Egy egyszerű játékos fejlesztése

A Civ# egy köralapú stratégiai játék, mely szoftver szempontból két fő komponenstípust tartalmaz:

1. Játékvezérlő: egy GUI alkalmazás, mellyel egy játékot vezényelhetünk le. Egy ilyen játékban 2-4 játékos vehet részt.
2. Játékos: egy .NET-es DLL-ben, melyet a Játékvezérlő Players mappájában kell elhelyezni (részletekért olvasd el az „A játékosok elhelyezése, egy összecsapás archiválása” fejezetet).

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 egy 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 2010-ben hozzunk létre egy Class Library típusú projektet
2. Adjunk benne referenciát a mellékelt CivSharp.Common dll-re
3. Az projektünkben hozzunk létre egy osztályt, mely implementálja a CivSharp.Common.IPlayer interfészt. Ezzel van egy formailag teljesen érvénye 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.

A játékos elnevezése

Első lépés, hogy adjunk egy egyedi nevet a játékosunknak:

```
public string PlayerName
{
      get { return "Adogatós Albatrosz"; }
}
```

Nagyon fontos, hogy a név egyedi legyen, mert a játékvezérlő nem indít el játékot, amiben két azonos nevű játékos szeretne játszani.

A játéktér (világ) adatainak fogadása

A következő fontos lépés, hogy tudjuk fogadni a játékvezérlő által küldött világ állapotot. Ez az információ mindig a RefereshWorldInfo 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 megtheted 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 városaimat 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 városaimat
      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 egyszer, a játékosunk első lépési lehetősége előtt hívódik, ezért a benne kapott adatok a következő körig nem frissülnek, azokat nekünk kell karbantartanunk (pl. a pénzünket csökkentenünk, ha vásároltunk valamit).

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 balra

	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 egységünkek van mozgáspontja, mely a lépésektők folymatosan 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.

```
// 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 elpusztul. Ha sikerül ellenséges városba lépni, akkor megszerezzük azt. Ha megszerezzük valaki utolsó megmaradt városát, 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 = "famegmunkálás" };
}
```

A lehetséges fejlesztések megtalálhatók a játékszabályok között is: famegmunkálás, íjászat, kerék, katapult, lándzsa, lótenyésztés, lovag, írás, bíróság, cölöpkerítés, kőfal. Ha semmit sem szeretnénk vásárolni, akkor pedig nullt kell visszaadnunk.

Figyeljünk oda, hogy minden fejlesztésnek ára van, melyet ki kell tudnunk fizetni (elég pénzünknek 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.

Városok alapítása

Városokat ott alapíthatunk, ahol harcoló egységünk áll és még nincsen másik város. Természetesen a városnak is ára van, megépítéséhez rendelkeznünk kell ezzel az összeggel.

