Civ# 2 Egyszerű játékos fejlesztése

Ennek a dokumentumnak az elolvasása előtt érdemes elolvasni a Játékszabályt. Ha ennek a dokumentumnak az olvasása közben olyan kérdéseid vannak, mint pl. „mikor vásárolhatok fejlesztést?”, akkor mindenképpen olvasd el először a Játékszabályt.

A Civ# 2, mint hagyományos társasjáték

A Civ# 2 egy köralapú stratégiai játék, melyet leginkább egy hagyományos táblás társasjátékként érdemes elképzelni, amit két játékos játszik egy játékvezető segítségével. A játékosok körbeülik a táblát és egymás után jönnek sorban, ahogy a játékvezető felszólítja őket:

- A játékvezető először kiosztja az aktuális játékosnak az aranyat, amit abban a körben a birodalma megtermelt.
- Utána megkérdezi, hogy melyik egységével szeretne lépni egyet, és ezután lépteti az egységet a játékos válaszának megfelelően. A játékvezető ezután újra megkérdezi, hogy melyik egységével szeretne lépni egyet. Ez egészen addig ismétlődik, míg a játékos azt nem mondja, hogy már nem szeretne többet lépni.
- Ekkor a játékvezető megkérdezi a játékost, hogy milyen fejlesztéseket szeretne vásárolni. Erre a játékos válaszként mondhat egy fejlesztést, amit a játékvezető megad neki. Ezt addig ismétlik, míg a játékos azt mondja, hogy nem szeretne már fejlesztést vásárolni.
- Ezután a játékvezető megkérdezi az aktuális játékost, hogy hol szeretne kolóniát alapítani, és a kért kolóniát felhelyezi a játéktérre. Ezt addig ismétli, míg a játékos azt mondja, hogy nem szeretne több kolóniát alapítani.
- Ezt követően a játékvezető megkérdezi az aktuális játékost, hogy szeretne-e egy egységet vásárolni. Ha a játékos szeretne venni egy egységet, akkor azt a játéktérre helyezi. Ezt addig ismétli, míg a játékos azt mondja, hogy nem szeretne több kolóniát alapítani.
- A játékvezető ezután kiosztja a következő játékos aranyát és őt kezdi kérdezgetni.

A játékmenet védelmében érvénytelen lépési kérés esetén a játékvezető abbahagyja az aktuális játékos kérdezgetését abban a körben, és áttér a következő játékosra.

A Civ# 2, mint szoftveres társasjáték

A Civ# 2 játékot két nagyobb szoftverkomponens valósítja meg:

1. Játékvezérlő: egy GUI alkalmazás, mellyel egy játékot vezényelhetünk le. Egy ilyen játékban 2 játékos vehet részt. Ez a komponens tölti be a hagyományos társasjáték játékvezetőjének szerepét. Feladata, hogy ügyeljen a játék szabályainak betartására és végigkérdezze a játékosokat, hogy mit szeretnének lépni és végül végrehajtsa a kért lépéseiket.

2. Játékos: egy .NET-es DLL, melyet a Játékvezérlő Players mappájában kell elhelyezni (részletekért olvasd el az „A játékosok elhelyezése, egy összecsapás archiválása” fejezetet). Ez a komponens tölti be a hagyományos társasjáték játékosának szerepét. Feladata, hogy válaszoljon a játékvezető kérdéseire.

A játékvezérlő feladata, hogy betöltse és egymás ellen futtassa a játékosokat, nagyon hasonlóan ahhoz, mintha egy társasjátékban lenne egy kijelölt ember arra, hogy minden műveletet ő végezzen el a játéktáblán, és a tényleges játékosok csak az ő kérdéseire válaszolva (Melyik egységeddel lépsz? Vásárolsz valamit?) irányíthatják a bábuikat.

A játékvezérlő a játékosokat egy kijelölt mappában keresi DLL-ek formájában. Ez a dokumentum azt mutatja be, hogy hogyan lehet egy ilyen egyszer játékos DLL-t készíteni.

Projekt léterhozása egy üres játékossal

1. Visual Studio-ban hozzunk létre egy **Class Library** típusú projektet
2. Adjunk benne referenciát a mellékelt **CivSharp.Common dll**-re
3. A projektünkben hozzunk létre egy osztályt, mely implementálja a **CivSharp.Common.IPlayer** interfészt. Ezzel van egy formailag teljesen érvényes játékosunk, ami azonban nem csinál semmit.

