

Instrukcja projektowa; projekt numer 2

Podstawy Programowania 2023/24, kierunek Informatyka

autor: Robert Ostrowski¹
wersja z dnia: 5.01.2024 r.

Projekt King Donkey

Cel

Celem projektu jest implementacja gry „King Donkey” i uniknięcie pozwu za podobieństwo z <https://www.youtube.com/watch?v=Pp2aMs38ERY>. Gra polega na sterowaniu ruchem i skokami postaci przemierzającej zdewastowany budynek i unikaniu beczek w drodze na szczyt (po punkty za projekt). Wybrane funkcjonalności/elementy gry, które należy zaimplementować podane są poniżej.

Opisy gry można znaleźć na niniejszej stronie:
[https://en.wikipedia.org/wiki/Donkey_Kong_\(arcade_game\)](https://en.wikipedia.org/wiki/Donkey_Kong_(arcade_game))

Środowisko programistyczne

Do instrukcji dołączony jest program startowy w którym zaimplementowano:

- obliczanie przyrostu czasu, co pozwala śledzić jego upływ
- wyświetlanie na ekranie plików graficznych w formacie BMP
- rysowanie piksela, linii, prostokąta
- wyświetlanie tekstu

Program działa w oparciu o bibliotekę SDL2 (2.0.3) – <http://www.libsdl.org/>. Jest ona dołączona do projektu startowego i nie trzeba pobierać jej źródeł. Kompilacja pod systemem Linux wykonujemy za pomocą komendy (w systemie 32-bitowym):

```
g++ -O2 -I./sdl/include -L. -o main main.cpp -lm -lSDL2 -lpthread -ldl -lrt
```

oraz (w systemie 64-bitowym)

```
g++ -O2 -I./sdl/include -L. -o main main.cpp -lm -lSDL2-64 -lpthread -ldl -lrt
```

W celu pomyślnej kompilacji projektu startowego, w katalogu, w którym znajduje się plik main.cpp powinny znajdować się

- Bitmapy z wymaganymi rysunkami (cs8x8.bmp, eti.bmp). Uwaga na wielkość liter w

¹ Uwaga: W razie niejasności lub niejednoznaczności w poniższym opisie proszę kontaktować się z autorem instrukcji pod adresem robert.ostrowski@pg.edu.pl; dalsze informacje kontaktowe znajdują się na stronie enauczania.

nazwach plików!

- Plik libSDL2.a (libSDL2-64.a przy kompilacji 64 bitowej).
- Katalog sdl dołączony do projektu.

Do projektu dołączone zostały skrypty, które mogą być użyte do kompilacji (comp w środowisku 32-bitowym oraz comp64 w środowisku 64-bitowym).

Prezentacja programu (zaliczenie tej części projektu) odbywać się będzie w wybranym przez studenta środowisku spośród dwóch poniższych opcji:

- w systemie Linux. Student jest zobowiązany sprawdzić przed przybyciem na zaliczenie czy program poprawnie się kompiluje i uruchamia pod dystrybucją dostępną w laboratorium,
- w systemie Windows, w środowisku MS Visual C++ w wersji zgodnej z tą dostępną w laboratorium.

Uruchomienie programu podczas zaliczenia jest warunkiem koniecznym uzyskania punktów z projektu nr 2.

W programie nie należy używać biblioteki C++ **stl**.

Wymagania obowiązkowe (5 pkt.)

Wszystkie wymienione tutaj elementy należy zaimplementować. Wersja do wymagań podstawowych jest bardzo uproszczona, w szczególności sterowanie można wykorzystać do testowania kolejnych wymagań. Brak któregokolwiek z poniższych elementów skutkuje otrzymaniem 0 pkt. z tego projektu.

1. Przygotowanie graficznej oprawy gry: obrys planszy, przygotowanie miejsca na wyświetlanie dodatkowych informacji: czasu, który minął od początku etapu.
Implementacja klawiszy sterujących:
 - a. **Esc**: wyjście z programu – program jest natychmiast zakończony,
 - b. **'n'**: nowa rozgrywka.
2. Implementacja jednego etapu gry. Scena tego etapu gry powinna posiadać podłogę oraz kilka poziomych, wiszących platform połączonych drabinkami. Rozmiar sceny można ograniczyć do rozmiaru okna.
3. Implementacja mechaniki poruszania postacią przy pomocy klawiszy kierunkowych. Ruch w lewo i prawo powinien być ograniczony platformami, a w górę i dół drabinkami. Pozycja postaci powinna być ograniczona rozdzielczością gry, a nie pozycją bloków z których zbudowana jest scena.
4. Nie ma konieczności implementacji warunku końca etapu (porażki lub zwycięstwo), ale należy poprawnie mierzyć czas przeznaczony na rozgrywkę w tym etapie. Powinien on być wyświetlany wraz z informacją o wykonanych podpunktach w wyznaczonym miejscu.

