COZY GAME FOR COZY EVENINGS

Fojt Sara
Hapunik Jan
Kiełbus Mateusz
Kubiszyn Alicja

SPIS TREŚCI

Opis problemu

Inspiracja

Projekt zakładał stworzenie prostej gry polegającej na zbieraniu punktów w minigrach. Wspólnie stwierdziliśmy, że chcielibyśmy stworzyć coś innego niż popularne “platformówki”. Stąd pojawił się pomysł gry wzorowanej na znanej pozycji “Animal Crossing”.

Początek pracy

Naszym pierwszym założeniem było stworzenie mapy, po której będzie przemieszczał się gracz, robi to za pomocą klawiszy “WASD”. Mapa ta zawiera również przeszkody, na które gracz nie ma wstępu. Dodatkowo umieszczone zostały na niej różne ozdobniki, takie jak przemieszczające się kurczaki (które zatrzymują się, gdy użytkownik się do nich zbliży oraz wydają dźwięki) lub grająca w tle muzyka.

Główna część

Na mapie znajdują się również tzw. NPC, czyli postacie, z którymi gracz może wchodzić w interakcje. Gdy wejdzie on na to samo miejsce, w którym stoi postać NPC pojawia się okno rozmowy. Zaprasza ono do zagrania w minigrę, na co można odpowiedzieć wybraniem odpowiedniego znaku na klawiaturze.

Minigry

Clicking Game

Gra opiera się na klikaniu myszką w pojawiające się na ekranie owoce. Należy klikać tylko w owoce świeże - za kliknięcie w zgniły owoc jest zabierany punkt życia. Punkt życia jest również zabierany, kiedy dobry owoc wypadnie poza okno gry. Owoce pojawiają się w różnych rozmiarach i z biegiem czasu zwiększają szybkość spadania.

Jumping Game

Gra ta polega na skakaniu po platformach i nie spadnięciu na dół okna. Postać w grze to kurczak i poruszamy nim klawiszami “a” i “d”. Gdy gracz dotknie jednej z platform zwiększa się jego wynik gry oraz skacze on w górę. Okno gry przesuwa się w dół proporcjonalnie do gracza poruszającego się w górę. Platformy generowane są losowo, gdy platforma dosięgnie końca mapy na dole, pojawia się na losowej szerokości za górną krawędzią okna gry. Na samym początku gry jest ona zamrożona do czasu kliknięcia “a” lub “d”.

Destroy Block Game

Gra polega na zniszczeniu wszystkich bloczków i niedopuszczeniu do spadnięcia piłki poza dolną granicę mapy. Do odbijania piłeczki służy paletka poruszająca się prawo-lewo za pomocą strzałek. Gra jest szerzej znana pod nazwą Arkanoid.

Snake Game

Gra która polega na budowaniu coraz dłuższego węża. W naszej grze głową węża jest gracz, a jego wydłużającym się ogonem - kolejne kurczaki, które gracz zbiera. Za każdego zebranego kurczaka gracz otrzymuje jeden punkt. Gra przerywa się, kiedy głowa węża uderzy w jego ogon. Gracz porusza się za pomocą klawiszy WSAD.

Struktura

Lista użytych klas

- Game
    - główna klasa gry, to właśnie za jej pomocą uruchamiamy całą grę
- Game_map

    - klasa mapy, która zawiera 16x16 mniejszych elementów (Game_square) - w tym te niedostępne dla gracza, obiekty klasy Chicken oraz NPC
- Game_square
    - klasa pojedynczego elementu mapy, zawierająca jego tekstury oraz informację o dostępności dla użytkownika
- Player
    - klasa gracza, która uwzględnia jego animację, ruch po mapie i kolizje z obiektami niedostępnymi
- Creature
    - klasa abstrakcyjna łącząca podklasy NPC i Chicken, zawiera wspólne metody podklas
- NPC
    - klasa postaci odsyłającej do minigier
- Chicken
    - klasa kurczaka, odpowiada za jego poruszanie się i animację oraz za zatrzymywanie się i wydawanie dźwięki, gdy zbliży się do niego gracz
- Exceptions
    - klasa wyjątków - opisuje błędy, które nasz program może napotkać

- Chickens
    - klasa potrzebna do gry w Snake'a, przechowuje pojedyncze kurczaki
- Menu
    - klasa uruchamiana na początku gry - daje do wyboru opcje play oraz exit

Opis najważniejszych klas

Game

Klasa sterująca, służy jako interfejs dla całej gry.
Jej najważniejsze parametry to:

