UNIVERSITÀ DEGLI STUDI DI NAPOLI
"PARTHENOPE"
FACOLTÀ DI SCIENZE E TECNOLOGIE

CORSO DI LAUREA IN INFORMATICA

RELAZIONE PROGETTO PROGRAMMAZIONE III

GIOCO AUTO

DOCENTE
Angelo Ciaramella

CANDIDATO
Salvatore Capobianco

Anno Accademico 2020-2021

Indice

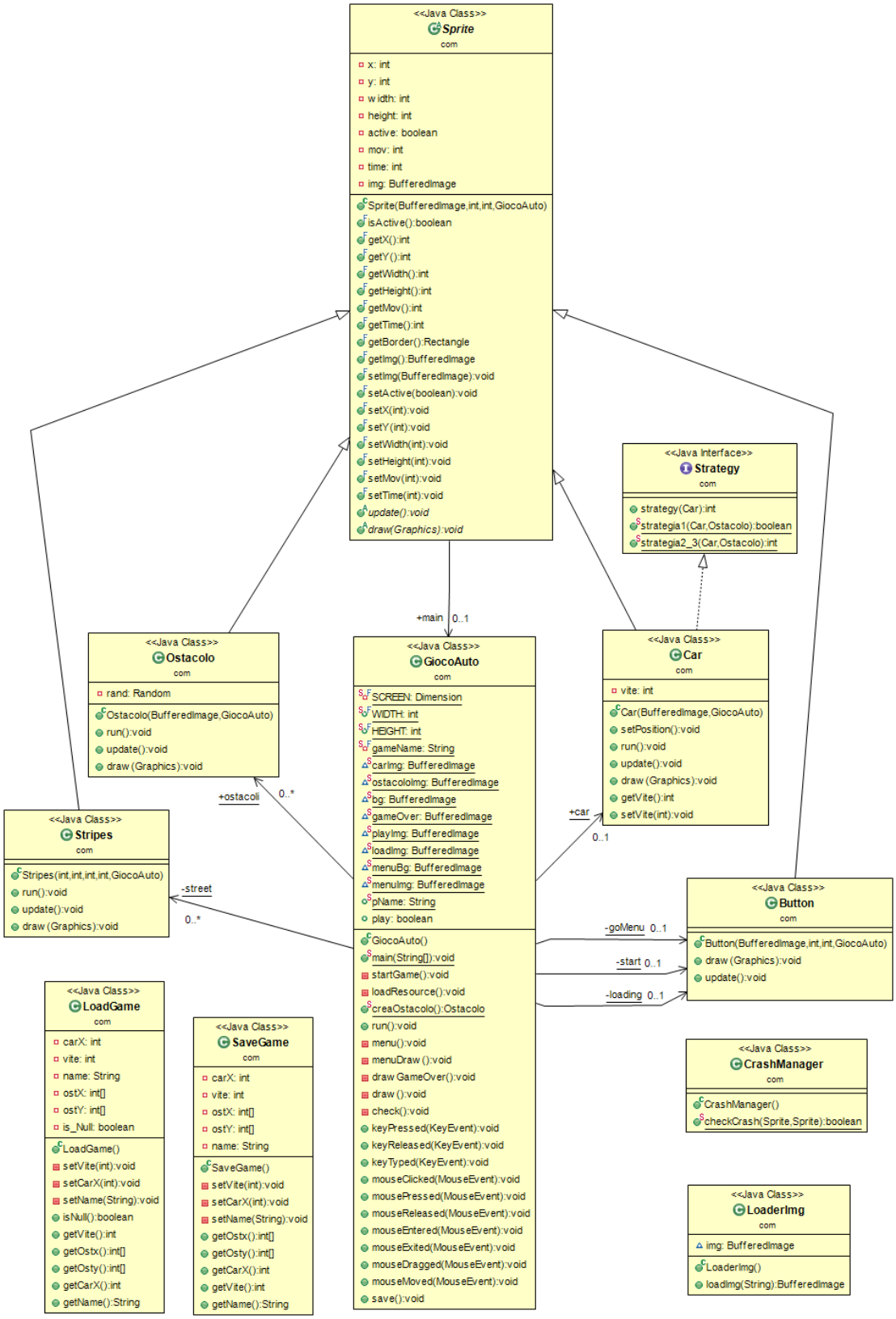
Requisiti

Si vuole realizzare un mini-gioco autonomo, il cui protagonista è un'auto che si muove lungo un percorso. Lungo di esso sono disposti casualmente e automaticamente degli ostacoli, che l'auto tenterà di evitare mettendo in pratica tre strategie di movimento. Ogni qualvolta l'auto scontra un ostacolo, cambia la strategia.