Játékoslogika készítése

A játékoslogika elkészítéséhez az IPlayer interfész metódusait (és tulajdonságait) kell megvalósítanunk. Ezeket a metódusokat fogja a játékvezérlő adott sorrendben végighívni. A játékosunk azt fogja csinálni a játékban, amit ezen hívások során a játékvezérlőnek visszatérési értékként átadunk.

Alapelvként érdemes megjegyezni, hogy **ha a játékosunk forráskódja egy érvénytelen akciót ad visszatérési értékként a játékvezérlőnek, akkor a játékvezérlő az adott körben már többször nem hívja meg a játékosunk metódusait**. Ha például olyan egységet szeretne vásárolni a játékos, amire nincs kreditje, akkor a vásárlás sikertelen lesz és a játékvezérlő a következő játékos metódusait kezdi el hívni.

A játékos elnevezése, faj meghatározása

Első lépés, hogy megadjuk a játékosunk nevét:

```
        public string PlayerName
        {
            get { return "A csapatom neve, amivel regisztráltam"; }
        }
```

Nagyon fontos, hogy a játékos neve pontosan egyezzen a csapatod nevével, amivel a versenyre regisztráltál. Erre két okból van szükség: (i) fontos, hogy a név egyedi legyen, mert a játékvezérlő nem indít el játékot két azonos nevű játékossal és (ii) a szervezők kizárhatják a nem saját csapatnév alatt játszó játékosokat.

Második lépés, hogy megadjuk a játékosunk civilizációját:

```
        public string PlayerRace
        {
            get { return "civ1"; }
        }
```

Fontos hogy egy létező civilizáció nevét adjuk meg pontosan, különben nem fog elindulni a játék. A játékban létező civilizációk nevei:

- civ1
- civ2

A játéktér (világ) adatainak fogadása

A következő fontos lépés, hogy tudjuk fogadni a játékvezérlő által küldött világ állapotot. Ez az információ mindig a RefreshWorldInfo metódus bemenő paramétereként érkezik. Az alábbi kódrészlet későbbi felhasználásra eltárolja ezt a beérkező világ-információt:

```
private int turn;
private WorldInfo world;

public void RefreshWorldInfo( int turn, WorldInfo world )
{
      this.turn = turn;
      this.world = world; }
```

Ha minden ilyen világfrissítéskor szeretnéd egy külön tagváltozóba eltárolni a saját egységeidet, akkor ezt megteheted úgy, hogy a beérkező WorldInfo objektum Units tulajdonságából kiválogatod azokat az egységeket, amiknek Te vagy a tulajdonosa, vagyis ahol az Owner megegyezik a Te neveddel:

```
private int turn; private
WorldInfo world; private
UnitInfo[] myUnits;

public void RefreshWorldInfo( int turn, WorldInfo world )
{
      this.turn = turn;
      this.world = world;
      // kikeresem a saját egységeimet: ezek tulajdonosa (Owner) én vagyok
      this.myUnits = this.world.Units.
            Where( unit => unit.Owner == this.PlayerName ).ToArray(); }
```

A fenti kódrészlethez nagyon hasonlóan lehet kikeresni a kolóniáimat is, vagy a saját játékosomra vonatkozó más adatokat is:

```
private int turn; private
WorldInfo world; private
UnitInfo[] myUnits; private
PlayerInfo myPlayer; private
CityInfo[] myCities;

public void RefreshWorldInfo( int turn, WorldInfo world )
```

```
{

      this.turn = turn;
      this.world = world;

      // kikeresem a saját egységeimet: ezek tulajdonosa (Owner) én vagyok
      this.myUnits = this.world.Units.
            Where( unit => unit.Owner == this.PlayerName ).ToArray();
      // kikeresem a saját játékosomat is a világ leírásából
      this.myPlayer = this.world.Players
            .Where( player => player.Name == this.PlayerName ).Single();
      // kikeresem a kolóniáimat
      this.myCities = this.world.Cities
            .Where( city => city.Owner == this.PlayerName ).ToArray();

}
```

A fenti megoldás a világ adataiból történő válogatásra csak egy példa, a játékos működésétől függően lehet, hogy vannak ennél sokkal jobb megoldások is.

A RefreshWorldInfo metódus minden körben meghívódik a játékosunk első lépési lehetősége előtt. A világ információit viszont lépéseink után is megkapjuk az ActionResult függvényben, tehát meg tudjuk nézni, hogy pl. sikeres volt-e egy lépésünk.