- Player player;
    - obiekt gracza
- Game_map map;
    - obiekt mapy, na którą składają się m.in. obiekty Game_square
- std::vector<std::unique_ptr<Minigame>> minigames;
    - vector zawierający wskaźniki na minigry - później odwołujemy się do minigry poprzez index w vectorze

Najważniejsze metody:

- update_game()
    - metoda, która zawiera praktycznie całą mechanikę gry, na jej działanie składa się głównie wywoływanie kolejnych metod, ma wysoki poziom abstakcji. Jedyną rzeczą jaką głównie obsługuje w ciele metody to odtwarzanie muzyki w tle gry

Game_map

Klasa zawierająca mapę i wszystkie jej komponenty (oprócz gracza) oraz obsługująca ich działanie - dostępność, animacje, poruszanie się.
Najważniejsze parametry:

- std::map<unsigned int, std::map<unsigned int, Game_square>> squares_first;
    - pierwsza warstwa kwadratów tworzących mapę, wszystkie zawierają teksturę trawy
- std::map<unsigned int, std::map<unsigned int, Game_square>> squares_second;
    - druga warstwa kwadratów tworzących mapę, mogą zawierać różne tekstury, niektóre nie zawierają żadnych
- std::vector<std::unique_ptr<Chicken>> animals;
    - wektor zawierający w sobie zwierzęta (wskaźniki) poruszające się na mapie, w podstawowej wersji gry, tworzonej automatycznie są to 3 kurczaki, które animują się i poruszają w różne strony
- std::vector<std::unique_ptr<NPC>> npcs;
    - wektor zawierający w sobie postacie npc (wskaźniki do nich), są one również automatycznie tworzone na z góry założonych pozycjach

Najważniejsze metody:

- void set_up_initial_state();
    - ustawia i renderuje na odpowiednich (z góry założonych miejscach) zwierzęta i postaci NPC oraz wywołując metodę set_up_squares() renderuje kwadraty z odpowiednimi teksturami
- void update_game_map(sf::RenderTarget&);
    - nakłada zmiany na obiekty na mapie, które nastąpiły podczas przejścia pętli while (window.isOpen())

Player

Klasa ta odpowiada za wygląd, zachowanie i metody gracza.
Najważniejsze parametry:

- bool moving = false;
    - zawiera informację o tym czy gracz się porusza, jest potem ona wykorzystywana przy animacji
- parametry odpowiadające za teksturę i kształt
    - Sprite i Texture - > tworzą gracza, z długiej grafiki wycinają tylko fragment odpowiadający danej klatce animacji

Najważniejsze metody:

- void updateMovement(Game_map&);
    - jest wywoływana automatycznie. Od razu ustawia flagę moving na false oraz sprawdza, który klawisz (w, a, s czy d) został naciśnięty (jeśli żaden z nich to po prostu się kończy). Następnie porusza gracza o nieznaczną długość w odpowiednią stronę i wywołuje updateAnimations(). Na sam koniec sprawdza czy ten ruch nie spowodował kolizji ze zwierzęciem lub postacią NPC, jeśli tak to zatrzymuje zwierzę lub zaczyna rozmowę z postacią.
- void updateAnimations();
    - po upłynięciu odpowiedniego czasu zmienia klatkę animacji postaci na kolejną

- bool check_collision(Game_map&, float, float);
    - sprawdza czy gracz nie próbuje wejść na zabronione pole, jeśli tak, to zwraca wartość true, która blokuje ruch gracza (w metodzie updateMovement()).

Minigame

Klasa abstrakcyjna, która łączy wszystkie minigry. Posiada dwie metody, które dzieli z podklasami, są to:

- virtual void start(sf::RenderWindow&);
    - uruchamia minigrę
- virtual unsigned int get_score() const;
    - zwraca wynik minigry

Mniej ważne klasy

- Game_square
    - zawiera pozycję, tekstury oraz informację o tym, czy jest dostępna dla gracza
- Animal
    - NPC
        - podklasa klasy Animal, zawiera grafikę z rozmową wyświetlaną w oknie przed minigrą oraz funkcję talk(), która obsługuje rozpoczynanie rozmów i gier
    - Chicken
        - podklasa klasy Animal, posiada parametry i metody niezbędne do animowanego ruchu po określonym torze, posiada również flagę stop i metodę, które zatrzymują obiekt, gdy spotka się z graczem
- Exceptions
    - klasa zawierająca własne wyjątki, takie jak niezaładowanie tekstury lub ustawienie/zmiana pozycji na niewłaściwą (poza oknem). Są one później wykorzystywane do testów.
- Menu
    - okno tej klasy wita użytkownika oraz daje dwie opcje do wyboru - rozpoczęcie nowej gry lub wyjście, po opcjach przemieszcza się strzałkami, a wybiera się klawiszem Enter

