Wasylkowski Łukasz, Zimnoch Maciej
Wrocław 2012r.

Analiza obiektowa projektu DynaBomber

Architektura systemu

Całość gry jest zarządzana przez klasę DynGame, która w zależności od obecnego stanu gry wywołuje metody aktualizacji poszczególnych klas. Główną klasą gry jest DynGame, która odpowiada za pojedynczą grę (która trwa do wyłączenia, lub do momentu kiedy na planszy pozostał nie więcej niż jeden żywy gracz). Największa możliwa liczba akcji jest delegowana na poszczególne, szczegółowe klasy, podczas gdy klasy większe zajmują się tylko tym, do czego potrzebna jest całościowa wiedza (np. dostęp do wszystkich obiektów, żeby obsłużyć kolizje).

Większość klas posiada metodę Initialize, której celem jest przekazanie załadowanych wcześniej zasobów. Klasy implementują też dwa interfejsy - DynICollidable, obsługujący zderzanie się obiektów, oraz DynIDrawable, który mają wszystkie klasy rysujące coś na ekranie.

Spis klas

1. **DynaBomber**
    a. Główna klasa programu
    b. Dziedziczy po klasie Microsoft.Xna.Framework.Game
        i. Metody LoadContent, Update, Draw i Initialize
        ii. Wywoływane przez framework XNA w celu aktualizacji gry
    c. Zarządza grą, aktualizując obecny stan programu i rysując oraz aktualizując klasy odpowiedzialne za odpowiednie stany (menu, gra itp)
    d. Opis wybranych metod:
        i. *LoadContent*: ładuje z dysku niezbędne zasoby
        ii. *Update*: w zależności od obecnego stanu gry aktualizuje wybrane klasy, aktualizuje też stan gry, jest odpowiedzialna za przechwytywanie klawiszy
        iii. *Draw:* Rysuje odpowiednie klasy (które mają własne metody Draw)
2. **DynBoard**
    a. Klasa przechowująca informację o planszy (w tym listę kolizji, eksplozji, bonusy)
    b. Implementuje interfejs DynIDrawable
        i. Metodę Draw()
    c. Odpowiada za większość akcji dziejących się wyłącznie w ramach planszy
    d. Wybrane metody:
        i. *Reset:* Resetuje planszę do stanu na początku gry
        ii. *ExplodeBomb*: Obsługuje wybuch przekazanej bomby, w razie potrzeby usuwając ściany, tworząc eksplozję oraz przechowując ją na liście
        iii. *SetTile*: Zmienia wybrane pole planszy, obsługuje pojawianie się bonusów
        iv. *Update*: Aktualizuje wszystkie elementy na listach (eksplozje, bonusy), usuwając nieaktywne

3. **DynGame**
    a. Klasa obsługuje pojedynczą grę
    b. Wybrane metody:
        i. *StartGame():* Przygotowuje klasę do rozpoczęcia nowej gry (resetuje się)
        ii. *Update:* Obsługuje kolizje (iteruje elementy implementujące interfejs DynICollidable, używając jego metod do sprawdzenia czy jest kolizja (*bool DynICollidable.Collides(DynICollidable object)*), oraz inicjując wykonanie akcji z nią związanej (*void DynICollidable.OnCollision(DynUtils.ObjectType o)*). Klasa obsługuje też wejście, stawianie bomb (które potrzebuje dostępu do klasy zarówno gracza jak i planszy) i ruch potworów
        iii. *GetResult:* Służy sprawdzeniu czy gra nadal trwa, wykorzystywana przez klasę DynaBomber do aktualizacji obecnego stanu gry oraz wyników punktowych
4. **DynCAnimation**
    a. Klasa obsługująca animacje
    b. Animuje obiekty wykorzystując przekazane grafiki (w postaci *sprite'ów*), zmieniając klatkę po określonym czasie
    c. Implementuje interfejsy DynIDrawable oraz DynICollidable (gdyż czasem obszar kolizji odpowiada dokładnie obszarowi animacji)
5. **DynPauseMenu, DynSettings, DynMenu, DynPostGame**
    a. Klasy wyświetlające i obsługujące różne menu
    b. Implementują interfejs DynIDrawable
6. **DynBomb**
    a. Klasa obsługująca pojedynczą bombę
    b. Implementuje DynIDrawable oraz DynICollidable
    c. Usuwana przez klasę DynGame gdy właściwość *active* będzie *false*
7. **DynExplosion**
    a. Obsługuje animację i kolizję z eksplozją
    b. Implementuje DynIDrawable
    c. Zawiera wiele animacji, przy wywołaniu jej metody Update lub Draw wywoływane są odpowiednie metody każdej instancji animacji
8. **DynPlayer**
    a. Klasa gracza, zawiera podstawowe informacje o postaci (ilość bomb, ich siłę, czy gracz żyje, pozycja itp)
    b. Klasa obsługuje ruch gracza, ale nie np. stawianie bomb (musiałaby mieć dostęp do kalsy planszy, dlatego robi to DynGame)
    c. Implementuje interfejsy DynICollidable oraz DynIDrawable
    d. Wybrane metody:
        i. *Move*: Obsługuje ruch gracza, sprawdzając ewentualne kolizje. Lista kolizji i bomb jest przekazywana do klasy w celu obsłużenia ruchu, gdyż w innym przypadku mocno by to zaciemniło pozostałe klasy (a ruch wpływa w zasadzie wyłącznie na gracza)
        ii. *die*: Metoda obsługująca śmierć gracza. Ustawia *alive* na *false*, w celu poinformowania klasy DynGame o śmierci gracza

