Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

DIPLOMOVÁ PRÁCE

Strategická multiplayer hra po sítí

2021

Jiří Hausner

Vedoucí práce:
RNDr. Trnečka Martin, Ph.D.

Studijní obor: Aplikovaná informatika,
prezenční forma

Bibliografické údaje

Autor:	Jiří Hausner
Název práce:	Strategická multiplayer hra po sítí
Typ práce:	diplomová práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2021
Studijní obor:	Aplikovaná informatika, prezenční forma
Vedoucí práce:	RNDr. Trnečka Martin, Ph.D.
Počet stran:	69
Přílohy:	1 CD
Jazyk práce:	český

Bibliograhic info

Author:	Jiří Hausner
Title:	Multiplayer strategy networking game
Thesis type:	master thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2021
Study field:	Applied Computer Science, full-time form
Supervisor:	RNDr. Trnečka Martin, Ph.D.
Page count:	69
Supplements:	1 CD
Thesis language:	Czech

Anotace

Výsledkem práce je strategická multiplayer hra po síti vytvořena v herním enginu Unity3D. Hra obsahuje stavbu budov, nákup jednotek a jejich vylepšení a soubojový systém.

Synopsis

The result of this work is multiplayer strategy networking game made in Unity3D game engine. Game consists of construction of building, purchase of units and their upgrades and combat system.

Klíčová slova: strategická hra; multiplayer; hra po síti; C#; Unity3D

Keywords: strategy game; multiplayer; networking game; C#; Unity3D

Děkuji vedoucímu práce RNDr. Matrinu Trnečkovi, PhD. za vedení práce a cenné rady, které mi pomohly při návrhu a vývoji aplikace.

Místopřísežně prohlašuji, že jsem celou práci včetně příloh vypracoval samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

datum odevzdání práce podpis autora

Obsah

Seznam obrázků

Seznam zdrojových kódů

1 Úvod

Cílem práce bylo naprogramovat libovolnou strategickou hru v herním enginu `Unity3d`, která měla umožnovat hru po síti. Herní mechanismus měl obsahovat stavbu budov, jednotek, jejich vylepšování a soubojový systém. Hra si měla korektně poradit se všemi stavy sítě. Součástí práce měla být i technická dokumentace.

Práce je rozdělena do pěti kapitol. V první kapitole se zabývám popisem. V další kapitole se věnuji použitým technologiím. Třetí kapitola nastíní koncept síťové architektury. Čtvrtá kapitola obsahuje programátorskou příručku popisující klíčové třídy programu aplikace hry a aplikací serverové části práce. Poslední kapitola je věnována uživatelské příručce nastinující možnosti ovládání aplikací a průvedce uživatelským rozhraním aplikace hry. Nakonec je v příloze práce zahrnutý herní manuál.

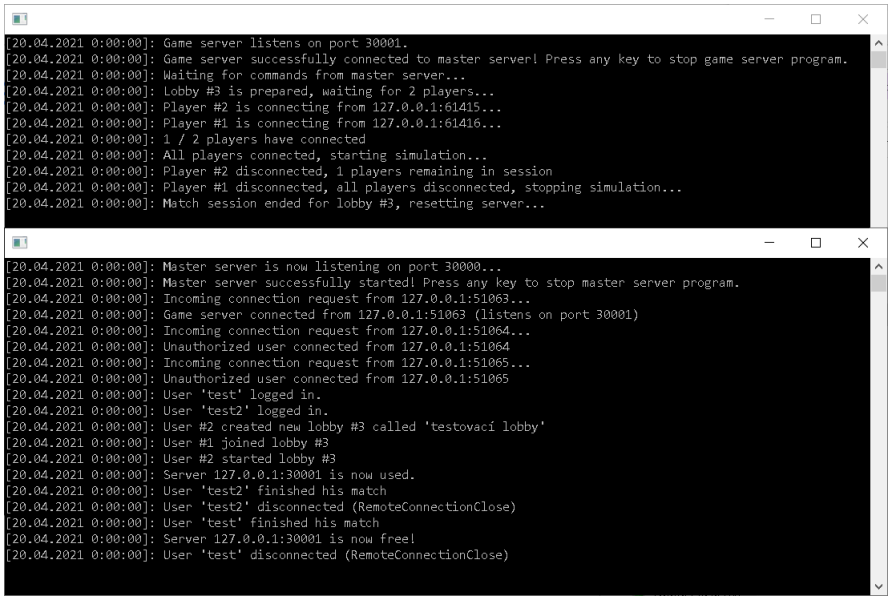
1.1 Strategická multiplayer hra

Strategická hra [1] (též strategie) je videoherní žánr, u kterého ovládáme větší množství herních objektů a manipulujeme s nimi s cílem dosažení vítězství. Obecně musí hráč plánovat řadu taktických kroků, ať už obraných nebo útočných, dopředu. Podmínkou vítězství v strategických hrách bývá zneškodnit herní objekty všech nepřátelských hráčů.

Existuje několik podžánrů strategických her jako například tahové strategie, budovatelské, deskové apod. V mé práci se zabývám implementací realtimové strategické hry podobné hrám jako jsou StarCraft nebo Age of Empires, jež je určena pro více hráču po síti.

1.2 Ukázky ze spuštěné aplikace

Obrázek 1: Ukázka ze samotné hry



Obrázek 2: Ukázka spuštěných aplikací serverové části

2 Použité technologie

2.1 JetBrains Rider

JetBrains Rider [2] je cross-platformní vývojové prostředí (IDE) od firmy JetBrains založené na technologii IntelliJ a Resharper, určené k vývoji aplikací nad platformou .NET. Kromě editaci a analýzy kódu, refaktorizaci, testování (unit) a debugování také nabízí integraci pro herní engine `Unity3D` v podobě pluginu. IDE také dokáže pracovat s mnoha projekty nad platformou .NET, cross-platformním .NET Core a platformou Mono. Studentskou verzi JetBrains Rider jsem použil k vývoji a debugovaní věškerého kódu spojeného s prací.

2.2 C#

C# [3] je vysokoúrovňový objektově orientovaný programovací jazyk vyvinutý firmou Microsoft spolu s platformou .NET Framework. C# lze využít k tvorbě databázových programů, webových aplikací a stránek, webových služeb, formulářových aplikací systému Windows a her. Jazyk se odvíjí od jazyků C++ a Java. V mé práci jsem využil jazyka C# verze 4.0.

2.3 Unity3D

Unity3D [4] je multiplatformní herní engine vyvinutý společností Unity Technologies. Používá se pro vývoj her pro PC, konzole, mobily a web. Engine poskytuje možnosti vývoje 2D i 3D her libovolného žánru a zaměření. Kromě grafického prostředí pro tvorbu, podporuje také tvorbu skriptů v jazyce C#. V práci jsem využil bezplatné verze pro vytvoření programu samotné hry.

2.4 LiteNetLib

LiteNetLib [5] je knihovna určená pro spolehlivou komunikaci po protokolu UDP. Nabízí jednoduchou správu spojení, peer to peer komunikaci, různorodé mechaniky přenosu dat (např. spolehlivé s pořadím, bez pořadí aj.), multicasting a hlavně podpora pro herní engine `Unity3D`. Knihovna byla použita pro veškeré síťování ve všech programech práce.

2.5 MySqlConnector

MySqlConnector [6] je ADO.NET poskytoval dat databází jako je MySQL Server, MariaDB, Percona Server, Amazon Aurora, Azure Database for MySQL, Google Cloud SQL for MySQL aj. Poskytuje implementace tříd potřebných k dotazování a aktualizaci databází. Knihovnu jsem použil pro navázání spojení a dotazovaní MySQL databáze v programu hlavního serveru.

3 Koncept síťové architektury

Při návrhu konceptu síťové architektury a psaní této kapitoly jsem čerpal informace ze zdrojů [7], [9] a [8].

3.1 Volba síťového modelu

Při volbě síťového modelu určeného pro multiplayer hry nikdy neexistuje jedno obecné řešení (ke každému typu hry se hodí jiná kombinace síťových modelů). Síťové modely se mohou dělit podle způsobu komunikace mezi uživateli – buď *Peer to Peer* (každý s každým), nebo klient-server (každý se serverem). Další rozdělení je podle typu přeposílaných dat – většina moderních her přeposílá stav herních entit (několikrát za vteřinu). Avšak u strategické hrách by tento způsob mohl znamenat obtíže, zejména po stránce zatížení připojení, protože by bylo potřeba zasílat stav všech jednotek a budov (těch můžou být stovky a více) několikrát během vteřiny. Tím se nabízí druhá varianta, která je u strategických hrách velice populární, a sice přeposílat pouze vstupy od uživatele. Tak snížíme šířku přenosového pásma a jediné co musíme zaručit je, že po přijetí vstupů od všech uživatelů zůstane hra synchronizovaná u všem uživatelů. Toho docílí tzv. *lockstep model* (viz další podkapitola).

3.2 Lockstep model

Hlavní myšlenkou `Lockstep modelu` je, že herní logika pokročí pouze pokud byly přijaty vstupní příkazy (přesun jednotky, postavení budovy apod.) od všech uživatelů, čímž zaručuje, že bude stav hry na všech zařízeních stejný (vyžaduje determinismus). Jednoduchou analogií je tahová hra, akorát tahy probíhají každých 100ms (pokud uživatel neprovedl žádnou akci, je odeslán prázdný příkaz).

Samotná simulace je rozdělena na tzv. *lockstep kroky* (případně kola) – analogie tahu. Během `lockstep kroku` shromažďujeme vstupy daného uživatele, které na konci *lockstep kroku* odešleme všem uživatelům. Ty jsou po přijetí naplánované k vykonání několik *lockstep kroků* do budoucna. Pokud simulace dojde do lockstep kroku, který ještě neschromáždil příkazy od všech uživatelů, dojde k okamžitému zastavení hry, které trvá, dokud všechny příkazy neschromáždí. Každý lockstep krok se skládá z několika *herních snímků*, které slouží k aktualizaci herní scény. Čím více těchto snímků během vteřiny je vykonáno, tím plynulejší simulace bude.

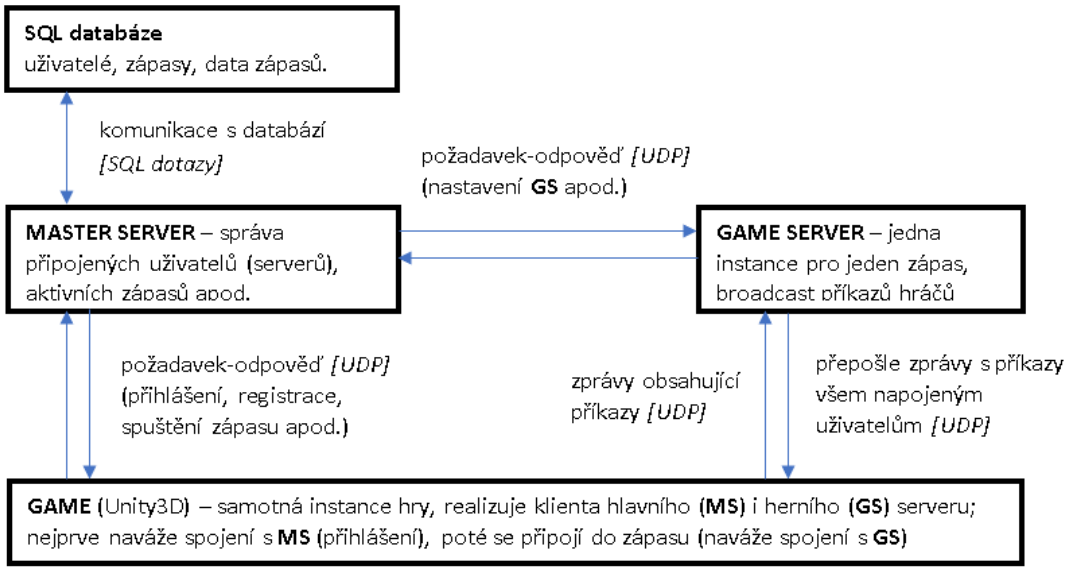
Použitím tohoto modelu získáváme více výhod než jen snížené zatížení připojení. Jednou z nejaktrativnějších výhod je, že program multiplayer hry se vůbec neliší od programu hry pro jednoho hráče. Další výhodou je možná detekce podvádění, jelikož uživatel, který by ve hře podváděl byl rychle desynchronizovaný a tím pádem odstraněný ze hry.

3.3 Schéma zvolené síťové architektury

V minulosti využívaly strategické a bojové hry při použití *lockstep* architektuře tzv. *Peer to Peer* model. To však sebou přinášelo spoustu nevýhod a problémů. Problémové v tomto přístupu bylo už jen napojení všech uživatelů do stejné hry (potřeba veřejné IP adresy, NAT punching apod.).

Tento zastaralý přístup jsem v práci nahradil `klient-server modelem`. Každý uživatel se tedy připojuje a komunikuje pouze s herním serverem, který slouží jako relé pro přeposílaní zpráv s příkažy. Tím se odstraní výše uvedené problémy a navíc se tím naskytne možnost dynamického nastavování délky lockstep kroku.

Do práce navíc přidávám tzv. *hlavní server*, který se stará o správu uživatelů, herních serverů a jejich přiřazování k samotným hrám. Navíc jsou veškeré tyto data ukládány v databázi (viz obrázek 3).



Obrázek 3: Schéma zvolené síťové architektury

3.4 Řešení problémů lockstep modelu

V tradičním síťovém modelu *lockstep* jsou přeposílány pouze vstupy uživatele, což nechává na každém uživateli, aby simulaci podle příkazů od všech uživatelů deterministicky aktualizoval. To není nejjednotušší úkol, jelikož některé herní logiky je velice obtížné vyřešit deterministicky. Navíc musíme zaručit, že všechny tyto procesy udrží stejné pořadí vykonávání.

Nedeterministické jsou zároveň některé aritmetické operace čísel s plovoucí řádovou čárkou. Protože je většina herních enginů postavena na aritmetice čísel s plovoucí řádovou čárkou (včetně herního enginu `Unity3D`), bylo potřeba nahradit některé části enginu vlastním deterministickým řešením. Vytvořil jsem tedy herní simulaci využívající celočíselné aritmetiky (s použitím násobku, čímž simuluji nekolik desetinných míst – dostatečné pro strategické hry) spolu se systémy

detekce kolizí, jednoduché fyziky herních objektů a systémem hledání nejkratší cesty.

Pokud neexistuje jiné než nedeterministické řešení k danému problému, je proces tohoto problému vykonán pouze na straně jednoho uživatele, který jeho výsledek odešle všem ostatním uživatelům jako příkaz (takto jsou řešeny všechny výpočty zahrnující náhodu). Tím zůstane simulace deterministická u všech uživatelů.

Dalším problémem *lockstep modelu* jsou záseky, případně uplné zastavení. Jelikož *lockstep model* vždy čeká na příkazy od všech uživatelů, může nastat situace, kdy uživatelovi příkazy nejsou doručeny před pokusem o vykonání daného lockstep kroku. Tím je celá simulace zastavena, dokud zpráva s příkazy nebude doručena, přičemž vznikne viditelný zásek (viditelný především na realtimové hře). Tento problém lze vyřešit plánováním příkazů několik lockstep kroků dopředu (v práci jsem pevně nastavil na tři kroky dopředu) a mechanizmem, který dynamicky mění délku `lockstep kroku` podle odezvy uživatele s nejpomalejším připojením. Navíc jsem aplikaci rozdělil na simulační vrstvu, která deterministicky vypočítává celou hru, a prezentační vrstvu, která pouze interpoluje aktuální stav simulace uživateli (tím je průběh celé hry plynulejší).

Posledním problémem, se kterým jsem se při použití *lockstep modelu* setkal, bylo řešení pádu klienta. Jakmile jeden z uživatelů ztratí spojení s serverem, ostatní uživatelé čekají, dokud toto spojení znovu nenaváže. Avšak k tomu nemusí dojít. Tím se nabízí nejlehčí (a zároveň nejpoužívanější) řešení – daného uživatele odebrat ze simulace a nenávratně bez něj pokračovat.

4 Programátorská příručka

4.1 Struktura projektů

Práce je rozdělena na 5 částí – projekt klienta jakožto aplikace herního enginu `Unity3D`, projekt hlavního serveru, projekt herního serveru, knihovna sdíleného kódu napříč projekty a struktura databáze napsaná v jazyce SQL.

4.2 Popis tříd aplikace hry

Tato podkapitola se věnuje popisu významných tříd projektu knihovny samotné hry, která je kompilována v prostředí editoru herního enginu `Unity3D`.

4.2.1 Třída AppManager

Třída `AppManager` je nejdůležitejší třída celé aplikace, spravuje celou aplikaci a jako jediná třída zůstává živá po celou dobu spuštění aplikace. Přes instanci této třídy se dá dostat ke všem objektům využívaných ve hře. Kromě toho si taky uchovává data o přihlášeném uživateli, jeho statistiky, seznam vytvořených herních lobby a informace o herním lobby, ve kterém se uživatel aktuálně nachází.

Třída je rozdělena do čtyř sektorů – *monobehaviour metody* (ty se starají o chod komponenty), *hlavní metody* (spuštění a zastavení simulace, načtení úrovně, změna uživatelského rozhraní apod.), metody spravující *údálosti připojení k hlavnímu serveru* a *pomocné metody*.

Při spuštění hry dojde k zavolání metody `Awake()`. Během jejího vyhodnocení se nejprve přichystají všechna uživatelská rozhraní, připraví se logování, pokusí se připojit na hlavní server a zobrazí přihlašovací uživatelské rozhraní. Herní engine během chodu aplikace volá metodu `Update()`, při které se kromě aktualizace simulace aktualizuje i spojení jak s hlavním serverem, tak i se serverem herním, pokud je uživatel uvnitř nějaké hry.

4.2.2 Třída MasterServerListener

Objekt třídy `MasterServerListener` se stará o navázání spojení a komunikaci s hlavním serverem. Třída definuje několik akcí, které fungují jako reakce na přijatou zprávu od hlavního serveru (např. při přijetí dat o uživateli, dat o všech herních lobby apod.). Jako klient hlavního serveru v této třídě používám objekt třídy `Client`, o které je více napsáno v podkapitole 4.7.1.

Důležitou metodou je `Connect()`, jejímž zavoláním dojde k pokusu o připojení se na hlavní server pomocí jeho IP adresy a portu, který definujeme pomocí argumentů. Další metodou, která stojí za zmínku, je `OnDataReceived()`, ve které dochází k zavolání správné události podle druhu přijaté zprávy.

