SnakeC 2.1

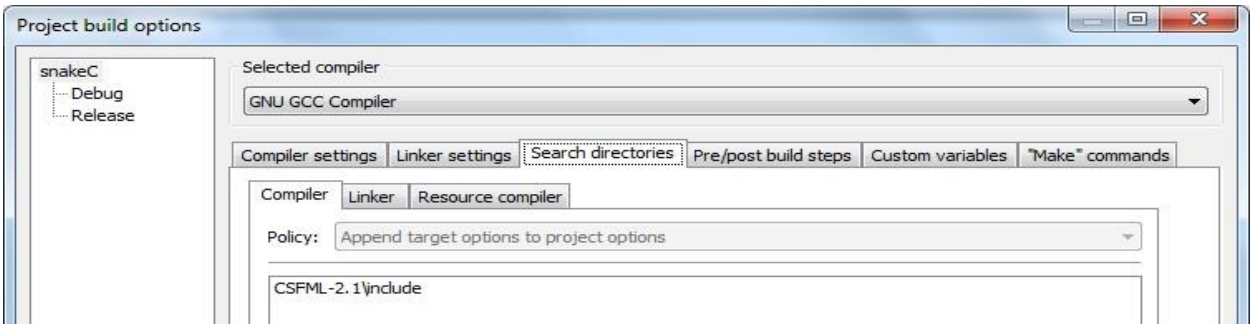
Zawartość archiwum

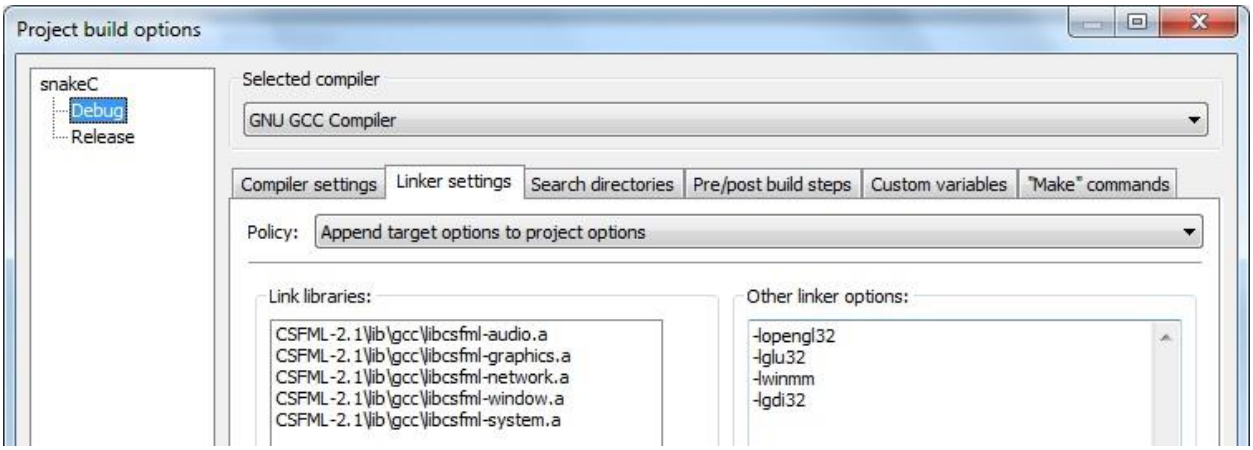
- **snakeC.cbp** – plik projektu
- **main.c** – plik z kodem źródłowym
- **snake.png** – obrazek głównego menu z nazwą gry
- **font.otf** – czcionka napisów programu
- **icon.ico** – ikona pliku wykonywalnego exe
- **icon.png** – ikona okna aplikacji
- **background.jpg** – tło programu
- **resources.rc** – plik odpowiedzialny za ustawienie ikony pliku exe
- Folder **CSFML-2.1** – folder z biblioteką graficzną CSFML (SFML for C)

Kompilacja

Projekt **snakeC.cbp** należy skompilować używając biblioteki graficznej **CSFML 2.1** (SFML for C) [*dołączonej do archiwum*]

Z ustawieniami (opcja „Build options" projektu):





Użytkowanie programu

Jest to popularna gra Snake (celem gry jest uzyskanie jak najwyższego wyniku [*score*]). Opcje w menu wybiera się za pomocą kursora, gdzie mamy do wyboru kolejno:

- **PLAY** – start nowej gry
- **DIFFICULTY** – wybór poziomu trudności (prędkość węża)
- **EXIT** – wyjście z gry

Wężykiem steruje się używając klawiszy:

- **W** – Ruch do góry
- **S** – Ruch w dół
- **A** – Skręt w lewo
- **D** – Skręt w prawo

Dokumentacja programu

- Oprócz biblioteki graficznej SFML zostały dołączone:
  - <windows.h> - do funkcji min()
  - <time.h> - do losowania liczb (a szczególnie pozycji „jabłka")
  - <math.h> - do funkcji abs() zwracającej wartość bezwzględną
  - <stdbool.h> - do zmiennych boolowskich
- **struct elementListy** – element listy dwukierunkowej
- **struct lista** – lista dwukierunkowa posiadająca wskaźnik na pierwszy oraz ostatni element listy. Jest ona stworzona specjalnie do przechowywania danych o wężu, dlatego posiada ona również zmienne takie jak:
  - **score** – do przechowywania aktualnego wyniku danej rozgrywki
  - **frozen** – wykorzystywana do przedłużania węża, gdy „jabłko" zostanie „zjedzone"
  - **up** – *true* gdy wąż idzie w dół lub w prawo, *false* - gdy idzie w górę lub w lewo
  - **lose** – *true* - gdy rozgrywka zostanie zakończona przegraną, *false* – w przeciwnym przypadku
  - **horizontal** – przechowująca informacje czy wąż porusza się pionowo, czy poziomo
  - **speed** – prędkość węża (im większa tym wolniej się porusza)

- lista funkcji programu:
  - void **mainMenu()** - funkcja obsługująca główne menu
  - void **game()** - funkcja obsługująca grę (snake)
  - void **settings()** - funkcja wyboru poziomu trudności gry
  - bool **isOptionMarked(sftext* text, sfcolor on, sfcolor off)** - funkcja sprawdzająca czy dana opcja z menu jest zaznaczona (czy jest na niej kursor i czy została kliknięta)
  - void **showMenu()** - funkcja animacji pojawiania się menu
  - void **hideMenu()** - funkcja animacji znikania menu
  - void **addToSnake(int val)** - funkcja dodająca jeden element do listy (co daje przy rysowaniu węża o jeden prostokąt więcej)
  - void **removeFirst()** - funkcja usuwająca pierwszą zmienną z listy(co odpowiada zniknięciu ostatniego prostokąta)
  - void **clearSnake()** - funkcja czyszcząca listę *snake*
  - void **startSnake()** - funkcja ustawiająca wszystkie początkowe wartości przy rozpoczęciu nowej rozgrywki
  - void **setApple()** - funkcja ustawiająca „jabłko" w odpowiednim miejscu
  - bool **isCorPos()** - funkcja sprawdzająca czy wybrana pozycja „jabłka" jest poprawna
  - bool **isEaten()** - funkcja sprawdzająca czy „jabłko" zostało „zjedzone"
  - bool **isConflict()** - funkcja sprawdzająca czy wąż się ze sobą zderzył
  - bool **collisionWall()** - funkcja sprawdzająca czy wąż zderzył się ze ścianą
  - void **intToString(int a)** - funkcja zamieniająca *int* od liczby uzyskanych pkt. w *string*

Inne informacje

- Wąż jest tworzony za pomocą prostokątów tworzonych w czasie rysowania. Każdy taki prostokąt jest uzyskiwany z 3 elementów listy(jeden dodatkowy element listy = jeden prostokąt więcej) dzięki czemu program zużywa mniej pamięci (nie musi zapamiętywać osobno współrzędnych każdego punktu zgięcia węża). Wąż porusza się z każdym wywołaniem pętli o 1px co daje płynność wizualną,

jednak skręcać może co 15px, aby poruszać się tylko po punktach kratowych. Gdy wąż zje jabłko jego ogon zostaje zatrzymany w miejscu na czas **X** kolejnych iteracji pętli (gdzie **X = snake.frozen**).

- „*Dlaczego jest tyle zmiennych globalnych*?"

  Nie bez powodu zostało użytych tyle zmiennych globalnych.
  Cały aplikacja działa w jednej pętli:
  while(sfRenderWindow_isOpen(App))
  Aby nie tworzyć w każdej iteracji pętli nowych obiektów i nie przypisywać im za każdym razem tych samych parametrów (co bardzo by spowolniło działanie pętli), to zostały stworzone one raz i tylko raz zostały im nadane stałe, niezmienne parametry (w funkcji **int main()**).
  Niby można by było wywołać pętlę
  while(sfRenderWindow_isOpen(App)) kilka razy, jednak czasami skutkuje to dziwnymi nieoczekiwanymi zmianami w działaniu programu.

Projekt w całości wykonany przez: Konrad Cielecki