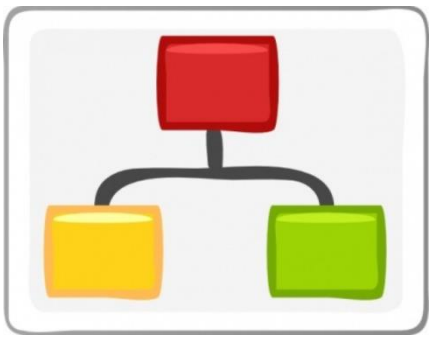


Praktyka programowania 2019/2020

Instrukcja projektowa cz. 2

Zadanie 2 – Gra Frogger

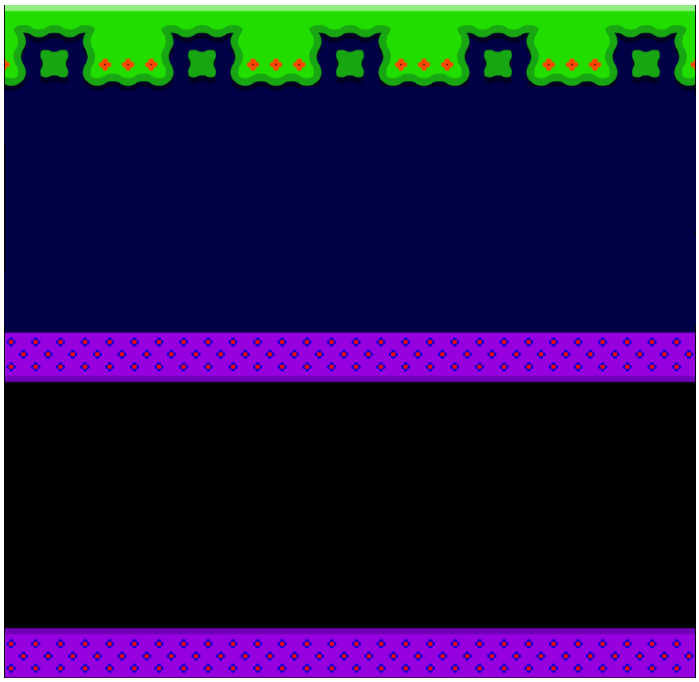
Autor: Tomasz Goluch

Wersja: 1.0

I. Temat projektu

Cel: Zapoznanie z tematem drugiego zadania części projektowej przedmiotu.

Tematem drugiego zadania projektowego jest napisanie gry podobnej do gry Frogger. Gra rozgrywa się na planszy przedstawionej na rysunku. 1.



Gracz steruje żabką za pomocą klawiszy strzałek lub WSAD. Celem jest przedostanie się pięcioma żabkami na drugą stronę ulicy, a następnie rzeki. Podczas rozgrywki gracz musi uważać na pojazdy znajdujące się na ulicy oraz aby nie wpaść do rzeki, ponieważ kończy się to utratą życia. Po utracie wszystkich żyć gra się kończy, wynik (jeśli znajduje się w pierwszej dziesiątce) zostaje zapamiętany, a gracz może podać swój login.

II. Środowisko pracy

Cel: Zapoznanie z wykorzystywanym w projekcie oprogramowaniem.

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 (2.0.10) – http://www.libsdl.org/. Jest ona dołączona do projektu startowego (nie trzeba jej ściągać).

Kompilacja pod systemem Linux wykonujemy (w systemie 32) za pomocą komendy:

