

Załącznik: Instrukcja edycji kodu źródłowego gry

Potrzebne aplikacje

Możliwe, że w przyszłości ktoś chciałby rozwijać stworzoną grę. W tym celu niezbędna jest instalacja programów **Unity3d** i **Visual Studio**. Program Unity3d można pobrać z oficjalnej strony (<http://unity3d.com/unity/download>), aplikacja jest darmowa, ale istnieje również jej płatna wersja, posiadająca więcej funkcjonalności. Na oficjalnej stronie internetowej dostępna jest również pełna dokumentacja środowiska oraz udostępnianego API (<http://unity3d.com/learn/documentation>).

Budowanie projektu

Po instalacji obu programów można otworzyć projekt w aplikacji Unity3d. W tym celu należy wybrać z menu **File/Open Project** i wybrać folder **game\sources\empiresstrategy**. Środowisko pozwoli otworzyć akurat ten folder, ponieważ zawiera on pliki odpowiednie dla projektu Unity3d. Tworzona aplikacja składa się z obiektów charakterystycznych dla środowiska (prefaby, skrypty, grafika, modele 3d, itd), a także bibliotek **.NET** stworzonych w **Visual Studio**.

Aby edytować wykorzystywane biblioteki należy otworzyć za pomocą Visual Studio plik **game\sources\empiresstrategy\source\empiresstrategy\source.sln**. Przed kompilacją należy upewnić się, iż projekt odwołuje się właściwie do plików. W ustawieniach projektu w zakładce **Reference Paths** powinno być odwołanie do ścieżki **game\sources\empiresstrategy\Assets\libraries**. Do tej samej ścieżki powinna wskazywać właściwość **Output Path** z zakładki **Build**. Po edycji i skompilowaniu biblioteki należy umieścić w odpowiednim katalogu **game\sources\empiresstrategy\Assets\libraries**. Następnie można zbudować pełną aplikację przy pomocy środowiska Unity3d. Środowisko pozwala również na testowanie gry i edycje kodu w trakcie jej działania. Unity3d pozwala na budowanie projektu w wersjach nie tylko przystosowanych do systemu Windows. Możliwa jest budowa, aplikacji webowej czy działającej na systemie Android. Stworzone biblioteki **.NET** mogą jednak sprawić, iż te wersje nie będą działać poprawnie.

Struktura plików oraz kodu źródłowego została opisana w rozdziale 4.2. Struktura programu.

Rozwój mechanizmów sztucznej inteligencji

Rozwój aplikacji pod względem algorytmicznym mógłby polegać na wprowadzeniu modyfikacji algorytmu A*, tak by był on wydajniejszy lub bardziej realistyczny. Innym pomysłem na modyfikację mogłoby być wprowadzenie własnego systemu do zarządzania stanami obiektów oraz akcjami wykonywanymi w danych stanach. Mogłoby to uatrakcyjnić rozgrywkę i uwiarygodnić wykorzystywaną sztuczną inteligencję. Możliwe jest również wprowadzenie innych mechanizmów, jak drzewa gry, należałoby jednak najpierw to przemyśleć pod kątem wydajności oraz powodu, dla którego miałyby one działać. Inne pomysły na rozwój aplikacji zostały przedstawione w rozdziale 6.3. Możliwość rozwoju.





















Struktura programu

Wykorzystywane środowisko Unity3d podczas tworzenia aplikacji wymusza własną strukturę katalogów. Struktura ta została rozwinięta o dodatkowe foldery, co przedstawia rysunek (Rys. 4.7). Widać tu foldery i pliki stworzone przez środowisko Unity3d:

- Assets (modele, dźwięki, tekstury wykorzystywane w projekcie)
- Library (pliki potrzebne do kompilacji)
- Obj (pliki potrzebne do kompilacji)
- ProjectSettings (ustawienia projektu)
- Temp (pliki tymczasowe)
- Pliki projektu Visual Studio

Poza tym dodane zostały foldery i pliki:

- Logs (logi wyjątków)
- Pomiar (logi poprawności algorytmów oraz pomiary ich czasu trwania)
- Source (pliki źródłowe bibliotek zawierających kod źródłowy gry)
- Pliki map (zapis planszy gry w formacie tekstowym).