Egységek mozgatása

Azt hogy melyik egységünkkel mit szeretnénk lépni a játékvezérlő az OnMovement metódus meghívásával kérdezi meg. Ennek a metódusnak az elkészítésekor a feladatunk egy MovementData objektum elkészítése és visszaadása. Ha nem szeretnénk vagy nem tudunk lépni azt egy null átadásával jelezhetjük:

```
public MovementData OnMovement()
{
      if( this.myUnits.Length == 0 ) // nincs egységem => nem tudok lépni
      return null;
  var unit = this.myUnits[ 0 ]; // a legelső egységemmel fogok lépni  var
cmd = new MovementData(); // ebben adom meg, hogy mit lépek  cmd.UnitID
= unit.UnitID;
      cmd.FromX = unit.PositionX; cmd.FromY = unit.PositionY; // innen lépek
      cmd.ToX = unit.PositionX + 1; cmd.ToY = unit.PositionY; // egyet jobbra


      return cmd;
}
```

A játékvezérlő egy körben többször is meghívja az OnMovement metódust, egészen addig, míg null-t nem adunk vissza, így lehetőségünk nyílik egy körben többször több egységet is mozgatni. Figyeljünk oda, mert minden egységünknek van mozgáspontja, mely a lépésektől folyamatosan csökken. Egy egységgel csak akkor tudunk lépni, ha van még mozgáspontja. Ezt a mozgáspontot az egység MovementPoints tagváltozóján keresztül érjük el. Fontos, hogy két RefreshWorldInfo metódsuhívás között a MovemenPoints karbantartását ne felejtsük el elvégezni (pl. csökkenteni eggyel, ha léptünk egy

bizonyos egységgel, vagy az ActionResult függvényben visszajött világ-állapottal írjuk felül mindig az általunk lokálisan eltárolt információkat.)

```
// csak akkor léphet ez az egység, ha van még mozgáspontja
if( unit.MovementPoints > 0 )
{
	// mozgatás
	...
	// mozgáspontok csökkentése
	unit.MovementPoints--; }
```

Ha olyan területre próbálunk lépni, ahol ellenséges egység tartózkodik, abból csata lesz, melynek során az egyik egység életponto(ka)t veszít. Ha egy egységnek elfogynak az életpontjai, akkor meghal. Ha sikerül ellenséges kolóniába lépni (tehát nincs már ott további ellenséges egység), akkor megszerezzük azt. Ha megszerezzük valaki utolsó megmaradt kolóniáját, akkor a birodalma teljesen a miénk lesz. Erről és a csatáról a játékszabályokban olvashatsz pontosabban.

Fejlesztések vásárlása

Azt, hogy mit szeretnénk kifejleszteni, az OnResearch metódusban mondhatjuk meg:

```
public ResearchData OnResearch()
{
	return new ResearchData() { WhatToResearch = "őrzők tornya" }; }
```

A lehetséges fejlesztések megtalálhatók a játékszabályok között is: ***falu, város, őrzők tornya, kovácsműhely, barakk, harci akadémia, városháza, bank, barikád, fal***. Ha semmit sem szeretnénk vásárolni, akkor null-t kell visszaadnunk.

Figyeljünk oda, hogy minden fejlesztésnek ára van, melyet ki kell tudnunk fizetni (elég aranyunknak kell lenni rá). Ezenkívül néhány fejlesztést csak akkor tudunk megvásárolni, ha rendelkezünk már egy bizonyos másik fejlesztéssel is. Az egyes fejlesztések ára és a fejlesztések közötti függőségek szintén a játékszabályok között találhatók meg.

Kolóniák alapítása

Kolóniákat ott alapíthatunk, ahol harcképes egységünk áll és még nincsen másik kolónia. Természetesen a kolóniának is ára van, megépítéséhez rendelkeznünk kell ezzel az összeggel.

A játékvezérlő az **OnBuilding** metódussal kérdezi meg tőlünk, hogy hol szeretnénk kolóniát alapítani. A játékvezérlő egészen addig folyamatosan újrakérdez minket, amíg null-lal nem válaszolunk: így lehetőségünk nyílik egy körben több kolónia alapítására is.

```
public BuildingData OnBuilding()
{
	if( this.myPlayer.Money < 300 ) // nincs elég aranyam		return null;

	if( this.myUnits.Length == 0 ) // nincs alapító egységem
		return null;
```

```
      // veszek egy kolóniát az első egységem helyén       // vigyázat nem
ellenőrzőm, hogy itt van-e már kolónia!
      var unit = this.myUnits[ 0 ];
      var cmd = new BuildingData();
      cmd.PositionX = unit.PositionX;
      cmd.PositionY = unit.PositionY;
      return cmd;

}
```

Egységek kiképzése

Új egységeket a kolóniáinkba vásárolhatunk aranyért. A játékvezérlő az OnTraining metódusban kérdezi megy tőlünk, hol szeretnénk új egységeket felállítani. Ha nem szeretnénk több egységet venni azt null visszatérési értékkel jelezhetjük.

