UNIWERSYTET WROCŁAWSKI

WSTĘP DO PROGRAMOWANIA W JĘZYKU C

PROJEKT

Tron

Autor
Michał Maras

7 lutego 2019

1 Zasady gry

Tron to gra przeznaczona dla dwóch graczy. Na samym początku są oni rozstawieni symetrycznie na planszy. Gracze poruszają się na zmianę. W każdej kolejce muszą się przemieścić na jedno z sąsiednich pól. W momencie przesunięcia się na dane pole zostaje ono zablokowane do końca gry. Przegrywa ten gracz, który jako pierwszy przemieści się na zablokowane wcześniej pole.

2 Opis programu

2.1 Menu główne

Po uruchomieniu programu użytkownik zobaczy menu główne. Może on kliknąć w różne przyciski za pomocą myszy. Przycisk "Exit" wychodzi z programu. Natomiast po zaznaczeniu opcji "Start Game" gra pozwoli nam wybrać ustawienia naszej rozgrywki. Możemy tu wybrać:

- czy dany gracz będzie sterowany przez użytkownika, czy sztuczną inteligencję
- klawisze sterowania w przypadku użytkownika lub poziom trudności sztucznej inteligencji
- kolory graczy
- ilość wygranych rund koniecznych do wygrania rozgrywki

Po kliknięciu przycisku "Play" program przenosi użytkownika do właściwej rozgrywki.

2.2 Rozgrywka

W przypadku opcji sterowania przez sztuczną inteligencję dany gracz będzie sam wykonywał ruchy. W przeciwnym przypadku gracz będzie zawsze poruszał się na wprost. Po kliknięciu odpowiedniego przycisku przez użytkownika (strzałki w lewo lub w prawo, albo klawisze A lub D), gracz skręci we właściwą stronę.

3 Implementacja

Program podzielony jest na dwie sceny - scenę Main Menu oraz Game Screen. Main Menu odpowiada za wybór ustawień gry i włączenie nowej rozgrywki. Game Screen za to zajmuje się już samą grą.

3.1 defines.h

W tym pliku są wszystkie makra często używane w programie (np. wielkość planszy, rozdzielczość ekranu, kolory, itp.)

3.2 main.c

Znajdują się tu trzy funkcje:

- main() – wywołuje funkcje CreateWindow(), StartApp() (funkcja z application.c) oraz DestroyWindow()
- CreateWindow() – tworzy i ustawia opcje okna
- DestroyWindow() – zwalnia pamięć zajętą przez okno

3.3 application.c / application.h

Znajduje się tu jedna funkcja - StartApp(), odpowiadająca za główną pętlę programu. Inicjalizuje ona scenę Main Menu, a następnie odpowiada za ciągłe odświeżanie logiki obecnej sceny oraz jej rysowanie na ekranie. Jeśli w pewnym momencie okno programu zostanie zamknięte, pętla się przerywa.

3.4 screens.c / screens.h

W tych plikach znajduje się deklaracja struktury sceny. Posiada ona m.in. Game Manager'a (odpowiadającego głównie za ustawienia rozgrywki i rozmowę z użytkownikiem), wszystkie potrzebne zasoby (czcionki, tekstury, itp.) oraz UI (interfejs użytkownika - tekst, obrazki i przyciski). Zawiera też odpowiednie funkcje (a właściwie wskaźniki na funkcje w zależności od rodzaju sceny) odświeżające logikę danej sceny, rysujące ją na ekranie i zamykające ją.

Znajdują się tu funkcje:

- CreateNewScreen() – funkcja tworząca nową scenę i ustawiająca ją domyślnie na Main Menu.
- ChangeScreen() – funkcja zmieniająca obecną scenę na inną lub zamykająca ją. Podczas tego zwalniana jest również pamięć po wszystkich rzeczach używanych tylko w obecnej scenie - wszystkie tekstury, teksty, przyciski, itp. zostają zniszczone. W ten sposób zapobiegane są wszelkie wycieki pamięci.
- CheckLoadedResources() – funkcja sprawdzająca, czy wszystkie zasoby sceny zostały poprawnie stworzone.
- DestroyResources() – funkcja usuwająca wszystkie zasoby sceny

3.5 UI.c / UI.h

Te pliki odpowiadają za interfejs użytkownika. Całość interfejsu w każdej ze scen podzielona jest na warstwy. Każda z nich zawiera odpowiednie tekstury, tekst i przyciski. Dzięki temu możemy szybko chować lub pokazywać daną warstwę, np. w szybki i łatwy sposób ukryć i wyłączyć wszystkie przyciski znajdujące się na ekranie. Jest to szczególnie istotne w przypadku warstw, które powinny pojawić się tylko po jakimś

konkretnym wydarzeniu. Przykładem takiej warstwy jest wyskakująca na samym końcu rozgrywki informacja o zwycięzcy oraz przycisk do powrotu do menu głównego.

