury

[1] Dave Ellis. *Sid Meier's Civilization II: The Official Strategy Guide (Secrets of the Games Series)*. Prima Games, mar 1996. ISBN 0761501061.

[2] Richard Van Eck. Digital game-based learning: It's not just the digital natives who are restless. *Educause review*, 41(2):16, 2006.

[3] Prima Games. *The Sims 4: Prima Official Game Guide (Prima Official Game Guides)*. Prima Games, sep 2014. ISBN 0804162190.

[4] Richard Moss. Build, gather, brawl, repeat: The history of real-time strategy games, sep 2017. URL `https://arstechnica.com/gaming/2017/09/build-gather-brawl-repeat-the-history-of-real-time-strategy-games/`. [cit. 25.6.2019].

[5] Nathan Istvan. The RTS Genre and What Killed It, jun 2017. URL `https://geeks.media/the-rts-genre-and-what-killed-it`. [cit. 25.6.2019].

[6] Andrej Brabec. Legendární strategie Dune 2 vznikla v domnění, že nebude mít konkurenci, jul 2014. URL `https://www.idnes.cz/hry/magazin/retro-pohled-dune-2.A140711_112917_bw-magazin_anb`. [cit. 25.6.2019].

[7] Keith R.A. DeCandido. *Command & Conquer (tm): Tiberium Wars*. Del Rey, may 2007. ISBN 0345498143.

[8] Bart G. Farkas. *Warcraft III: Reign of Chaos Official Strategy Guide (Bradygames Take Your Games Further)*. Brady Games, jun 2002. ISBN 0744000807.

[9] Selby Bateman. *Unlock the Secrets of Total Annihilation*. GW Press, oct 1997. ISBN 1568939035.

[10] Rick Barba. *StarCraft II: Field Manual*. Insight Editions, nov 2015. ISBN 1608874508.

[11] André Ferreira. *Basics of Age of Empires 2: On a Journey to be Great or Simply to Win Your Friends*. apr 2016.

[12] Edwin Evans-Thirlwell and Edwin Evans-Thirlwell. The decline, evolution and future of the RTS, mar 2016. URL `https://www.pcgamer.com/the-decline-evolution-and-future-of-the-rts/`. [cit. 25.6.2019].

[13] Andrew Hayward. The 10 biggest esports of 2018 by total prize pool, Dec 2018. URL `https://esportsobserver.com/10-biggest-prize-pools-2018/`. [cit. 25.6.2019].

[14] Aram Cookson. *Unreal Engine 4 Game Development in 24 Hours, Sams Teach Yourself*. Sams Publishing, jun 2016. ISBN 0672337622.

[15] Ben Tristem. *Unity Game Development in 24 Hours, Sams Teach Yourself (2nd Edition)*. Sams Publishing, dec 2015. ISBN 0672337517.

[16] Chris Bradfield. *Godot Engine Game Development Projects: Build five ross-platform 2D and 3D games with Godot 3.0*. Packt Publishing, jun 2018. ISBN 1788831500.

[17] Unity3D vs Godot detailed comparison as of 2019, 2019. URL `https://www.slant.co/versus/1047/1068/~unity3d_vs_godot`. [cit. 29.6.2019].

[18] Brass Harpooner. Choosing a game engine: Unity vs Unreal Engine vs Godot, 2018. URL `https://cyberglads.com/making-cyberglads-1-choosing-a-game-engine.html`. [cit. 29.6.2019].

[19] Aras Pranckevičius. Releasing the Unity C# source code – Unity Blog, mar 2018. URL `https://blogs.unity3d.com/2018/03/26/releasing-the-unity-c-source-code/`. [cit. 29.6.2019].

[20] Paul Bettner and Mark Terrano. 1500 archers on a 28.8: Network programming in Age of Empires and beyond. *Presented at GDC2001*, 2:30p, 2001.

[21] treeform. Don't use lockstep in rts games, Oct 2016. URL `https://medium.com/@treeform/dont-use-lockstep-in-rts-games-b40f3dd6fddb`. [cit. 29.6.2019].

[22] Exit Games. Photon Bolt engine, perfect for Steam & PC. URL `https://www.photonengine.com/bolt`. [cit. 29.6.2019].