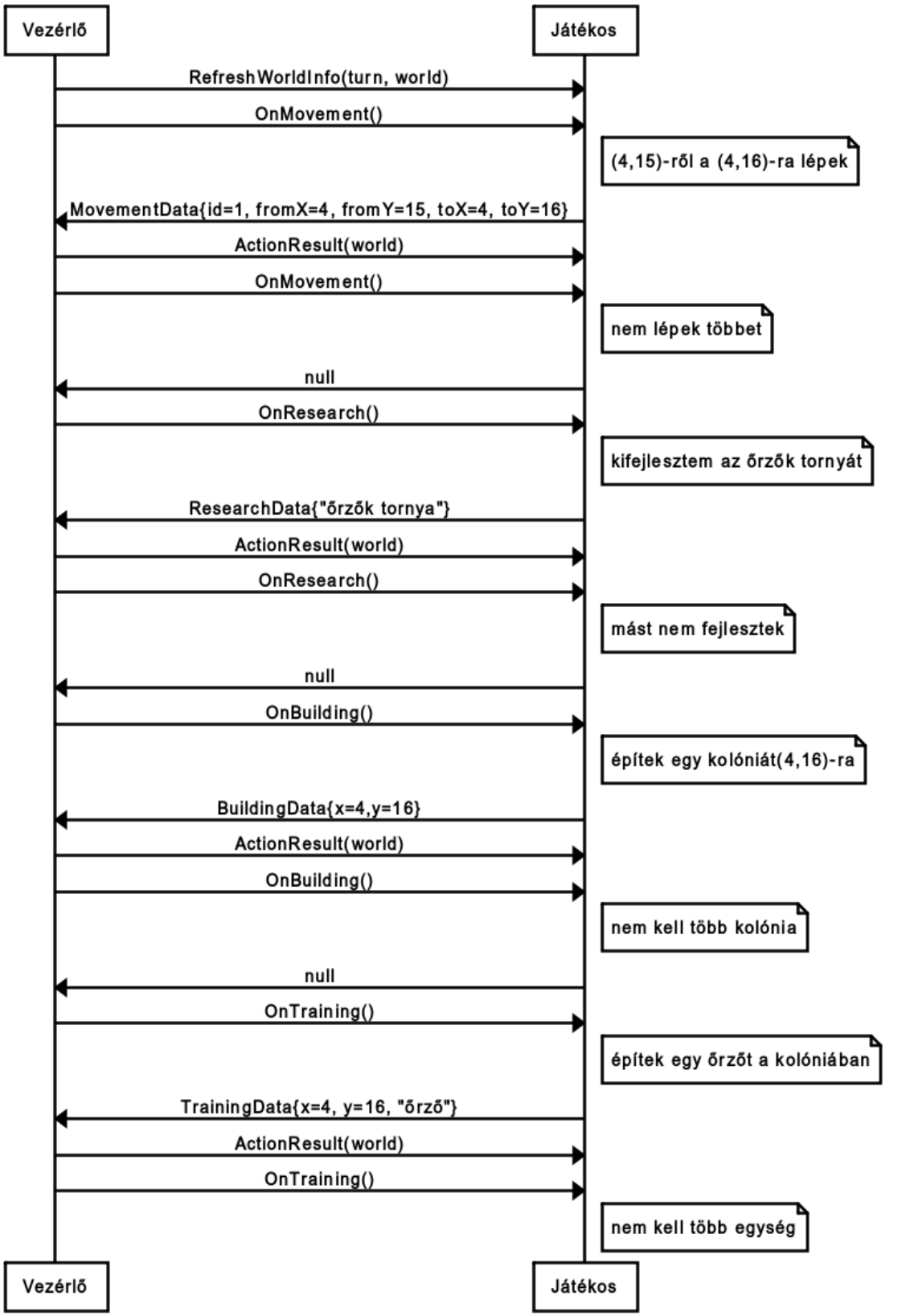
```
public TrainingData OnTraining()
{
      if( turn < 10 ) // a 10. kör előtt nem vásárlok
            return null;

      if( this.myPlayer.Money < 50 ) // nincs elég kreditem
            return null;

      // az első kolóniámban veszek lovagot
      var city = this.myCities[ 0 ];
      var cmd = new TrainingData();
      cmd.PositionX = city.PositionX;
      cmd.PositionY = city.PositionY;
       cmd.UnitTypeName = "lovag";
      return cmd;
}
```

A fenti kódrészlet a 10. kör után annyi lovagot vásárol az első kolóniába, amennyit a kreditje enged.