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

```
public BuildingData OnBuilding()
{
	if( this.myPlayer.Money < 140 ) // nincs elég pénzem
		return null;

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

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

Egységek kiképzése

Új egységeket a városainkba vásárolhatunk pénzért. 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 pénzem
		return null;

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

A fenti kódrészlet a 10. kör után annyi talpast vásárol az első városába, amennyit a pénze enged.

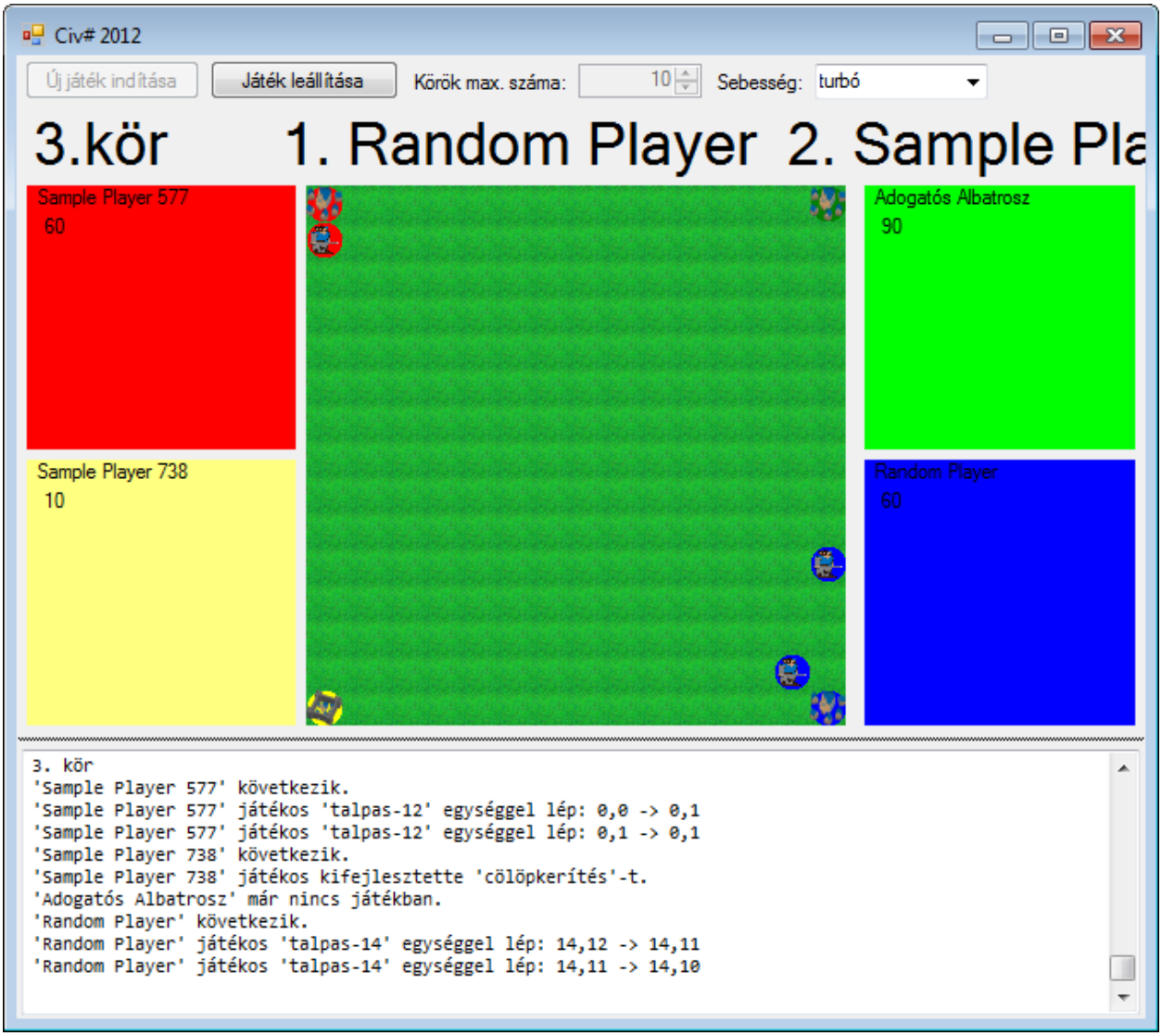
További metódusok

A jó játékoslogika implementálását további segédmetódusok segítik:

- ActionResult(bool succeeded): azt közli velünk a játékvezérlő, hogy az előző akciónk sikeres volt-e.