Sposób testowania

Ze względu na fakt oparcia naszego projektu głównie na interfejsie graficznym, na jaki pozwala nam biblioteka SFML, testowanie nie było najprostszym zadaniem. Jednak wykonaliśmy testy jednostkowe do metod i klas, które nam na to pozwalały. Testowanie naszego projektu opiera się więc na:

- testach jednostkowych
    - stworzone przy pomocy catch2, testują konstruktory i niektóre funkcje
- testach metodą tradycyjną
    - testowaliśmy drobne fragmenty programu po kolei (przy ich tworzeniu), to złożyło się na względnie dobrze przetestowany finałowy program. Testy takie

polegały na uruchomieniu danego fragmentu programu i symulowaniu ruchów użytkownika.

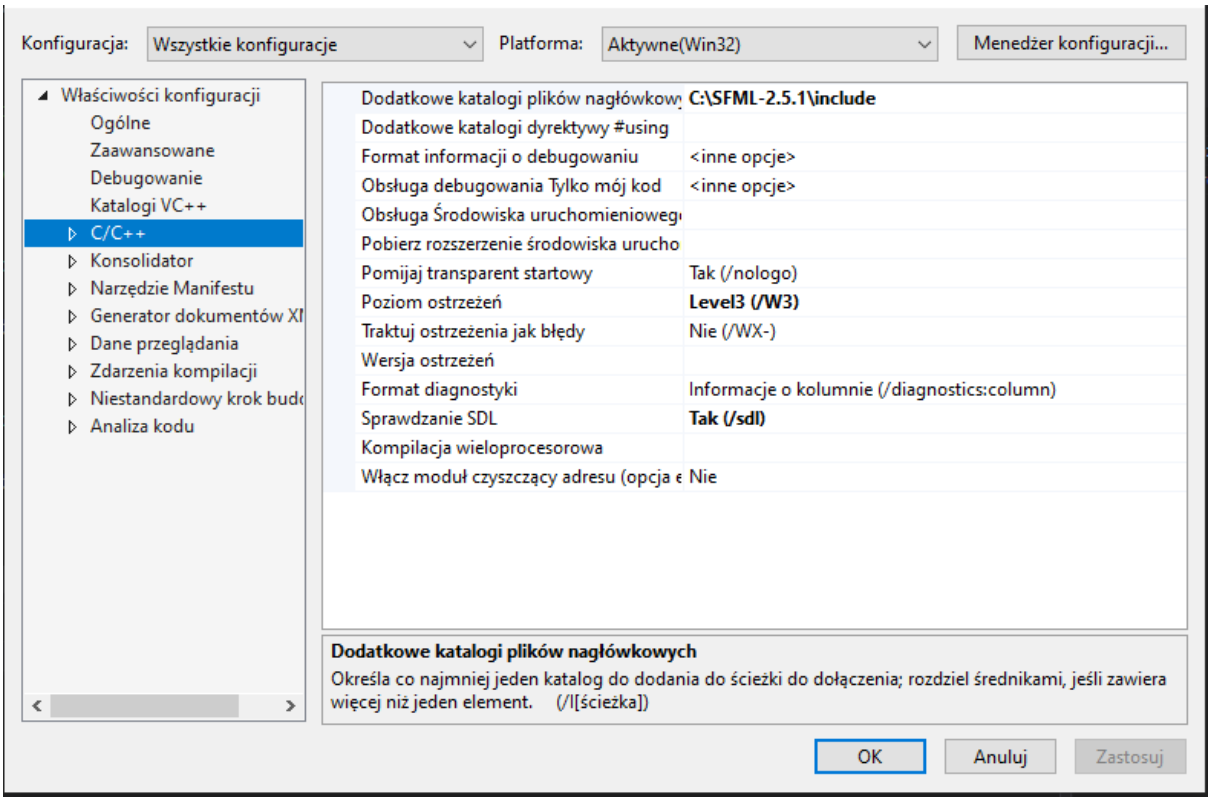
Sposób instalacji i obsługi

Instalacja