További metódusok

A jó játékoslogika implementálását további segédmetódusok segítik:

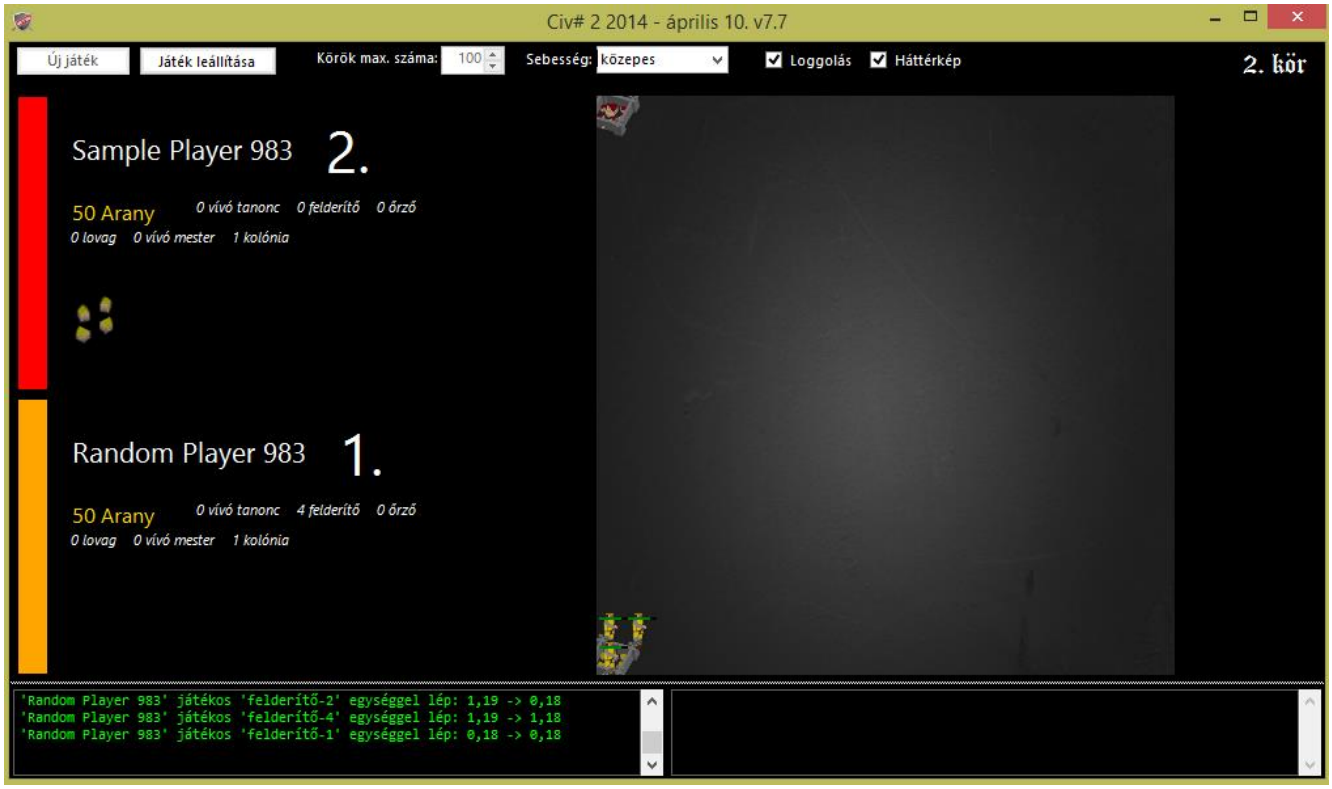
- **void ActionResult(WorldInfo world):** lépéseink után visszaadja nekünk a világ aktuális (a lépésünk utáni) állapotát.
- **GameOver(bool winner, string message):** vége a játéknak. A winner parameter azt mondja meg, hogy győztünk-e. A message-ben egy magyarázó üzenetet találhatunk.
- EnemyDestroyed(string playerName): megöltük az egyik ellenfelünket. Szuper, csak így tovább!
- UnitLost(string unitID): elvesztettük az egyik egységünket.
- CityLost(int positionX, int positionY): elvesztettük az egyik kolóniánkat.



www.websequencediagrams.com

A játékos kipróbálása

A játékosunkat úgy tudjuk kipróbálni, hogyha egy másik játékossal együtt bemásoljuk a játékvezérlő mellett található Players mappába, majd elindítunk a játékvezérlőben egy új játékot az új játék gomb megnyomásával.