La partita termina quando l'auto "esaurisce le vite" scontrando tre ostacoli.

In qualsiasi momento è possibile mettere in pausa il gioco, salvare il suo stato e riprendere la partita successivamente, dal punto in cui era stata interrotta.

Diagramma UML delle classi



Esecuzione

All'avvio del gioco, viene visualizzato il menu principale che permette il caricamento di una partita precedente o l'inizio di una nuova:

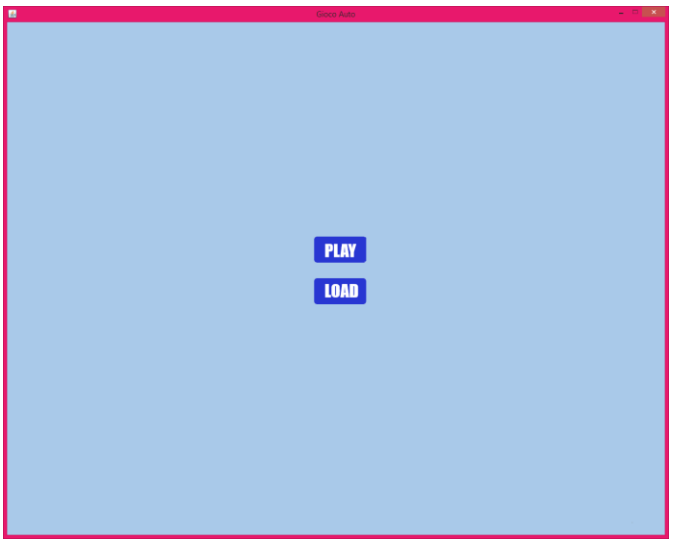


Figura 1: Menu iniziale

Cliccando sul bottone "PLAY", verrà richiesto il nome del giocatore ("PLAYER 1" di default se non viene specificato):

