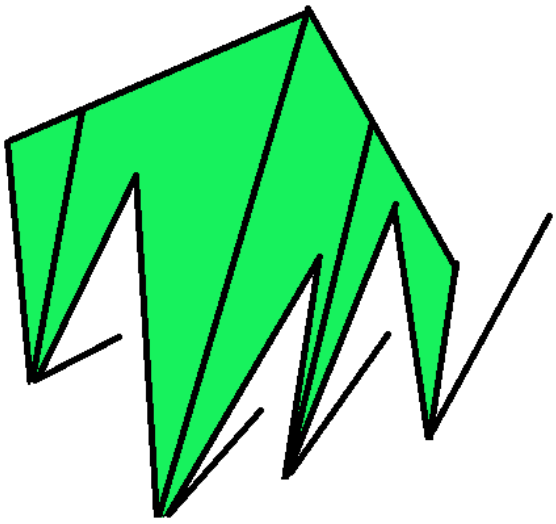
1/18/2021

Projekt PO- Spike Engine



SPIKE ENGINE

Juliusz Łosiński 46155

1. **Nazwa projektu:**

Spike Engine – silnik do tworzenie gier 2D.

2. **Cel projektu:**

Stworzenie silnika, który by umożliwił tworzenie gier wideo. Inaczej mówiąc, ten silnik zapewni programiście gier specjalne API, które umożliwi wpisywanie instrukcji, które dotyczą samej gry np. poruszanie się postaci, wątek fabularny. Programista nie będzie musiał ingerować w silnik, gdyż pewne komponenty silnika (Engine Components) będą zapewnione na starcie, czyli tworzenie grafik (Render component), kontrola klawiszy oraz myszy (Input component), możliwość dodawania różnych komponentów do obiektów gry (Game object component) np. komponent odpowiadający za identyfikację kolizji.

3. **Opis projektu:**

„GameEngine" pakiet zawiera cały kod dotyczący silnika (Game engine), a „Game" pakiet zawiera cały kod dotyczący samych przedmiotów gier (Game objects).

Sam silnik zawiera w sobie komponenty:

- **Render component** – umożliwia tworzenie grafik na ekranie gry. Inaczej mówiąc, pozwala na rysowanie grafik postaci, prostokątów oraz różnych napisów. Owy komponent bazuje na mapie pixeli obrazu, który został uzyskany z obrazu (BufferedImage), który jest rysowany na ekranie przy pomocy canvas a dokładnie Buffer startegy. Owa mapa pixeli jest po prostu naszym ekranem gry.

- **Input component** – umożliwia kontrolę klawiatury oraz myszy.

- **Window component** – tworzy ekran z grą oraz nakłada na nią (na canvas) Buffer strategy, które umożliwi, posiadanie kilku obrazów gry w buforze (RAM) a następnie po prostu zamienienia bieżącego obrazu gry na kolejny z RAM'u, który jest aktualny.

Działanie silnika bazuje na wątku, który wywołuje pętle działania gry, która będzie uwzględniała zmiany np. przesuniecie obiektu na ekranie.

Ogólna zasada działania silnika:

Główną klasą jest „GameContainer” czyli klasa, która zawiera wszystkie komponenty silnika, do niej jest przesyłana obiekt konfiguracyjny gry oraz instrukcje zapisane obiekcie klasy „Game API”, które te instrukcje są później umieszczane w dogodne miejsca w pętli działania gry, do renderowania oraz update’towania.

Do poszczególnych komponentów silnika jest przesyłana referencja do głównej klasy, która ma w sobie odnośnik do innych komponentów . Dzięki temu, możemy np. dodać słuchaczy do canvas, który jest w window component. InputComponent -> GameContainer -> WindowComponent.

Silnik również wspiera obsługę **Audio**, która umożliwia standardową funkcję np. play, stop, loop. Należy tylko przesłać w konstruktorze obiektu ścieżkę do pliku z muzyką.

Sam proces renderowania grafik, dzięki render component:

- Do metody komponentu renderowania możemy przesłać „**sprites**”, czyli zestaw kilku obrazków w jednym. „Sprite” jest to grafika, która jest umieszczania na obiektach pierwszo planowych np. player czy NPC. Dzięki manipulowaniu na jednym obrazku, na którym jest kilka obrazków, możemy płynnie się przemieszczać między obrazkami (sprite’ami).

- Również metody do komponentu renderowania możemy przesyłać „**tiles**”, czyli zestaw kilku obrazków w jednym, tylko one są już grafikami, które będą funkcjonowały jako obrazki drugo planowe np. otoczenie, domki, góry. Zatem sprites oraz tile są do siebie podobne, ale czynią życie łatwiejsze, gdyż mamy pewien dobry podział.

