Relazione progetto di metodologie di programmazione Gennaio/Febbraio 2025

Filippo Fanesi
matricola 124132
Informatica (L-31)
consegnato il 09/02/2025

[Specifiche](#)
[Introduzione](#)
[Analisi delle Responsabilità delle Classi](#)
- [Package Model](#)
  - [Classi di Base](#)
  - [Struttura del Gioco](#)
  - [Giocatori](#)
- [Package View](#)
  - [Renderer](#)
- [Package Controller](#)
  - [GameEngine](#)
  - [AIDirector](#)
  - [InputHandler](#)

[Implementazione classi](#)
- [Package Controller](#)
  - [GameEngine](#)
  - [AIDirector](#)
  - [InputHandler](#)
- [Package Model](#)
  - [Vector2D](#)
  - [Car](#)
  - [Bot](#)
  - [Track](#)
- [Package View](#)
  - [Renderer](#)
  - [TrackElement](#)

[Istruzioni per l'uso](#)
- [Requisiti di Sistema](#)
- [Esecuzione del Gioco](#)
- [Configurazione del Tracciato](#)
- [Visualizzazione](#)
- [Note Importanti](#)

Specifiche

Formula 1 è un gioco di carta e matita che si gioca su un foglio di carta quadrettata, sul quale viene disegnato un circuito automobilistico di fantasia, con una linea di partenza e una linea di arrivo (che possono anche coincidere se il circuito è circolare). Il gioco simula una gara tenendo conto dell'inerzia dei veicoli (per esempio, è necessario frenare quando si affronta una curva). L'obiettivo del progetto è quello di realizzare le classi che permettano di gestire una gara dove giocatori interattivi (umani) e giocatori bot (programmati) concorrono per vincere la gara.

Introduzione

Il progetto implementa una versione digitale del gioco "Vector Rally", un gioco di corse che simula il movimento di veicoli su una griglia, tenendo conto dell'inerzia e della necessità di gestire attentamente la velocità nelle curve. L'obiettivo principale è stato quello di sviluppare un'architettura software robusta e flessibile che permetta di gestire partite tra giocatori umani e bot con diverse strategie di gioco.
L'implementazione segue rigorosamente il pattern architetturale Model-View-Controller (MVC), che ha permesso di ottenere una netta separazione delle responsabilità tra:

- **Model**: gestisce la logica di business e i dati del gioco, includendo le classi che rappresentano il tracciato (Track), i veicoli (Car), i giocatori (Player, Bot) e la fisica del movimento (Vector2D).
- **View**: si occupa della presentazione dei dati all'utente attraverso la classe Renderer, che fornisce una rappresentazione testuale del gioco sulla console.
- **Controller**: coordina l'interazione tra Model e View attraverso le classi GameEngine, AIDirector e InputHandler, gestendo il flusso del gioco e le azioni dei giocatori.

Questa organizzazione ha permesso di ottenere un codice modulare e facilmente estensibile, come dimostrato dall'implementazione di diverse strategie di gioco per i bot e dalla possibilità di aggiungere nuove funzionalità senza impattare le componenti esistenti.
Nel seguito della relazione verranno analizzate in dettaglio le scelte implementative, le responsabilità delle singole componenti e come queste cooperano per realizzare le funzionalità richieste dalle specifiche del progetto.

Analisi delle Responsabilità delle Classi

Package Model

Classi di Base

- **Vector2D**: Rappresenta le coordinate e i vettori nel gioco, gestendo le operazioni matematiche fondamentali (addizione, sottrazione, moltiplicazione) necessarie per il movimento.
- **TrackElement**: Enum che definisce i possibili elementi della pista (BOUNDARY, START, FINISH, TRACK, OCCUPIED, PLAYER, NEXTPROJECTION) con i relativi simboli di rappresentazione.

Struttura del Gioco

- **Track**: Gestisce la struttura del tracciato, caricando la configurazione da file e mantenendo lo stato della griglia di gioco. Responsabile di:
    - Caricamento della pista da file
    - Gestione della griglia di gioco
    - Verifica delle posizioni valide
    - Mantenimento delle linee di partenza/arrivo

- **Car**: Classe base che rappresenta un veicolo nel gioco, gestendo:
    - Posizione attuale
    - Vettore velocità
    - Calcolo della prossima posizione
    - Logica del movimento considerando l'inerzia

Giocatori

- **Player**: Gestisce il giocatore umano, elaborando gli input dell'utente e applicando le mosse al veicolo.
- **Bot**: Classe astratta che definisce il comportamento base dei bot, con due implementazioni:
    - Gestione della posizione e movimento
    - Implementazione di diverse strategie di gioco attraverso il metodo findNextMove()

Package View

Renderer

Responsabile della visualizzazione del gioco, incluso:

- Rendering della pista
- Visualizzazione delle posizioni dei veicoli
- Proiezione dei movimenti futuri

Package Controller

GameEngine

Nucleo del gioco che:
- Inizializza tutti i componenti
- Gestisce il loop principale del gioco
- Controlla le condizioni di vittoria/sconfitta
- Coordina l'interazione tra i vari componenti

AIDirector

Gestisce il comportamento dei bot:
- Coordina il movimento dei bot
- Aggiorna le posizioni dei bot
- Gestisce le collisioni

InputHandler

Gestisce l'input dell'utente:
- Lettura degli input da console
- Validazione degli input

Questa struttura rispetta il pattern MVC:

Model: Track, Car, Player, Bot, Vector2D, TrackElement
View: Renderer
Controller: GameEngine, AIDirector, InputHandler

La separazione delle responsabilità permette una facile estensione del codice, come l'aggiunta di nuove strategie per i bot o l'implementazione di diverse modalità di visualizzazione.

Implementazione classi

Package Controller

GameEngine

- constructor: Inizializza il gioco con il percorso del tracciato e l'input handler
- startGame: Carica il tracciato, inizializza renderer e crea i giocatori (bot e player)
- renderCarsPosition: Aggiorna la posizione di tutte le auto sulla griglia
- gameloop: Loop principale del gioco che controlla vittoria/sconfitta e aggiorna lo stato
- update: Aggiorna le posizioni dei veicoli e il display

AIDirector

- updatePlayerPosition: Aggiorna la posizione del giocatore per i bot
- moveBots: Calcola e esegue il movimento di ogni bot

InputHandler

- getIntInput: Legge e valida input numerici da console
- getStringInput: Legge input di testo da console
- cleanup: Chiude lo scanner di input

Package Model

Vector2D

- add/subtract: Operazioni vettoriali base
- multiply/divide: Operazioni scalari
- vectorEquals: Confronta due vettori
- getters/setters: Accesso alle coordinate

Car

- getPosition/getSpeed: Accesso allo stato del veicolo
- getNextPosition: Calcola la proiezione del prossimo movimento
- move: Esegue il movimento considerando l'inerzia

Bot

- findNextMove: Determina la prossima mossa usando algoritmi di pathfinding
- isValidMove: Verifica se una mossa è valida
- moveTo: Esegue il movimento verso una posizione target
- walkbackPath: Ricostruisce il percorso dal punto finale

Track

- loadTrack: Carica e interpreta il file del tracciato
- getTrackElement: Restituisce il tipo di elemento in una posizione
- getMaxCar/getCarQuantity: Gestione dei limiti dei veicoli
- getters: Accesso alle dimensioni e elementi del tracciato

Package View

Renderer

- printTrack: Visualizza lo stato corrente del tracciato
- setOccupied/setOccupiedByPlayer: Marca le posizioni occupate
- clearAllOccupiedSpaces: Resetta le posizioni occupate
- projectCarCenterPos: Visualizza la proiezione del movimento
- clearProjection: Rimuove la proiezione precedente

TrackElement

- getSymbol: Restituisce il carattere associato all'elemento
- fromChar: Converte un carattere nel corrispondente elemento

Istruzioni per l’uso

Requisiti di Sistema

Java Development Kit (JDK) installato
Gradle build tool installato

Esecuzione del Gioco

Aprire un terminale nella directory principale del progetto
Eseguire i seguenti comandi:

1. gradle build
2. gradle run

Configurazione del Tracciato

Il gioco utilizza file di testo per definire i tracciati. È necessario fornire il percorso (assoluto) del file quando è richiesto dal programma al primo prompt DOPO l’esecuzione, non è necessario passare alcun argomento quando si esegue l’applicativo. È possibile creare nuovi tracciati seguendo queste regole:

- il primo carattere denota il numero di giocatori
- Il tracciato deve essere rettangolare o quadrato
- Deve iniziare dalla prima riga del file
- Utilizzare i seguenti simboli:
  - “#” : Bordo/Fuori pista
  - “ ” (spazio) : Pista percorribile
  - “S” : Linea di partenza
  - “F” : Linea di arrivo
- la linea di partenza “S” deve avere spazio a sufficienza per ogni giocatore

è disponibile un esempio di file chiamato testTrack1.txt sotto le Resources del progetto

Visualizzazione

Il gioco mostra:

- P : Posizione del giocatore
- O : Posizione dei bot
- N : Proiezione del prossimo movimento
- La griglia viene aggiornata dopo ogni mossa

Note Importanti

Il movimento tiene conto dell'inerzia: la nuova posizione è influenzata dalla velocità precedente
Il gioco termina quando:

- Un giocatore raggiunge il traguardo
- Un giocatore si schianta contro un muro o esce dalla pista