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

oraz (w systemie 64-bitowym) za pomocą komendy:

```
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 nazwach plików!
- Plik libSDL2.a (libSDL2-64.a przy kompilacji 64 bitowej).
- Katalog SDL2-2.0.10 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.

III. Kryteria oceny

Cel: Zapoznanie z wymaganymi, dodatkowymi oraz bonusowymi kryteriami oceny projektu.

Wymagania obowiązkowe (5 pkt.)

Wszystkie wymienione tutaj elementy należy zaimplementować. Brak któregokolwiek z poniższych elementów skutkuje otrzymaniem 0 pkt. z tego projektu:

- **Oprawa graficzna** – rysowanie planszy, Frogger'a oraz poruszających się pojazdów, żółwi i bali. Poruszające się elementy powinny zawijać się na szerokości 2 ekranów. Oznacza to, że znikające po prawej stronie bale, samochody itp. powinny jeszcze „przejeżdżać" wirtualną jedną szerokość ekranu i dopiero wyjeżdżać ponownie z lewej strony.
- **Sterowanie** – Frogger'em sterujemy za pomocą klawiszy ze strzałami: ↑,↓,←,→ lub klawiszy: W, S, A, D. Powinien on przeskakiwać o jedno (pole) w kierunku ostatnio wciśniętego klawisza.
- **Implementacja jednego etapu gry** – scena tego etapu gry powinna zawierać poruszające się pojazdy, żółwie i bale. Nie muszą być animowane, w szczególności żółwie nie muszą nurkować. Nie ma konieczności implementacji efektu osiągania końca etapu (zwycięstwo/porażka).
- **Kolizje** – implementacja mechanizmu kolizji pozwalającego na stratę życia przez Frogger'a w przypadku zetknięcia się z pojazdem bądź utopieniem się w wodzie. Frogger ginie również w przypadku wypłynięcia poza ekran (na balu, żółwiu itp…)

oraz próbie wskoczenia na górny brzeg w miejscu innym niż przeznaczone do tego i nie zajęte wcześniej. Po stracie życia program powinien się kończyć.

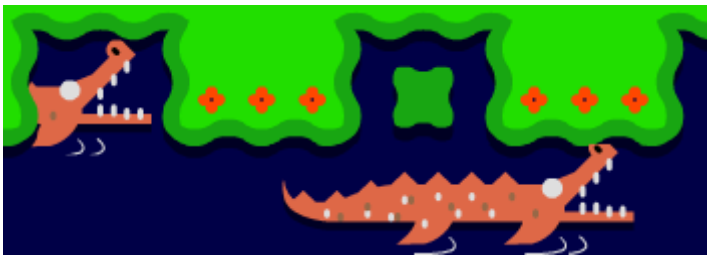
Wymagania dodatkowe (10 pkt.)

- **Życia** – gracz powinien posiadać pewną ustaloną liczbę żyć. Po rozjechaniu przez pojazd bądź utopieniu się w rzece ich liczba zmniejsza się o jedno, a gra powinna zaczynać się od pozycji startowej. W momencie, kiedy gracz straci ostatnie życie wyświetla się plansza z napisem **GAME OVER** i z pytaniem czy zakończyć grę? – (**1pkt**).
- **Pauza i zakończenie gry** – Naciśniecie przycisku **p** w trakcie trwania rozgrywki powoduje zatrzymanie się gry i wyświetlenie napisu **PAUSED** (zakończenie pauzy następuje poprzez ponowne naciśnięcie przycisku **p**), a przycisku **q** napisu: **OUIT GAME? Y/N** – co pozwala na powrót no menu głównego – (**1pkt**).
- **Czas** – w dolnej prawej części ekranu powinien pojawić się wskaźnik upływu czasu. Po przekroczeniu 40 sekund kolor paska powinien się zmieniać na czerwony (ostrzeżenie) a po kolejnych 10 sekundach gracz traci jedno życie i gra zaczyna się od ekranu startowego – (**1pkt**).
- **Punktacja** – w dolnej części planszy powinien być wyświetlany wynik. Punktacja jest następująca:
    - osiągnięcie pozycji bliższej górnemu brzegowi planszy, niż dotychczasowa o jeden skok = 10 pkt;
    - przeprowadzenie Froggera do gniazda na bezpiecznym brzegu rzeki = 50pkt + pozostały czas w sekundach * 10pkt (jeśli zaimplementowano upływ czasu);
    - zjedzenie pszczoły (jeśli zaimplementowano bonusy) = 200 pkt;
    - przeprowadzenie zagubionej żabki (jeśli zaimplementowano bonusy) = 200 pkt.

  Po przejściu na drugą stronę rzeki powinna nad żabką pojawić się liczba informująca o wielkości uzyskanego bonusu. Uwaga jeżeli bonusy sumują się (przeprowadzenie żabki i zjedzenie pszczoły) powinna wyświetlać się ich suma – (**1pkt**).
- **High scores** – Najlepsze 10 wyników powinno być zapisywane w pliku i wyświetlane po wybraniu opcji HIGH SCORES z początkowego menu.

  Zapisanie wyniku powinno być możliwe po zakończeniu rozgrywki, gracz może podać z klawiatury swoją nazwę, która również powinna być zapamiętana w pliku – (**1pkt**).