- struktura UI layer – jedna warstwa - zawiera wszystkie przyciski, tekst i tekstury do niej należące
- funkcja DestroyLayer() – zwalnia pamięć po danej warstwie
- struktura UI button – przycisk - zawiera informacje o swoim położeniu oraz czy użytkownik obecnie na niego najeżdża lub klika
- struktura UI image – obrazek - zawiera obraz oraz swoją pozycję
- struktura UI text – tekst - zawiera informacje o swoim położeniu na ekranie, tekst, który powinien wyświetlać oraz czcionkę i kolor.

3.6 board.h

Zawiera strukturę Board reprezentującą planszę gry.

3.7 gameManager.c / gameManager.h

Pliki te zawierają strukture Game Manager'a i wszystkie funkcje z nią związane. Game Manager posiada wszystkie informacje o grze (planszy, graczach, wyniku, obecnej rundzie, itp.). Ponadto zajmuje się ustawieniem i działaniem samej rozgrywki oraz wymianie informacji z użytkownikiem (pobieranie wejścia).

Funkcje w tych plikach to:

- ResetGame() – resetuje grę - ustawia na nowo planszę, generuje nowe pozycje początkowe graczy, itp.
- UpdateGame() – odświeża grę - uruchamia funkcje CheckInput(), MakeTurn(), oraz sprawdza warunki zakończenia gry
- CheckInput() – sprawdza, czy użytkownik wcisnął przycisk odpowiadający za skręcanie
- MakeTurn() – wykonuje turę gracza - odpowiednio przesuwa gracza, jeśli uderzy on w ścianę to zwraca informację o przegranej rundzie

3.8 players.c / players.h

Zawierają strukturę player reprezentującą gracza. Posiada ona informacje: czy dany gracz jest sterowany przez sztuczną inteligencję czy użytkownika, które przyciski odpowiadają za skręcanie gracza, obecne jego położenie na planszy, następny ruch, który gracz zamierza wykonać, poziom trudności sztucznej inteligencji oraz kolor gracza.

Posiada również trzy funkcje odpowiadające za poruszanie się gracza - sterowanie użytkownika (getHumanInput()), łatwej sztucznej inteligencji (getAIInputEasy()) i trudnej sztucznej inteligencji (getAIInputHard()).

3.9 AI.c / AI.h

Sztuczna inteligencja w grze jest podzielona na dwa rodzaje - łatwą i trudną. Łatwa polega na, jeśli jest to możliwe, zbliżaniu się do przeciwnika.

W przypadku trudnej następny ruch przewidywany jest w nieco bardziej skomplikowany sposób. Używany jest do tego rekurencyjny algorytm minimax. Generuje on drzewo wszystkich możliwych stanów w grze do kilku tur do przodu. Dla każdego z liści drzewa wyznacza odpowiednią wagę (im ruch jest bardziej korzystny dla gracza obecnie wykonującego ruch, tym waga jest większa). Następnie dla każdego wierzchołka nie będącego liściem, jeśli następny ruch w tym stanie gry należy do gracza sterowanego sztuczną inteligencją to wybierana jest wartość największa spośród jego dzieci. W przeciwnym przypadku wybierana jest wartość najmniejsza. Gracz wykonuje ruch odpowiadający największej wartości wybranej przez korzeń drzewa. Do wyznaczenia wagi danego stanu używana jest funkcja heurystyczna. Za pomocą algorytmu BFS rozdziela ona planszę na dwie części - pola, do których obecny gracz dostanie się przed przeciwnikiem i na odwrót. Waga danego stanu to $(200 * (V_m - V_o) + 55 * (E_m - E_o))$, gdzie V_m, V_o to ilość pól, do których szybciej dotrze kolejno gracz wykonujący ruch i przeciwnik, a E_m, E_o to ilość dostępnych krawędzi znajdujących się w części gracza wykonującego ruch i przeciwnika.

W tych plikach znajdują się:

- struktury bfsQuery, Node i Queue oraz funkcje CreateNewQuery(), pushQueue() i popQueue() – struktury i funkcje pomocnicze do algorytmu BFS w celu wyznaczenia wagi stanu
- minimax() – rekurencyjna funkcja minimax opisana wyżej

3.10 MainMenu.c / MainMenu.h

W tych plikach znajdują się wszystkie funkcje inicjalizujące, odświeżające i rysujące warstwy UI wykorzystywane przez scenę Main Menu oraz funkcje tworzącą, aktualizującą, rysującą oraz niszczącą tą scenę.

3.11 GameScreen.c / GameScreen.h

W tych plikach znajdują się wszystkie funkcje inicjalizujące, odświeżające i rysujące warstwy UI wykorzystywane przez scenę Game Screen oraz funkcje tworzącą, aktualizującą, rysującą oraz niszczącą tą scenę.