4.2.3 Třída GameServerListener

Třída `GameServerListerer` pracuje na podobném principu jako třída `MasterServerListener`, avšak místo spojení a komunikaci s hlavním serverem, pracuje s servery herními.

Mezi významné metody patří metoda `SendCommands()`, která serializuje seznam příkazů lokálního uživatele v aktuálním lockstep kroku pro odeslání na herní server, a metoda `BuildCommandList()`, která deserializuje zprávu obsahující seznam příkazů od herního serveru, jež se mají naplánovat k vykonání během daného lockstep kroku.

4.2.4 Třída Simulation

Instance třídy `Simulation` se stará o připravení a chod simulace hry. Jejím hlavním úkolem je správa lockstep manažéra a veškerých herních entit. Zároveň poskytuje detektor kolizí a manažéra vyhledávání nejkratších cest.

Při aktualizaci herního enginu je zavolána metoda `Update()`, ta však aktualizuje pouze lockstep manažéra. Samotná aktualizace simulace je zavolána metodou `UpdateSimulation()`, která je volána skrz událost lockstep manažéra `OnUpdate` (viz zdrojový kód 1).

Zdrojový kód 1: Průběh aktualizace simulace

```
private void UpdateSimulation(int frameTime)
{
    // vymažeme odstraněné entity
    DeleteRemovedEntities();

    // vložíme přidané entity
    InsertAddedEntities();

    // aktualizujeme pozice pohybujících se entit
    UpdatePositionOfMovingEntities();

    // aktualizujeme logiku všech entit
    foreach (var entity in Entities)
    {
        entity.SimulationUpdate(frameTime);
    }

    // aktualizujeme stav hry
    Game.SimulationUpdate(frameTime);
}
```

Další zajímavou metodou je metoda `LoadGameObjects()`, během které dochází k načtení herních objektů herního enginu `Unity3D` určených k inicializaci simulace (spawnery hlavních budov, herní entity jako jsou surovinové zdroje apod.).

Zbylé metody poskytují rozhraní pro práci s herními entitami jako je například přidání nové entity, odebrání stávající, vyhledání entity podle jejího id apod.

4.2.5 Třída GameEntity

Abstraktní třída `GameEntity` slouží pro inicializaci, správu, aktualizaci a terminaci herních entit v simulaci. Definuje způsob, jakým všechny herní entity vykonávají tyto činnosti (viz rozhraní IGameEntity zobrazené ve zdrojovém kódu 2).

Zdrojový kód 2: Rozhraní herní entity

```
public interface IGameEntity
{
    // vlastnosti
    int Id { get; }
    int PlayerId { get; }
    Vector2Int Position { get; set; }
    Vector2Int Velocity { get; set; }
    Bounds Bounds { get; }

    // zavoláno po přidání entity do simulace
    void Initialize();

    // volané při každé aktualizaci simulace
    void SimulationUpdate(int frameTime);

    // zavoláno po odstranění entity ze simulace
    void Terminate();
}
```

4.2.6 Třída LockstepManager

Třída `LockstepManager` se stará o synchronizaci simulace napříč všemi hráči dané relace. Při vytvoření nové simulace se vytvoří nový objekt této třídy, přičemž se přichystá první lockstep krok (s využitím nejvyšší hodnoty odezvy připojení k serveru mezi hráči). Následně herní engine `Unity3D` opakovaně volá metodu `Update()`, která se stará o aktualizaci lockstep modelu i při nízkých snímcích za sekundu. Toho se docílí tak, že se při každém herním snímku akumuluje délka jeho výpočtu a vykreslení, a pokud tato akumulovaná hodnota dosáhne délky snímku aktuálního lockstep kroku, dojde k aktualizaci lockstep modelu zavoláním metody `UpdateLockstep()` (viz zdrojový kód 3) a akumulovaná hodnota se sníží o délku snímku daného lockstep kroku.

Zdrojový kód 3: Aktualizace lockstep modelu

```
private void UpdateLockstep()
{
    // aktualizace kroku dochází pouze při prvním snímku
    if (IsFirstFrameInTurn)
    {
        // odešleme příkazy, které uživatel zadal během posledního kroku
        SendIssuedCommands();

        // protože se všechny příkazy plánují několik kroků dopředu,
        // tak zde musíme udělat výjimku pro prvních několik kroků
        if (_currentTurn >= TurnBuffer.TurnsInAdvance)
        {
            // zastavíme simulaci, pokud není daný krok připraven
            if (!IsReady)
            {
                StartWaitingForOtherPlayers();
                return;
            }

            // vykonáme příkazy v aktuálním kroku
            ProcessCurrentTurn();
        }

        // postoupíme do dalšího kroku
        AdvanceToNextTurn();

        // vypočítáme délku dalšího kroku a délku jeho snímků
        CalculateTurnAndFrameLengths();
    }

    // aktualizujeme simulaci
    UpdateRuntime();
}
```

Při výkonu metody `UpdateRuntime()` kromě aktualizace simulace stopujeme délku času trvání výpočtu a uchováváme si největší stopovanou hodnotu po dobu daného lockstep kroku. Tuto hodnotu spolu s aktuální odezvou připojení k hernímu serveru odesíláme spolu s vydanými příkazy daného uživatele. Každý uživatel si tak vypočítá maximální odezvu aktualizace simulace a připojení k serveru, pomocí kterých nastaví parametry pro výpočet dalšího lockstep kroku.

Pokud chce uživatel zadat nový příkaz, zavolá metodu `IssueCommand()`, kde nový příkaz uvede jako první argument.

4.2.7 Třída TurnBuffer

Tato třída slouží k bufferování několika lockstep kroků do budoucnosti. Obsahuje pole objektů spravujících lockstep kroky (více v dalším odstavci), pole aktuálně vypočítané odezvy každého hráče a pole aktivních hráčů. Důležitou metodou je

metoda `InsertCommands()`, která se stará o zařazení dat příkazů daného hráče do správného lockstep kroku. V okamžiku, kdy se *lockstep manažér* dostane do fáze vykonání aktuálního lockstep kroku, je zavolána metoda `ProcessCurrentTurn()`. Ta vykoná všechny příkazy aktuálního lockstep kroku a vypočítá odezvy všech uživatelů. Třída je připravena řešit odhlášení hráče ze hry, přičemž je volána metoda `RemovePlayer()`. Ta vynuluje data lockstep kroků daného hráče a odebere ho z výpočtu maximální odezvy. Zároveň třída definuje vlastnost `IsReady`, která je pravdivá, pokud v aktuálním lockstep kroku dostal uživatel příkazy od všech aktivních uživatelů.

Třída obsahuje podtřídu `Turn`, která reprezentuje samotný lockstep krok. Tato podtřída obsahuje pole seznamů přijatých příkazů a pole odezvy každého hráče. Mezi hlavní metody třídy patří `InsertCommands()`, která do daného lockstep kroku doplní správnému hráči jeho příkazy a uloží jeho odezvu, a metoda `ExecuteCommands()`, která uložené příkazy vykoná. Objekt této třídy zároveň hlídá počet přijatých příkazů. Po vykonání daného *lockstep kroku* je objekt vynulovaný zavoláním metody `Reset()` (třída `TurnBuffer` uchovává omezený počet objektů třídy `Turn`, která použití těchto objektů recykluje z důvodů optimalizace).

4.2.8 Rozhraní ICommand a třídy realizující příkazy

Rozhraní ICommand definuje zapouzdření tříd, které realizují herní příkazy (přesun jednotky, postavení budovy apod.). Rozhraní definuje getter vlastnosti udávající typ příkazu, metodu `Execute()`, která slouží k vykonání daného příkazu, a metody `Serialize()` a `Deserialize()`, jež jsou určené k serializaci a deserializaci pro další zpracování. Ukázka takových příkazů je k dispozici ve zdrojových kódů 4 a 5.

Zdrojový kód 4: Vykonání příkazu přesunu jednotky

```
public override void Execute()
{
  // najdeme herní entitu, jež má být přesunuta
  var entity = AppManager.Simulation.GetEntityById(ObjectId);

  // nastavíme cíl dané jednotce
  ((Unit)entity)?.Target(Position);
}
```

Zdrojový kód 5: Vykonání příkazu nákupu v budově

```
public override void Execute()
{
    // najdeme herní entitu budovy podle jejího id
    var building = AppManager.Simulation.GetEntityById(BuildingId);
    if (building == null)
        return;
```

```
    // získáme objekt spravující daného hráče
    var player = AppManager.Simulation.Game
                    .GetPlayer(building.PlayerId);

    // zkontrolujeme hráčovi prostředky nutné k nákupu
    if (player.Wood < WoodCost || player.Stone < StoneCost ||
        player.Gold < GoldCost)
        return;

    // snížíme počet prostředků daného hráče
    player.Wood -= WoodCost;
    player.Stone -= StoneCost;
    player.Gold -= GoldCost;

    // provedeme samotný nákup
    ((Building)building).PurchaseItem(ItemId);
}
```

4.2.9 Třída CollisionDetector

Pro správu kolizí je v programu připravena třída `CollisionDetector`. Jelikož hra může obsahovat velké množství herních entit, u kterých je nutné kontrolovat kolize, bylo nutností ulevit každé entitě seznamem potencionálních kolizí.

Z tohoto důvodu jsem zvolil strukturu *quad tree* [10], která herní pole rozděluje vždy na čtyři stejně velké kvadranty, čímž se sníží počet entit, které budou kontrolovány. Každá entita může být součástí více kvadrantů (ten v programu nese název `Node` – uzel).

Při přidání herní entity do stromu se nejdříve zkontroluje, zdali náleží danému uzlu (to vždy platí pro kořenový uzel). Pokud uzlu náleží, tak jej do uzlu přidáme. Překročí-li počet entit v daném uzlu maximální hodnoty, dojde k rozdělení uzlu na čtyři pod-uzly, do kterých jsou entity daného uzlu přesunuty. Analogicky struktura pracuje s odebíráním entit. Po odebrání entity je cestou zpět ke kořenu stromu kontrolován počet entit. Pokud je v nějakém uzlu počet entit roven nule a daný uzel má potomky, budou poduzly daného uzlu sjednoceny v jeden.

Pokud chce entita získat seznam kolizí, stačí zavolat metodu `GetCollisions()`. Ta projde strukturou stromu směrem k listům a přidává do seznamu entity, se kterými koliduje. Viz zdrojový kód 6.

Zdrojový kód 6: Vyhledání potencionálních kolizí

```
public void GetCollisions(
    Bounds bounds, ICollection<IGameEntity> collisions)
{
    // zkontrolujeme, zdali náš čtverec náleží tomuto uzlu
    if (!_bounds.Intersects(bounds))
        return;

    // pokud má uzel potomky, převedeme vyhledání na ně
```

```
    if (HasChildNodes)
    {
        _topLeftChild.GetCollisions(bounds, collisions);
        _topRightChild.GetCollisions(bounds, collisions);
        _bottomLeftChild.GetCollisions(bounds, collisions);
        _bottomRightChild.GetCollisions(bounds, collisions);

        return;
    }

    // jinak zkontrolujeme kolize se všemi entitami v uzlu
    foreach (var entity in _entities)
    {
        if (bounds.Intersects(entity.Bounds))
            collisions.Add(entity);
    }
}
```

Problém nastává, když entita změní svoji pozici (například posunem). Poté je nutné entitu před samotnou změnou pozice ze stromu odebrat, provést změnu pozice a nakonec entitu znovu do stromu přidat.

Samotná detekce kolize je pak jednoduchý *AABB test*, což je vlastně zjištění, zdali dva čtverce mají společný průnik (čtverec je v projektu definován jako instance třídy `Bounds`) – viz zdrojový kód 7.

Zdrojový kód 7: Detekce kolize (AABB test)

```
public bool Intersects(Bounds other)
{
    return _min.x <= other._max.x && _max.x >= other._min.x &&
           _min.y <= other._max.y && _max.y >= other._min.y;
}
```

4.2.10 Třída PathFinder

Třída `PathFinder` poskytuje rozhraní pro vyhledávání nejkratších cest po úrovni. Tato funkcionalita je založena na variaci algoritmu A* [11]. Řešení algoritmu nabízí nejkratší cestu v osmi směrech s preferencí diagonálních cest (to zaručí plynulejší pohyb jednotek po mapě). Pro vyhledání cesty je volána metoda `FindShortestPath()`, která buď najde nejkratší cestu, nebo taková cesta neexistuje. Pokud je cíl cesty obsazený překážkou, vyhledáme nejdříve nový nejbližší cílový bod, který je volný.

Kromě nalezení nejkratší cesty tato třída nabízí i rozhraní pro dynamické přidávání a odebírání překažek, za které jsou v případě této aplikace brány budovy hráčů. Při inicializaci herních entit, které lze považovat za překážku, je volána metoda `TakeSpace()`. Pro test, zdali je pozice volná, je volaná metoda `IsSpaceTaken()`. Pro případ, že byla herní entita odstraněna, je zde dostupná metoda `FreeSpace()`, která uvolní místo po překážce.

4.2.11 Třída Game

Třída nesoucí název `Game` se stará o stav hry. Uchovává si seznam objektů třídy `Player` a aktualizuje jej při každé aktualizaci simulace. Stejně jako třída `GameEntity` i tato třída implementuje podobné rozhraní – řeší inicializaci, aktualizaci i terminaci pouze přes objekt třídy `Simulation`.

4.2.12 Třída Player

Tato třída spravuje zvolenému hráči jeho statistiky (počet surovin, počet ztracených jednotek, počet postavených budov apod.), hlídá jeho stav, jestli hráč již nevyhrál nebo neprohrál, spravuje hráčovy budovy a zakoupené vylepšení. K vyhodnocení procesů v této třídě dochází vždy při aktualizaci simulace (při aktualizaci objektu hry).

4.2.13 Třída CameraHandler

Jedná se o třídu, která zajišťuje ovládání pohledu uživatele v aktuální herní scéně. Uživatel může ovlivňovat pozici pohledu a může pohled přiblížit a oddálit.

Při aktualizaci herního objektu kamery se zavolá metoda `Update()`, která nejdřív aktualizuje pohyb kamery (směr pohybu získá otestováním vstupu od uživatele), potom aktualizuje přiblížení pohledu kamery a nakonec omezí pozici kamery podle velikosti aktuální herní úrovně (specificky podle velikosti terénu aktuální herní scény).

4.2.14 Třída Selection

Hlavním účelem třídy `Selection` je výběr a ovládání herních entit patřících danému uživateli. Funkcionalita je rozdělena na dvě části – správa selekce jednotek a budov uživatele a správa režimu stavění nové budovy. Většinu této práce odvádí metoda `Update()`, jež je zavolána během každého herního snímku.

Pokud uživatel nestaví žádnou budovu, nachází se třída `Selection` v režimu výberu herních entit. O to se stará metoda `UpdateSelection()`, která nejprve kontroluje vstupy uživatele (myš, klávesnice), jestli nedochází k výběru herních entit, následně hlídá, zdali uživatel nezrušil aktuální výběr, a nakonec kontroluje, jestli uživatel nedal nějaký příkaz vybraným entitám (pouze pro entity typu jednotky) – viz metoda `CommandSelectedUnits()`. Pokud uživatel při mnohonásobném výběru zvolí více druhů herních entit a je mezi daným výběrem alespoň jedna jednotka, tak se pro ostatní herní entity výběr zruší (preference výběru jednotek). Pokud dojde k výběru budovy, zobrazí se nákupní menu dané budovy. Pokud uživatel zakoupí stavbu některé z budov, zavolá se metoda `BeginBuilding()` a třída přejde do režimu stavby budovy.

Přepnutím do režimu stavby budovy dojde nejdříve ke zrušení aktuálního výběru, připravíme stavbu budovy, zabráníme uživateli interakce s uživatelským rozhraním a zobrazíme placeholder stavěné budovy na místo kurzoru uživatele. Následovně je při aktualizaci třídy `Selection` volána metoda `UpdateBuilding()`,

která reaguje na vstupy uživatele během stavění budovy – pokud uživatel přesune kurzor, přesune se placeholder, pokud je placeholder na nežádoucím místě, zabráníme uživateli dokončit stavbu (přebarvění placeholderu na červeno). Jestliže je placeholder na vhodném místě a uživatel dokončí stavbu kliknutím levého tlačítka myši, dokončíme stavbu zavoláním metody `FinishBuilding()`, která dokončí stavbu a přepne třídu znovu do režimu výběru.

4.2.15 Třída Targetable

Jde o abstraktní třídu rozšiřující funkcionalitu herních entit simulace (rozšiřují třídu `GameEntity`). Každý herní objekt, se kterým může hráč interagovat, by měl dědit od této třídy.

Tato třída řeší veškerou funkcionalitu od správy bodu životů dané herní entity, správy zvukového zdroje a přehrávání zvuků, vykreslení lišty s body zdraví, až po přichystání herního objektu k vykreslení při průchodu druhé kamery (vykreslení minimapy).

Všechny třídy dědící od třídy `Targetable` se stanou komponentou objektu herního enginu `Unity3D`, takže je tato třída jakýmsi mezníkem mezi simulační části lockstep modelu a prezentací herním objektem ve scéně herního enginu `Unity3d`.

4.2.16 Třída Unit

Jedná se o abstraktní třídu dědící od třídy `Targetable`. Tato třída implementuje veškerou herní logiku, která by se dala očekávat od jednotek ve hrách typu strategie – správa atributů herní jednotky (síla útoku, rychlost pohybu apod.), správa animací modelu jednotky, funkcionalita vyhledávání nepřátelských entit v okolí jednotky, soubojová logika, systém nahánění cílové nepřátelské entity aj.

Při aktualizaci simulace se nejprve aktualizuje systém nahánění (soukromá metoda `UpdateTargeting()`). Pokud jednotka nemá vybraný cílový objekt, zkusí takový vyhledat v blízkém okolí, jinak je jednotka vedena směrem k cílovému objektu, přičemž při změně pozice cílového objektu dochází k přepočítání trasy. Následuje aktualizace efektů, kterými daná jednotka disponuje (jsou-li takové). Nakonec se aktualizuje soubojový a obsazovací systém, má-li jednotka vybraný cíl (metoda `UpdateAttackingAndConquering()`).