- **Bonusy** – pojawiająca się w drugim rzędzie zagubiona żabka którą należy przeprowadzić na drugą stronę rzeki, co daje dodatkowych 200 pkt, oraz pojawiająca się losowo pszczoła na drugim końcu rzeki, której zjedzenie daje również dodatkowe 200 pkt. Cyklicznie pewna grupa żółwi powinna nurkować powodując utopienie się przebywającej na nich żabki – (**1pkt**).
- **Animacje** – skakanie żabki, płynące i nurkujące grupy żółwi, obracające się opony niektórych pojazdów, animacje krokodyli (drugi etap – jeśli zaimplementowano) – (**1pkt**).
- **Kolejne etapy** – Powinny być zaimplementowane przynajmniej pięć kolejnych etapów gry. W drugim etapie powinny pojawiać się krokodyle (zarówno pływające –

na które można wskoczyć, jak i te ujawniające się z ukrycia na drugim brzegu) oraz pędzące wyścigówki.



W trzecim etapie pojawia się pływający piesek który dopływa do balu i znika (wchodzi na niego). Można na niego wskoczyć ale należy zeskoczyć przed jego zniknięciem. Pojawiają się również węże które są zagrożeniem, ale tylko głowa, wejście na ogon nie zagraża Froggerowi.

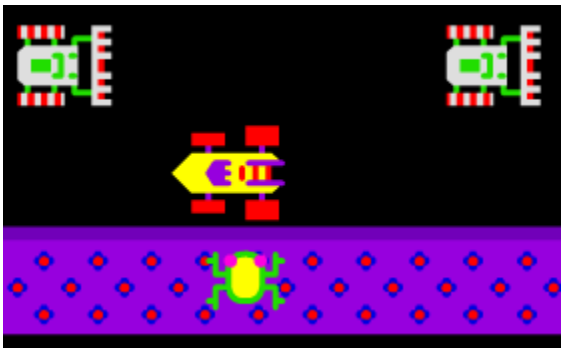
Dodatki czwartego etapu polegają na zmianie ilości (mniej bali, więcej krokodylów, piesków, znikających żółwi lub pędzących wyścigówek …) oraz na przyśpieszaniu pewnych ruchomych elementów wybranych rzędów. Mile widziane są również własne, nietrywialne innowacje – (**1pkt**).

- **Kodowanie wyglądu etapu gry w pliku** – 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 informacje o elementach wymienionych w wymaganiach podstawowych, czyli:
  - położenie początkowe Froggera,
  - czasu przeznaczonego na rozgrywkę w tym etapie,
  - początkowego położenia oraz prędkości wszystkich elementów ruchomych znajdujących się w tym etapie,
  - powinien również określać takie parametry jak prawdopodobieństwo pojawienia się elementów bonusowych takich jak: zagubione żabki, pszczoły, przyczajone krokodyle itp.

  Łączna liczba etapów jest automatycznie rozpoznawana przez program na podstawie liczby plików umieszczonych w bieżącym katalogu i posiadających określone przez studenta nazewnictwo. Np. zaleca się aby pliki z kolejnymi etapami nazwać etap1.txt, etap2.txt, etap3.txt itd. Wówczas liczba etapów jest równa najwyższej liczbie x takiej, że bieżący katalog zawiera kolejne x ponumerowanych w powyższy sposób plików. Dzięki takiemu rozwiązaniu dodanie kolejnego etapu realizuje się tylko poprzez umieszczenie nowego pliku (o odpowiednim numerze) w bieżącym katalogu. Koniec bieżącego etapu i rozpoczęcie następnego następuje gdy Mario dotrze do oznaczonej lokalizacji – (**1pkt**).
- **Zapis stanu gry** – naciśnięcie klawisza **a** w dowolnym momencie powinno włączać/wyłączać automatycznego gracza. – (**1pkt**).

Wymaganie bonusowe (**3 pkt.**)

- **Automatyczny gracz** – naciśnięcie klawisza **a** w dowolnym momencie gry powinno uruchamiać/zatrzymywać automatycznego gracza. Automatyczny gracz stara się jak najszybciej przeprowadzić Froggera do gniazda na bezpiecznym brzegu rzeki. Algorytm działa zachłannie, starając się iść/skakać przeważnie do góry, jednak powinien być w stanie przewidzieć sytuacje które mogą zakończyć się śmiercią Froggera, np. skok na bal z którego nie da się ani pójść dalej ani zawrócić. Generalnie automatyczny gracz dysponuje całą wiedzą na temat stanu gry i na jej podstawie w odpowiednim momencie generuje zdarzenia odpowiadające naciśnięciu jednego w czterech klawiszy sterujących: ↑,↓,←,→ lub W, S, A, D. Jeśli pewne zdarzenia dzieją się w sposób pseudolosowy, np. atak krokodyla z ukrycia, automatyczny gracz nie jest w stanie przewidzieć pewnych sytuacji ale powinien momentalnie reagować. W takim przypadku dopuszczalna jest, w niektórych przypadkach nawet śmierć Froggera – (**1 pkt**).