Wymagania nieobowiązkowe (10 pkt.)

A. (1 pkt) Implementacja skoków:

- Po naciśnięciu **spacji** postać powinna skoczyć. Wysokość skoku powinna być ograniczona. Możliwe powinno być przeskoczenie beczek (jeśli zaimplementowano punkt C), a także wskoczenie na platformę. Po wyjściu poza krawędź platformy postać powinna spadać aż do napotkania platformy lub podłoża niżej.
- Parametry ruchu postaci i sterowania powinny być łatwe do zmiany aby osiągnąć płynne sterowanie.

B. (1 pkt) Przygotowanie wielu etapów.

- Przynajmniej 3 różne etapy powinny być dostępne.
- W celu ułatwienia demonstracji zmiana etapu następuje po naciśnięciu klawisza numerycznego mu odpowiadającego.
- Gra powinna wykryć wejście w obszar kończący etap i zasygnalizować to zdarzenie (kontynuacja w punkcie D).

C. (1 pkt) Beczki

- Beczki poruszają się zaczynając w określonym miejscu i podróżują stałą trasą do punktu końcowego lub według prostych zasad.
- Zachowanie beczek i powinno być sparametryzowane.
- Gra powinna wykryć kolizję z beczką zasygnalizować te zdarzenie (kontynuacja w punkcie D).



D. (1 pkt) Śmierć, liczba żyć i menu:

- Po uruchomieniu gry powinno wyświetlone zostać menu umożliwiające wybór wszystkich opcji: wyjście, sprawdzenie wyników (punkt G), wybór etapu (punkty B oraz I). Wybór nie zaimplementowanej opcji powinien wyświetlić komunikat o jej niedostępności.
- Tekst wpisywany przez gracza (np. podczas wpisywania pseudonimu) powinien zawsze być widoczny i klawisz **backspace** powinien umożliwić usuwanie liter.
- Wyświetlanie na ekranie pozostałej liczby żyć postaci w formie graficznej.
- Utrata życia po dotknięciu beczki (jeśli zaimplementowano punkt C).
- Utrata życia powinna wyświetlić zapytanie o kontynuację i (jeśli zaimplementowano punkt F) liczbę zdobytych punktów.

- f. Utrata wszystkich żyć powinna spowodować wyświetlenie menu głównego i (jeśli zaimplementowano punkt *H*) po wcześniejszym zapytaniu o zapis wyniku.
- g. Po dotarciu na szczyt należy przejść do kolejnego etapu. (Proponowane jest przygotowanie prostego sposobu demonstracji tej funkcjonalności wcześniej)

E. **(2 pkt.)** Animacje (wymaga przynajmniej 2 z 3 dodatkowych podpunktów zaznaczonych poniżej):

- a. Animacje biegu, skoku (punkt *A*) i wspinania postaci.
- b. Animacje tytułowego antagonisty (tj. źródła beczek) odpowiadającego za zrzucanie beczek (tj. sygnalizację, że beczka za chwilę się pojawi). Panika nie jest wskazana, gdyż piękno grafiki nie jest celem projektu.
- c. Animacje beczek (punkt *C*).
- d. Zdobyte punkty bonusowe (punkt *F*) powinny być na chwilę wyświetlane na ekranie.
- e. **Uwaga:** szybkość animacji nie powinna być zależna od szybkości komputera (przy założeniu, że spełnia on minimalne wymagania gry)!

F. **(1 pkt)** Liczenie punktów:

- a. Liczba punktów powinna być wyświetlana na ekranie i dynamicznie zmieniana.
- b. Ukończenie etapu zwiększa liczbę punktów. Powinny być one zachowane przy przejściu do kolejnego etapu (lub restart obecnego w przypadku braku punktu *B*).
- c. Przeskakiwanie beczek (jeśli zaimplementowano punkt *C*) powinno dawać dodatkowe punkty.
- d. Umieszczenie trofeum dającego dodatkowe punkty po dotknięciu w określonym miejscu etapu.



G. **(1 pkt)** Zapamiętywanie najlepszych wyników:

- a. Po zakończeniu gry powinno być możliwe wpisanie swojego pseudonimu i wyniku do pliku. Ilość zapisanych w pliku wyników nie powinna być ograniczona
- b. Z poziomu menu można wyświetlić posortowaną listę wyników.
- c. Liczba wyników na ekranie, w przeciwieństwie do zawartych w pliku, powinna być ograniczona.
- d. Jeśli nie wszystkie wyniki mieszczą się na ekranie program powinien umożliwić dotarcie do nich np. poprzez przełączanie się między stronami lub przewijanie listy.
- e. W przypadku braku punktu *F* zapisywać można czas zamiast wyników.