- 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 városunkat.

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.

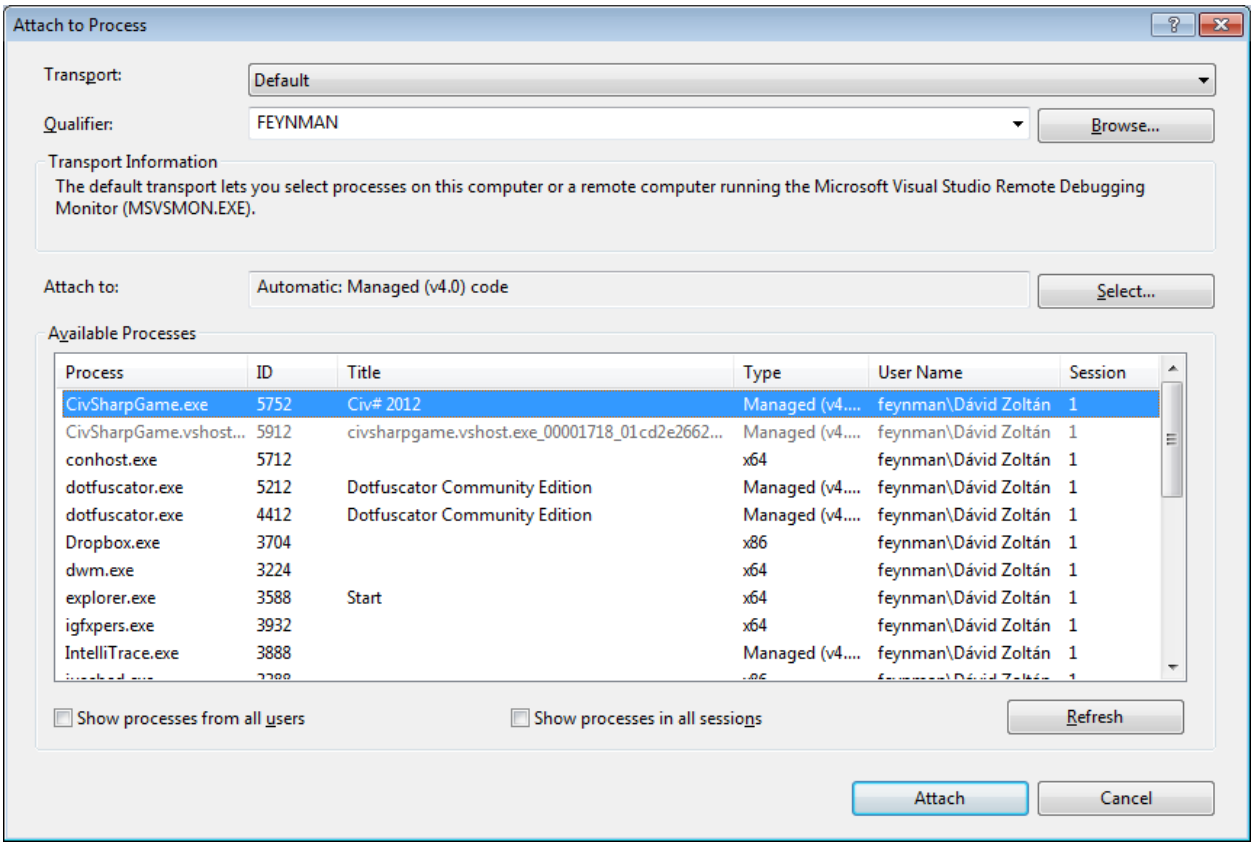


A fejlesztés idejére tesztellenfélként használható a mellékelt RandomPlayer, mely nagyjából random akciókat valósít meg, ráadásul játékon belül állandó, de játkosonként random névvel, így több példányban is elndítható.

A játékos debuggolása

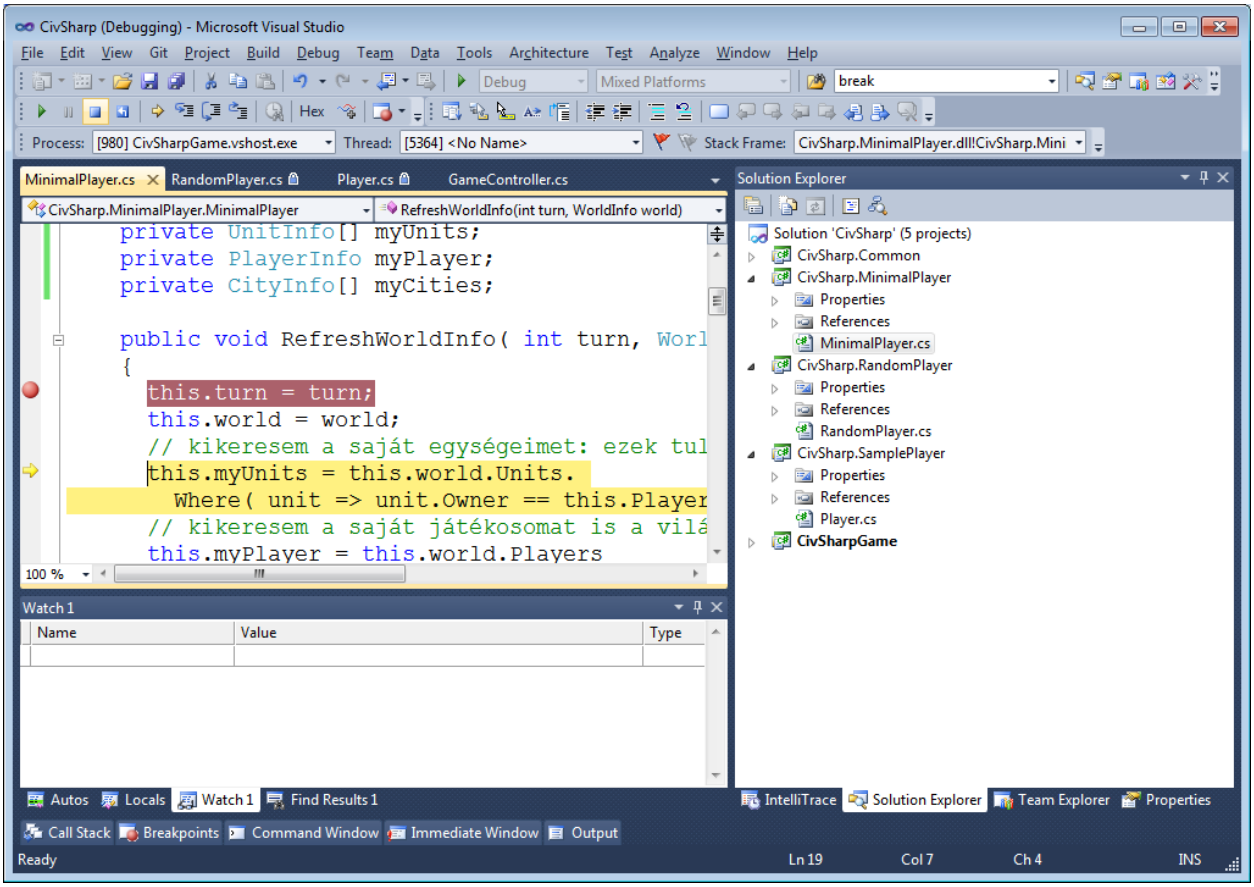
Lehetőség van arra, hogy a játékosunk kódjában töréspontot helyezzünk el, és debuggoljunk. Ehhez az alábbi lépések szükségesek:

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\cisharp-players
- A vezérlő mellett található players mappa

Ezért egy-egy játék játékosait (2-4 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.