9. **DynGameOptions**
    a. Klasa zawierająca opcje gry
    b. Opcje są zamknięte w klasie, aby ich dodawanie nie wiązało się ze zmianą liczby parametrów w metodach, a więc aby ułatwić późniejsze zmiany
10. **DynMonster**
    a. Klasa obsługująca ruch i kolizje pojedynczego potwora. Ich lista jest przechowywana w klasie DynBoard (żeby koordynować np. ruch potworów i kolizję ze ścianami)
    b. Implementuje DynIDrawable, DynICollidable
11. **DynBonus**
    a. Klasa przechowująca informacje o bonusie
    b. Implementuje DynIDrawable, DynICollidable
12. **PlayerKeys**
    a. Klasa umożliwiająca obsługę klawiszy, wczytuje ustawienia użytkowników
13. **FileManager**
    a. Klasa udostępniająca metody ładowania i zapisywania opcji do pliku
14. **DynUtils**
    a. Klasa pomocnicza, zawierająca metody, właściwości i obiekty (np. Random) wspólne dla różnych klas

Wybrane wykorzystane wzorce projektowe

Fasada

Klasa *FileManager* udostępnia interfejs do ładowania i zapisywania plików INI oraz plików konfiguracji konsoli XBOX jako metody osobnej klasy. Złożony jest proces uzyskiwania dostępu do konfiguracji (import bibliotek DLL itp), natomiast klasa *FileManager* udostępnia najbardziej potrzebne metody, tym samym implementując strukturalny wzorzec projektowy *fasada*.

Stan

Klasa *DynaBomber* na podstawie obecnego stanu aplikacji (*DynUtils.GameState*) decyduje które elementy aplikacji uaktualniać oraz rysować, przełączając na podstawie informacji uzyskiwanych od klas oraz akcji użytkownika obecny stan. Jest to czynnościowy wzorzec projektowy *stan*, który zmienia zachowanie obiektu (klasy *DynaBomber*) w zależności od stanu aplikacji.

Zarządca

Klasa *DynaGame* jest zarządcą, który koordynuje pozostałe klasy. Aby nie przekazywać stanu o całej grze każdej klasie z osobna, oraz aby nie powtarzać działań, poszczególne klasy z zasady zajmują się tylko tym, do czego potrzebują informacji wyłącznie o swojej klasie (oraz w naszym przypadku, zwykle, listy kolizji), oraz wpływają wyłącznie na siebie. Zatem wszystkie akcje, w których jeden obiekt wpływa na drugi, wykonywanie są używając wzorca projektowego *zarządcy*, który łącząc w sobie wiele klas, wywołuje odpowiednie metody pozwalając im na interakcje ze sobą.

Inne zastosowania i rozszerzenia

Najbardziej **uniwersalną** klasą w projekcie jest DynCAnimation, która w zasadzie może być wykorzystana w każdym kontekście, w którym poruszają się obiekty animowane, dla których grafikę mamy w pojedynczym pliku *sprite*. Kolejną klasą może być FileManager oraz PlayerKeys, które implementują mniej lub bardziej uniwersalne interfejsy czytania i zapisywania ustawień do pliku oraz obsługi klawiszy. Architektura pozostałych klas raczej nie pozwala na ich *reusing*, jako że są ze sobą powiązane i działają tylko w ramach systemu dla którego powstały.

Możliwe zmiany w projekcie stosunkowo łatwo wprowadzić, choć ciężko wymyślić nietrywialną zmianę do gry tego rodzaju. Przykładowe modyfikacje:

1. Dodanie większej ilości graczy

Wiąże się ze zmianą zmiennej odpowiadającej za maksymalną ilość graczy, zapewnieniem odpowiedniej ilości plików graficznych oraz dodania do listy *PlayersStartingPositions* klasy *DynGame* kolejnych pól startowych. Reszta akcji związana z listą graczy jest wykonywania iteracyjnie przez pętle.

2. Dodanie nowych bonusów

Wystarczy dodać nowy typ bonusu na liście *ObjectType* w klasie *DynUtils*, oraz odpowiednią odpowiedź na kolizję w klasie gracza

3. Możliwość przesuwania bomb

Nietrywialna zmiana, wymagałaby obsłużenia dodatkowego przycisku w klasie *DynGame*, odpowiadającego za popchnięcie bomby, po czym nowa flaga w klasie DynBomb informowałaby o ruchu. Aktualizacja pozycji następowałaby jako dodatkowa funkcja metody Update() (aż do napotkania kolizji). *DynGame* z kolei sprawdzałoby kolizję poruszającej się bomby z innymi elementami gry, w razie której wystąpienia zatrzymywałaby ruch przedwcześnie (przed uderzeniem o ścianę).

4. Możliwość gry jednoosobowej

Wymagałaby utworzenia nowej klasy, która jednak korzystałaby w całości i bez zmian z obecnych klas. Zawierałaby informacje tylko jednego gracza, możliwość zakończenia gry następowałaby w momencie zabicia wszystkich potworów (zmiana metody *GetResult)*, po wejściu (także po zabiciu potworów) w nowopowstałe miejsce na planszy, ukrywające się w trakcie gry pod jednym z możliwych do zniszczenia pól.

Diagram klas

Poniżej lekko uproszczony diagram klas (nie wszystkie klasy implementujące interfejsy zostały zaznaczone, nie zostały podane wszystkie metody i właściwości klas oraz nie wszystkie powiązania, gdyż nie byłoby to w ogóle czytelne)