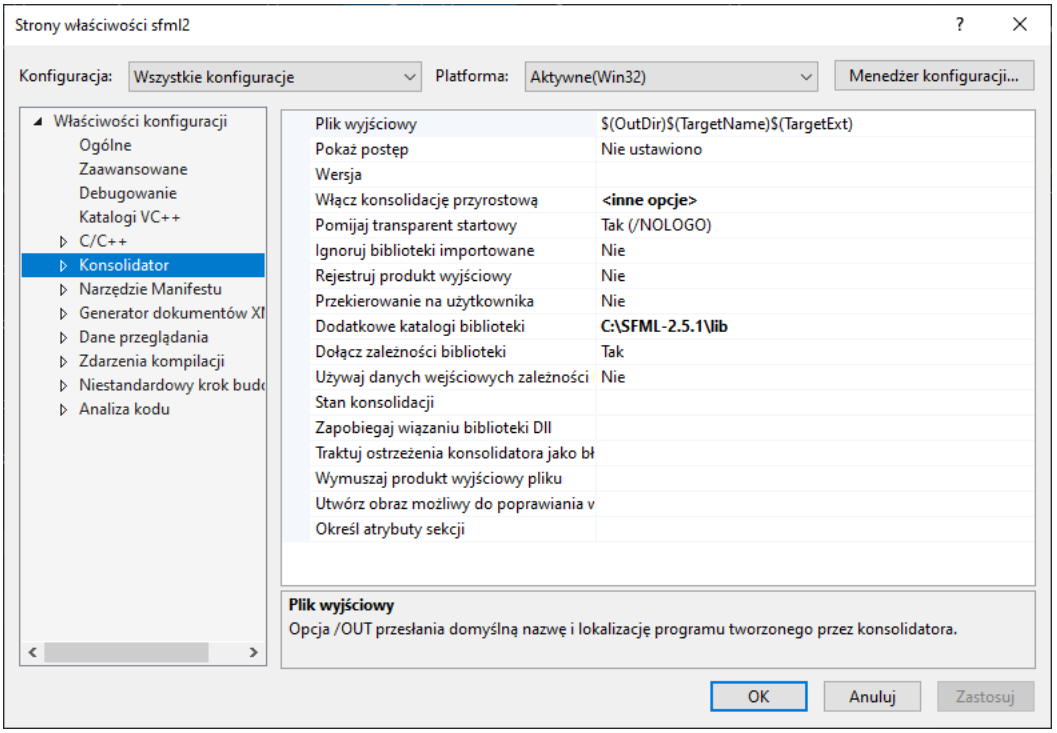
Instalacja polega na pobraniu ostatniego commita z gałęzi main w projekcie “proi projekt” na platformie gitlab. Należy zachować taki sam układ folderów - image, music i fonts. Potrzebne do działania programu są również biblioteki dynamiczne SFMLa, a do kompilacji - biblioteki statyczne SFMLa.

Konfiguracja SFML

Aby poprawnie skonfigurować SFML w środowisku Visual Studio należy wejść w Projekt > Właściwości… oraz w lewym górnym rogu wybrać opcję wszystkie konfiguracje. Należy wejść w zakładkę C/C++ i w pierwszym polu dodać ścieżkę do folderu include w sfml (najlepiej aby folder SFML był na dysku C)



Następnie wchodzimy w zakładkę konsolidator i w dodatkowych katalogach biblioteki wpisujemy ścieżkę do folderu lib w SFML