	Assets	2013-12-11 12:46	File folder	
	Library	2014-01-22 10:36	File folder	
	Logs	2014-01-22 10:28	File folder	
	obj	2013-09-06 22:46	File folder	
	pomiar	2014-01-11 15:25	File folder	
	ProjectSettings	2014-01-07 21:27	File folder	
	source	2013-09-06 22:50	File folder	
	Temp	2014-01-22 11:03	File folder	
	Assembly-CSharp.csproj	2013-09-06 23:06	Visual C# Project f...	4 KB
	Assembly-CSharp.csproj.vspsc	2013-09-06 22:49	Visual Studio Sour...	1 KB
	Assembly-CSharp.pdb	2013-09-15 17:40	PIDB File	7 KB
	Assembly-CSharp-vs.csproj	2013-12-14 22:27	Visual C# Project f...	5 KB
	empiresstrategy.sln	2013-09-06 22:49	Microsoft Visual S...	2 KB
	empiresstrategy.suo	2013-12-14 21:44	Visual Studio Solu...	86 KB
	empiresstrategy.userprefs	2013-09-15 17:40	USERPREFS File	1 KB
	empiresstrategy.vsscc	2013-09-06 22:49	Visual Studio Sour...	1 KB
	empiresstrategy-csharp.sln	2013-09-06 22:46	Microsoft Visual S...	2 KB
	empiresstrategy-csharp.suo	2014-01-11 17:39	Visual Studio Solu...	31 KB
	map.txt	2014-01-11 22:43	Text Document	314 KB
	map1.txt	2014-01-12 20:58	Text Document	314 KB

Rys. 4.7 Struktura plików.

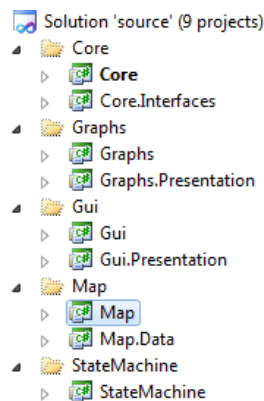
W środowisku Unity3d do każdego obiektu można dodawać skrypty obsługujące jego działanie. Możliwe jest obsłużenie między innymi zdarzenia pojawienia się obiektu lub zachowania obiektu podczas rysowania każdej klatki animacji. Skrypty podpięte do obiektów wykorzystują stworzone do tego celu osobne biblioteki stworzone w języku C#. Biblioteki te zawierają większość kodu źródłowego gry, a ich strukturę prezentuje rysunek (Rys. 4.9). Najważniejsze elementy gry takie jak obsługa graficznego interfejsu użytkownika, kamery czy zarządzania głównymi danymi gry obsługiwane są poprzez obiekt **CoreObject** widoczny na rysunku (Rys. 4.8). Prócz tego obiekty występujące w grze wielokrotnie tworzone są na podstawie bytu nazywanego **prefab** [14]. Byt ten może być zdefiniowany przed rozgrywką, a następnie wielokrotnie dodawany lub usuwany do sceny, idea jego istnienia polega właśnie na definiowaniu skomplikowanych konfiguracji bytu. Najważniejsze z tych bytów również przedstawione są na rysunku (Rys. 4.8).



Rys. 4.8 Najważniejsze obiekty gry w środowisku Unity3d.

Do omawianych obiektów przypisane są skrypty dziedziczące po klasie **MonoBehaviour** [15], co pozwala na obsługę zdarzeń dostarczanych przez środowisko. Wykorzystywane zdarzenia to głównie:

- Awake – uruchamiane po załadowaniu skryptu
- Update – uruchamiane z każdym przerysowaniem animacji
- OnGUI – uruchamiane z każdym przerysowaniem graficznego interfejsu
- Start – uruchamiane w pierwszej ramce animacji
- OnMouseOver – uruchamiane, gdy kursor znajdzie się w obrębie obiektu przy uwzględnieniu położenia kamery.



Rys. 4.9 Struktura projektu.

Program oparty jest o serwisy bazujące na idei singletonów. Występujące w programie serwisy to:

- SceneLoaderService (przeładowywanie scen gry)
- AgentService (zarządzanie agentami graczy)
- MatchDataService (przechowywanie danych rozgrywki)
- PlayerDataService (przechowywanie danych gracza)
- CameraService (sterowanie kamerą)
- PrefabService (zarządzanie predefiniowanymi obiektami graficznymi)
- TextureService (zarządzanie teksturami gry)
- PathFinderService (wyszukiwanie ścieżki)
- ServiceA (implementacja algorytmu A*)
- ServiceDijkstra (implementacja algorytmu Dijkstry)
- ServiceFord (implementacja algorytmu Forda-Bellmana)
- GuiService (wyświetlanie przycisków graficznego interfejsu)

W programie występuje kilka głównych klas będących ważnymi elementami kodu, są to:

- Player (przechowuje dane konkretnego gracza)
- Agent (przechowuje informacje o agencie oraz odpowiada za jego sztuczną inteligencję)
- MyGameObject (jest obiektem bazowym dla wszystkich pozostałych obiektów występujących na planszy)