3 VI 2024

Gra Autoslalom

Autoslalom to sowiecka gra elektroniczna typu LCD, wydana przez firmę Elektronika w latach 80. Gra polegała na sterowaniu samochodem, który poruszał się po torze pełnym przeszkód. Celem gracza było unikanie kolizji z przeszkodami i zdobywanie jak największej liczby punktów. Sterowanie odbywało się za pomocą dwóch przycisków umożliwiających ruch w lewo i w prawo. Przykładową rozgrywkę można zobaczyć pod linkiem

Rysunek 1: Poglądowy wygląd gry

1 Opis zasad gry

- cel gry: Gracz kontroluje samochód poruszający się po torze pełnym przeszkód. Celem będzie zdobywanie jak największej liczby punktów poprzez unikanie kolizji z przeszkodami i utrzymanie jak najdłuższego czasu jazdy.
- Sterowanie: Gra będzie obsługiwana za pomocą klawiatury, dwoma przyciskami - jeden do skręcania w lewo, drugi do skręcania w prawo. Gracz będzie musiał używać tych przycisków w odpowiednim momencie, aby manewrować samochodem i omijać przeszkody.
- Przeszkody: Na torze będą znajdowały się przeszkody w postaci barier. Kolizja z nimi kończy grę.
- Punkty: Gracz zdobywa punkty za bezkolizyjnie przejechany rząd przeszkód. Im dłużej uda się utrzymać samochód na trasie, tym więcej punktów się zdobędzie.
- Trudność: Gra stopniowo zwiększa swoją trudność poprzez zwiększanie prędkości samochodu co skutkuje szybszym przybliżaniem się przeszkód do samochodu.
- Koniec gry: Gra kończy się w momencie, gdy samochód uderzy w przeszkodę lub gracz osiągnie 999 punktów.

2 Opis projektu

Wykorzystując komponenty biblioteki do budowy graficznych interfejsów użytkownika SWING zaimplementuj autorską wersję gry Autoslalom, przyjmując następujące założenia:

- logikę gry i reprezentację planszy gry należy umieścić w pakiecie `p02.game` jako klasę `Board` implementującą jednocześnie interfejs `KeyListener`. Wciskanie klawiszy sterujących w aplikacji będzie obsługiwane przez bezpośrednią implementację metod;

- plansza gry będzie reprezentowana jako jednowymiarowa tablica zmiennych `int` o rozmiarze 7 elementów, gdzie wiersz o indeksie 0 reprezentuje pozycję samochodu, a pozostałe pola opisują przeszkody na torze;

- przeszkody na torze będą generowane losowo, przyjmując że pomiędzy wierszami z przeszkodami będzie tyle pustych wierszy ile jest obiektów klasy `SevenSegmentDigit` reprezentujących cyfrę 0. Znaczy to że jeżeli reprezentowana wartość to 000 wówczas pojawi się jeden wiersz z przeszkodami na każde cztery zdarzenia `TickEvent`. Natomiast gdy reprezentowana wartość to $00X$ wówczas pojawi się jeden wiersz z przeszkodami na każde trzy zdarzenia `TickEvent`. Z kolei gdy reprezentowana wartość to $0XX$ wówczas pojawi się jeden wiersz z przeszkodami na każde dwa zdarzenia `TickEvent`. itd;

- przeszkody na torze będą generowane losowo, należy jednak zapewnić grywalność, a co zatem idzie:

  - nie mogą wystąpić 3 sąsiadujące ze sobą przeszkody, np.

    ```
    = = =
    ␣ ␣ ␣
    ```

  - w dwóch kolejnych rzędach nie mogą wystąpić dwa elementy na tej samej pozycji, np:

    ```
    ␣ = =
    ␣ = ␣
    ```

- detekcja kolizji pomiędzy przeszkodą a samochodem, zostanie zrealizowana przez operację koniunkcji bitowej pomiędzy wartością wierszy o indeksach 0 i 1. W przypadku stwierdzenia kolizji nastąpi wygenerowanie zdarzenia `ResetEvent` i poinformowanie o tym fakcie obiektów nasłuchujących;

- część graficzna gry będzie umieszczona w pakiecie `p02.pres`

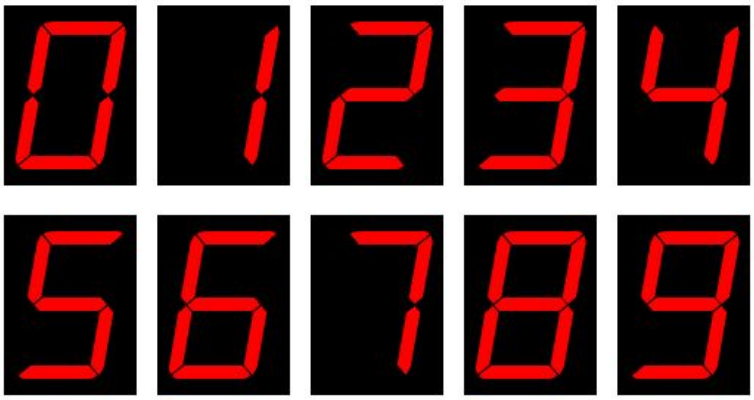
- na część graficzną składają się:

  - komponent `JTable` odpowiadający za reprezentację toru jazdy wraz z poboczem (czarne elementy ekranu, patrz rysunek 1). Należy dołożyć wszelkiej staranności aby komponent ten implementował wzorzec MVC, ze szczególnym uwzględnieniem nie odświeżania tych komórek tabeli, których wartość w modelu danych nie uległa zmianie. Należy również zwrócić szczególną uwagę aby operacje (np. zmiana wartości komórki) były realizowane poprzez zdarzenia modelu MVC;
  - komponent graficzny `JPanel` z samodzielną implementacją metody `paintComponent(Graphics)` odpowiedzialny za wyrysowanie obrazka przedstawiającego tor i pobocze (czerwone i zielone elementy gry, patrz rysunek 1);
  - licznik punktów (lewy górny róg ekranu), zbudowany z 3 komponentów graficznych (reprezentujących odpowiednio od prawej do lewej rzędy jedności, dziesiątek i setek), z których każdy będzie realizowany przez własną implementację klasy `SevenSegmentDigit`. Obiekty tej klasy będą ze sobą sekwencyjnie powiązane, tak aby każdy następny nasłuchiwał na zdarzenia poprzedniego. Np. jeżeli obiekt klasy `SevenSegmentDigit` reprezentujący rząd jedności wygeneruje zdarzenie `PlusOneEvent` to obiekt reprezentujący rząd dziesiątek zwiększy swój stan o jeden, itd adekwatnie do kontekstu przychodzących zdarzeń;

- obsługa klawiatury będzie realizowana za pomocą implementacji interfejsu `KeyListener` i zakłada wykorzystanie klawiszy:
  - `s` - rozpoczęcie rozgrywki;
  - `a` - przesunięcie samochodu w lewo;
  - `d` - przesunięcie samochodu w prawo.

- klasa `SevenSegmentDigit` będzie klasą dziedziczącą po `JPanel` i przesłaniającą metodę `paintComponent(Graphics)`. Celem tej metody będzie własnoręczna wizualizacja pojedynczej cyfry (za pomocą metod klasy `Graphics`, wyłączając metody rysujące obrazki). Obiekt będzie odbierał i generował zdarzenia zgodnie z delegacyjnym modelem zdarzeń. Koniecznym zatem będzie implementacja mechanizmu generującego i przyjmującego następujące zdarzenia:
  - `StartEvent` - ustalający stan cyfry na 0 i generując zdarzenie `StartEvent` rozsyłane do wszystkich nasłuchujących komponentów;
  - `PlusOneEvent` - dodający kolejny punkt i zwiększający cyfrę o jeden. Jeżeli zwiększana cyfra będzie cyfrą 9 to powraca do reprezentowania cyfry 0, jednocześnie generując zdarzenie `PlusOneEvent` rozsyłane do wszystkich nasłuchujących komponentów;
  - `ResetEvent` - wyłączający wszystkie segmenty cyfry, jednocześnie generując zdarzenie `ResetEvent` rozsyłane do wszystkich nasłuchujących komponentów;

  Otrzymanie określonego zdarzenia będzie musiało prowadzić do odświeżenia oblicza tego komponentu. Przykład segmentowej cyfry przedstawiono na rysunku 2



Rysunek 2: Przykład siedmiosegmentowych cyfr

- uruchomienie rozgrywki następuje po wciśnięciu i puszczeniu klawisza `s`, skutkiem czego będzie uruchomienie lub wznowienie niezależnego od interfejsu graficznego wątku (realizowanego przez własną implementację klasy `Thread`). Celem tego wątku będzie:
  - cykliczne generowanie zdarzenia `TickEvent` i rozsyłanie go do wszystkich nasłuchujących, zgodnie z delegacyjnym modelem zdarzeń;
  - interwał pomiędzy cyklami wątku będzie się stopniowo zmniejszał (symulując przyspieszenie auta);
  - wątek zostanie wstrzymany, gdy:
    * obiekt klasy `SevenSegmentDigit` reprezentujący ostatnią (rząd setek) cyfrę wygeneruje zdarzenie `PlusOneEvent`;
    * samochód uderzy w przeszkodę i obiekt klasy wątku otrzyma zdarzenie `ResetEvent`;

    Niezależnie od przyczyny wstrzymania wątku interwał pomiędzy cyklami wątku powróci do wejściowej wartości.

  Wykorzystaj wzorzec programistyczny `Singleton`, aby ponowne lub wielokrotne kliknięcie przycisku `s` nie skutkowało uruchomieniem kolejnych wątków.

- w toku rozgrywki występują następujące zdarzenia:
  - `TickEvent`, który powiadamia klasę reprezentującą planszę iż należy przesunąć wiersze z przeszkodami;
  - przesuwanie wierszy w modelu danych może prowadzić do "kolizji" pomiędzy przeszkodą a samochodem. W takim przypadku należy wygenerować zdarzenie `ResetEvent`;
  - wciskanie klawiszy `a` i `d` będzie obsługiwane przez bezpośrednią implementację interfejsu `KeyListener` w klasie `Board`.