Koszęcin, dnia 03.04.2020

Laboratorium ZTPGK

Fizyka w grach komputerowych: Specyfikacja

Karol Kozuch
AEI Informatyka, IGT SII
Semestr: 1
E-mail: karokoz247@student.polsl.pl

1. Opis gry

Celem gry stworzonej na potrzeby laboratorium jest zniszczenie minimalnej wymaganej ilości celów poprzez oddanie do nich celnych strzałów z działa, które jest kontrolowane przez gracza. Warunkiem zwycięstwa jest zniszczenie przynajmniej 80% celów. Za cel uznawane są animowane obiekty stojące, wiszące łańcuchy i kurtyny. Gra rozpoczyna się tuż po jej włączeniu, a ilość dostępnej amunicji dla gracza nie jest ograniczona. Segmenty celu zniszczonego zmieniają kolor na różowy dla odróżnienia od pozostałych celów.

2. Specyfikacja wewnętrzna

2.1. Wykorzystane wzorce projektowe

W trakcie implementacji gry wzorowano się na wzorcu *Observer* (ang. obserwator). Celem użycia wzorca było zmniejszenie ilości powiązań między poszczególnymi klasami – jeżeli dana klasa chce, by druga klasa o czymś ją poinformowała, to może jej przekazać referencję na jedną ze swoich metod. W ten sposób druga klasa będzie mogła przekazać informację bez potrzeby posiadania referencji do pierwszej klasy. Co więcej, mechanizm ten pozwala na podłączenie wielu metod z wielu klas, które zostaną wywołane przez klasę obserwującą, gdy warunki wywołania zostaną spełnione. Klasami, które korzystają z tego wzorca, są między innymi klasy *Target* (jako klasa obserwująca) oraz *TargetComponent* (jako klasa obserwowana).

2.2. Klasy

W projekcie można znaleźć klasy wymienione poniżej. W ważniejszych z nich zostały także opisane metody:

- *BulletDetector* – zadaniem klasy jest raportowanie zmian w obecności obiektów oznaczonych jako pociski w określonej komponentem typu *Collider* przestrzeni obiektu, do którego należą tenże komponent i skrypt zawierające tę klasę. W klasie znajdują się następujące metody:
  - *OnTriggerEnter*() - Metoda wywoływana automatycznie przez środowisko Unity w momencie gdy obiekt należący do określonej w edytorze warstwy rozpocznie kolidować z komponentem typu *Collider* ustawionym jako wyzwalacz. Wywołana tylko w momencie wejścia. Jeżeli obiekt faktycznie należy do określonej w edytorze warstwy – wywoływane są wszystkie zarejestrowane metody obsługujące to zdarzenie.
  - *OnTriggerExit*() - Odpowiednik *OnTriggerEnter*() dla zdarzenia opuszczenia obszaru *Collidera* przez obiekt wcześniej w nim będący.
- *CannonLoader* – Zajmuje się procesem przeładowania działa po wystrzale. Posiada następujące metody:
  - *Start*() - zajmuje się sprawdzeniem, czy wymagane komponenty zostały przypisane w edytorze oraz rejestruje metodę obsługującą zdarzenie wciśnięcia przycisku przeładowania.
  - *Reload*() - tworzy nowy obiekt pocisku w komorze działa, jeżeli żadnego pocisku obecnie tam nie ma
  - *BulletInChamber*() - Metoda, poprzez którą klasy zewnętrzne mogą poinformować *CannonLoader* o tym, że działo jest już załadowane i nie wymaga przeładowania.
  - *BulletLeftChamber*() - Jak *BulletInChamber*() z tą różnicą, że użyta w przypadku braku pocisku w dziale.
- *CannonTrigger* – Odpowiada za wystrzelenie pocisku i zapewnienie, że póki to nie nastąpi, zostanie on w komorze.
  - *Start*() - Rejestruje metodę *Shoot*() jako obsługę zdarzenia wciśnięcia przycisku wystrzału. Rejestracja następuje w klasie *InputReader*.
  - *ArmProjectile*() - Konfiguruje komponent sprężyny tak, by utrzymywał nowo stworzony pocisk w komorze działa.
  - *Shoot*() - Odpowiada za wykonanie wystrzału – zwalnia sprężynę i aplikuje siłę.
- *InputHandler* – Odpowiada za sprawdzanie i odbieranie danych wejściowych od gracza.