Při aktualizaci prezentační stránky, ke které dochází častěji jak u stránky simulační, se pouze interpoluje simulovaná pozice na prezenční, aktualizují se animace modelu jednotky a daný herní objekt se natočí směrem k cílovému objektu nebo podle směru jeho trasy přesunu.

4.2.17 Třída Building

Další abstraktní třídou, která dědí od třídy `Targetable`, je třída `Building`. Ta se zabývá herní logikou, na které staví všechny hráčem postavitelné budovy ve hře.

Hlavní mechanikou, kterou třída `Building` disponuje, je fronta nákupního košíku budovy spolu se seznamem nabízených předmětů. Tento seznam může každý potomek této třídy doplnit přepsáním metody `Initialize()` o libovolný předmět definovaný objektem třídy `BuyMenuItem`. Ten definuje id a typ předmětu, jeho název, dobu výroby, jeho cenu apod.

Při zavolání metody aktualizující simulaci budovy pouze aktualizuji frontu nákupního košíku pomocí metody
`UpdateItemQueue()`. Tato metoda odečte uplynulý čas od zbývajícího času potřebného k dokončení výroby předmětu a pokud se tento čas rovná nule provede výrobu předmětu (spawnutí jednotky, převod surovin apod.).

Významnou metodou je `PurchaseItem()`, která se volá po vykonání příkazu zakoupení předmětu (je tedy výhradně volaná skrz simulaci). Při jejím vykonání se přidá daný prvek do nákupní fronty.

4.2.18 Třída ResourceSource

Stejně jako `Building` i tato abstraktní třída dědí od třídy `Targetable`. Její hlavní úděl je implementace logiky využité u budov surovinýh zdrojů. Ty využívají dvou systémů – obsazování a sběr surovin. Systém obsazování je postavený na tabulce bodů obsazení (zdroj surovin vlastní hráč s nejvíce body). Pokud hráč začne obsazovat zdroj, který vlastní jiný hráč, tak nejprve odebírá body nepřátelského hráče a pak teprve přidává body do svého políčka.

Během aktualizace simulace se musí aktualizovat systém obsazování a systém sběru surovin. Při aktualizaci obsazování (metoda `UpdateConquering`) se kontroluje stav tabulky obsazení. Pokud je zdroj surovin vlastněn nějakým hráčem a jeho hodnota nedosahuje maximální hodnoty, je mu vlastnictví odebráno. Pokud je hráčova jednotka v dostatečné vzdálenosti od budovy zdroje surovin, zavolá metodu `Conquer()`. Ta provede přidání, respektive odebrání bodu cizího hráče způsobem popsaným výše. Pokud po přídání bodu dosahuje hráčova hodnota v tabulce obsazení maximální hodnoty, získá hráč vlastnictví zdroje surovin.

Při aktualizaci sběru surovin (metoda `UpdateResourceGathering()`) se odečítá čas potřebný k sběru další várky surovin. Pokud je tento čas roven nule dojde k zavolání metody `GatherResources()`, která vyzvedne novou várku surovin. Pokud zdroj surovin nevlastní žádný hráč, aktualizace systému sběru surovin neprobíhá.

4.2.19 Třída Effect

Tato abstraktní třída definuje vlastnosti a způsob interakce s hráčovými, případně protihráčovými jednotkami. Efekty jsou poté přiřazeny třídám dědícím od třídy `Unit` během vykonávání metody `Initialize()`. Samotný systém aktualizace efektů je vykonán při aktualizaci simulace ve třídě `Unit`.

4.2.20 Třídy uživatelských rozhraní

Všechny třídy uživatelských rozhraní dědí od abstraktní třídy `MenuScreen`. Tato abstraktní třída požaduje po dědících třídách, aby implementovaly metody `OnEnabled()` a `OnDisabled()`. Metoda `OnEnabled` připravuje elementy uživatelského rozhraní k práci s daným uživatelským rozhraním a metoda `OnDisabled()` tyto elementy bezpečně odstraní.

Veškerá data, se kterými by mohlo uživatelské rozhraní pracovat, jsou dostupná přes třídu `AppManager`. Pokud má dojít k přechodu na jiné uživatelské rozhraní, zavolá se statická metoda `SetScreen()` instance třídy `AppManager`, která následovně deaktivuje aktuální uživatelské rozhraní a aktivuje nové.

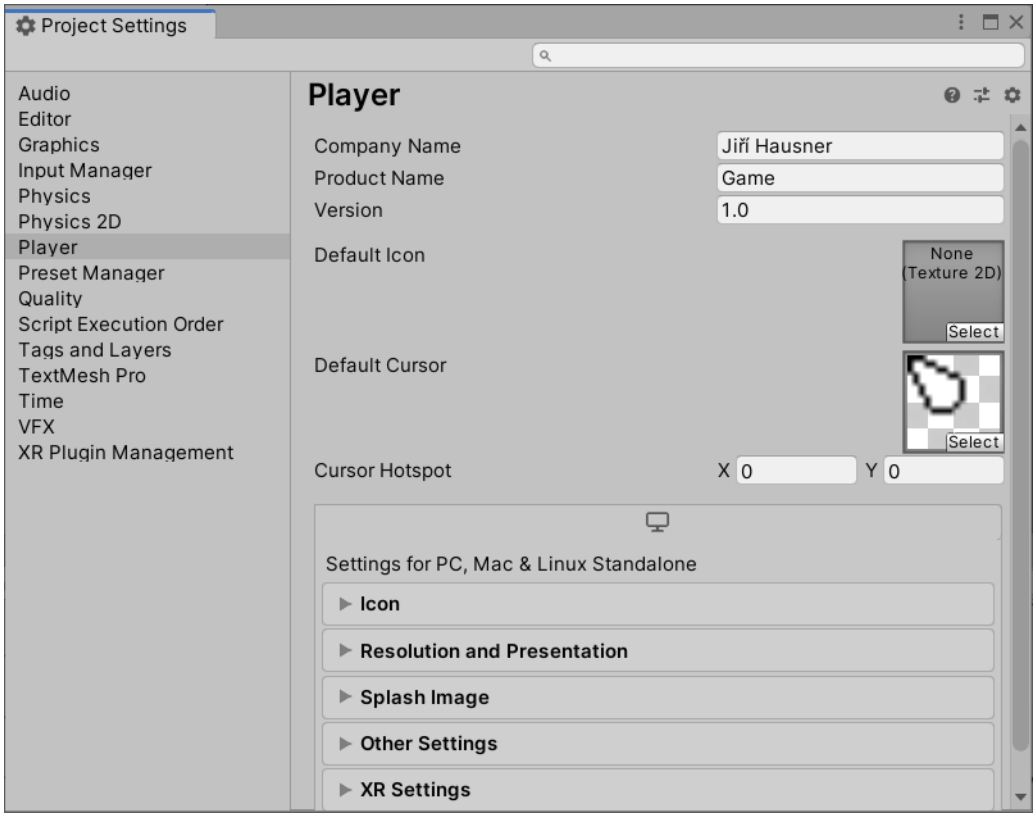
Nejzajímavější třídou uživatelského rozhraní je `GameScreen`. Tato třída kromě základní práce s uživatelským rozhraním definuje funkcionalitu aktuálního stavu surovin hráče, správu nákupní nabídky budovy, zobrazení a práci s minimapou, herní chat, správa kurzorů apod.

4.3 Integrace do herního enginu Unity3D

Při tvorbě této podkapitoly a samotné hry jsem čerpal informace z oficiálního manuálu [12] k hernímu enginu `Unity3D`.

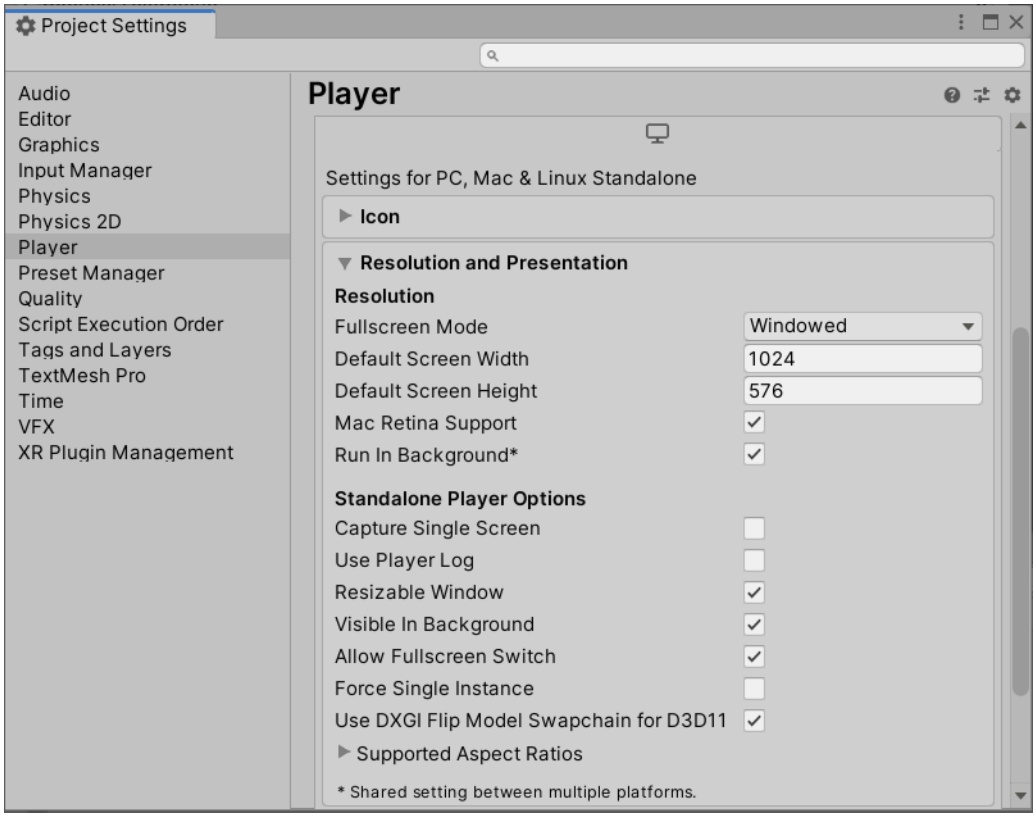
4.3.1 Nastavení herního projektu

Po vytvoření nového herního projektu nastavíme přehrávač (player) v okně `Project Settings`, které je dostupné pomocí stisknutí položky Edit→Project Settings v hlavním nabídkovém pruhu. Nejprve nastavíme základní vlastnosti projektu (název, verze, výchozí kurzor – viz obrázek níže).



Obrázek 4: Nastavení projektu v Unity3D

následně je potřeba nastavit výchozí rozlišení aplikace a nastavit předvolby samotné aplikace (viz obrázek níže). Výchozí rozlišení aplikace je v poměru 16:9, jelikož je uživatelské rozhraní relativně rozloženo přávě v poměru 16:9. Dalším důležitým krokem je povolit chod aplikace na pozadí (*Run In Background*), vypnout logování (*Use Player Log*), povolit změnu velikosti rozlišení a vypnout vynucené samotné instance aplikace (*Force Single Instance*).

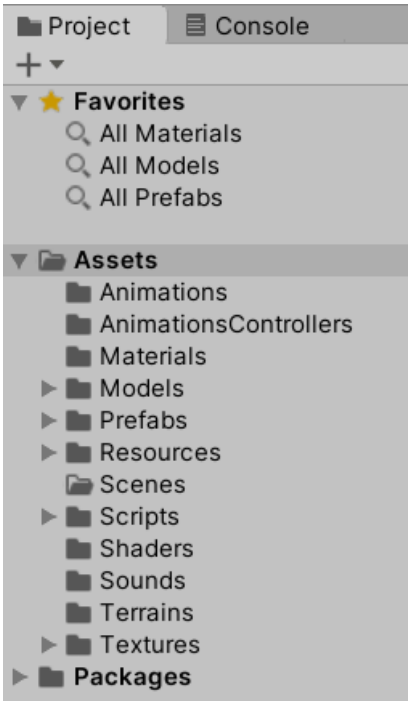


Obrázek 5: Nastavení rozlišení a prezentace projektu

Nyní je projekt připraven a můžeme začít s vývojem samotné hry.

4.3.2 Struktura herního projektu

Veškerý herní obsah se nachází ve složce `/Assets`, která obsahuje následující podsložky:



Obrázek 6: Struktura projektu v Unity3D

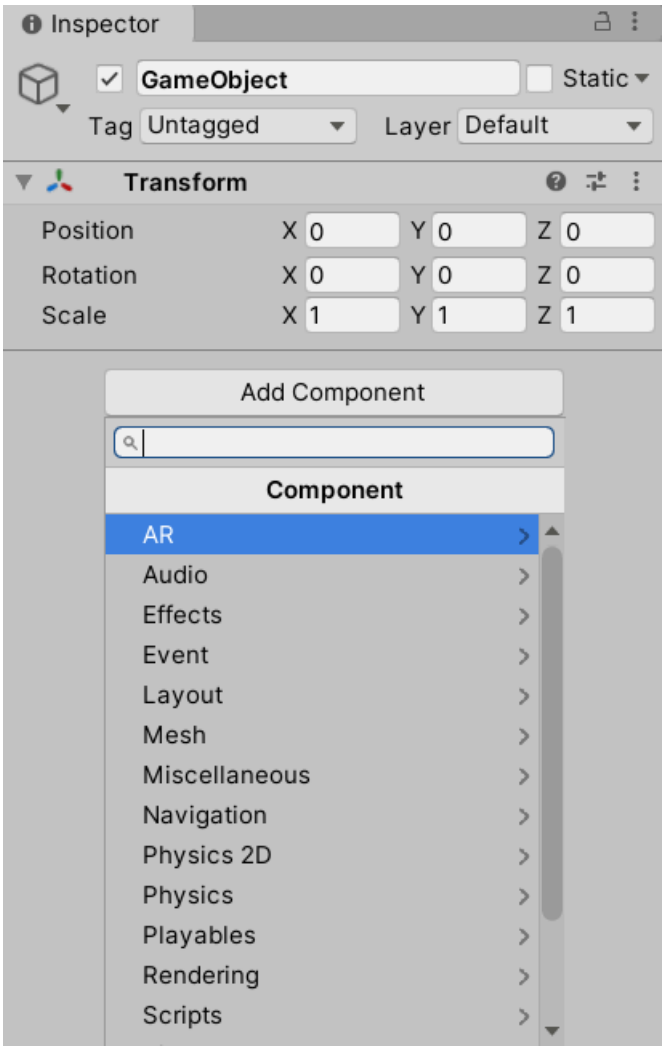
Herní kód napsaný v programovacím jazyce C# je k dispozici ve složce `/Assets/Scripts`.

4.3.3 Scény, herní objekty a prefaby

Scéna je kontejnerem herních objektů, se kterými chceme zrovna pracovat. V jeden okamžik může být aktivní pouze jedna scéna, avšak je možné inkrementálně načíst obsah další scény do právě aktivní scény, čímž v podstatě bude aktivních scén více. Každá scéna musí obsahovat objekt kamery a každá scéna by měla obsahovat objekt směrového světla a objekt spravující události uživatelského rozhraní (pokud daná scéna obsahuje nějaké uživatelské rozhraní).

Herní objekt je základní stavební strukturou charakterizovanou pomocí unikátního identifikačního čísla, několika vlastnostmi a kontejnerem pro jednotlivé komponenty. Mezi hlavní vlastnosti herního objektu patří jmenovka (tag) a vrstva (layer – určena zejména pro renderování). Pokud herní objekt během svého života nemění svůj vnější stav (pozici, renderování apod.), může být nastavený jako statický, tím lze docílit optimalizace výkonu. Pokud si nepřejeme objekt zrovna používat, můžeme jej deaktivovat a kdykoliv později znovu aktivovat. Samotný herní objekt bez komponent moc neumí. Základní komponentou je komponenta `Transform`. Každý herní objekt tuto komponentu obsahuje a jako jedinou nelze od objektu odloučit. Herní engine `Unity3D` obsahuje velké množství zabudovaných komponent (např. Camera, Light, MeshRenderer, BoxCollider apod.). Hlavní sílou je možnost implementace vlastní komponenty pomocí možnosti `New script` při výběru komponent (více v podkapitole 4.3.4). Správa her-

ního objektu je k dispozici v okně `Inspector`. Zde je možné pomocí tlačítka `Add Component` přidat novou komponentu dle vlastního výběru (viz obrázek níže).



Obrázek 7: Struktura herního objektu v Unity3D

Herní objekty taky mohou obsahovat další herní objekty. Každý takový objekt potom odvíjí svoji pozici relativně od pozice rodiče. Herní objekty je možné vyhledat několika způsoby – podle jména, umístění v hierarchii scény (metoda `Find()`), podle jmenovky (metoda `FindWithTag()`) apod. Zároveň je stejným způsobem možné vyhledat komponenty v daném herním objektu pomocí metody `TryGetComponent()`. Herní engine `Unity3D` nabízí několik předpřipravených herních objektů. Mezi tyto objekty patří 3D objekty (primitives, terén, model apod.), 2D objekty, efekty, světlo, audio, video, objekt uživatelského rozhraní a objekt kamery.

Prefab je v herním enginu `Unity3D` chápán jako šablona nebo prototyp herního objektu, z kterého je možné vytvořit plnohodnotný herní objekt třemi způsoby – přetáhnutím ze struktury projektu do hierachie herní scény, static-

kým doplněním reference na prefab do komponenty skriptu a následovným zavoláním metody `Instantiate()`, a nebo uložením prefabu do speciální složky `/Assets/Resources`, čímž lze reference na prefab získat dynamicky přes cestu k prefabu pomocí zavolání metody `Resources.Load()`. Samotné nastevení prefabu je obnobné nastavení herního objektu (přes okno `Inspector`).

4.3.4 Vkládání skriptů psaných v jazyce C#

Jediným způsobem jak v herním enginu `Unity3D` spouštet uživatelem vytvořené skripty, napsané v programovacím jazyce C#, je přidělit komponentu typu skript některému hernímu objektu nebo prefabu herního objektu jako komponentu. Vytvořit nový skript můžeme buď vytvořením nového assetu typu `Script` a nebo přes tlačítko `Add Component` výběrem položky `New script` v okně inspektoru herního objektu nebo prefabu.

Každý skript musí obsahovat stejnojmennou třídu, která musí dědit od třídy `MonoBehaviour`. Ta implementuje základní logiku herní komponenty a obsahuje odkazy na herní objekt, ve kterém je komponenta skriptu zrovna používána. Nově vytvořený skript bude mít obsah zobrazený ve zdrojovém kódu 8.