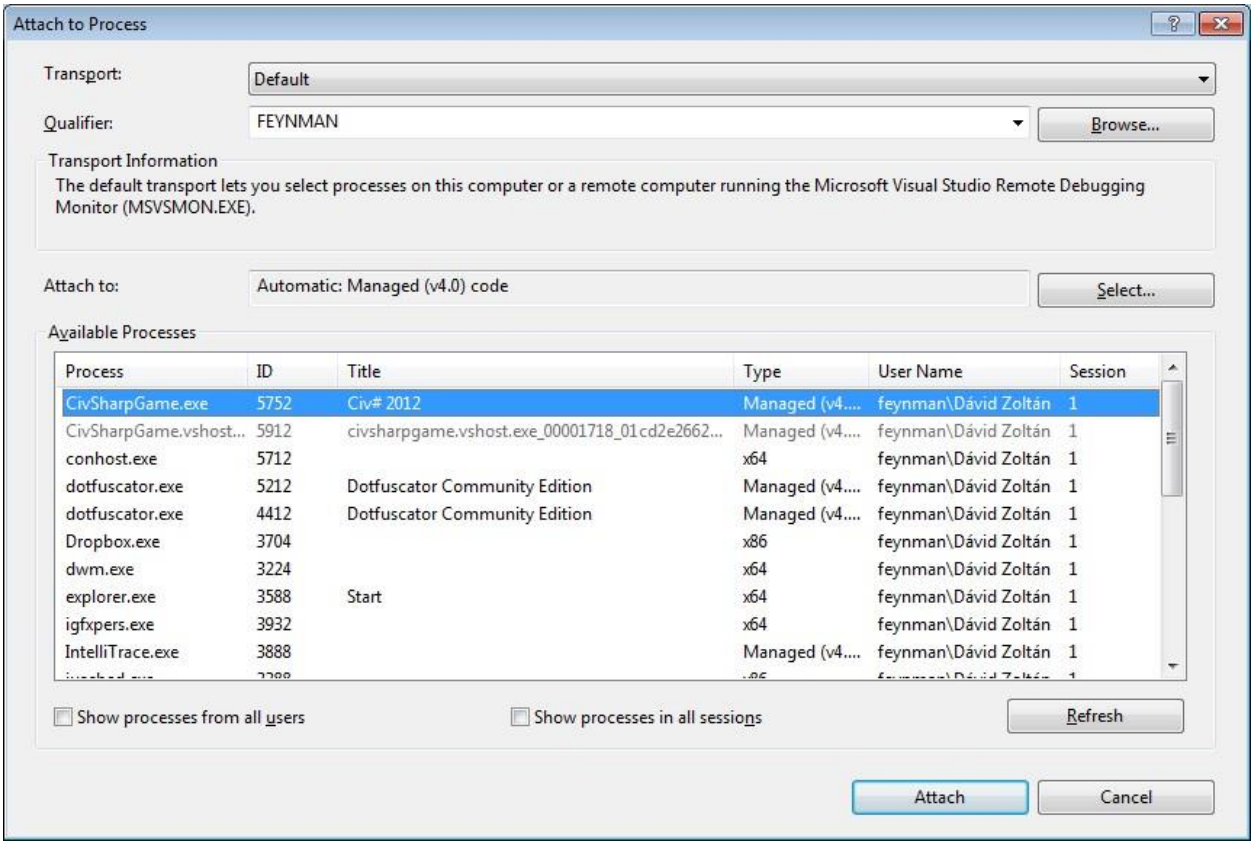
Játékvezérlő

A fejlesztés idejére tesztellenfélként használható a mellékelt RandomPlayer.