- **Automatyczny gracz z wykorzystaniem bonusów** – jeśli istnieje możliwość zarobienia dodatkowych punktów związanych z możliwością wykorzystania bonusu np. zabranie zagubionej żabki albo zjedzenie pszczoły automatyczny gracz powinien z tego skorzystać. Powinien znaleźć (może w sposób zachłanny) i zacząć się poruszać jak najszybszą drogą do bonusu. Chyba, że stwierdzi, że bonus jest poza jego zasięgiem. Po zdobyciu bonusu jeśli gra się nie zakończyła powinien kontynuować algorytm z poprzedniego punktu. Wymaga implementacji **Automatycznego gracza** – (**1 pkt**).
- **Doskonały automatyczny gracz** – automatyczny gracz powinien grać w sposób pozwalający na maksymalizację wyniku aktualnie sterowanego Froggera, biorąc pod uwagę nie tylko punkty z bonusów ale również wybierając w każdym momencie gry najbardziej optymalną drogę. Algorytm powinien znać wszystkie możliwe drogi oraz ich czasy do wszystkich jeszcze nie zajętych gniazd po bezpiecznej stronie brzegu i wybierać najbardziej optymalną. Analizowane drogi mogą być tworzone w następujący sposób. W sytuacji początkowej widocznej poniżej na rysunku Frogger może od razu poruszyć się w prawo i w lewo albo poczekać i poruszyć się do przodu, do tyłu poruszyć się nie może. Taka sytuacja generuje trzy możliwe warianty dróg a z każdego takiego wariantu mamy kolejne maksymalnie 4 możliwe itd. itd.



Generowane w ten sposób drzewo możliwych posunięć może być bardzo duże a jego analiza bardzo czasochłonna. Ponadto analizując możliwość poruszania się Froggera do tyłu liczba takich dróg może być nieskończona. W celu pozbycia się problemów

wprowadzanych przez opóźnienia czasowe na stan gry (aby wyeliminować przypadki podejmowania decyzji na nieaktualnych danych) należy przed wykonaniem obliczeń wstrzymać grę (wykorzystując wymaganie dodatkowe pauza). W celu poradzenia sobie z problemem powrotów (ruchy w dół planszy) Froggera, należy stosować je tylko w przypadku pojawienia się adekwatnego bonusu (np. zagubionej żabki). Możliwość powrotu powinna być zawsze ograniczona do dolnego brzegu, powrót na ulicę raczej nie przyniesie nam korzyści. Ponadto można testować algorytm uruchamiając go już na wodzie, co zmniejsza odległości do niezajętych gniazd i powinno znacznie ograniczyć wymagane obliczenia. Wymaga implementacji **Automatycznego gracza** i **Automatycznego gracza z wykorzystaniem bonusów** – (**1 pkt**).

Wymagania bonusowe będą wymagać wcześniejszej implementacji pewnych wymagań dodatkowych (np. pauza, upływ czasu, punktacja) i nie będą oceniane w innym przypadku. Pomimo, że **nie wszystkie** 10 wymagań dodatkowych musi być zaimplementowane aby otrzymać punkty z implementacji wymagań bonusowych to jednak te drugie wymagają dużo więcej wiedzy i nakładów sił. Zatem nie jest korzystne dla studenta skupianie się na wymaganiach bonusowych w miejsce dodatkowych. Wymagania bonusowe się zawierają np. implementacja **Automatycznego gracza z wykorzystaniem bonusów** wymaga implementacji **Automatycznego gracza** i w tym przypadku student otrzymuje **2 pkt** za realizację obydwu podpunktów. W przypadku zaimplementowania wszystkich podstawowych i dodatkowych oraz części bądź wszystkich bonusowych wymagań można otrzymać z tej części projektu 16 do 18 pkt.

Ocenie będzie podlegał także sposób implementacji i styl kodowania. W zależności od tej oceny wynik może być przemnożony przez liczbę z zakresu 0.5–1.5. Jeśli odpowiedzi na zadane przez prowadzącego pytania sugerują niesamodzielną pracę, prowadzący może obniżyć wynik o dowolną wartość.

Dobrym przykładem na którym można się wzorować jest: http://www.happyhopper.org/, a tutaj można zobaczyć mistrzowski wynik odkrywający późniejsze etapy gry: https://www.youtube.com/watch?v=u23kf-XBack.