Zdrojový kód 8: Ukázka výchozího kódu nového skriptu

```
using UnityEngine;
using System.Collections;

public class NewScript : MonoBehaviour
{
    // pro inicializaci skriptu
    void Start() {}

    // pro aktualizaci skriptu
    void Update() {}
}
```

Uživatel může ovládat cyklus života komponenty pomocí metod reagujících na události. Pro inicializaci skriptu je možno vybrat ze dvou metod. Metoda `Awake()` je vykonána při inicializaci komponenty (pouze jednou) a metoda `Start()` je volána vždy po aktivaci herního objektu, který vlastní danou komponentu. Pro aktualizaci skriptu je tu metod hned několik. První metodou je `Update()`. Ta je volána při aktualizaci herního objektu. Další metodou je `FixedUpdate()`, která je výhodná pro fixní aktualizaci (například výpočtu fyziky apod.). Poslední metodou je `LateUpdate()`, která je volána po zavolání předchozích dvou metod. Herní engine `Unity3D` těchto metod poskytuje celou řadu, pro více informací doporučuji dokumentaci na oficiálních stránkách. Odkaz na samotný herní objekt je k dispozici pomocí vlastnosti `gameObject`.

Každou veřejnou vlastnost třídy skriptu je možné nastavovat v okně inspektoru. Pokud si přejeme nastavit soukromou vlastnost, musíme před danou vlastnost doplnit atribut `[SerializeField]`. Serializovat lze každou zabudovanou

třídu herního enginu `Unity3D`, základní datové struktury (int, bool apod.), pole a seznamy serializovatelného typu, enumerace a struktury.

4.3.5 Rozdělení a struktura herních scén

V této práci jsem herní scény rozdělil na scénu určenou pro uživatelské rozhraní využívané před spuštěním samotné hry, scény úrovní nesoucí název ve formátu *level{id}* (např. *level0*) a testovací scénu (*testlevel*).

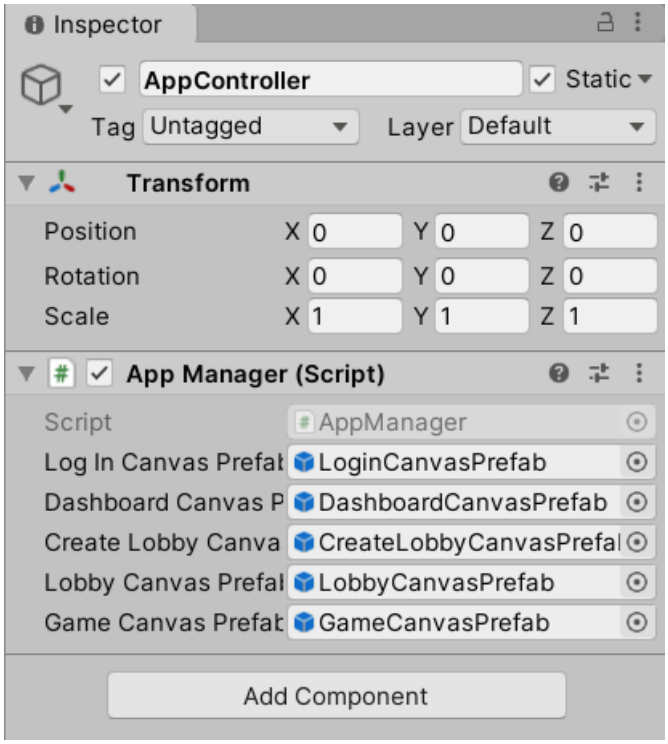
Úvodní scéna je určena k zobrazení herního menu – přihlašovací obrazovky, vstupního lobby, tvorby nového herního lobby a zobrazení aktuálního herního lobby. Tato scéna zároveň slouží k inicializaci singletonu třídy `AppManager`, tedy jako jediná scéna obsahuje herní objekt nesoucí komponentu skriptu stejnojmenné třídy. Pro správné nastavení je potřeba doplnit odkaz na prefaby objektů uživatelských rozhraní v komponentě skriptu. Posledním krokem přípravy této scény je vytvoření herního objektu spravující události uživatelských rozhraní. Toho docílíme kliknutím na položku GameObject→UI→ EventSystem v hlavním nabídkovém pruhu.

Názvy scén herních úrovní musí odpovídat názvu a id úrovni ve třídě `Levels`, jinak může dojít k chybě při spuštění zápasu herního lobby. Co vše musí herní scény obsahovat více v podkapitole 4.3.9.

Při vývoji aplikace hry jsem většinu času strávil v testovací scéně *testlevel*. Ta obsahuje prototypy herních objektů a je to jediná herní scéna zabalena do binárních souborů nesoucí název `offlinetest`.

4.3.6 Nastavení hlavních herních objektů

Jako první nastavíme nejdůležitejší herní objekt celé aplikace, a sice objekt `AppController` spravující celou aplikaci pomocí komponenty skriptu `AppManager`. Tento objekt vytvoříme pouze v první spouštěné herní scéně (tou je v mé práci scéna `MenuScene`). Vybereme tedy položku GameObject→Create Empty a přidáme novému hernímu objektu komponentu skriptu `AppManager`. následně dosadíme do této komponenty reference na všechny prefaby uživatelských rozhraní (viz obrázek níže).

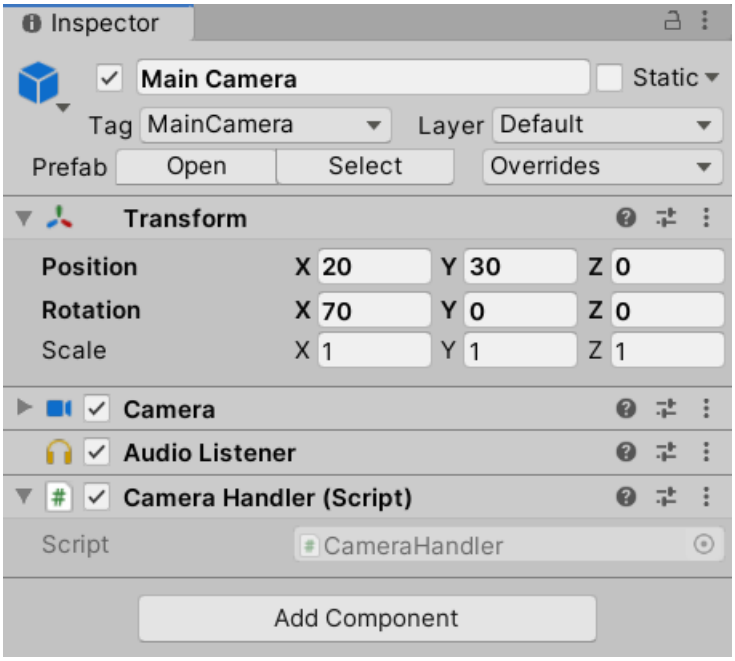


Obrázek 8: Nastavení objektu spravující celou aplikaci

Nakonec nastavíme tento herní objekt jako statický.

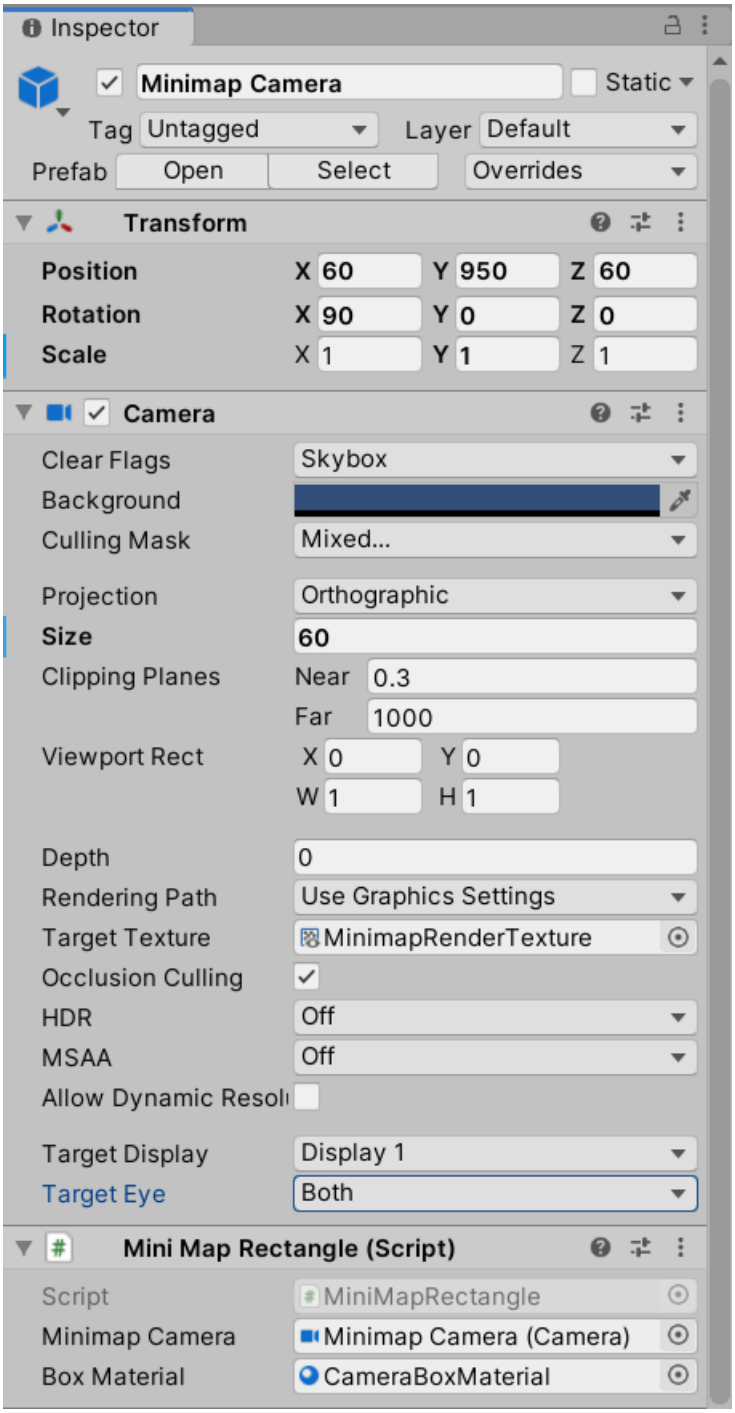
Každá herní scéna v mé práci ještě musí obsahovat objekt správy událostí `Event System`. Ten stačí přidat pomoci kliknutí na položku GameObject→UI→Event System v hlavním nabídkovém pruhu.

Jako další doplníme hernímu objektu hlavní kamery (`Main Camera`) komponentu skriptu `CameraHandler`, a nastavíme rotaci osy x na 70 v komponentně `Transform` (finální náklon kamery, viz obrázek níže). Tento herní objekt nemusíme vytvářet, jelikož jej každá herní scéna při vytvoření již obsahuje.



Obrázek 9: Nastavení objektu hlavní kamery

Posledním důležitým objektem, který je potřeba nastavit, je objekt spravující kameru, jež bude použita pro vykreslenení minimapy. Přidáme tedy herní objekt kliknutím na položku GameObject→Camera v hlavním nabídkovém pruhu. Objektu nastavíme pozici a rotaci (viz obrázek níže) a v komponentě `Camera` přepneme režim projekce na ortografický.



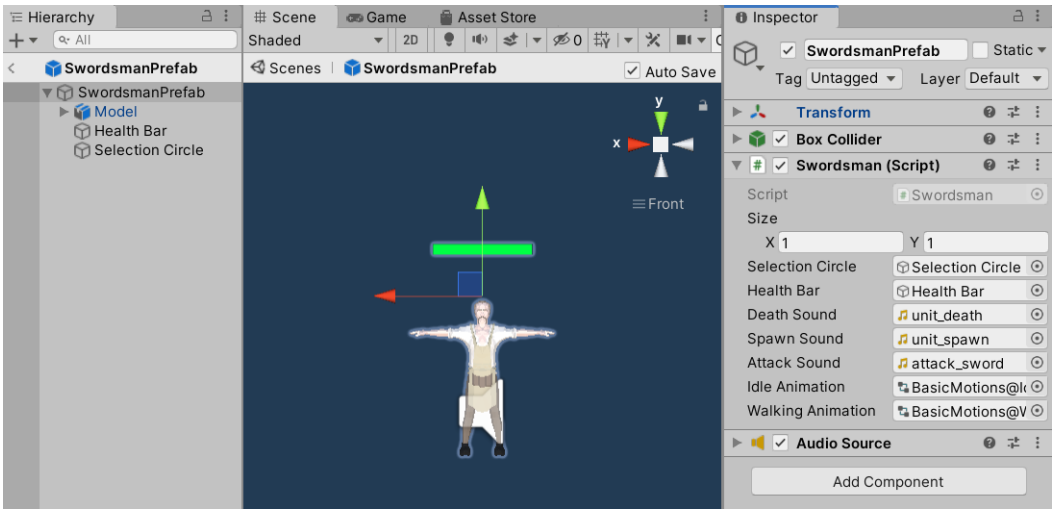
Obrázek 10: Nastavení objektu kamery vykreslující minimapu

Dále nastavíme v položce `Culling Mask`, že chceme vykreslovat pouze vrstvu terénu a minimapy. Tyto vrstvy bude potřeba nejprve vytvořit v nabídce nastavení projektu pod lištou `Tags and Layers`. Jako poslední vytvoříme ve složce `/Assets` novou texturu, jejíž referenci dosadíme do políčka `Target Texture` v komponentě `Camera` objektu kamery minimapy. Právě do této texture bude vy-

kreslen pohled kamery minimapy. následně přidáme hernímu objektu minimapy komponentu skriptu `Minimap Rectangle`, do něhož doplníme nutné závislosti. Tím je herní objekt kamery pro vykreslení minimapy připraven.

4.3.7 Příprava prefabů herních objektů

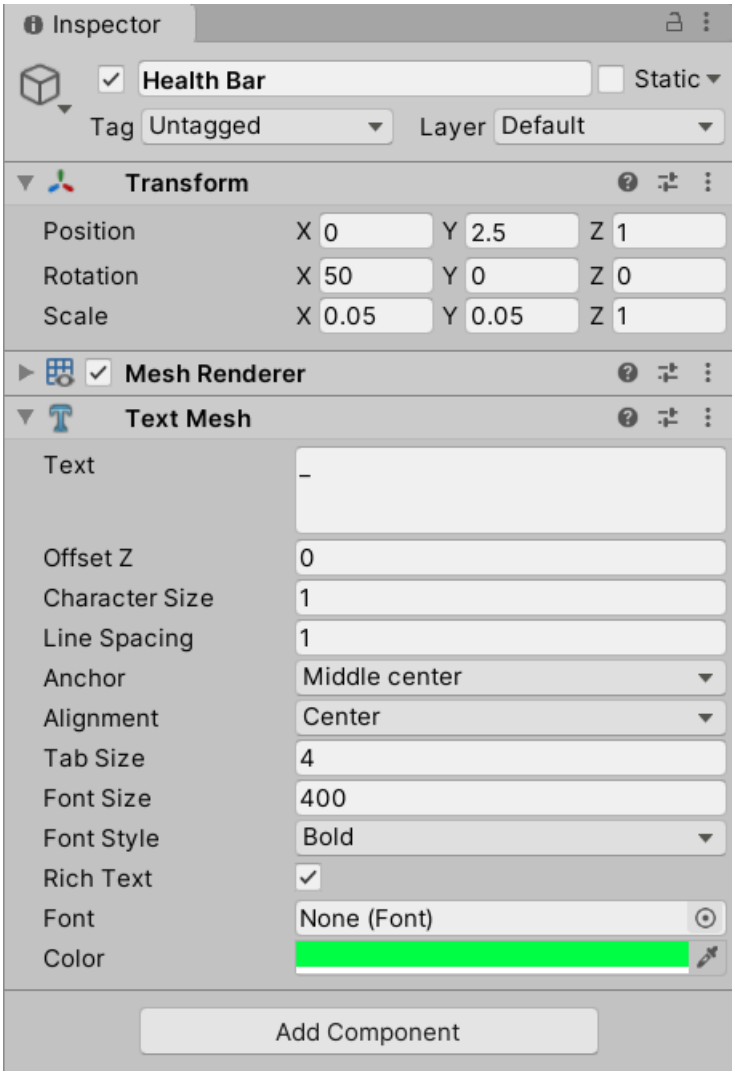
Samotná hra nabízí spoustu různých jednotek, budov, přírodních překážek apod. Každý z těchto prvků hry musí být dopředu připraven k použití. Všechny jsou založeny na podobné struktuře – kořenový objekt obsahuje komponentu skriptu spravující danou herní entitu (například pro entitu jednotky šermíře to bude skript `Swordsman`), objekt vykreslující model dané herní entity, objekt vykreslující body zdraví a objekt vykreslující označení, že je dané entita zvolena (poslední dvě pouze u herních objektů jednotek a budov).



Obrázek 11: Nastavení objektu spravující terén

Nejprve přidáme kořenovému objektu komponentu skriptu určenou pro daný typ herní entity a komponentu `Box Collider`, kterou nastavíme tak, aby odpovídala zvolenému modelu.

