Sviluppo del software Formula 1 - RaceTrack Game

Camilletti Samuele

03/09/2023

Contents

1 Definizione dell'obiettivo

L'obiettivo del progetto è realizzare l'implementazione avanzata per il software Formula 1 (anche detto Racetrack).

2 Analisi

2.1 Individuazione degli attori

Dopo aver effettuato l'analisi della specifica del progetto, sono stati individuati i seguenti attori:

- **Macchina**
- **Pilota**
- **Tracciato**
- **Gara**
- **Vista tramite console**
- **Vista tramite GUI**

2.2 Individuazione delle responsabilità

Per ognuno degli attori citati emergono le seguenti responsabilità:
Macchina:

- Gestione della macchina e dei suoi spostamenti nello spazio.
- Gestione del caricamento da file di una o più macchine.

Pilota:

- Gestione delle informazioni del pilota.
- Gestione della prossima mossa da compiere (accelerare, frenare, etc.).
  Da cui emergono altre due responsabilità:
  - Gestione dell'acquisizione in input della mossa nel caso di pilota umano.
  - Gestione della generazione di una prossima mossa 'intelligente' nel caso di pilota bot.

Tracciato:

- Gestione dei bordi, traguardo e inizio del tracciato.
- Gestione della rappresentazione geometrica del tracciato nello spazio.
- Gestione del caricamento da file del tracciato.

Gara:

- Gestione della simulazione passo per passo di una gara. Gestione del regolamento di gara. Gestione della creazione di nuove gare.

Viste:

- Gestione di una vista realizzata in JavaFX.
- Gestione di una vista tramite console.
- Gestione di un controllore per regolare le interazioni tra le viste e modello (gara).

2.3 Assegnazione delle responsabilità

Le responsabilità descritte sono state implementate attraverso le seguenti interfacce e classi:

- Il comportamento di una macchina è definito dall'interfaccia **Car** che viene implementata dalla classe **RacetrackCar** la quale definisce il modo in cui una macchina si sposta in uno spazio vettoriale a due dimensioni e del suo stato. L'informazione di un punto nello spazio è definito dalla classe immutabile **Point** che verrà riutilizzata in composizione ad altri oggetti, per la rappresentazione di componenti geometrici.
- Il caricamento da file è definito dall'interfaccia **CarLoader** che viene implementata dalla classe **RacetrackCarLoader**.

Per il pilota:

- Il comportamento di un pilota è definito dall'interfaccia **Driver** che viene implementata dalla classe **RacetrackDriver**.
- La responsabilità di calcolare la prossima direzione è assegnata all'interfaccia sealed **InputResolver** per ottenere la direzione in cui "guidare" la macchina. L'interfaccia permette l'implementazione a due interfacce **HumanResolver** e **BotResolver** che a loro volta sono non-sealed. La classe BaseBot implementa BotResolver e fornisce un'intelligenza artificiale di base per il calcolo della prossima mossa. La classe InputLoader implementa HumanResolver e viene realizzata utilizzando una variabile condivisa che viene letta/scritta da due thread concorrenti (il thread di input, e il thread della simulazione di gara).
- Il caricamento da file è definito dall'interfaccia **DriverLoader** che viene implementata dalla classe **RacetrackDriverLoader**.

Per il tracciato:

- Il tracciato è definito dall'interfaccia **Track** che viene implementata dalla classe **Racetrack**. Il circuito nello spazio viene implementato utilizzando la seguente gerarchia di classi, che vanno ad assumersi la responsabilità di gestire il circuito dal punto di vista geometrico:
  - Segment: Definisce un segmento con un Point iniziale e uno finale.
  - Un interfaccia PolygonalShape per definire una linea spezzata composta più vertici (Points).
  - L'interfaccia viene implementata dalla classe PolygonalChain (linea spezzata) ed estesa dalla classe Polygon che introduce l'invariante di classe in cui l'ultimo vertice si collega con il primo formando una linea chiusa. La classe fornisce un metodo per verificare che un punto sia nella figura o meno basato su una variante dell'algoritmo di Ray Casting, e un metodo per verificare se un segmento si interseca con uno dei suoi lati.
- Il caricamento da file è definito dall'interfaccia **TrackLoader** che implementata dalla classe **RacetrackLoader**.

Per la simulazione:

- Il motore di gioco è definito dall'interfaccia **Race** che viene implementata dalla classe **RaceEngine**. Mantiene una lista di piloti, un tracciato, e uno stato.
- Per garantire l'estendibilità è state definita una interfaccia **RaceFactory** per la creazione di nuove Race.
- La responsabilità di definire quali mosse sono legali in una race è assegnata alla interfaccia RaceRule che viene implementata dalla classe RaceTrackRule. Questa classe viene utilizzato in composizione nel RaceEngine.

Per la vista:

- La responsabilità di gestire la vista JavaFX è assegnata alle classi **JavaFXView** e **JavaFXController**. JavaFX configura lo stage (leggendo il rispettivo file FXML) e i controller. Mentre il controller della javafx gestisce gli eventi che vengono eseguiti sulla GUI ed esegue le eventuali modifiche grafiche.
- La vista da console è gestita attraverso la classe **ConsoleView**.

- La classe **SetupController** mette a disposizione il modello (della simulazione di gara) alle viste e mette a disposizione i metodi per aggiornare il model. Tra cui il metodo per avviare una gara, eseguire uno step, inserire l'input per il driver di turno.

3 Guida all'uso

1. Eseguire gradle build e gradle run.
2. Avviata la GUI cliccare su 'Load Track' e selezionare uno dei circuiti di prova presenti in formato .csv denominati "DefaultTrackX". Se non dovesse aprirsi di default, il path in cui sono locati è api.src.main.resources.
3. Ripetere il passo 2 ma selezionare 'Load Cars' e un file denominato "DefaultCarsX" in cui X è lo stesso del passo 2.
4. Cliccare Start Race.
5. Proseguire cliccando Next Step per avanzare con i turni.
6. Nel caso fosse il turno di un player umano il keyPad in basso a destra si accenderà consentendo di inserire la mossa.
7. La gara termina quando una macchina taglia il traguardo (disegnato in verde) o tutte le macchine sono fuori dal circuito.

3.1 Diagramma UML