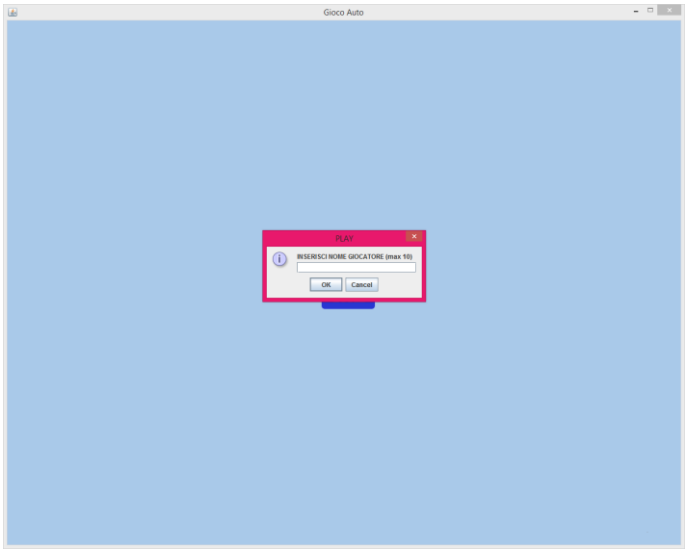


Figura 2: Scelta nome giocatore

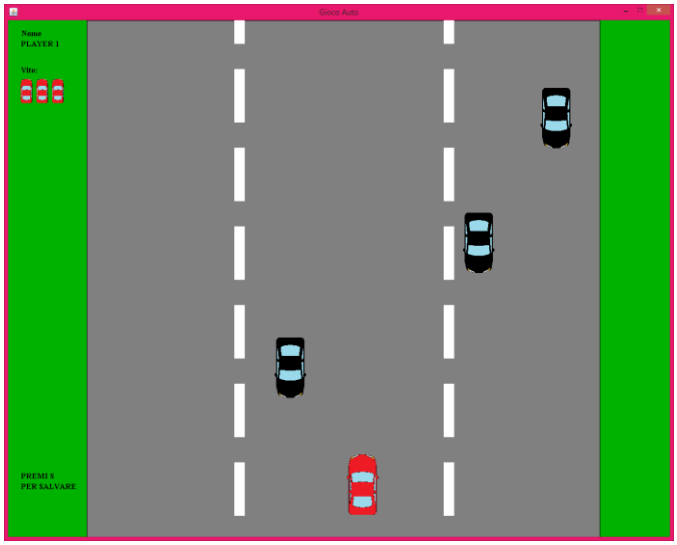


Figura 3: Partita avviata

Ad intervalli regolari, vengono creati ostacoli che compaiono dalla parte superiore della finestra; secondo la strategia corrente (indicata dal colore) l'auto metterà in atto dei comportamenti per evitarli.

Il gioco permette di salvare il suo stato, in qualsiasi momento della partita, semplicemente premendo il tasto "S" su tastiera. Un messaggio a schermo mostrerà l'esito del salvataggio, dopodiché, si ritorna al menu principale.

Figura 4: Salvataggio stato

Ogni qualvolta un ostacolo colpisce il giocatore, il gioco si riavvia e una vita viene sottratta; l'auto cambia colore, nonché comportamento (strategia) in base alle vite rimanenti.

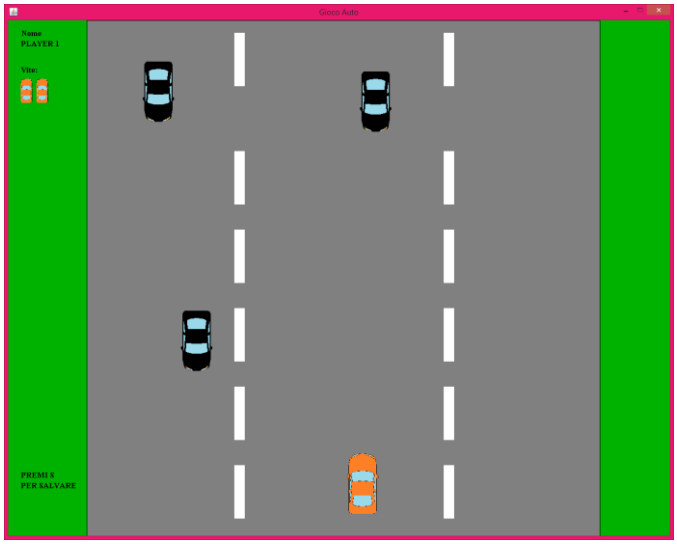


Figura 5: Due vite

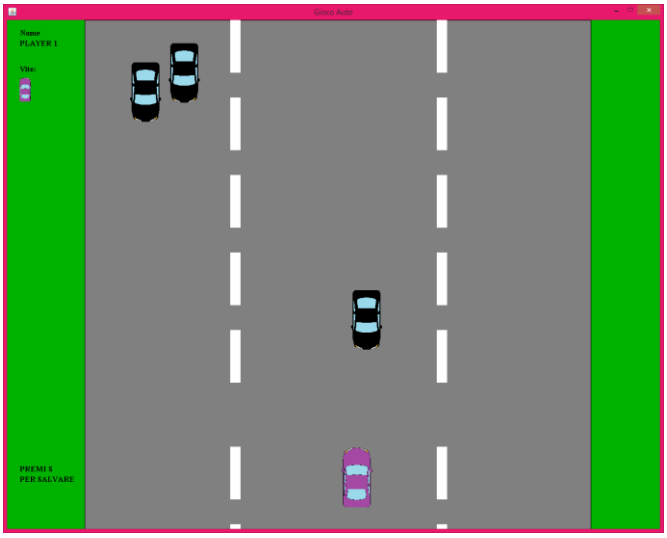


Figura 6: Una vita

Quando si esauriscono le vite, la partita termina e si può tornare al menu principale.

Figura 7: Game over

È possibile caricare, ripristinare lo stato precedente di una partita cliccando il pulsante “LOAD” nel menu principale.