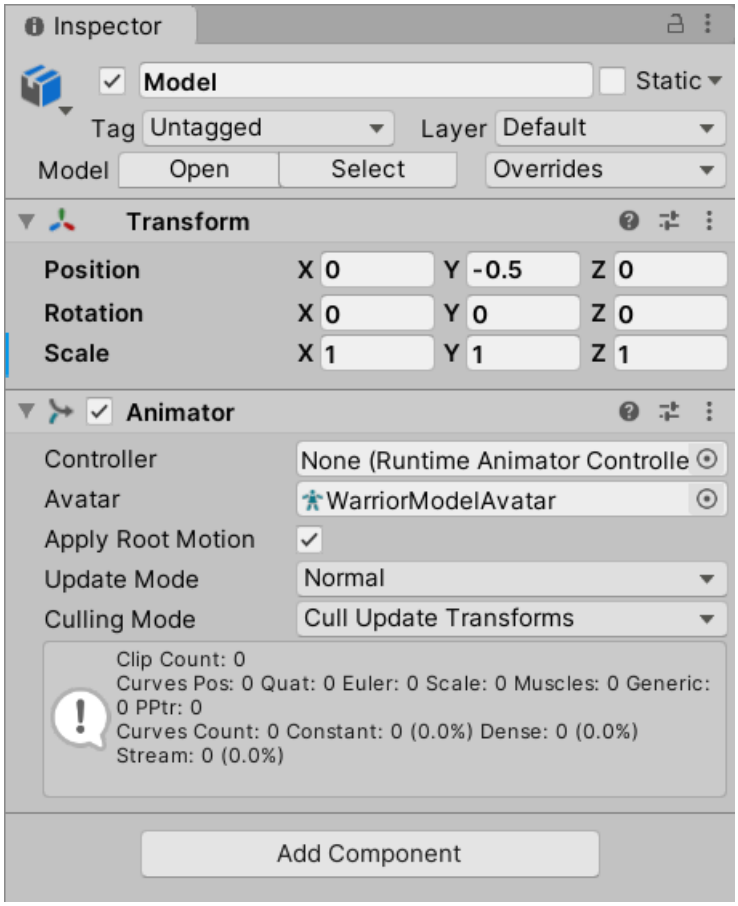
následně přidáme další herní objekt přes položku GameObject→ 3D Object→3D Text v hlavním nabídkovém pruhu. Jako obsah textu nastavíme znak podtržítka a přidělíme mu zelenou barvu. Pak objekt natočíme a přesuneme podle hodnot v obrázku níže. Tímto je objekt pro vykreslení počtu bodů zdraví připraven.



Obrázek 12: Nastavení objektu vykreslující body zdraví

Dále nás čeká připřavit objekt, který vyznačuje, že je daný herní objekt zvolený. Vytvoříme teda nový objekt přes GameObject→3D Object→Plane. Nyní jen natočíme a přesuneme plochu pod model herního objektu a přidáme mu materiál `SelectionBox`, který je připravený ve složce `/Assets/Materials`. Nakonec deaktivujeme komponentu `Mesh Renderer`, která bude sama automaticky aktivována, bude-li daný herní objekt vybraný.

Nyní přichystáme herní objekt určený k vykreslení modelu entity. K tomu nám stačí pouze přetáhnout soubor modelu ze složky assetů přímo do hierarchie herního objektu pod kořenový objekt. Nakonec, je-li daným objektem jednotka, přidáme komponentu `Animator`, do které dosadíme odkaz na animátora avataru (viz obrázek níže).



Obrázek 13: Nastavení objektu vykreslující model

Nyní už jen přiřadíme odkazy na potomky kořenového objektu do komponenty skripty kořenového objektu, nastavíme velikost herní entity v simulaci a doplníme odkazy na zvuky (vyžaduje-li je daná herní entita).

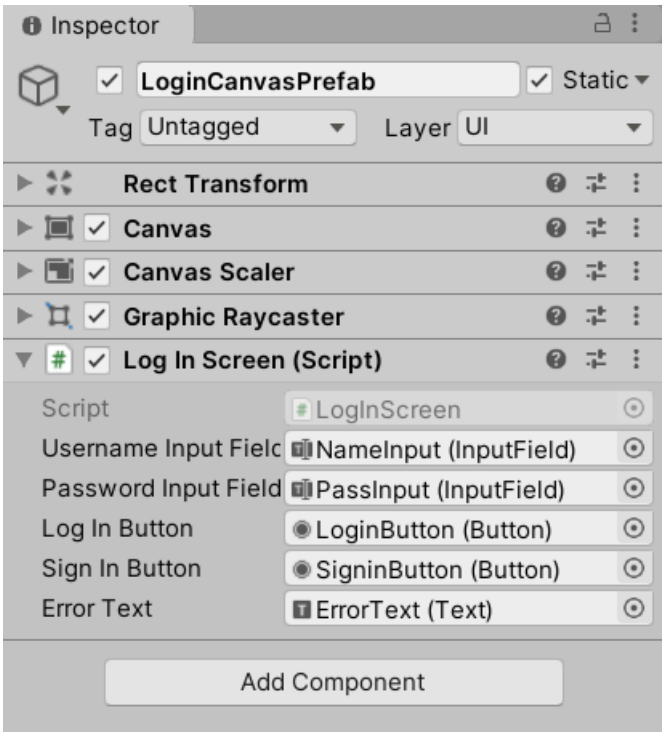
Jelikož je příprava všech herních entit velice podobna, stačí jen připravit jeden základní prefab, ze kterého budou všechny další prefaby vycházet, a pozměnit jen název prefabu, komponentu skriptu, model a velikost objektu podle typu dané herní entity.

4.3.8 Tvorba uživatelského rozhraní

Herní engine `Unity3D` implementuje všechny prvky uživatelského rozhraní jako herní objekt, případně jako komponentu herního objektu. Každé uživatelské rozhraní musí mít alespoň jeden skript, který jej ovládá (žádný objekt reprezentující prvek uživatelské rozhraní neimplementuje logiku daného prvku), a každá herní scéna obsahující uživatelské rozhraní musí obsahovat herní objekt spravující události uživatelského rozhraní (`Event System`).

Pro vytvoření uživatelského rozhraní vytvoříme nejprve herní objekt typu `Canvas`. Tomuto objektu přidáme komponentu skriptu, který jej bude spravovat, a vytvoříme nový objekt typu `Panel`, který bude jeho potomkem. Všechny další

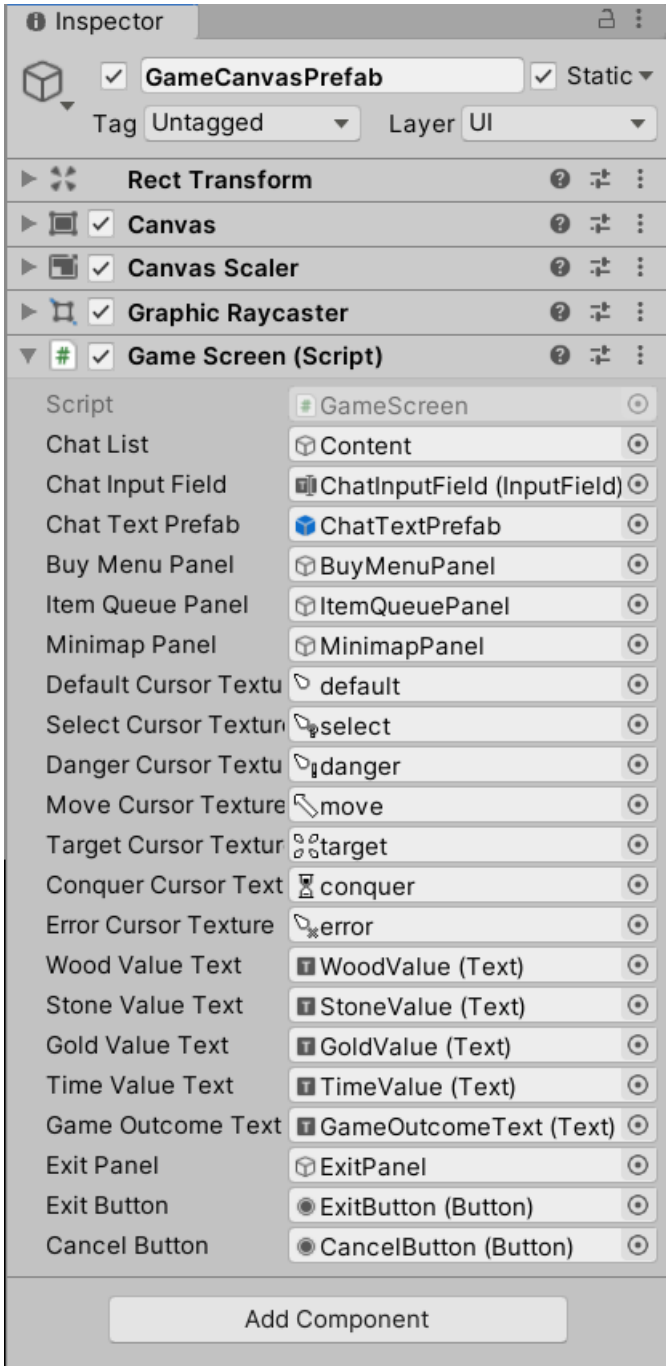
objekty daného uživatelského rozhraní budou rovněž potomkém objektu panelu. Pokud bude od prvku daného uživatelského rozhraní požadovaná nějaká funkcionalita, měl by něj existovat odkaz v komponentě skriptu daného uživatelského rozhraní (viz obrázek níže).



Obrázek 14: Nastavení objektu uživatelského rozhraní přihlašovacího formuláře

Například uživatelské rozhraní přihlašovacího formuláře registruje odkazy na dvě vstupní pole (uživatelské jméno a heslo), dvě tlačítka (příhlášení, registrace) a text pro chybové hlášky.

Uživatelské rozhraní použité při samotné hře navíc očekává odkazy na textury kurzorů (viz obrázek níže). Ty jsou následně nastavovány podle typu objektu, na který ukazuje kurzor myši, případně pokud je zvolena nějaká jednotka. Na rozdíl od ostatních uživatelských rozhraní je uživatelské rozhraní samotné hry inicializováno při každém spuštění nového zápasu, a to kvůli vazbám na herní objekty, které jsou při skončení hra odstraněny.



Obrázek 15: Nastavení objektu uživatelského rozhraní samotné hry

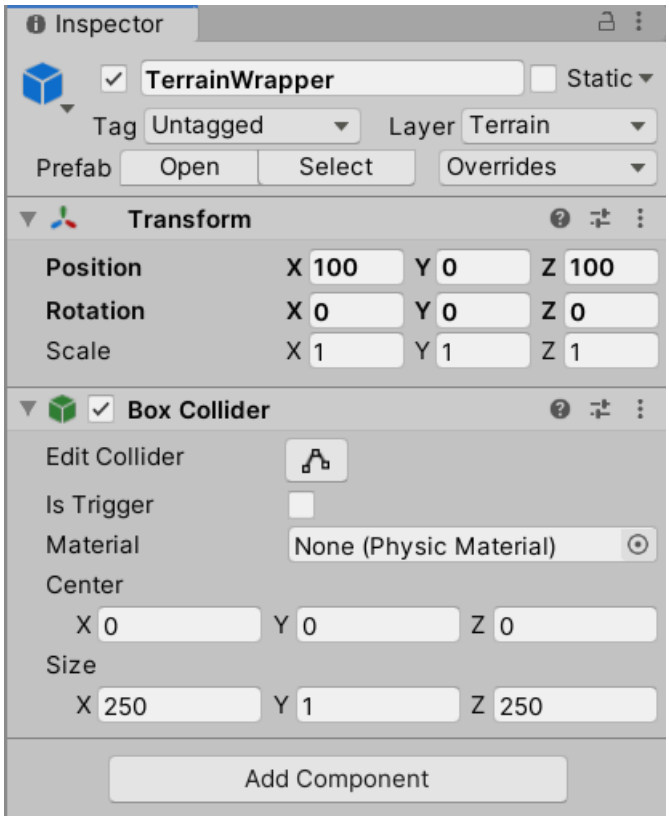
Ve mé práci jsou všechny herní objekty uživatelských rozhraní uloženy jako prefaby ve složce `/Assets/ Prefabs/UI/Canvas` a předána komponentě skriptu (třída `AppManager`) herního objektu `AppController` (ten je dostupný v hierarchii scény `MenuScene`), který se stará o jejich inicializaci a správu. V každém okamžiku hraní hry jsou načtená všechna uživatelská rozhraní, avšak pouze to používané je aktivní.

4.3.9 Tvorba herních úrovní

Každá herní úroveň je součástí vlastní scény. Nová herní scéna se vytváří kliknutím na položku Asset→ Create→Scene v hlavním nabídkovém pruhu. Po úspěšném vytvoření nové scény odstraníme veškeré výchozí herní objekty, které nová scéna obsahuje, protože je v dalším kroku nahradíme předefinovanými objekty.

Než se dostaneme k samotnému návrhu herní úrovně, je potřeba scénu doplnit o potřebné prefaby důležitých herních objektů. Všechny tyto prefaby se nachází ve struktuře projektu na cestě */Assets/Prefabs/ Scene* a pro jejich doplnění do scény stačí tyto soubory přetáhnout do okna hierarchie scény.

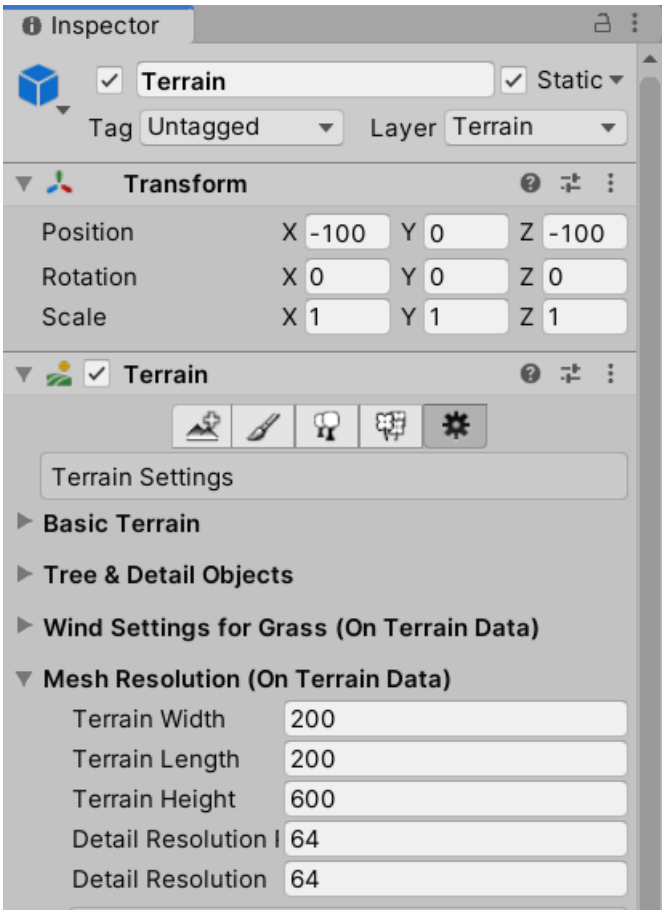
Nyní následuje část nastavení a vytvoření terénu nové herní úrovně. Nejprve nastavíme herní objekt nesoucí název `TerrainWrapper`. Správné nastavení tohoto objektu je nezbytné k fungování minimapy. Dejme tomu, že nová herní úroveň bude mít rozměr 200×200 (pro představivost rozměr herní jednotky šermíře je 1×1×1), nastavíme tedy rozměr položky `Size` komponenty `Box Collider` o 50 jednotek rozměru více, než je námi zvolena velikost terénu (rezerva pro správnou funkcionalitu minimapy).



Obrázek 16: Nastavení objektu spravující terén

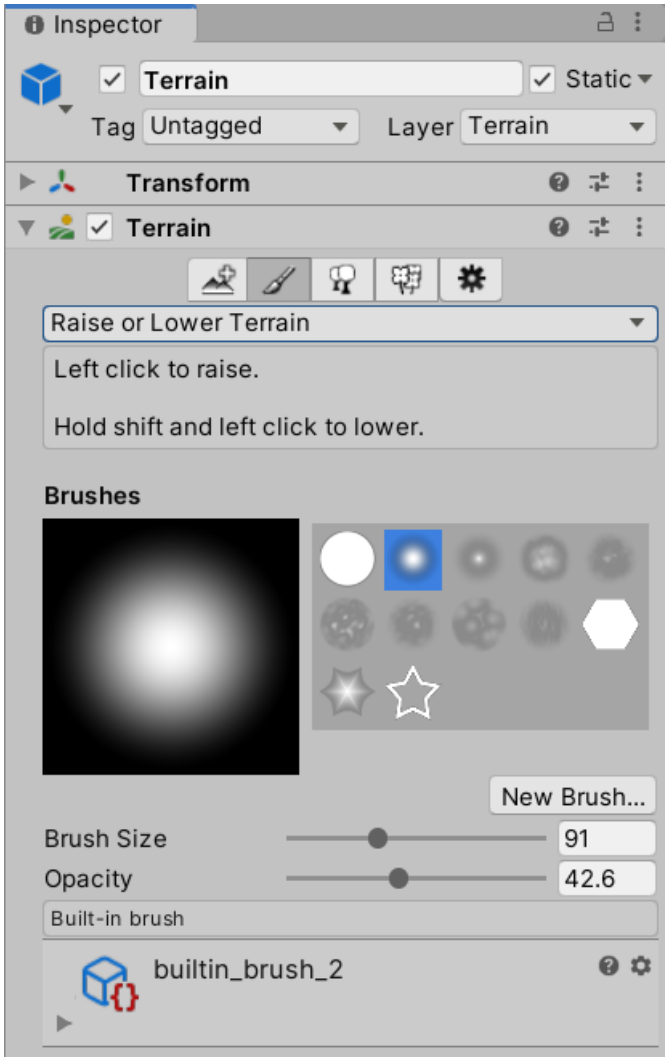
Dimenze Y je ve většině případů v mé aplikaci vynechávaná, jelikož není používaná. Nyní přidáme hernímu objektu `TerrainWrapper` nový objekt terénu. Toho docílíme kliknutím na herní objekt `TerrainWrapper` a následujícím kliknutím na

GameObjekt→3D Object→Terrain v hlavním nabídkovém pruhu. Nejprve posuneme objekt terénu o polovinu jeho rozměru zpět a následně nastavíme rozměr samotného terénu uvnitř komponenty `Terrain` pod lištou `Terrain Settings` (viz obrázek níže).