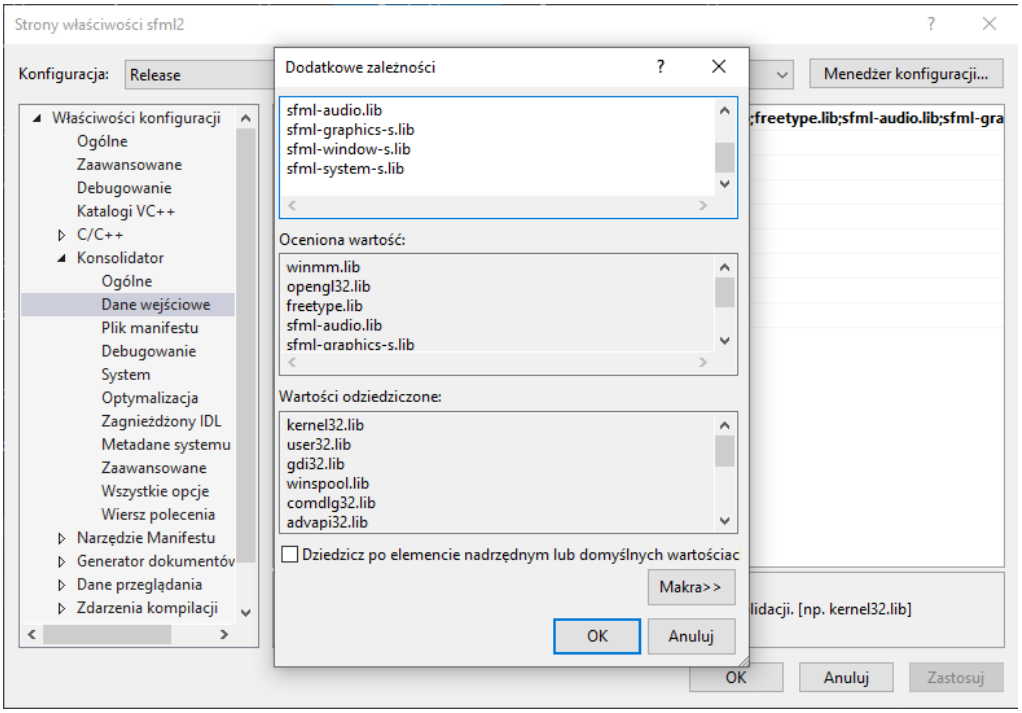


Następnie w podzakładce Konolidatora - dane wejściowe podajemy ścieżki do bibliotek sfml, które są wykorzystywane w naszym projekcie:

- sfml-audio.lib
- sfml-graphics.lib
- sfml-window.lib
- sfml-system.lib

Przechodzimy do opcji Debug zamiast Wszystkie konfiguracje i dopisujemy do dodanych bibliotek -d w następujący sposób

- sfml-audio-d.lib
- sfml-graphics-d.lib
- sfml-window-d.lib
- sfml-system-d.lib



Środowisko jest już skonfigurowane.

Obsługa

Aby uruchomić grę należy uruchomić plik main.cpp. Wita użytkownika oknem klasy Menu. Stwarza on również klasę Game, która obsługuje całą grę. Gracz może wybrać opcję play albo exit. Po wybraniu “play” pojawi się plansza z postaciami NPC (mniejsze), graczem (większy) i obiektami na mapie. Postać musi poruszać się dokładnie po kwadratach - nie ma to znaczenia, jeśli będzie się poruszać po dostępnych kwadratach, ale ma znaczenie, jeśli spróbuje wejść na niedozwolony kwadrat chociaż częścią.

Jeśli gracz wejdzie na kwadrat z postacią NPC pojawia się okno z propozycją gry. Można ją odrzucić (naciskając Esc) lub przyjąć (naciskając Enter);

Wykorzystane narzędzia

- Środowisko pracy - głównie Visual Studio 2019
- Canva - tworzenie grafik
- Cozy Asset - gotowe grafiki
- Biblioteka SFML - interfejs graficzny oraz duża część gry (okna, obiekty itp.)
- Catch2 - testy jednostkowe
- Biblioteka Audio YT - muzyka i dźwięki do gry
- Ogólnodostępne fonty

Ciekawostki

- dużym problemem było ustalenie rozmiaru okna głównej gry, nie można było też za każdym razem wpisywać nowej wartości pasującej do danego ekranu. Dodatkowo przy zmienianiu okna trzeba również przeskalować obiekty, które się w nim znajdują. Po wielu próbach i błędach udało nam się stworzyć okno wykorzystujące realny rozmiar naszego ekranu i dostosowuje się do niego. Dlatego nasza gra będzie wyglądać dobrze, zarówno w małym telefonie, jak i na dużym monitorze;
- wykorzystanie unique pointerów - wykorzystaliśmy unique_ptr w kilku miejscach w naszym kodzie. Z unique pointerami powiązane są kurczaki i postaci npc znajdujące się na mapie, ale również minigry. Oszczędza to pamięć (może w tej skali nie dużo), jednak wymagało od nas zastosowania funkcji restartGame(), która zeruje uzyskany wcześniej wynik i ustawia grę znów w stanie początkowym.

Znane błędy i problemy

- tworzenie się nowego okna przy rozmowie z postacią npc - przekazanie okna gry głównej do tak zagnieżdżonego elementu jakim była funkcja talk() było niezmiernie trudne i powodowało więcej problemów niż było warte
- problemy techniczne - awarie sprzętu, co znacznie utrudniało pracę
- inna wielkość okna JumpingMinigame - gra ta została stworzona w stałym wymiarze 400x533, co powoduje odpalenie się nowego okna. Nie wygląda to najbardziej estetycznie, ale ten sposób zachowuje estetykę gry. (z grą Destroy Block sytuacja wygląda podobnie)

Changelog

https://gitlab-stud.elka.pw.edu.pl/jhapunik/proi-projekt.git