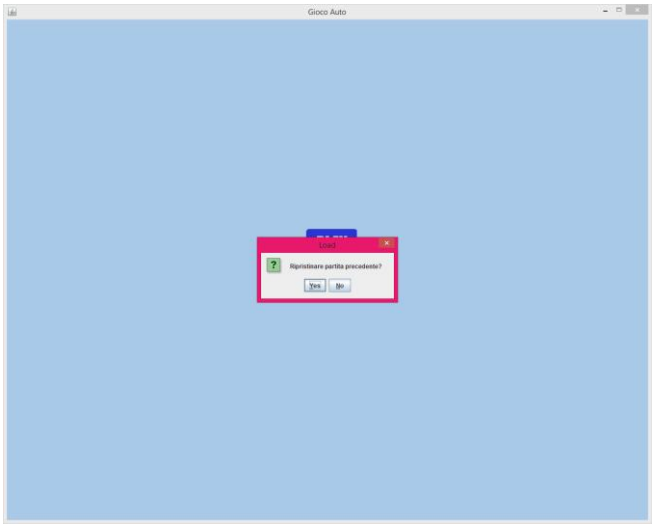


Figura 8: Caricamento

Confermando l’operazione, il gioco riprenderà dalla partita salvata in precedenza.

Codice

INTERFACCIA STRATEGY

L'interfaccia Strategy è la parte fondamentale e funzionale del giocatore che, permette di applicare dei comportamenti, al fine di evitare gli ostacoli autonomamente.

Il cuore di quest'interfaccia è il ciclo for (nel metodo di default strategy) che controlla se c'è almeno un ostacolo che si trova nella traiettoria del giocatore.

```
22      for(Ostacolo o : GiocoAuto.ostacoli) {
23        //Se si trova sulla sua traiettoria
24        Rectangle b = new Rectangle(o.getBorder().x,o.getBorder().y,
25            o.getBorder().width, car.main.getHeight()-o.getBorder().y);
26        if(car.getBorder().intersects(b)||car.getBorder().contains(b)) {
27          //Se non si muove
28          if(car.getMov()==0) {
29            //STRATEGIA 1 (Se si trova nel suo intorno)
30            if(strategia1(car,o) && (car.getVite()>0 && car.getVite()<=3)) {
31              Random rand=new Random();
32              int rnd;
33              do {
34                rnd=rand.nextInt(2+3)-2;
35              }while(rnd!=2 && rnd!=-2);
36              return rnd;
37            }else
38            //STRATEGIA 2 & 3 (destra e sinistra)
39            if(car.getVite()==3 || car.getVite()==2) {
40              switch(strategia2_3(car,o)) {
41                case 1:
42                    if(car.getVite()==3 || car.getVite()==2)
43                      return 2;
44                    break;
45                case 2:
46                    if(car.getVite()==3)
47                      return -2;
48                    break;
49                case 0:
50                    return 0;
51                default:
52                    break;
```

```
53                  }
54                }
55              }
56              //Se si trova ai bordi cambia movimento
57              else{
58                if(car.getX() >= GiocoAuto.WIDTH-250 || car.getX() <= 150)
59                  return(car.getMov()*-1);
60              }
61            }
62            else {
63              count++;
64            }
65          }
```

I metodi statici, “strategia1” e “strategia2_3” dell’interfaccia Strategy, sono le varie strategie da intraprendere, che sono:

1. Casualmente nel caso un ostacolo si trovi nel suo intorno
2. A sinistra se la maggioranza degli ostacoli si trovano alla sua destra
3. A destra se la maggioranza degli ostacoli si trovano alla sua sinistra

```
75  /**
76   * Controllo che un ostacolo si trovi nell'intorno del giocatore
77   * @param car Il giocatore
78   * @param o L'ostacolo
79   * @return true se si trova nel suo intorno false altrimenti
80   */
81  public static boolean strategia1(Car car, Ostacolo o) {
82    if(o.getY() >= GiocoAuto.HEIGHT-car.getHeight()*4)
83      return true;
84    return false;
85
86  }
```

```java
/**
 * Controllo della traiettoria di un ostacolo,
 * il giocatore si sposta a destra se ha la maggioranza degli
 * ostacoli alla sua sinistra,
 * altrimenti si sposta a sinistra.
 * @param car Il giocatore
 * @param o L'ostacolo
 * @return 1 se la maggioranza degli ostacoli si trova a sinistra,
 * 2 se la maggioranza e' a destra, 0 se non si trova nella traiettoria
 */
public static int strategia2_3(Car car, Ostacolo o) {
  if(o.getY() >= GiocoAuto.HEIGHT/4-o.getHeight()) {
    int sx=0,dx=0;
    for(Ostacolo ost : GiocoAuto.ostacoli) {
      if(ost.getX() < car.getX())
        sx++;
      else
        dx++;
    }
    if(sx >= dx) {return 1;}
    else {return 2;}
  }
  return 0;
}
```