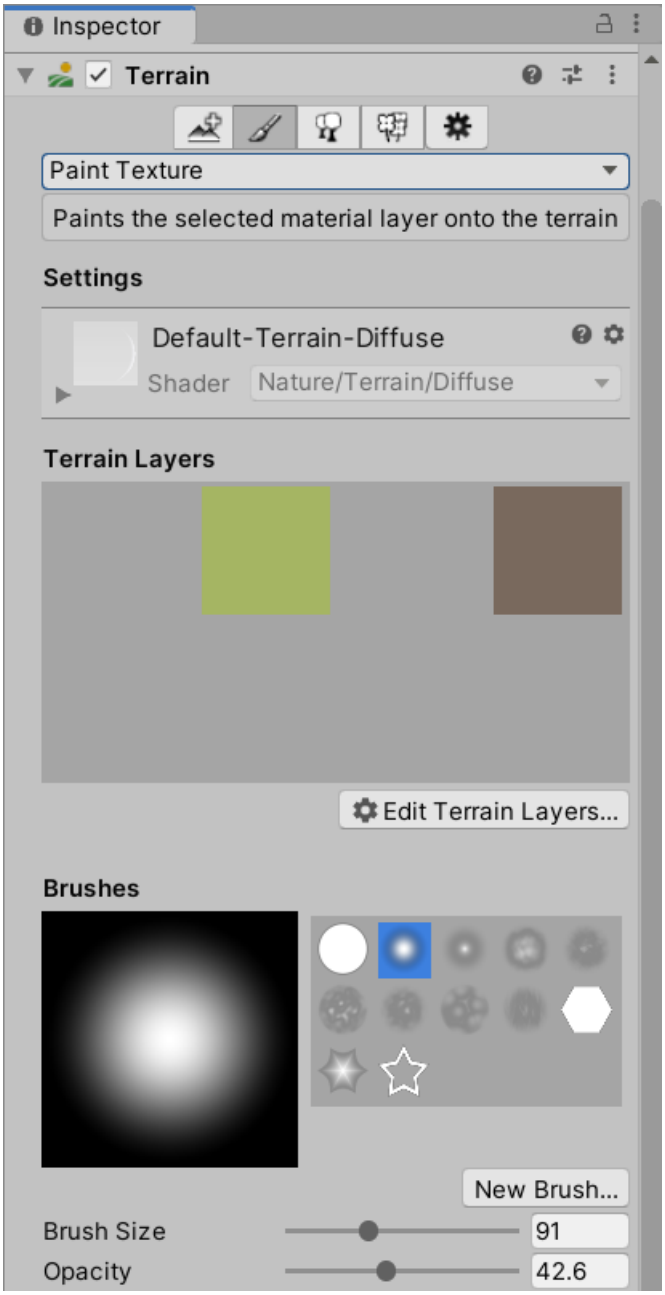
Obrázek 17: Nastavení komponenty spravující terén

Teprve teď můžeme začít upravovat samotný terén úrovně. Přepneme na lištu `Paint Terrain` v komponentě `Terrain` stejnojmenného objektu. Máme zde několik možností práce s terénem – úprava tvaru terénu, malování po textuře terénu, nastavení výšky terénu, vyhlazení terénu apod. Nejprve se vrhneme na úpravu tvaru (na pořadí těchto úprav nezáleží). Vybereme položku `Raise or Lower Terrain`, následně zvolíme tvar štětce pro úpravy a levým tlačítkem myši měníme tvar terénu.



Obrázek 18: Úprava tvaru terénu

Kromě tvaru štětce můžeme nastavit i jeho velikost a průsvitnost tahu. Pro úpravu textury terénu přepneme na lištu `Paint Texture`. Před samotným malováním musíme nastavit vrstvy terénu. To provedeme kliknutím na tlačítko `Edit Terrain Layers...` a výběrem možnosti `Create Layer...`. V zobrazeném okně vybereme texturu pro novou vrstvu. Tímto způsobem vytvoříme všechny vrstvy, jež chceme používat při malování po terénu. Samotné malování je potom totožné úpravě tvaru terénu (viz obrázek níže).

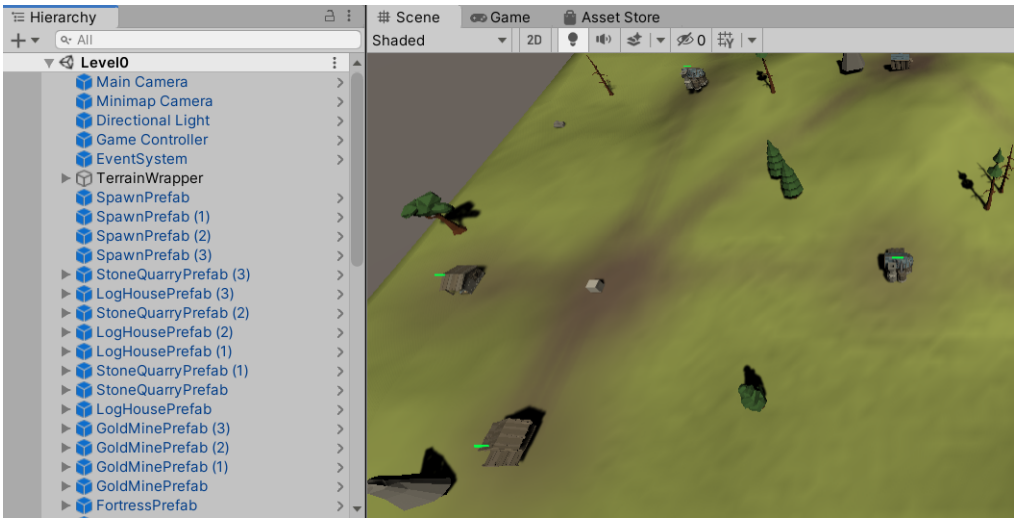


Obrázek 19: Malování po textuře terénu

Po dokončení úprav terénu nové herní úrovně je ještě potřebné přidat prefaby nezbytných herních objektů. Nejprve přidáme herní objekty spawnerů hlavních budov. To provedeme přetáhnutím prefabu `SpawnPrefab`, který se nachází ve složce */Assets/Prefabs*, do hierarchie naší scény. Těchto spawnerů vytvoříme podle maximálního počtu hráčů naší herní úrovně. Pro každý spawner ještě nastavíme id hráče, kterého má spawnout. Toho docílíme nastavením položky `Player Id` v jeho komponentě skriptu `Spawn` (první hráč nese id 0).

Nyní už stačí jen do hierarchie scény přetahovat prefaby statických herních

objektů jako jsou stromy, kameny (složka */Assets/Prefabs/Environment*) a surovinové zdroje (složka */Assets/Prefabs/ResourceSources)*. Nyní by scéna nové herní úrovně měla být hotova. Už jen stačí scénu správně pojmenovat a přidat nový záznam do třídy `Levels`. Po sestavení bude otevřena prohlížeč souborů ve složce, kde byl projekt aplikace sestaven.

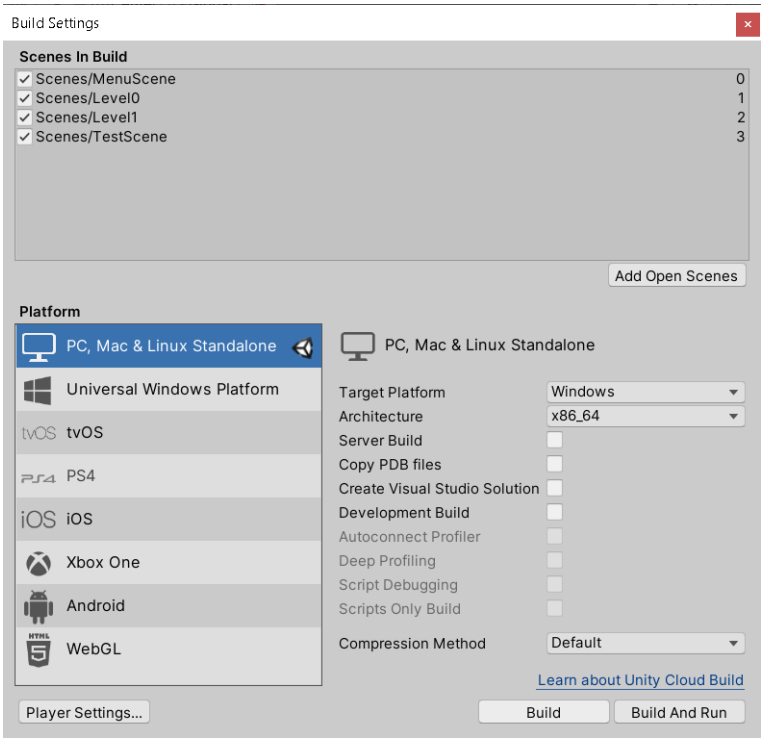


Obrázek 20: Konečná podoba scény nové herní úrovně

4.4 Kompilace a sestavení aplikace hry

Máme-li otevřený editor herního enginu `Unity3D`, dochází k automatické kompilaci při každé změně ve složce `/Assets` (ať už úpravou herního skriptu nebo úpravou jakéhokoliv assetu). Herní kód je kompilován do souboru pod názvem `Assembly-Csharp.dll`.

Pro sestavení aplikace hry vybereme položku File→Build Settings v hlavním nabídkovém pruhu nebo stiskneme kombinaci kláves Ctrl+Shift+B. Zde přidáme scény, které bude sestavený balíček obsahovat a stiskneme tlačítko `Build`. Následně se nám otevře dialog, pomocí kterého vybereme složku, kam má být hra sestavena.



Obrázek 21: Obrazovka sestavení aplikace hry

4.5 Popis tříd hlavního serveru

V této podkapitole se věňuji popisu významných tříd projektu hlavního serveru (neboli `MasterServer`).

4.5.1 Třída Program

Tato třída obsahuje vstupní bod aplikace hlavního serveru (statická metoda `Main()`). Při spuštění aplikace se nejprve zkontrolují a naparsují vstupní parametry aplikace (možnost nastavit port hlavního serveru), potom načte konfigurační soubor `masterserver.txt` a pomocí načtených hodnot se program pokusí připojit k databázi. Při úspěšném připojení se následně vytvoří objekt třídy `Lobby`, který se otevře zavoláním metody `Open()`. Tato metoda vytvoří server, který je určený ke komunikaci s jednotlivými uživateli a herními servery. Předposledním krokem této metody je hlavní smyčka programu, ve které se aktualizují síťové události. Tato smyčka končí pouze při stisknutí libovolné klávesy. Nakonec se zavolá metoda `Close()` objektu třídy `Lobby`, čímž se ukončí činnost hlavního serveru.

4.5.2 Třída Lobby

Jednou z nejrozsáhlejších tříd projektu je třída `Lobby` aplikace `MasterServer`. Tato třída implementuje server, který obsluhuje uživatele (herní klienty) a herní

servery. Její hlavní součástí jsou seznamy připojených uživatelů, seznam čekajících nebo probíhajících herních lobby a seznam připojených herních serverů. Všechny tyto seznamy jsou realizované jako slovníky nad třídami kontejnerů (viz podkapitola 4.5.4).

Třída je rozdělena do tří hlavních skupin – hlavní metody obsluhy serveru, metody vykonávající reakci na přijaté zprávy od uživatele nebo serveru a skupina pomocných metod.

Do první skupiny patří například metoda `OnConnectionRequest()`, která zajistí, že se na hlavní server mohou napojit pouze klienti poslední verze hry a klienti poslední verze herního serveru. Metoda `OnClientConnected()` klienta přidá do správného seznamu (uživatele do seznamu uživatelů, herní server do seznamu herních serverů). Nejvýznamnější metodou je metoda `OnDataReceived()`, která dekóduje přijaté zprávy a podle typu zprávy spustí metodu, která řeší odpověď na daný dotaz (metody druhého regionu).

Všechny metody reakcí na příchozí zprávy pracují v podstatě na stejném principu – nejprve dojde k deserializaci příchozí zprávy, potom změní vnitřní stav lobby hlavního serveru (například změna vnitřního stavu uživatele, herního lobby, herního serveru apod.) a nakonec serializace a odeslání odpovědi (klientovi, množině klientů nebo klientovi herního serveru).

Mezi pomocné metody patří například serializace zprávy obsahující informace o všech herních lobby nebo metoda vyhledávající volný herní server.

4.5.3 Třída Database

Třída `Database` slouží jako přostředník mezi hlavním serverem a samotnou databází. Zajišťuje připojení k databázi případně oznámí důvod neúspěšného připojení a poskytuje metody, které realizují dotazy na samotnou databázi (abstrakce dotazu pomocí metody). Příkladem takových dotazů jsou metody pracující s uživatelem `LogIn()`, `SignIn()`, `GetUserData()` apod., nebo metody pracující s herními lobby jako například `CreateMatchData()` a `UpdateMatch()`. Spojení s databází je postaveno na knihovně `MySqlConnector`.

4.5.4 Třídy UserData, LobbyData a ServerData

Tyto třídy slouží jako kontejnéry dat, které využívá a zpracovává třída `Lobby`. Každý přihlášený uživatel je reprezentovaný jedním objektem třídy `UserData`. Každý uživatel může být členem jednoho herního lobby, které je reprezentováné objektem třídy `LobbyData` a každý přihlášený herní server je reprezentován jedním objektem třídy `ServerData`. Třídy `UserData` a `LobbyData` mají navíc reprezentaci v podobě tabulky uvnitř databáze.

4.6 Popis tříd herního serveru

Tato podkapitola se věnuje popisu významných tříd projektu herního serveru (neboli `GameServer`).

4.6.1 Třída Program

Obsahuje vstupní bod aplikace herního serveru (statická metoda `Main()`). Tato metoda nejprve zkontroluje a naparsuje vstupní parametry aplikace (možnost nastavit port serveru), potom načte konfigurační soubor `masterserver.txt` a pomocí načtených hodnot vytvoří objekt třídy `Lobby`, který následně spustí zavoláním metody `Open()`. Připojí se k aplikaci hlavního serveru a vytvoří server, který je určený ke komunikaci s jednotlivými hráči daného lobby. Předposledním krokem je hlavní smyčka programu, ve které se aktualizují síťové události. Tato smyčka končí v okamžiku konce hry, při ztrátě spojení s aplikací hlavního serveru nebo pokud tak určí uživatel stisknutím libovolné klávesy. Nakonec se zavolá metoda `Close()` objektu třídy `Lobby`, čímž se ukončí činnost herního serveru.

4.6.2 Třída Lobby

Tato třída reprezentuje správce herního lobby. Lobby se může nacházet v jednom ze dvou stavů – pasivní (*idle* – čeká na instrukce od hlavního serveru) nebo aktivní (*running* – má informace o herním lobby a realizuje komunikaci mezi herními klienty). Kromě údajů o aktuálním herním lobby si tato třída uchovává seznam připojených uživatelů a parametry, jež jsou potřebné pro synchronizaci hry (např. společný seed náhod, výchozí minimální síťová latence, apod.).

Důležitou metodou této třídy je metoda `Open()`, která nastavuje a registruje reakce na události klienta, který komunikuje s hlavním serverem, a serveru, který slouží jako prostředník komunikace mezi připojenými uživateli.

Jednou s takových reakcí na události je metoda `OnConnectionRequest()`. Tato metoda zajistí, že se k serveru může uživatel připojit pouze přes aplikaci hry, a to jen tehdy, pokud patří do daného herního lobby. Při potvrzení je připojenému uživateli přiřazené identifikační číslo, které je používané při synchronizované komunikaci pomocí Lockstep modelu.

Další významnou metodou je `PrepareLobby()`, která je zavolána v okamžiku, kdy hlavní server přidělí tento herní server k některému z herních lobby (hernímu serveru jsou předány informace o herním lobby).

Poslední metoda, která stojí za zmínku, je metoda `StartSimulation()`. Tato metoda je volána po připojení všech uživatelů nebo po vypršení doby připojení (nutnost připojení alespoň jednoho uživatele). Při vykonávání této metody se nejprve všem připojeným uživatelům odešle zpráva, která obsahuje informace o lobby, synchronizační paramentry a identifikační číslo daného uživatele, a potom00 je 0 hlavnímu serveru oznámeno, že byl stav herního serveru změněn (tím se zajistí, že je jeden herní server používán pouze jedním herním lobby). Uživatelé po přijetí této zprávy spouští herní simulaci.

4.7 Knihovna sdílených souborů Shared

Tato podkapitola se věnuje popisu tříd knihovny sdílených souborů `Shared` napříč projekty hlavního serveru, herního serveru a projektu hry.

Jelikož je proces kompilace aplikace samotné hry oddělený od kompilace programů serverové části (kvůli omezení importace knihoven v herním enginu Unity3D), jsou třídy, které využívá projekt samotné hry, realizované jako reference na soubor v adresáři projektu hry.

4.7.1 Třída Client

Třída `Client` zajišťuje základní operace komunikace se serverem po síti, registruje reakce na události aktuálního připojení (jako například přijetí dat, aktualizace odezvy připojení apod.). Pomocí metody `Connect()` se spustní klient, který se pokusí připojit na zvolený server. Důležitou metodou je `Send()`, která slouží k odeslání dat na server. Pro aktualizaci událostí se volá metoda `PollEvents()`. Pro ukončení spojení se serverem se zavolá metoda `Disconnect()`, čímž dojde k odpojení se od serveru.

4.7.2 Třída Server

Třída `Server` zajišťuje základní operace komunikace s klienty po síti. Stejně jako třída `Client` i tato třída registruje reakce na události (jako například žádost o připojení, samotné připojení uživatele, odhlášení uživatele, přijetí dat apod.). Pro spuštění serveru se volá metoda `Run()`, jež obsahuje obsahuje také metodu `Send()`, která je zde navíc rozšířena jako metoda `Distribute()`, která zasílané byty rozešle všem připojeným klientům. Pro aktualizaci událostí se volá metoda `PollEvents()`. Po dokončení práce s objektem třídy `Server` se zavolá metoda `Stop()`.

4.7.3 Třída Logger

Tato třída slouží k výpisu a logování zpráv, chyb a výjimek do konzole a souboru logu ve složce `/log`. Poskytuje statickou metodu `GetInstance()`, pomocí které je možné získat jinak nezískatelnou instanci třídy Logger (např. pro nastavení vnitřního stavu singletonu) a statickou metodu `Put(string message)`, jejímž zavoláním vypíšeme určenou zprávu do příkazového řádku, kde je aplikace serveru spuštěna, a zároveň do logovacího souboru.

4.7.4 Třídy komunikačních zpráv

Třídy komunikačních zpráv jsou rozděleny do dvou kategorií – zprávy komunikace uživatele a herního serveru s hlavním serverem a zprávy komunikace uživatele s herním serverem. Všechny tyto třídy implementují rozhraní `INetSerializable`, jež vyžaduje implementovaci metod serializace a deserializace zprávy.

4.8 Kompilace serverových aplikací

Narozdíl od kompilace aplikace samotné hry je kompilace serverových komponent jednoduchá. Pro kompilaci spustíme vývojové prostředí `JetBrains Rider`

(případně `Visual Studio 2019`), se kterým jsou soubory projektů kompatibilní) a spustíme samotnou kompilaci buď pomocí klávesové zkratky *Ctrl+F9*, nebo pomocí kontextové nabídky Build.

Jedinou podmínkou úspěšné kompilace programů serverové části je mít nainstalovaný balíček knihovny `LiteNetLib` verze 0.9.4 a balíček `MySqlConnector` verze 1.0.0, jež se instaluje pomocí manažéra balíčků `NuGet` (ten je součástí obou výše zmíněných vývojových prostředí).