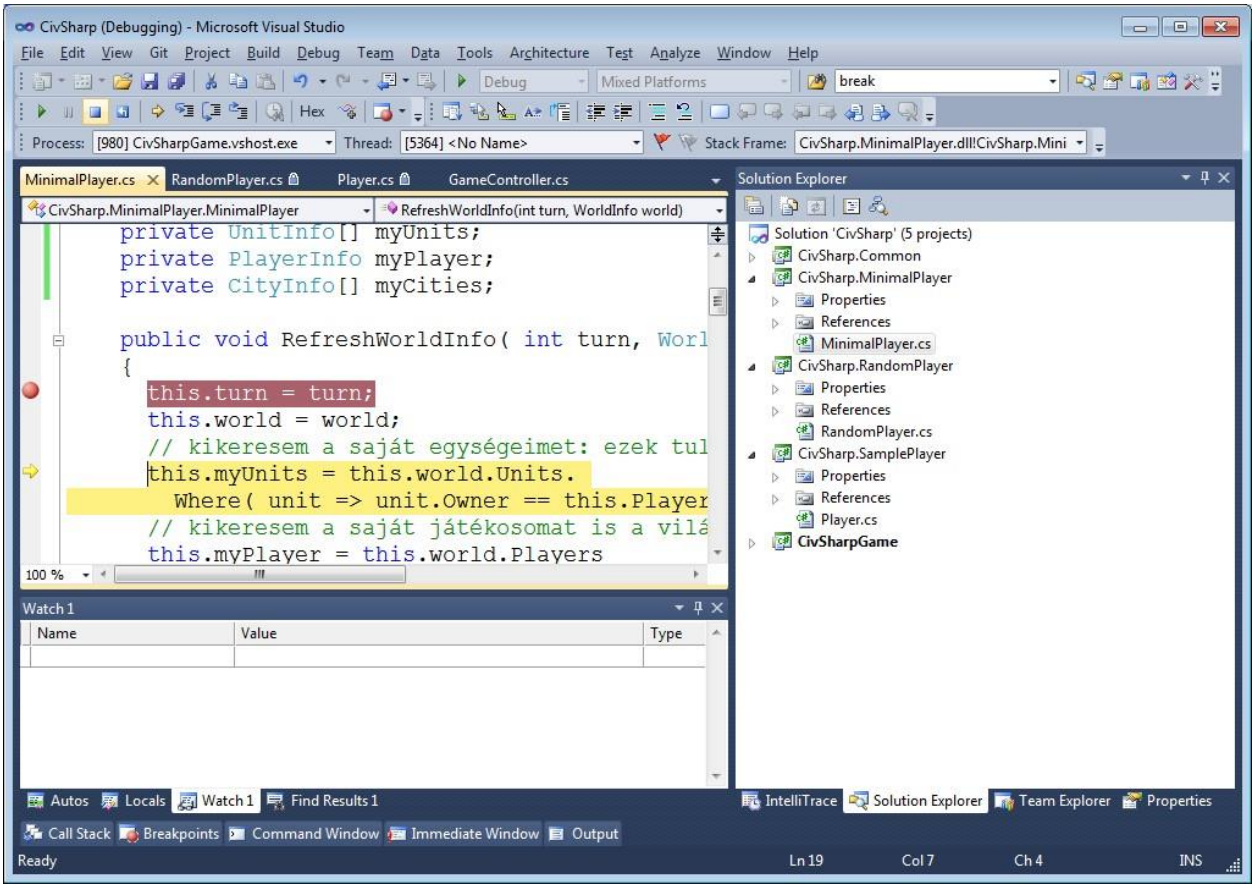
A játékos debuggolása

Lehetőség van arra, hogy a játékosunk kódjában töréspontot helyezzünk el, és debuggoljunk. Ehhez az alábbi lépések szükségesek:

1. Helyezzük el a töréspontot a játékosunk kódjában.
2. Fordítsunk egy Debug verziót a játékosból.
3. Másoljuk be ezt a frissen fordított DLL-t a játékosok mappájába.
4. Indítsuk el a játékvezérlőt
5. Visual Studioban válasszuk a Debug → Attach to Process… menüpontot
6. A felbukkanó ablakban válasszuk ki a játékvezérlőt (CivsharpGame)



7. Nyomjuk meg az Attach gombot.
8. Indítsunk el egy játékot a játékvezérlőben.
9. A Visual Studio debuggere meg fogja állítani a játékosunk futását a töréspontnál és innen lehetőségünk van a soronkénti léptetésre, változók kiértékelésre a megszokott módon.