H. **(1 pkt)** Zaawansowany wygląd etapu gry:

- a. Wiszące platformy mogą być pochylone (lub przekonująco udawać pochylone).



I. **(1 pkt)** Kodowanie wyglądu etapu gry w pliku.

- a. Należy zaprojektować własny (edytowalny, np. w edytorze tekstowym) format pliku etapu gry. Format taki powinien być znany studentowi, tak aby był w stanie wytłumaczyć oraz dokonać wskazanych edycji etapu podczas odpowiedzi. Podczas startu danego etapu, powinien nastąpić odczyt wyglądu tego etapu z pliku. Plik powinien zawierać przynajmniej informacje o położeniu wszystkich przeszkód znajdujących się w tym etapie oraz wymiarach etapu.
- b. Dodatkowo w pliku powinny znajdować się informacje o wszystkich elementach o które rozszerzono implementację podczas realizacji wymagań nieobowiązkowych.
- c. **Uwaga:** program nie powinien nakładać limitów na maksymalną liczbę obiektów różnego typu znajdujących się w pliku. Oznacza to, że program analizuje zawartość pliku i następnie przydziela pamięć wystarczającą na przechowanie danych o wszystkich obiektach znajdujących się w pliku. Format kodowania tych informacji w pliku należy dobrać samodzielnie, co




oznacza, że dla pewnego ułatwienia wczytywania opisu etapu z pliku można zdecydować się na taki format, w którym na początku pliku znajdują się informacje (preambuła) o liczbie poszczególnych obiektów, a następnie znajduje się sam opis planszy. W ten sposób można wczytać preambułę, zaalokować pamięć na podstawie znajdujących się tam danych a następnie wczytać etap. (Przy takim rozwiązaniu należy dodać sprawdzenie poprawności danych – program powinien rozpoznać niezgodność preambuły z zawartością)

Wymagania „z gwiazdką” (3 pkt.)

Uwaga: poniższe wymagania rozszerzają podpunkty: wygląd etapu gry, kolizje, kodowanie wyglądu etapu gry i są oceniane wyłącznie gdy zaimplementowane zostały wszystkie pozostałe punkty.

- J. (1 pkt) Fizyka
- Platformy mogą być pod dowolnym kątem i modelowane równiami pochyłymi.
 - Beczki powinny poruszać i zderzać się zgodnie z określonymi zasadami “fizyki” w grze, zamiast z góry wyznaczonej trajektorii. Za kryterium zniknięcia można przyjąć ograniczony parametrem czas trwania.
- K. (2 pkt) Dynamiczne tworzenie etapu - specjalny etap w trybie nieskończonym
- Zasady generowania powinny być sparametryzowane.
 - Gdy gracz wespnie się powyżej połowy ekranu wygenerowany powinien zostać kolejny fragment etapu. Kamera podąża za graczem w górę odsłaniając wygenerowaną scenę
 - Platformy i drabiny powinny być wygenerowane w taki sposób aby gracz zawsze miał możliwość wspięcia się wyżej.
 - Punktowane powinno być dotarcie do coraz wyżej położonych, wyznaczonych miejsc etapu.
 - Antagonista (tj. źródło beczek) powinien również zmieniać pozycję po dotarciu do takiego miejsca.

Uwagi końcowe

-  Wymagania dotyczące szaty graficznej: wystarczające jest użycie dowolnych bitmap (dobrych właściwie pod względem rozmiaru).
-  Konfiguracja programu powinna umożliwiać łatwą zmianę wszelkich parametrów, nie tylko tych wyraźnie wskazanych w powyższym opisie. Przez łatwą zmianę rozumiemy modyfikację stałej w programie.
-  Projekt może być napisany w sposób obiektowy, ale całkowicie zabronione jest używanie biblioteki standardowej C++ (w tym typu string, cin, cout, vector itp.) (Uwaga: typu string z biblioteki C++ nie należy mylić w biblioteką string.h z C –

można używać funkcje znajdujące się w string.h)

- Obsługa plików powinna być zrealizowana przy użyciu biblioteki standardowej C (rodzina funkcji f???? - np. fopen, fread, fclose itd.)
- Każdy fragment przedstawionego do oceny kodu powinien być napisany samodzielnie przez studenta.
- Szybkość działania programu powinna być niezależna od komputera, na którym uruchomiono program. Stałe jednostki w programie powinny być opisane odpowiednimi komentarzami, na przykład:

```
const int SZEROKOSC_PLANSZY = 320;    // piksele
const double POZYCJA_X_NAPISU = 60.0; // procent szerokości ekranu
const double POZYCJA_Y_NAPISU = 5.0;  // procent wysokości ekranu
```