    - *Update*() - Wywoływana co klatkę symulacji, odczytuje wartości osi poziomej i pionowej kontrolek oraz wywołuje metody sprawdzające stan innych kontrolek.
    - *RegisterReload*() - Rejestruje podaną metodę jako obsługującą zdarzenie przeładowania działa.
    - *RegisterShoot*() - Rejestruje podaną metodę jako obsługującą zdarzenie wystrzału z działa.
    - *RegisterExit*() - Rejestruje podaną metodę jako obsługującą zdarzenie wyjścia z aplikajci.
    - *ChkKeys*() - Sprawdza, czy zostały wciśnięte kluczowe dla gry klawisze na klawiaturze.
- *PlayerCannonMover* – Nadzoruje ruch działa. Zawiera metody:
    - *Update*() - wywoływana co klatkę symulacji, wykonuje obrót działa zgodnie ze stanem wartości osi poziomej i pionowej, skalując ruch odpowiednio do czasu ostatniej klatki.
    - *GetAxesValues*() - Zwraca wartości osi poziomej i pionowej po konwersji na odpowiednie osie obrotu działa.
- *ChainHeadBreaker* – Przełącza oddziaływanie grawitacji na głowę łańcuchów zawieszonych w powietrzu.
    - *Start*() - inicjalizuje klasę, pobierając referencję na komponent typu *Collider* obiektu i sprawdza, czy komponent ten istnieje.
    - *OnTriggerEnter*() - wywołana w momencie kolizji z innym obiektem. Jeżeli obiekt kolidujący jest przypisany do warstwy określonej w edytorze, włącza oddziaływanie grawitacji na głowę łańcucha.
- *Target* – Klasa definiuje cel do zniszczenia przez gracza.
    - *Start*() - Odnajduje wszystkie składowe komponenty i rejestruje w nich metodę *TargetDestroyed()* jako obsługującą zdarzenie trafienia.
    - *RegisterObserveTargetDestroy()* – Umożliwia zarejestrowanie metod służących do poinformowania klas zewnętrznych o zniszczeniu obiektu.
    - *TargetDestroyed()* – Wywołuje zarejestrowane metody obsługi zdarzenia zniszczenia obiektu i propaguje sygnał o zniszczeniu obiektu na wszystkie jego składowe części.
- *TargetComponent* – Definiuje zachowanie najmniejszego elementu celu.
    - *RegisterTakingHitHandler*() - pozwala na zarejestrowanie przez inne klasy metod obsługi zdarzenia trafienia elementu.
    - *OnCollisionEnter*() - wywołana w momencie kolizji z innym obiektem. Jeżeli obiekt należy do określonej w edytorze warstwy – wywołuje zarejestrowane metody obsługi zdarzenia.
    - *ChangeColor*() – Zmienia kolor komponentu na określony w edytorze.
- *TargetCounter* – Zlicza ilość zniszczonych obiektów i sprawdza, ile jeszcze ich zostało do spełnienia warunku zwycięstwa.
    - *Start*() - Oblicza minimalną ilość obiektów do zniszczenia, odnajduje wszystkie cele i rejestruje w nich metodę *TargetDestroyed*() jako obsługującą zdarzenie zniszczenia celów.
    - *TargetDestroyed*() - Odnotowuje zniszczenie kolejnego celu i, jeżeli zniszczono odpowiednią ilość celów, wywołuje metody obsługi zdarzenia spełnienia warunku zwycięstwa.
- *TargetRotator* – Odpowiada za animację rotacji celów stojących.
    - *Start()* - Odnajduje komponent *Rigidbody* obiektu i upewnia się, że wektor siły jest znormalizowany.
    - *Update*() - metoda wywoływana co klatkę symulacji. Aplikuje siłę obrotową, jeżeli prędkość obrotowa jest zbyt niska (parametr ustawiany w edytorze).
    - *CalcPosition*() - oblicza pozycję przyłożenia siły obrotowej.
- *Comparator* – klasa statyczna, dostarcza metody porównujące wartości w niestandardowy sposób.

    - *CompareLayers*() - porównuje wartości warstw ustalonych w edytorze dla skryptu (typ *LayerMask*) oraz przypisanej do obiektu. Zwraca *true* jeżeli choć jedna warstwa się zgadza.
- *GameController* – Nadzoruje działanie całej gry. Zawiera metody:
    - *Start*() - rejestruje metodę *ExitGame*() jako obsługę naciśnięcia klawisza wyjścia z programu w klasie *InputReader*.
    - *GameObjectiveMet*() - Obsługuje zdarzenie spełnienia warunków zwycięstwa. Wywołuje wszystkie metody obsługujące dotyczące GUI.
    - *ExitGame*() - Wywołuje metodę wyjścia z aplikacji.
- *GuiController* – Kontroluje elementy GUI.
    - *EnableVictoryText*() - Obsługuje zdarzenie zwycięstwa gracza. Powoduje wyświetlenie tekstu informującego o zwycięstwie na ekranie.

2.3. Interfejsy

W projekcie wyróżnić można dwa interfejsy:

- *ITarget* – Dostarcza uniwersalnej metody rejestracji obsługi zdarzenia zniszczenia celu dla wzorca *Observer*.
- ITargetComponent - Dostarcza uniwersalne metody rejestracji obsługi zdarzenia otrzymania trafienia przez segment celu dla wzorca *Observer* oraz metody zmiany koloru przez segment.

3. Specyfikacja zewnętrzna

3.1. Sterowanie

Sterowanie w grze opiera się w pełni o klawiaturę:

- A, D – Przyciski te powodują odpowiednio obrót działa w lewo i w prawo,
- W, S – Przyciski te powodują odpowiednio obrót działa w górę i w dół,
- R – Przeładowanie działa, o ile nie ma już załadowanego pocisku
- Spacja – wystrzał
- Escape – wyjście z gry (w dowolnym momencie).

3.2. Cel gry

Celem gry jest zniszczenie przez gracza przynajmniej 80% celów znajdujących się przed działem. Cel może zostać zniszczony poprzez bezpośrednie trafienie go pociskiem. Cel nie musi zostać fizycznie zniszczony (rozpaść się na części pierwsze) – wystarczy, by został trafiony.

4. Testy

Aplikacja była testowana na bieżąco w trakcie tworzenia. Tworzenie poszczególnych elementów podzielono na fazy, do każdej następnej fazy przechodzono w momencie stwierdzenia spełnienia warunków działania poprzedniej. Dany element został dołączany do aplikacji w momencie spełnienia wszystkich warunków działania go dotyczących.

5. Podsumowanie

Projekt udało się wykonać zgodnie z zamierzeniami.