Propozycja projektu, Mateusz Oleszek, AiR, A7-L14

Jako projekt zamierzam użyć SFMLa do stworzenia gry typu twin-stick shooter/shoot 'em up, w której gracz porusza się po mapie WSADem, a za pomocą myszki celuje i strzela do różnych rodzajów przeciwników, zbierając przy tym ulepszenia i modyfikacje broni i starając się przetrwać jak najdłużej.

Aplikacja będzie wykorzystywała system stanów (np. stan menu, gry, ekranu końcowego, ustawień) od których będzie zależeć jak w danej chwili będzie się aplikacja zachowywała.

Stany (states)

Stan menu głównego, gry, ekranu końcowego, menu ustawień.

Dziedziczą one od bazowej klasy wirtualnej „State" , która wymusza implementację funkcji „update" i „render", w których będzie się znajdowała logika odpowiadająca za to co w danej chwili będzie się działo na ekranie.

Klasa bazowa posiada także funkcję odpowiadającą za odświeżanie pozycji myszki (w relacji do okna oraz świata gry) jako, że każdy stan będzie potrzebował tej funkcjonalności.

Wszystkie zasoby jak tekstury czy dźwięki będą przechowywane w samej grze, więc stany nie będą musiały ładować ich na nowo kiedy będą tworzone, tylko będą przechowywały do nich wskaźniki.

Stos stanów

Stany składowane są na stosie, i tylko ten znajdujący się na wierzchu będzie wykonywał swoje funkcje „update" i „render". Każda z instancji stanu posiada wskaźnik do stosu, więc może umieszczać na nim nowe elementy. Kiedy dany stan zakończy swoje działanie zostanie usunięty ze stosu kiedy będzie znajdował się na jego szczycie. Załączyłem grafikę przedstawiającą ideę jego działania.

Main gameplay loop

- Sprawdzić czas, który minął od ostatniej klatki/cyklu, tak zwane dt(delta time), które będzie przekazywane do funkcji „update" stanów w celu uniezależnienia szybkości ruchu obiektów od prędkości gry oraz zasilenia wszelkich liczników czasów.
- Zaktualizować wydarzenia SFMLa, głównie to, czy gracz zamknął okno gry.
- Zaktualizowanie obecnego stanu gry (tutaj się cała rozgrywka odbywa)
- Usunięcie ze stosu tych stanów, które zostały zakończone.
- Renderowanie zawartości aktualnego stanu.

Entity

Zarówno gracz jak i wrogowie dziedziczą z wirtualnej klasy entity. Przechowuje ona obiekt „sf::Sprite”, który reprezentuje ciało tej postaci. Posiada ona funkcje pozwalające sprawdzić pozycję, dystans lub kolizję tego sprite’a.

Moduły

W celu rozdzielenia zadań związanych z zachowaniem postaci, klasa Entity posiada wskaźniki, do których można przypisać moduły – klasy odpowiadające za poszczególne aspekty zachowania i interakcji postaci z otoczeniem.

Są to moduły:

- Ruchu
- Animacji
- Hitboxu
- Sztucznej Inteligencji

Wszystkie moduły przy konstrukcji wymagają pewnych parametrów określających ich działanie, co pozwala używać ich na graczu oraz na przeciwnikach, zachowując to samo ogólne działanie z odpowiednimi różnicami w zachowaniu.

Moduł ruchu

Przyjmuje znormalizowany wektor reprezentujący pożądany kierunek ruchu i na jego podstawie zajmuje się ruchem postaci. Początkowym przyspieszaniem do wartości maksymalnej, stopniowym zwalnianiem po ustaniu sygnałów do dalszego ruchu, co symuluje działanie pędu. Odpowiada także jak postać będzie się poruszała w przypadku specjalnych akcji jak unik.

Moduł Animacji

Zarządca dodawaniem i odgrywaniem animacji. Potrzebuje arkusza animacji, na którym będą wyrysowane poszczególne klatki. Następnie podając pozycję startową, ilość klatek, oraz ich wymiary, można postaci dodać animację. Przy odgrywaniu ich, może uściślić, że dana animacja ma priorytet nad innymi, np. animacja ataku przerwie tą chodzenia. Załączyłem grafikę która jest arkuszem animacji dla głównej postaci.

Moduł Hitboxu

Odpowiada on za granice „ciała” postaci i to jak będzie wchodziła w interakcje z innymi. „nadpisuje” więc tym pewne funkcje obecne w samej klasie „entity” jak obliczanie odległości od innych ciał, albo pozyskanie współrzędnych pozycji postaci (zarówno domyślnego górnego lewego rogu sprite’a, jak i bardziej pomocnego centrum).

Moduł sztucznej inteligencji

Oczywiście dodany tylko do przeciwników, zajmuje się determinowaniem następnego ruchu mając informacje o aktualnej pozycji gracza, rodzaju wroga, którego ruchami steruje oraz jego aktualnym stanem. Różne rodzaje przeciwników mają odmienne strategie ruchu. Np. podążanie za graczem lub trzymanie się od niego w pewnej odległości. A ich stan (zależny od aktualnego zdrowia, wpływający także na ich wygląd) także ma wpływ na ich poruszanie się, np. uciekanie od gracza w stanie bliskim zniszczenia.

Strzelanie

Każda postać w grze przechowuje kontener obiektów klasy pocisk, które wystrzeliła. Każdy z tych obiektów posiada charakterystyki takie jak szybkość, obrażenia które zada, średnicę (pociski reprezentowane są jako sf::CricleShape) oraz maksymalny dystans na jaki mogą się oddalić.

Po oddaniu akcji strzału nowy pocisk zostanie dodany do kontenera. Znajdujące się w nim pociski są aktualizowane co klatkę, w przypadku pocisków gracza jest sprawdzane czy kolidowały z jakimkolwiek obecnych aktualnie na planszy przeciwnikiem, a jeśli tak, zadają mu obrażenia i są usuwane. Podobnie w celu stwierdzenia kolizji z graczem, sprawdzane są wszystkie pociski wszystkich przeciwników.

Są one także usuwane, jeśli oddalą się na ustaloną odległość od postaci, która je wystrzeliła, w innym wypadku nawet te, które oddaliły by się od planszy na dalekie odległości zaśmiecałyby pamięć i spowalniałyby grę.

Statystyki pocisku jaki zostanie wystrzelony zależy w przypadku gracza od aktualnie zebranego Power-Upa, a u przeciwnika od jego rodzaju, który zostaje przypisany mu na stałe przy stworzeniu.

Powe Upy

Obiekty pojawiające się na polu gry. Po zetknięciu z graczem otrzymuje on na ograniczony czas dane ulepszenie, a sam obiekt zostaje usunięty.

GUI

W celu nawigowania po menu stworzyłem własne klasy elementów interfejsu graficznego, takie jak przycisk, pasek tekstowy lub rozwijana lista.

Zasoby

W celu łatwiejszego zarządzania teksturami, dźwiękami, elementami GUI, obiektami tekstowymi itp. Zbieram je w kontenerze std::map, w którym każdy obiekt ma swój unikatowy klucz reprezentujący go.

Pliki nagłówkowe

Żeby ułatwić zarządzanie plikami nagłówkowymi i wynikającymi z nich zależnościami, postanowiłem skorzystać z prekompilowanego pliku nagłówkowego, w którym umieściłem wykorzystywane odwołania do plików nagłówkowych biblioteki standardowej, SFMLa oraz kilka własnych funkcji i szablonów pomocniczych.