4.9 Struktura databáze

Při přihlášení nebo registraci uživatele jsou přihlašovací údaje odeslány hlavnímu serveru, který je porovnává oproti záznamům tabulky `user` (viz zdrojový kód 9). Místo hesla je do tabulky ukládán jeho hash. Pokud při registraci nastane během tvorby nového záznamu chyba, je registrace zrušena a uživatel je obeznámen, že nelze registraci s danými údaji provést.

Zdrojový kód 9: Tabulka uživatelů

```
CREATE TABLE `user` (
  `uid` int(11) NOT NULL PRIMARY KEY,
  `name` varchar(64) NOT NULL UNIQUE KEY,
  `pwd` varchar(64) NOT NULL
);
```

Když uživatel vytvoří herní lobby, hlavní server vytvoří záznam tabulky `lobby` (viz zdrojový kód 10). Id tohoto záznamu je pak odesláno zpět uživateli. Pokud vlastník herního lobby toto lobby před spuštěním hry zruší, záznam tabulky se smaže.

Zdrojový kód 10: Tabulka herních lobby

```
CREATE TABLE `lobby` (
  `lid` int(11) NOT NULL PRIMARY KEY,
  `owner` int(11) NOT NULL,
  `level` int(11) NOT NULL,
  `winner` int(11) DEFAULT NULL
);
```

Uživatel při hraní hry sbírá data o vytvořených, ztracených a zničených jednotek a budov, počtu získaných surovin apod. Tato data se až do konce hry nachází v objetku třídy `Player`. Po skončení hry se odesílají od klienta na hlavní server, kde se tyto statistiky uloží v databázi do tabulky `lobbydata` (viz zdrojový kód 11).

Zdrojový kód 11: Tabulka statistik uživatele z herního lobby

```
CREATE TABLE `lobbydata` (
  `lid` int(11) NOT NULL,
  `uid` int(11) NOT NULL,
  `u_created` int(11) NOT NULL DEFAULT '0',
  `u_destroyed` int(11) NOT NULL DEFAULT '0',
  `u_lost` int(11) NOT NULL DEFAULT '0',
  `b_created` int(11) NOT NULL DEFAULT '0',
  `b_destroyed` int(11) NOT NULL DEFAULT '0',
  `b_lost` int(11) NOT NULL DEFAULT '0',
  `wood_produced` int(11) NOT NULL DEFAULT '0',
  `stone_produced` int(11) NOT NULL DEFAULT '0',
  `gold_produced` int(11) NOT NULL DEFAULT '0',
  `time_played` int(11) NOT NULL DEFAULT '0',
  UNIQUE KEY `id` (`lid`,`uid`) USING BTREE
);
```

Jakmile hlavní server obdrží statistiky od všech uživatelů, kteří byli součástí daného herního lobby, vypočítá, který z uživatelů vyhrál a změní pole `winner` v tabulce `lobby`. Suma těchto statistik je poté použita při zobrazení statistik hráče v uživatelském rozhraní vstupního lobby v aplikaci samotné hry.

4.10 Možná rozšíření

Aplikace samotné hry nabádá k rozšíření ovládání vybraných jednotek (například rozkaz uživatele, aby jednotky udržovali jistou formaci apod.). Dále některé strategické hry skrývají uživateli neprozkoumanou část mapy (např. odkrýt pouze část mapy, ve které se nachází jednotky nebo budovy daného hráče).

Co se týče serverové části, ta by se dala rozšířit o výběr serverů podle lokace hráčů pomocí zprůměrování jejich odezvy připojení. Aplikace herního serveru by se mohla rozšířit o možnost znovunapojení hráče do hry (např. při pádu připojení). Řešením by mohlo být ukládání všech přeposílaných příkazů od připojených uživatelů.

5 Uživatelská příručka

5.1 Systémové požadavky

Aplikace hry požaduje následující:

1. Operační systém Windows 7+, MacOS 10.12+, Ubuntu 16.04+
2. .NET 2.0 Core Runtime (doporučeno 3.1 Core Runtime)
3. Procesor podporující instrukce setu SSE2
4. Grafickou kartu kompatibilní s DirectX 10 (model shaderu 4.0)

Aplikace serverové části mají za podmínky:

1. .NET 3.1 Core Runtime
2. MySQL Server (jakýkoliv kompatibilní s knihovnou MySqlConnector)

Aplikace hry i serverů byly zkompilované a testované pouze na platformě `Microsoft Windows 10`, avšak měly by být zkompilovatelné a spustitelné na všech platformách, které podporují závislosti `.NET Standard 2.0` pro aplikaci hry a `.NET Standard 2.1` pro programy serverové části.

5.2 Instalace a odinstalace

Pro instalaci serverové části stačí přetáhnout nebo zkopírovat obsah složek `/masterserver` a `/gameserver` ze složky `/bin` na zvolené místo. Dále se musí nainstalovat MySQL server a vytvořit novou tabulku, do které importujeme obsah souboru `online-rts.sql` ze složky `/db`.

Instalace samotné hry je provedena přetáhnutím nebo zkopírováním obsahu složky `/game`, která se nachází také ve složce `/bin`.

K odinstalaci stačí přesunutý/nakopírovaný obsah smazat a odinstalovat MySQL server (pokud jsme jej instalovali pouze pro spuštění serverové aplikace).

5.3 Spuštění aplikace

Pro spuštění aplikace `MasterServer` nejdříve zkontrolujeme, zda je správně nastavený konfigurační souboru zvaný `database.txt`. Ten musí obsahovat správné přihlašovací údaje pro připojení k databázi. Nyní můžeme spustit aplikaci spuštěním souboru `MasterServer.exe`. Aplikace navíc umožňuje spuštění s pomocí nastavení jednoho volitelného argumentu, kterým se dá nastavit přístupový port serveru.

Před spuštěním aplikace `GameServer` zkontrolujeme, jestli je správně nastavený konfigurační soubor `masterserver.txt`, který musí obsahovat správnou IP adresu a port aplikace `MasterServer`. Potom je aplikace připravena ke spuštění.

Stejně jako aplikace `MasterServer` i `GameServer` umožňuje nastavení přístupového portu pomocí prvního argumentu při spuštění aplikace.

Stejný postup kontroly nastavení konfiguračního souboru `masterserver.txt` musíme uskutečnit před spuštěním a distribucí aplikace hry. následně stačí spustit spustitelný soubor `Game.exe`.

Pro správný chod serverové části aplikace nejdříve spouštíme jednu instanci aplikace `MasterServer` a jednu nebo více instancí aplikace `GameServer`, které mají nakonfigurované připojení na daný `MasterServer`. Potom je možné spustit libovolný počet instancí samotné hry, které jsou nakonfigurovány na připojení se na daný `MasterServer`.

Spuštění všech aplikací je podmíněné splněním systémových požadavků popsaných v kapitole 5.1.

5.4 Ovládání hry

V této podkapitole je sepsán seznam vstupů uživatele určených k ovládnání samotné hry. K ovládání stačí pouze myš, avšak použití klávesových zkratek ovládání hry ulehčí.

Levé tlačítko myši
: Výběr jednotky/budovy. Při podržení výběr více jednotek. Při stavbě budovy odsouhlasí stavbu.

Pravé tlačítko myši
: Provedení akce se zvolenými jednotkami (přesun, útok, apod.). Při stavbě budovy zruší stavbu.

Pohyb myši po okraji obrazovky
: Pohyb kamery podle pozice myši.

Kolečko myši
: Přiblížení/oddálení kamery.

W, šipka nahoru
: Pohyb kamery nahorů.

S, šipka dolů
: Pohyb kamery dolů.

A, šipka vlevo
: Pohyb kamery vlevo.

D, šipka vpravo
: Pohyb kamery vpravo.

NUMPAD +
: Přiblížení kamery.

NUMPAD -
Oddálení kamery.

TAB
Zvětšení/zmenšení zobrazení minimapy.

U
Výběr všech jednotek.

ENTER
Odeslání zprávy v chatu.

ESCAPE
Zruší výběr jednotek, budov, stavby a psaní zprávy v chatu.

F2
Zobrazí nebo skryje statistiky simulace a odezvy připojení.

Více ohledně popisu a ovládání uživatelského rozhraní samotné hry najdete v podkapitole 5.5.5.

5.5 Průvodce uživatelských rozhraní ve hře

5.5.1 Přihlašovací obrazovka

Přihlašovací obrazovka je tvořena jednoduchým formulářem, který obsahuje vstupní pole pro uživatelské jméno a heslo a tlačítka pro přihlášení a registraci. Při stisknutí obou tlačítek nejprve dochází k validaci vstupních polí a teprve pak herní klient komunikuje s hlavním serverem.

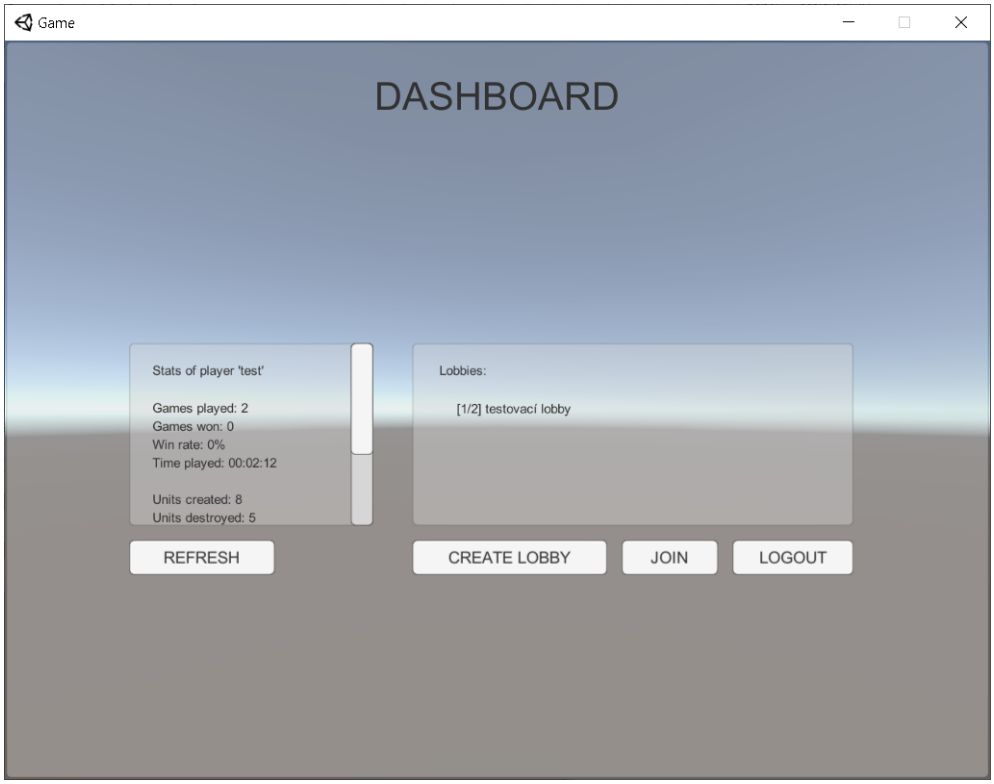
Při úspěšné registraci je uživatel zároveň přihlášen. Po úspěšném přihlášení následuje přesun do uživatelského rozhraní vstupního lobby.

Obrázek 22: Uživatelské rozhraní přihlašovací obrazovky

Pokud nastane chyba (například špatné heslo, registrace pod uživatelským jménem, které je již obsazené apod.), předchozí akce se zruší a chyba je vypsána červeně nad formulářem.

5.5.2 Vstupní lobby

Vstupní lobby po levé straně zobrazuje údaje o uživateli spolu s jeho dosavadními statistikami a po pravé straně seznam vytvořených herních lobby. Pod těmito panely se nachází tlačítko pro obnovení seznamu herních lobby, vytvoření nového herního lobby, přihlášení se do zvoleného herního lobby a nakonec odhlášení se ze hry.

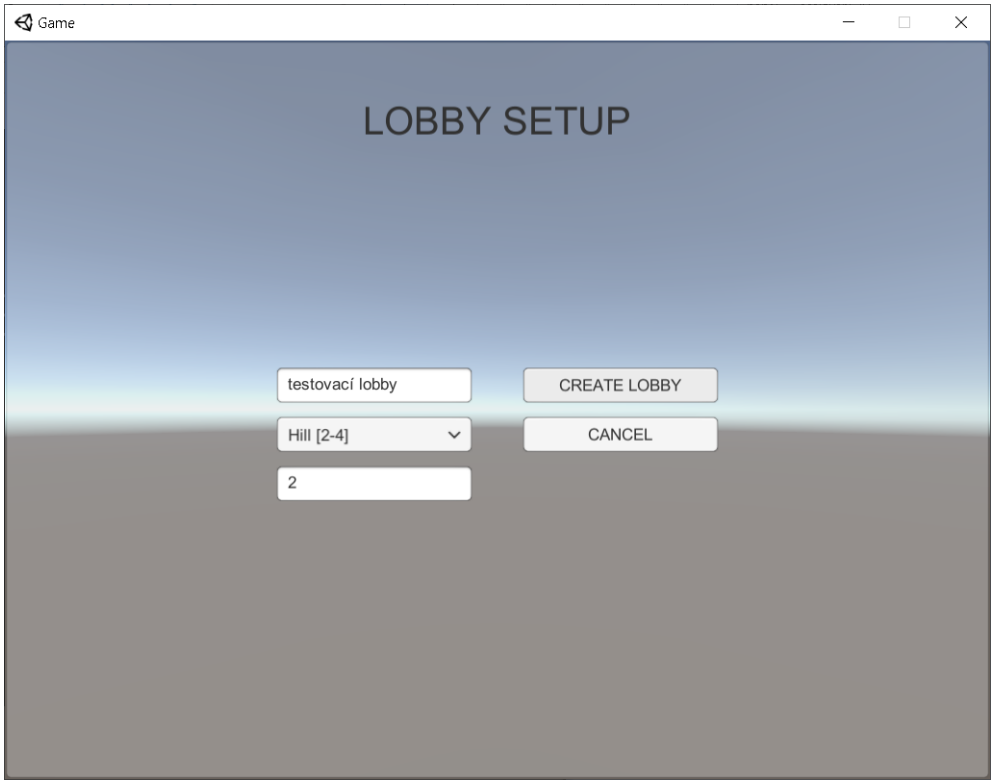


Obrázek 23: Uživatelské rozhraní vstupního lobby

Pro přihlášení se do herního lobby musí uživatel buď herní lobby vytvořit (pomocí tlačítka tvorby nového herního lobby) nebo stisknout levé tlačítko myši nad vybraným herním lobby a poté stisknout tlačítko připojení. Po úspěšném připojení je uživatel přesunut do uživatelského rozhraní herního lobby. Po stisknutí tlačítka tvorby herního lobby je uživatel přesunutý do uživatelského rozhraní tvorby herního lobby. Pro odhlášení uživatele stačí stisknout tlačítko odhlášení.

5.5.3 Tvorba nového lobby

Uživatelské rozhraní tvorby nového herního lobby obsahuje jednoduchý formulář, pomocí kterého se nastavuje samotné lobby.

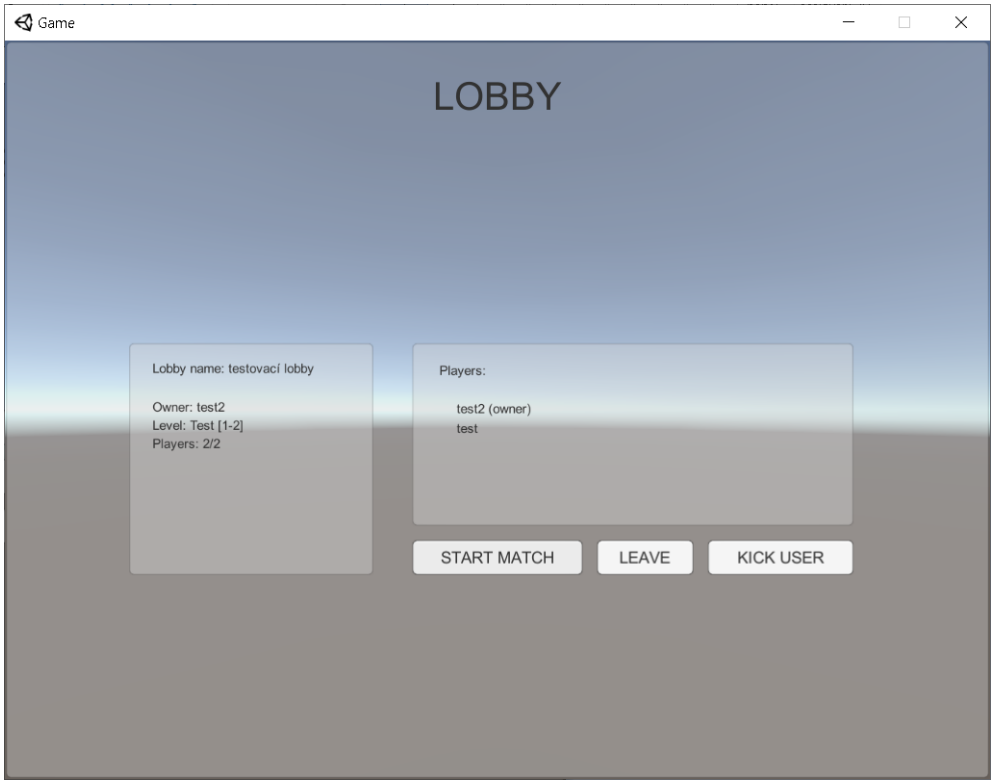


Obrázek 24: Uživatelské rozhraní tvorby nového lobby

Pokud při tvorbě herního lobby dojde k chybě, je stejně jako v předchozím uživatelským rozhraní uživateli tato skutečnost oznámena výpisem červeného upozornění nad formulářem.

5.5.4 Herní lobby

Uživatelské prostředí herního lobby je odlišné pro majitele a ostatní příslušníky daného lobby. Všichni uživatelé mohou dané lobby opustit (v případě opuštění majitele je lobby zrušeno). Majitel může navíc vykopnout zvoleného uživatele (stejný postup jako při volbě herního lobby v obrazovce vstupního lobby) nebo herní lobby při splnění podmínek spustit. Pokud se nějaký uživatel napojí nebo odpojí, tak se informace o daném lobby a seznam uživatelů aktualizuje.