METODO CHECK

Questo metodo privato (invocabile solo dal principale GiocoAuto) controlla eventuali collisioni tra l'auto e un ostacolo; in caso affermativo riduce le vite a disposizione e riavvia lo scenario.

Inoltre controlla la fuoriuscita degli ostacoli dallo schermo e nel caso, provvede a rimuoverli dal gioco.

```
377  /**
378   * Controllo collisioni o fuoriuscita dalla finestra degli ostacoli
379   */
380  private void check() {
381     // Controllo collisioni
382     for(Ostacolo o : ostacoli) {
383        // Collisione avvenuta
384        if(CrashManager.checkCrash(car,o)) {
385           // Pausa di qualche secondo
386           try {
387              Thread.sleep(1250);
388           } catch (InterruptedException e) {
389              // TODO Blocco catch generato automaticamente
390              e.printStackTrace();
391           }
392           // Elimina gli ostacoli presenti
393           ostacoli.clear();
394           // Decrementa le vite di 1 se possibile
395           if(car.getVite()>=1) {
396              car.setVite(car.getVite()-1);
397           }
398           // Altrimenti se esaurite, termina la partita
399           else {
400              car.setVite(0);
401              break;
402           }
403
404           // Riposiziona l'auto
405           car.setPosition();
406           break;
407        }
```

```
408      }
409
410      // Controllo fuoriuscita dallo schermo degli ostacoli e relativa rimozione
411      for(int i=0;i<ostacoli.size();i++) {
412        if(ostacoli.get(i).getY()>GiocoAuto.HEIGHT) {
413          ostacoli.remove(i);
414          break;
415        }
416      }
417    }
```

CLASSE CRASHMANAGER

Questa classe viene utilizzata durante l'esecuzione quando viene invocato il metodo "check". Ad essa è infatti delegato il controllo delle collisioni a basso livello: restituisce un valore booleano che indica se i due oggetti (Sprite) collidono oppure no.

```
2  /**
3   *
4   * Controllo collisioni tra auto e ostacoli
5   * @author Salvatore Capobianco
6   */
7  public class CrashManager {
8    /**
9     * Controlla l'intersezione di due rettangoli
10    * formati dal giocatore e l'ostacolo
11    * @param car il giocatore con le sue coordinate
12    * @param o l'ostacolo con le sue coordinate
13    * @return true se c'e' intersezione, false altrimenti
14    */
15   public static boolean checkCrash(Sprite car, Sprite o) {
16     //Confronta se i rettangoli si sovrappongono
17     return car.getBorder().intersects(o.getBorder());
18
19   }
20 }
```

METODO CREAOSTACOLO

Questo metodo statico, crea un oggetto di tipo Ostacolo controllando che non si sovrapponga con gli altri già presenti.

```
173  public static Ostacolo creaOstacolo(){
174    boolean ok;
175    Ostacolo temp;
176    do {
177      ok = true;
178      // Crea nuovo ostacolo
179      temp = new Ostacolo(ostacoloImg,null);
180      // Controlla tutti gli ostacoli presenti nella finestra della partita
181      for(Ostacolo o : ostacoli) {
182        // Rettangolo di traiettoria di temp
183        Rectangle b = new Rectangle(temp.getBorder().x,
184            temp.getBorder().y, temp.getBorder().width,
185            GiocoAuto.HEIGHT-temp.getBorder().y);
186        // Se si trova nella traiettoria di un altro ostacolo ripete la creazione
187        if(o.getBorder().intersects(b) || o.getBorder().contains(b)) {
188          ok = false;
189          break;
190        }
191      }
192    }while(!ok);
193    temp.start();
194    return temp;
195  }
196
```