A játékosok elhelyezése, egy összecsapás archiválása

A játékvezérlő az alábbi mappák közül sorrendben az első létezőből próbálja meg betölteni a játékosokat:

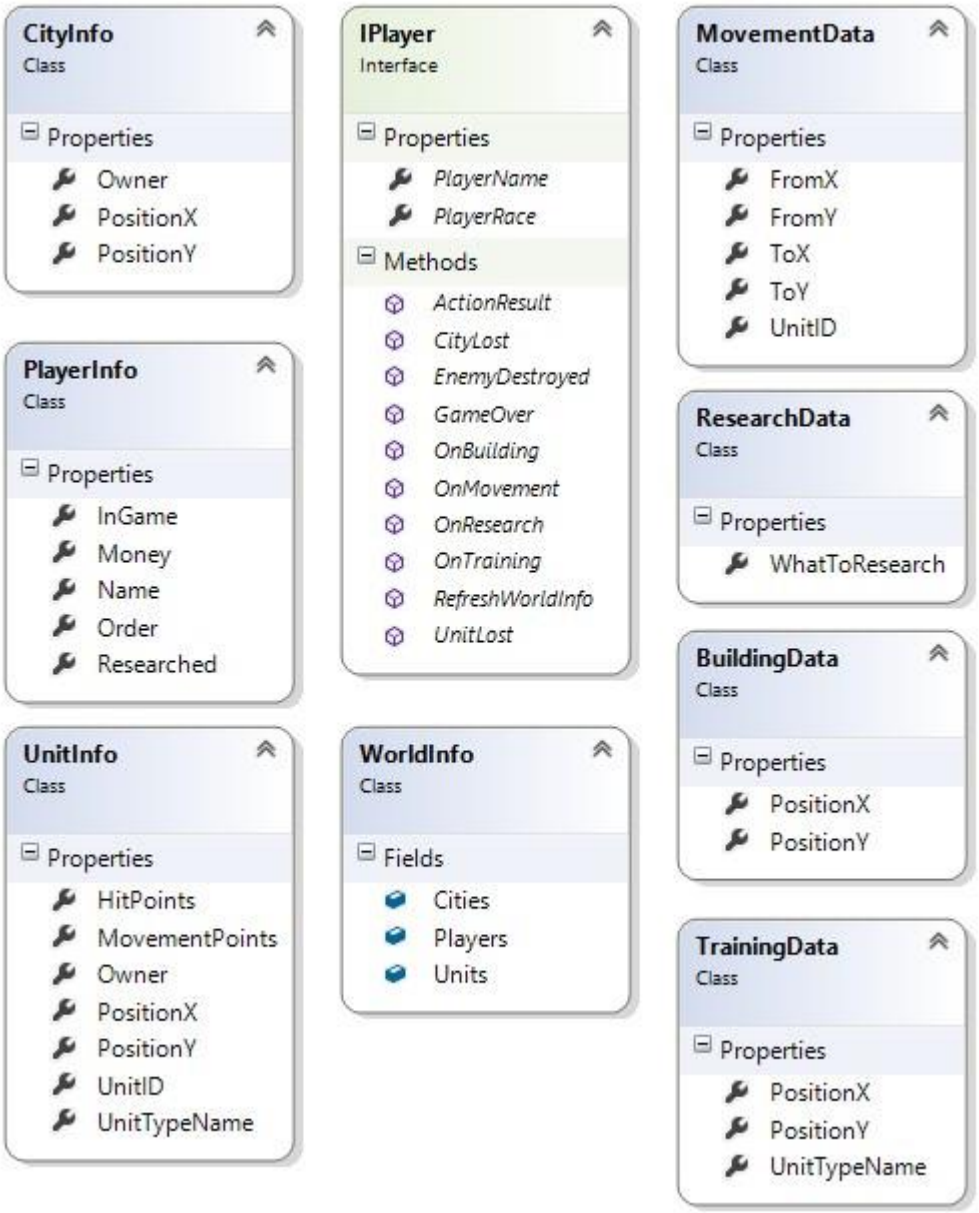
- C:\civsharp-players
- D:\civsharp-players
- D:\temp\civsharp-players
- A vezérlő mellett található players mappa

Ezért egy-egy játék játékosait (2 darab) a fenti mappák valamelyikébe kell másolni. A játékról készülő logfile szintén ebbe a mappába kerül.

Egy játék után érdemes a fenti mappát archiválni (pl. összezippelve átmozgatni máshová), mert abban benne lesznek a játékosok és az összecsapásukról készült log.

Melléklet

CivSharp.Common osztálydiagram



WhatToResearch: falu, város, őrzők tornya, kovácsműhely, barakk, harci akadémia, városháza, bank, barikád, fal

UnitTypeName: felderítő, őrző, lovag, vívó tanonc, vívó mester