Obrázek 25: Uživatelské rozhraní herního lobby

Jakmile se do lobby napojí potřebný počet uživatelů, může majitel dané lobby spustit stistknutním tlačítka pro nastartování hry. Po jeho stisknutí se všem účastníkům daného lobby přepne obrazovka na uživatelské rozhraní samotné hry spolu s načtenou herní úrovní.

5.5.5 Uživatelské rozhraní samotné hry

Uživatelské rozhraní samotné hry je velice jednoduché. Uprostřed horní části obrazovky je vypsán aktuální počet všech surovin a čas trvání dané hry. V levém dolním rohu se nachází vstupní pole pro zasílání zpráv. V pravém horním rohu je zobrazena minimapa, kterou je možné přiblížit. Při stisknutí levého tlačítka myši kdekoliv do prostoru minimapy dojde k přesunu herní kamery na dané místo na mapě. Při najetí kurzoru na jakýkoliv herní objekt se zobrazí informace o daném herním objektu.

Obrázek 26: Uživatelské rozhraní samotné hry

Pokud uživatel klikne na jednu z jeho budov, zobrazí se v pravém dolním rohu nákupní nabídka zvolené budovy. Ta obsahuje tlačítka, které reprezentují položky nakoupení daného produktu (jednotky, budovy nebo vylepšení). Při najetí kurzoru na položku je zobrazen popis dané položky spolu s cenou nákupu. Pokud uživatel nedisponuje daným počtem surovin, bude položka deaktivována. Po úspěšném nákupu je daná položka zavedena na konec nákupní fronty, která postupně zpracuje daný příkaz. Při nákupu budovy je nákupní nabídka schována a uživatel volí místo stavby zvolené budovy. Po odsouhlasení pozice stavby se stavba provede a nákupní nabídka bude znovu zobrazena.

5.6 Ovládání serverů

Ovládání serverů je velice jednoduché. Po spuštění se jak aplikace `MasterServer`, tak `GameServer` můžou ukočit stisknutím libovolného tlačítka klávesnice.

Závěr

Cílem práce bylo naprogramovat libovolnou strategickou hru v herním enginu `Unity3d`, která měla umožnovat hraní po síti. Samotná hra je posazena do prostředí raného středověku.

Herní mechanismus se skládá ze stavby budov, obsazování zdrojů surovin, nákup herních jednotek a jejich vylepšení a jednoduchý soubojový systém. Hra obsahuje typické prvky strategických her jako je kamera s pohledem z vrchu, minimapa a možnost ovládání jednotek skrz ni.

Serverová část se skládá z hlavního serveru, který je určený k správě uživatelů, herních lobby a herních serverů, které umožňují hrát hru po síti bez požadavků na veřejnou IP adresu nebo NAT punching. Hlavní server si ukládá data do MySQL databáze.

Conclusions

The aim of this work was to implement an arbitary networking game strategy in `Unity3D` game engine. Game is seated in early Middle Ages.

Game mechanism consists of construction of buildings, occupation of resources sources, purchase of game units and their upgrades and simple combat system. Game contains typical strategy game elements such as top-down camera view and minimap which allows controling game units.

Server part is consist of master server, which is designated to handle users, game lobbies and game servers, and of game servers, which allow networking game without necessities of public IP address or NAT punching.

A Obsah přiloženého CD

bin/
: Program hry Game.exe ve složce `/game`, program hlavního serveru ve složce `/masterserver`, program herních serverů v adresáři `/gameserver` a nakonec program testovací verze hry ve složce `/offlinetest`. Adresář obsahuje i všechny runtime knihovny a další soubory potřebné pro bezproblémový běh programu z CD.

doc/
: Text práce ve formátu PDF, vytvořený s použitím závazného stylu KI PřF UP v Olomouci pro závěrečné práce včetně všech příloh, a všechny soubory potřebné pro bezproblémové vygenerování PDF dokumentu textu (v ZIP archivu), tj. zdrojový text textu, vložené obrázky, apod.

src/
: Kompletní zdrojové texty aplikace Game, MasterServer a GameServer a sdílené knihovny Shared se všemi potřebnými (příp. převzatými) zdrojovými texty, knihovnami a dalšími soubory potřebnými pro bezproblémové vytvoření spustitelných verzí programu.

README.txt
: Instrukce pro instalaci a spuštění aplikací, včetně všech požadavků pro jeho bezproblémový provoz.

Navíc CD obsahuje:

install/
: Instalátory aplikací, runtime knihoven a jiných souborů potřebných pro provoz programů, které nejsou standardní součástí operačního systému určeného pro běh programu.

literature/
: Vybrané položky bibliografie, příp. jiná užitečná literatura vztahující se k práci.

U veškerých cizích převzatých materiálů obsažených na CD jejich zahrnutí dovolují podmínky pro jejich šíření nebo přiložený souhlas držitele copyrightu. Pro všechny použité (a citované) materiály, u kterých toto není splněno a nejsou tak obsaženy na CD, je uveden jejich zdroj (např. webová adresa) v bibliografii nebo textu práce nebo v souboru `README.txt`.

B Herní manuál

Tato přídavná kapitola popisuje veškeré herní jednotky, budovy, zdroje surovin a vylepšení spolu s popisem všech jejich atributů a vlastností.

B.1 Prostředí hry

Hra je zasazena do období raného středověku na území Evropy – tedy období měče, luku a kopí.

B.2 Cíl hry

Cílem hry je porazit ostatní hráče pomocí zničení všech jejich budov. Pokud je hráči zničená hlavní budova, tak mu je odepřeno stavět další budovy. Jedinou výjimkou je zaplacení daně v ruinách hlavní budovy, čímž se obnoví hlavní budova.

B.3 Seznam jednotek

B.3.1 Šermíř

Nejlevnějšími a nejdostupnějšími jednotkami jsou šermíři. Ty lze najmout v kasárnách za cenu 10 kusů zlata. Šermíři mají následující atributy:

Počet bodů zdraví – 100 bodů zdraví

Rychlost přesunu – 5 metrů za vteřinu

Vzálenost zpozorování nepřítele – 10 metrů

Síla útoku – 20 bodů zdraví

Dosah útoku – 1 metr

Cooldown dalšího útoku – 1 vteřina

B.3.2 Kopíník

Kopíníci jsou sice silnější jako šermíři, avšak pomalejší. Lze je najmout v kasárnách za cenu 20 kusů zlata. Šermíři mají následující atributy:

Počet bodů zdraví – 120 bodů zdraví

Rychlost přesunu – 4.5 metrů za vteřinu

Vzálenost zpozorování nepřítele – 10 metrů

Síla útoku – 30 bodů zdraví

Dosah útoku – 1 metr (1.2 metru při nákupu vylepší dlouhého kopí)

Cooldown dalšího útoku – 1.5 vteřiny

B.3.3 Lukostřelec

Lukostřelec je jediná jednotka, která je schopná boje na dálku. Také jako jediná jednotka má slabinu proti jezdcům na koni (zranění od útoku jezdců je dvojnásobné). Lukostřelce je možné najmout na střelnici za 5 kusů dřeva a 15 kusů zlata. Tato jednotka disponuje následujícími atributy:

Počet bodů zdraví – 75 bodů zdraví

Rychlost přesunu – 5 metrů za vteřinu

Vzálenost zpozorování nepřítele – 11 metrů

Síla útoku – 30 bodů zdraví (45 při nákupu vylepšení ohnivého šípu)

Dosah útoku – 11 metrů

Cooldown dalšího útoku – 1 vteřina

B.3.4 Jezdec na koni

Jezdec na koni je nejrychlejší jednotkou ve hře. Tento druh jednotky lze najímat ve stájích za cenu 1 kusu dřeva, 1 kusu kamene a 20 kusů zlata. Jezdci mají slabinu proti kopíníkům (zranění od kopíníků je dvojnásobné).

Počet bodů zdraví – 150 bodů zdraví

Rychlost přesunu – 10 metrů za vteřinu

Vzálenost zpozorování nepřítele – 10 metrů

Síla útoku – 50 bodů zdraví

Dosah útoku – 1.1 metru

Cooldown dalšího útoku – 1.2 vteřiny

B.3.5 Mnich

Mnich jako jediná jednotka nezpůsobuje nepřátelským jednotkám žádné poškození. Na druhou stranu dokáže mnich jako jediná jednotka uzdravovat okolní jednotky. Tuto jednotku lze najmout v chrámu za 10 kusů zlata.

Počet bodů zdraví – 50 bodů zdraví

Rychlost přesunu – 4 metry za vteřinu

Vzálenost zpozorování nepřítele – 1 metr

Síla útoku – 0 bodů zdraví

Dosah útoku – 1 metr

Cooldown dalšího útoku – 1 vteřina

B.3.6 Princ

Princ je nejvzácnější a nejdražší jednotkou ve hře. Kromě obrovského poškození má princ vedlejší efekt vůdcoství – všechny nepřátelské jednotky, které jsou na dosah zpozorování prince, přijímají dvojnásobné poškození (od jakékoliv jednotky). Stejně jako mnichy lze prince najímat v chrámu, a to za cenu 10 kusů dřeva, 10 kusů kamene a 50 kusů zlata. Jednotka prince má následující atributy:

Počet bodů zdraví – 300 bodů zdraví

Rychlost přesunu – 5 metrů za vteřinu

Vzálenost zpozorování nepřítele – 11 metrů

Síla útoku – 30 bodů zdraví

Dosah útoku – 1.1 metr

Cooldown dalšího útoku – 0.8 vteřiny

B.4 Seznam budov

B.4.1 Hlavní budova

Nejdůležitejší budova, bez které se žádný hráč neobejde. Také jako jediná budova vyrábí suroviny a to každých 5 vteřin vyrobí 1 kus dřeva, 1 kus kamene a 5 kusů zlata. Také jde o jedinou budovu, kde lze nakoupit stavbu jiných budov. Atributy hlavní budovy:

Počet bodů zdraví – 1000 bodů zdraví

Nákupní nabídka:

Kasárny
: 10 kusů dřeva a 10 kusů kamene; okamžitý nákup

Střelnice
: 15 kusů dřeva a 15 kusů kamene; okamžitý nákup

Stáje
: 20 kusů dřeva a 20 kusů kamene; okamžitý nákup

Trh
10 kusů dřeva, 10 kusů kamene a 10 kusů zlata; okamžitý nákup

Kovárna
15 kusů dřeva, 15 kusů kamene a 15 kusů zlata; okamžitý nákup

Chrám
10 kusů dřeva, 10 kusů kamene a 20 kusů zlata; okamžitý nákup

B.4.2 Ruiny hlavní budovy

Pokud je hráči zničena hlavní budova, na jejím místě zůstanou ruiny. Ruiny hlavní budovy mají jedinou nabídku nákupního menu – opravu hlavní budovy. Ruiny mají následující atributy:

Počet bodů zdraví – nekonečno (nezničitelná budova)

Nákupní nabídka:

Oprava hlavní budovy
50 kusů dřeva, 50 kusů kamene a 50 kusů zlata; okamžitý nákup

B.4.3 Kasárny

Kasárny slouží k nákupu šermířů, kopíníků a vylepšení dlouhého kopí.

Počet bodů zdraví – 500 bodů zdraví

Nákupní nabídka:

Šermíř
10 kusů zlata; nákup trvá 3 vteřiny

Kopíník
20 kusů zlata; nákup trvá 5 vteřin

Dlouhé kopí
10 kusů dřeva a 50 kusů zlata; nákup trvá 10 vteřin

B.4.4 Střelnice

Střelnice slouží k nákupu lučišníků a vylepšení ohnivého šípu.

Počet bodů zdraví – 500 bodů zdraví

Nákupní nabídka:

Šermíř
5 kusů dřeva a 20 kusů zlata; nákup trvá 5 vteřin

Ohnivé šípy
10 kusů dřeva a 100 kusů zlata; nákup trvá 20 vteřin

B.4.5 Stáje

Stáje slouží k nákupu jezdců na koni a mají následující atributy:

Počet bodů zdraví – 500 bodů zdraví

Nákupní nabídka:

Jezdec na koni
1 kus dřeva, 1 kus kamene a 20 kusů zlata; nákup trvá 5 vteřin

B.4.6 Trh

Trh může hráč využívat k výměně přebytečných surovin za ty, které hráčovi schází.

Počet bodů zdraví – 250 bodů zdraví

Nákupní nabídka:

1 kus dřeva
2 kusy zlata; okamžitý nákup

1 kus kamene
2 kusy zlata; okamžitý nákup

1 kus zlata
1 kus dřeva a 1 kus kamene; okamžitý nákup

B.4.7 Kovárna

Kovárna slouží k nákupu zbylých vylepšení – vylepšené opevnění, rychlejší výroba surovin a rychlejší nákup jednotek.

Počet bodů zdraví – 700 bodů zdraví

Nákupní nabídka:

Vylepšené opevnění
50 kusů dřeva, 50 kusů kamene a 10 kusů zlata; nákup trvá 10 vteřin

Rychlejší výroba surovin
10 kusů dřeva, 10 kusů kamene a 50 kusů zlata; nákup trvá 10 vteřin

Rychlejší nákup jednotek
10 kusů dřeva, 10 kusů kamene a 50 kusů zlata; nákup trvá 10 vteřin

B.4.8 Chrám

Chrám slouží k nákupu mnichů a princů:

Počet bodů zdraví – 200 bodů zdraví

Nákupní nabídka:

Mnich
10 kusů zlata; nákup trvá 3 vteřiny

Princ
10 kusů dřeva, 10 kusů kamene a 150 kusů zlata; nákup trvá 1 vteřinu

B.5 Seznam zdrojů surovin

Zdroje surovin jsou mimo hlavní budovu jediným zdrojem surovin. Na začátku hry je každý zdroj surovin neutrální (tedy nepatří žádnému hráči). Tyto zdroje se obsazují pomocí povelu jednotky (větší počet jednotek lineárně zrychlí obsazování surovinového zdroje). Vlastnictví surovinového zdroje je znázorněno barevným odstínem budovy. Hráči jsou suroviny přičteny pouze tehdy, je-li barva odstínu zdroje úplná.

B.5.1 Dřevorubecký srub

Dřevorubecký srub vyrábí vlastníkovi 1 kus dřeva každé 3 vteřiny.

B.5.2 Kamenolom

Kamenolom vyrábí vlastníkovi 1 kus kamene každé 3 vteřiny.

B.5.3 Zlatý důl

Zlatý důl vyrábí vlastníkovi 1 kus zlata každé 3 vteřiny.

B.5.4 Tvrz

Tvrz je nejefektivnější způsob získávání zlata – každých 5 vteřin vyrobí 10 kusů zlata.

B.6 Seznam vylepšení

Všechna vylepšení se nakupují pouze jedinkrát a přetrvávají tak u hráče až do konce hry (přetrvají i zničení budovy, ve které byly nakoupeny).

B.6.1 Dlouhé kopí

Nákupem vylepšení dlouhého kopí se navýší dosah útoku kopníků o 0.2 metru.

B.6.2 Ohnivé šípy

Vylepšení ohnivých šípů dodá 15 dodatečných bodů poškození pro všechny lukostřelce daného hráče.

B.6.3 Vylepšené opevnění

Nákupem vylepšeného opevnění se hráči doplní body zdraví budov, které utrpěly poškození a zároveň se zdvojnásobí maximální počet bodů zdraví všech budov (i takovým budovám, které hráč teprve postaví).

B.6.4 Rychlejší výroba surovin

Nákup vylepšení rychlejší výroby surovin zkrátí dobu výroby všech suroviny na polovinu.

B.6.5 Rychlejší nákup jednotek

Koupí rychlejšího nákupu jednotek se zkrátí doba všech nákupů na polovinu.

Literatura

[1] STRATEGICKÁ VIDEOHRA nebo též strategie (anglicky strategy video game) je videoherní žánr. Wikipedia [online]. 2020-11-14. [cit. 2021-04-28]. Dostupné z: https://cs.wikipedia.org/wiki/Strategická_videohra

[2] JETBRAINS RIDER, Fast and powerful cross-platform .NET IDE [online]. ©2021. [cit. 2021-04-12]. Dostupné z: https://www.jetbrains.com/rider/

[3] C# je cross-platformní objektově orientovaný programovací jazyk. Wikipedia [online]. 2020-04-13. [cit. 2021-04-12]. Dostupné z: https://cs.wikipedia.org/wiki/C_Sharp

[4] UNITY je multiplatformní herní engine vyvinutý společností Unity Technologies. Wikipedia [online]. 2021-04-27. [cit. 2021-04-28]. Dostupné z: https://cs.wikipedia.org/wiki/Unity_(herní_engine)

[5] LITENETLIB: Lite reliable UDP library for .NET Framework. GitHub [online]. 2020-12-02. [cit. 2021-04-12]. Dostupné z: https://github.com/RevenantX/LiteNetLib

[6] MYSQLCONNECTOR is ADO.NET data provider for MySQL. GitHub [online]. 2021-04-12. [cit. 2021-04-15]. Dostupné z: https://github.com/mysql-net/MySqlConnector/

[7] 1500 ARCHERS ON A 28.8: Network Programming in Age of Empires and Beyond [online]. 2001-03-22. [cit. 2021-04-16]. Dostupné z: https://www.gamasutra.com/view/feature/131503/

[8] RTS CLIENT-SERVER NETWORKING: Client-Server Lock Step Model. Medium [online]. 2019-03-15. [cit. 2021-04-17]. Dostupné z: https://medium.com/@evan_73063/rtsclientservernetworking36e8154ff740

[9] CLINTON BRENNAN: Lockstep Implementation in Unity3D [online]. 2013-12-06. [cit. 2021-04-18]. Dostupné z: http://clintonbrennan.com/2013/12/lockstep-implementation-in-unity3d/

[10] QUADTREE: A quadtree is a tree data structure in which each internal node has exactly four children. Wikipedia [online]. 2021-04-08. [cit. 2021-04-21]. Dostupné z: https://en.wikipedia.org/wiki/Quadtree

[11] A* SEARCH ALGORITHM: A* is a graph traversal and path search algorithm. Wikipedia [online]. 2021-04-21. [cit. 2021-04-22]. Dostupné z: https://en.wikipedia.org/wiki/A*_search_algorithm

[12] UNITY USER MANUAL: Use the Unity Editor to create 2D and 3D games, apps and experiences [online]. 2021-04-19. [cit. 2021-04-25]. Dostupné z: https://docs.unity3d.com/Manual/