Podstawy grafiki komputerwej 1 silnik do gier 2d

Rafał Grot, Kamil Gunia, Piotr Górski

November 30, 2023

1 Wymagania systemowe

- SFML
- cmake

2 Użycie biblioteki `engine`

Aby mieć dostęp do silnika gry należy wykorzystać plik nagłówkowy `engine.hpp` oraz zlinkować bibliotekę `engine` Klasa Engine jest singletonem. Zaleca się zaposanie z dokumentacją techniczną. Po ustawieniu parametrów należy wywołać metodę `buildWindow()`. Następnie o ile potrzebna wykonać inicjalizację pętli użytkownika. Na koniec należy wywołać metodę silnika `loop()`.

```
int main(){
Engine::getInstance().setMaxFps(3).setResolution({1000, 1000}).buildWindow();

// ...

Engine::getInstance().loop();
}
```

3 Wybrane klasy wykorzystywanie przez silnik.

3.1 Prymitywy

Przy wykorzystaniu Klasa z biblioteki SFML stworzyć obiekty dziedziczące po `sf::Drawable` oraz dodać je do silnika za pomocą metody silnika `Engine::add()`.

3.2 GameObject

GameObject to klasa reprezentująca obiekt w grze.

3.3 Drawable

Drawable jest to abstrakcyjna klasa bazowa wszystkich klas, które można narysować. Dokładnie jest to alias do `sf::Drawable`.

3.4 AnimatedObject

Jest to klasa zawierająca metodę wirtualną `animate()`, należy ją przeciążyć w celu realizacji animacji. Powinna być użyta w dla animacji jeśli inna klasa obsługująca animacje nie jest odpowiednia.

3.5 AnimatedSpriteSheet

Klasa reprezentująca obiekty, które są animowane przy użyciu bitmap. Dziedziczy po AnimatedObject.

- Obiekty tej klasy wczytują informacje o animacji z plików.
- Konstruktor przyjmuje ścieżkę do katalogu, który musi zawierać plik `config.txt`

Metadane animacji opisane w pliku konfiguracyjnym mają następujący format:

```
COMMAND
ARGS
```

Bezpośrednio za linią z `COMMAND` musi znajdować się linia z argumentami oddzielanym spacją. Linie które mają być traktowane jako komentarze zaczynają się znakiem "#".

3.5.1 Command

Prawidłowe kommendy `COMMAND` to:

`SPRITESHEET` Posiada jeden parametr – ścieżkę relatywną od katalogu, w którym znajduje się plik konfiguracyjny, do pliku zawierającego tablice spirtów.

`ANIMATION` Oznacza ładowanie animacji, kolejne wywołania oznaczają nowe typy animacji. Nie przyjmuje parametrów.

`FRAME` Zawiera informacje o pojedynczej klatce animacji. Przyjmuje parametry oznaczające kolejno

- pozycje x lewego górnego rogu sprita.
- pozycje y lewego górnego rogu sprita.
- rozmiar w osi x sprita.
- rozmiar w osi y sprita.
- czas trwania klatki

3.5.2 Przykład

```
#Comment lines start with '#'
#SPRITESHEET should be followed with with relative path in the next line
#(from config file directory) to the spritesheet

#COMMENTS CAN NOT be in between INFO
#example:
#SPRITESHEET
##SOMECOMMENT
#filename.png
#
#is not allowed

#<path>
SPRITESHEET
spritesheet.png

#Indicates new animation
ANIMATION
#FRAME represents frame of animation
#it is followed by line containing
#<pos x> <pos y> <size x> <size y> <duration>
FRAME
0 0 100 100 0.5
FRAME
100 0 100 100 0 0.5
```

3.6 UpdateableObject

Obiekty których stan logiczny się zmienia powinny przeciążać wirtualną metodę `update` klasy `UpdateableObject`. Silnik nie wywołuje tej metody, należy zadbać aby była ona wywoływana np. przy użyciu własnej pętli gry.

4 Przykład użycia silnika

Program wyświetlający animację.

```
#include "SFML/Graphics/Color.hpp"
#include "animatedSpriteSheet.hpp"
#include "engine.hpp"
#include <iostream>

namespace G {
std::string basePath = "resources/";
}; // namespace G

AnimatedSpriteSheet animation(G::basePath + "animation");

int main() {
  Engine::getInstance().setMaxFps(3).setResolution({1000, 1000}).buildWindow();

  animation.setPosition({300, 300});
  animation.setColor(sf::Color::Cyan);

  Engine::getInstance().add(&animation);

  Engine::getInstance().loop();
}
```

5 Z wymagań

5.1 Obsługa klawiatury i myszy

Wykorzystaj `setEventHandler()`. Po szczegóły zajrzyj do dokumentacji.

```
#include "engine.hpp"
#include <iostream>

int main() {
  Engine &eng = Engine::getInstance().setWindowTitle("dev").buildWindow();

  eng.setEventHandler(
      Engine::Event::MouseButtonPressed, [](const Engine::Event &ev) {
        std::cout << "Custom event handler Mouse button press "
                  << ev.mouseButton.button << '\t' << ev.mouseButton.x << '\t'
                  << ev.mouseButton.y << "\n";
      });
```

```
  eng.loop();
}
```

5.2 Obsługa współrzędnych (Point2D)

Klasa `Point2d`. Po szczegóły zajrzyj do dokumentacji.

5.3 Rysowanie prymitywów

Przy użyciu klass z bilbioteki SFML oraz metod `Engine::add()` i `Engine::remove()`. Szeczóły w dokumentacji technicznej oraz dokumentacji SFML.

5.4 Wypełnianie prymitywów kolorem

Przy użyciu metod obiektów z biblioteki SFML.

5.5 Przekształcenia geometryczne

Przy użyciu SFML.

5.6 Obsługa bitmap

Przy użyciu biblioteki SFML tak jak z primitywami.

5.7 Animowanie bitmap

Przy użyciu klasy `AnimatedSpriteSheet`. Szecgóły w dokumetnacji technicznej.