POLITECHNIKA ŚLĄSKA 2015

Podstawy zespołowego tworzenia gier komputerowych

Architektura

11.04.2015

Skład sekcji:
Duda Jacek
Gromadzki Tadeusz
Kurda Wojciech
Małek Mateusz
Mirota Jakub
Pomocnik Rafał
Purta Piotr
Rusin Michał
Uramowski Piotr

Prowadzący: dr inż. Michał Kawulok

1. Historia zmian dokumentu

Wersja	Data	Autor	Opis
1.0	28.03.2015	Mateusz Małek	Stworzenie dokumentu
1.1	11.04.2015	Piotr Purta	Scalenie dokumentu, poprawki i formatowanie

2. Spis treści

3. Wstęp

Dzięki temu, że projekt tworzony przez nasz zespół korzysta ze środowiska Unity, wiele istotnych elementów jest już dostarczanych przez silnik, a nasza rola ogranicza się wyłącznie do implementacji samej rozgrywki. Unity w swoich założeniach wspiera komponentową budowę aplikacji, dlatego też naszym zadaniem jest pisanie modułów, które komunikując się wzajemnie będą wpływały na grę. Budowa komponentowa oraz sposób tworzenia poszczególnych elementów (część z nich tworzy się automatycznie, a metodą „konstruktorową" w ich przypadku jest „Awake") w pewien sposób narzuca na nas konieczność zastosowania architektury opierającej się na wzorcu singleton.

4. Klasy

Sporą częścią tworzonych przez nas klas będą komponenty, które bezpośrednio dziedziczą po klasie MonoBehaviour udostępnionej przez Unity. Każdy komponent stworzony w ten sposób może być następnie „przypięty" do konkretnego obiektu w grze („Game Object") oraz komunikować się z innymi obiektami lub komponentami. Oprócz komponentów istnieć również będą klasy nie dziedziczące po MonoBehaviour – te będą jednak służyły przede wszystkim do zarządzania (menedżery) lub jako klasy wspomagające serializację/deserializację lub inicjalizowanie.

5. Komunikacja

Komunikacja pomiędzy wszystkimi klasami w grze odbywa się za pośrednictwem singletonów. Istnieje jeden główny singleton, który będzie dostępny ze wszystkich scen i będzie zawierał te najbardziej podstawowe elementy m.in. menedżer zasobów czy menedżer kamer. Dodatkowo dla każdej sceny istnieje oddzielny singleton, który zawiera elementy wyłącznie dla tej sceny. Przykładowo: dla sceny gry będzie zawierał menedżera przeciwników lub menedżera poziomów. Nie będzie możliwości odwołania się do singletona przypisanego do sceny, która nie jest aktualnie w użyciu. Singletony inicjalizują się na samym początku, zaraz po wczytaniu sceny, a mechanizm ten realizowany jest w obiektach „Init".

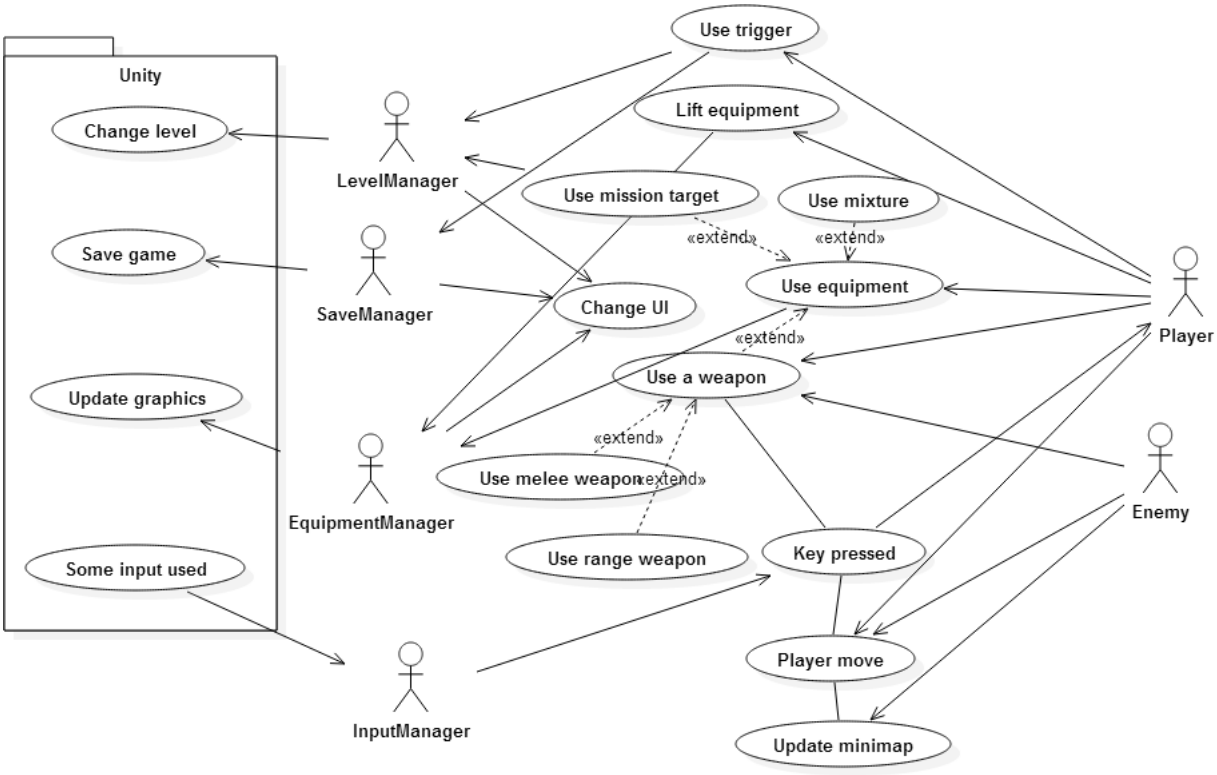
Dane pomiędzy scenami będą przesyłane również za pośrednictwem głównego singletona. Nie istnieje inna metoda przesyłu danych. Aby ułatwić testowanie gry, wszystkie elementy inicjalizujące scenę będzie można sobie dowolnie skonfigurować ustawiając odpowiednie wartości w obiekcie „Init". Po ich ustawieniu i odpaleniu gry na przykład bezpośrednio ze sceny „Game", ta będzie zainicjalizowana według naszej konfiguracji.

6. Menadżery

Typową konstrukcją dla naszej aplikacji będzie „menedżer-obiekty zarządzane". Na przykład będzie istniał menedżer przeciwników, który w trakcie inicjalizacji sceny stworzy instancje wszystkich przeciwników. Referencje do menedżerów znajdować się będą w singletonach. Część menadżerów będzie komponentami ze względu na to, że często koniecznością jest przypisanie pewnych elementów konfiguracyjnych danemu menedżerowi, a to można uczynić wyłącznie poprzez stworzenie menedżera jako komponentu.

W trakcie tworzenia singletona dla danej sceny w jej metodzie inicjalizacyjnej tworzone są wszystkie niezbędne menadżery. Te z kolei w swoich metodach inicjalizacyjnych tworzą elementy zarządzane. W związku z tym scena sama w sobie będzie posiadała wyłącznie elementy statyczne. Te dynamiczne będą tworzone przy jej inicjalizacji. Przy wyjściu ze sceny każdy ze stworzonych menedżerów jest usuwany, a co za tym idzie, elementy przez niego zarządzane również.

7. Diagram przypadków użycia



8. Diagram klas