[23] Welcome to forge networking developers! URL `https://developers.forgepowered.com/`. [cit. 29.6.2019].

[24] Unity Technologies. Multiplayer overview, . URL `https://docs.unity3d.com/Manual/UNetOverview.html`. [cit. 29.6.2019].

[25] Unity Technologies. Navigation and pathfinding, . URL `https://docs.unity3d.com/Manual/Navigation.html`. [cit. 29.6.2019].

[26] Aron Granberg. A* pathfinding project. *Get Started with The A* Pathfinding Project*, 2014.

[27] Jordan Russel and Martijn Laan. Inno setup. URL `http://www.jrsoftware.org/isinfo.php`. [cit. 17.7.2019].

[28] Unity Technologies. Network lobby, Jul 2018. URL `https://assetstore.unity.com/packages/essentials/network-lobby-41836`. [cit. 16.7.2019].

[29] Unity Technologies. Spawning gameobjects, . URL `https://docs.unity3d.com/Manual/UNetSpawning.html`. [cit. 1.7.2019].

[30] Xiao Cui and Hao Shi. A*-based pathfinding in modern computer games. *International Journal of Computer Science and Network Security*, 11(1):125–130, 2011.

[31] Nils J Nilsson and Nils Johan Nilsson. *Artificial intelligence: a new synthesis*. Morgan Kaufmann, 1998.

Přílohy

Tabulka 2: Jednotky a budovy

Název	Věk	Jídlo	Dřevo	Zlato
Jednotka	Dřevěný	150	50	20
Hlavní budova	Dřevěný	1000	1000	1000
Dům	Dřevěný	0	500	0
Kasárny	Kamenný	0	1000	200
Ošetřovna	Železný	100	500	500
Knihovna	Kamenný	0	1000	1200
Mlýn	Dřevěný	0	1000	500
Pila	Kamenný	500	500	500
Banka	Železný	1000	1500	300

Tabulka 3: Vylepšení v hlavní budově

Název	Jídlo	Dřevo	Zlato
Stone Age	2000	0	2000
Iron Age	5000	1000	3000
Diamond Age	10000	2000	7000

Tabulka 4: Vylepšení v kasárnách

Název	Věk	Jídlo	Dřevo	Zlato	Nutná vylepšení	Šermířství
Gear 1	Kamenný	200	500	500	-	20
Gear 2	Kamenný	300	700	700	Gear 1, Books 1	35
Gear 3	Železný	700	1000	1500	Gear 2	55
Gear 4	Železný	1000	1500	2500	Gear 3, Books 2	75
Gear 5	Diamantový	2000	3000	6000	Gear 4, Books 4	100

Tabulka 5: Vylepšení v knihovně

Název	Věk	Jídlo	Dřevo	Zlato	Podmínky	Efekt
Books 1	II.	0	1000	1500	-	Inteligence 20
Books 2	III.	500	1500	2000	Books 1	Inteligence 35
Books 3	III.	1000	3000	3500	Books 2, 30 jednotek	Inteligence 55
Books 4	IV.	2000	5000	5000	Books 3, jeden mlýn	Inteligence 75
Books 5	IV.	3000	6000	7000	Books 4, Gear 5	Inteligence 100
Building Books	II.	500	1500	1500	Books 2, jedna pila	Odemkne zaměření na stavitelství

Tabulka 5: Vylepšení v knihovně

Název	Věk	Jídlo	Dřevo	Zlato	Podmínky	Efekt
Medicine Books 1	III.	1500	500	1500	Books 3, jedna ošetřovna	Odemkne zaměření na lékařství
Medicine Books 2	III.	2500	500	2500	Medicine Books 1	Zvýší maximální počet pacientů
Intelligence	II.	100	0	0	-	Zaměření na inteligenci
Building	II.	0	100	0	Building Books	Zaměření na stavitelství
Healing	III.	0	0	100	Medicine Books 1	Zaměření na lékařství

Tabulka 6: Vylepšení v mlýně

Název	Věk	Jídlo	Dřevo	Zlato	Nutná vylepšení
Flour	Kamenný	500	1000	1500	-
Bread	Železný	1000	1500	2000	Flour