- Też metody tego komponentu umożliwiają **tworzenie łańcuchów znaków** na ekranie, czyli dobieranie odpowiednich znaków oraz ich wartości UTF. Inaczej mówiąc, do metody przesyłany jest obrazek, który zawiera w sobie alfabet oraz liczby. Potem uzyskujemy unicode litery jaka chcemy wyrysować na ekranie, następnie uzyskujemy index tej litery, a konkretniej, która pozycja w wierszu obrazka, ta litera odpowiada.

Np. kiedy mam obrazek 300x16, czyli długość 300 pixeli oraz 16 wysokości. To, kiedy będziemy mieli np. 10 znaków to podzielimy 300/10 to uzyskamy, że jeden znak ma długość 30 pixeli oraz wysokości 16 pixeli. Kiedy uzyskamy index litery za pomocą unicode UTF, to będziemy mogli się przemieścić po tym wierszu. Np. kiedy chcemy wyrysować literę C to musimy przejść o 2 pozycje do przodu, czyli u nas to będzie 60 pixeli w prawo (w wierszu).

Schemat działania przedmiotów gier (Game Objects):

Każdy obiekt w grze dziedziczy po klasie Game Object, czyli każdy obiekt ma pewne określone atrybuty (pola): wysokość, długość, indywidualne ID oraz inne, które są np. przydatne do detekcji kolizji.

Klasa Game object zawiera również w sobie listę wszystkich stworzonych w grzKlasae obiektów.

Obiekt klasy game object, zawiera w sobie listę komponentów (game object components), która umożliwia przechowywanie dodanych komponentów do przedmiotu gry. Dzięki temu, możemy różne obiekty gry, które różnie reagują na środowisko, np. niektóre reagują na kolizję lub działają na ich różne siły (np. grawitacja). Jest to bardzo przydatne, bo dzięki temu mamy możliwość separacji rzeczy miedzy drugoplanowymi oraz pierwszoplanowymi.

Celem jest to, żebyśmy mogli dodawać oraz usuwać komponenty bez ingerencji w resztę systemów, które nie ingerują ze sobą, ale są też niektóre komponenty, które muszą komunikować się ze sobą np. komponent animacji oraz sprite komponent. Sprite component umożliwia rysowanie określonego rysunku z mapy spritów, który pokazują aktualny obrazek dotyczący danego obiektu gry np. obiekt gry, które nazywa się żaba będzie miała dodany sprite komponent, który umożliwi jej narysowanie rysunku żaby na ekranie. A komponent animacji umożliwi dostosowanie przejeść miedzy różnymi obrazkami żaby np. jak skacze lub kiedy nic nie robi.

Każda klasa przedmiotu komponentu dziedziczy po klasie abstrakcyjnej GameObjectComponent, w celu późniejszego wykonywania metod (Update, Render) podczas iterowania przez komponenty, które zawiera w sobie lista, którą ma game object.

Czyli inaczej mówiać każdy game object component ma dwie metody (Render, Update) podobnie jak Game API. Dzięki temu, że każdy game object component ma je w sobie, możemy iterować przez listę komponentów przedmiotu oraz wywoływać te funkcje. Lista komponentów jest ogólna więc może być w niej różny komponent ale dziedziczący po GameObjectComponent.

Zaimplementowane komponenty gry:

- **BoxColliderComponent** – umożliwia detekcje kolizji z innymi obiektami, czy są obok lub jeden na drugim oraz dany przedmiot gry co może zrobić, np. może iść w prawo lub lewo.

- **SpriteComponent**- umożliwia rysowanie grafik przedmiotu gry na ekranie, czyli np. obiekty gry żaba może narysować żabe na ekranie.

- **RigidBodyComponent** – umożliwia dodanie różnych reakcji fizycznych obiektów, np. działanie na siły w tym grawitację.

Zatem jeżeli połączymy RigidBodyComponent oraz BoxColliderComponent będziemy mogli stworzyć obiekt, który spadnie z określonej wysokości z określoną prędkością oraz wyląduje na obiekcie, z którym będzie kolidował.

Jak stworzyć grę?

Żeby stworzyć grę na podstawie tego silnika, należy utworzyć klasę np. Game (tak jest jako przykład), która implementuje interfejs Game API (dwie metody: update oraz render, w tych metodach są instrukcje programisty, który tworzy grę). Później należy stworzyć obiekt klasy game container i w konstruktorze przesłać referencje do obiektu Game.