

# Programmazione 2 – a.a. 2021-22

## Esercitazione Java: Un gioco di bocce

1. Descrizione
  2. Modello matematico (semplificato) del moto delle bocce
  3. L'interfaccia Gioco
- Cave canem
  - Cosa fare per adesso

# Descrizione

Campo di gioco rettangolare con sponde.

Da 1 a 12 bocce, tra cui il boccino.

Da 1 a 4 buche rotonde.

Il giocatore tira il boccino.

In assenza di ostacoli, la boccia in moto (inizialmente il boccino) prosegue in linea retta.

- Se il centro viene a trovarsi dentro una buca, cade in buca.
- Se urta contro una sponda, rimbalza indietro.
- Se urta un'altra boccia, si ferma e parte l'altra boccia.

In assenza di urti, la velocità diminuisce col tempo.

Il giocatore ha un numero massimo di tiri.

- Fine del gioco = boccino in buca oppure numero di tiri esaurito.
- Punteggio = numero di bocce, diverse dal boccino, finite in buca.


Simulazione discreta a scatti, con intervallo di tempo  $\Delta$  fissato (piccolo).

# Cosa viene dato, cosa bisogna fare

Dati : a) Interfaccia Gioco  
b) Visualizzatore grafico

Da fare:

- scrivere una classe che implementa l'interfaccia Gioco
- collegarla al visualizzatore
- avrete un gioco funzionante

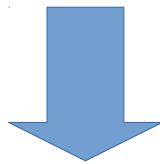


-classe principale che implementa Gioco  
-altre classi per i vari oggetti coinvolti...  
-relazioni di ereditarieta' e clientela...

Potete usare classi predefinite di Java e tutti gli strumenti presentati nel corso.

# Ordini di grandezza

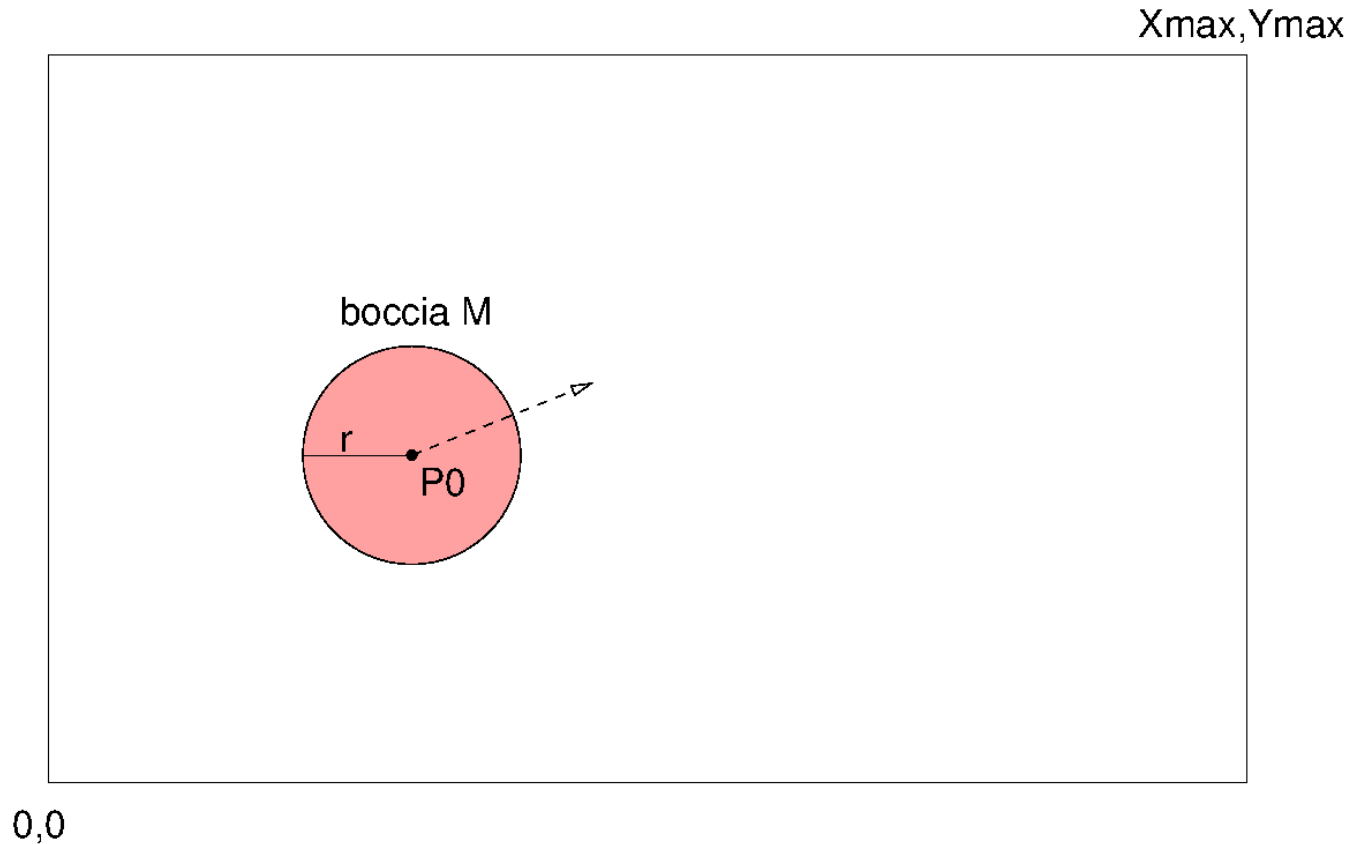
- larghezza e lunghezza del campo = tra 150 e 250 cm
- diametri delle bocce = tra 5 e 20 cm
- diametri delle buche = massimo 25 cm
- simulazione a scatti con  $\Delta = 20 \text{ msec}$  (= 0.02 secondi)
- velocità iniziale = massimo 50 cm/secondo
- decremento ad ogni scatto = 0.05 cm / secondo



Il campo accoglie comodamente tutte le bocce e le buche.

In uno scatto la boccia si sposta meno di un centimetro.

# Modello matematico (semplificato)



Posizione di una boccia = posizione del suo **centro**.

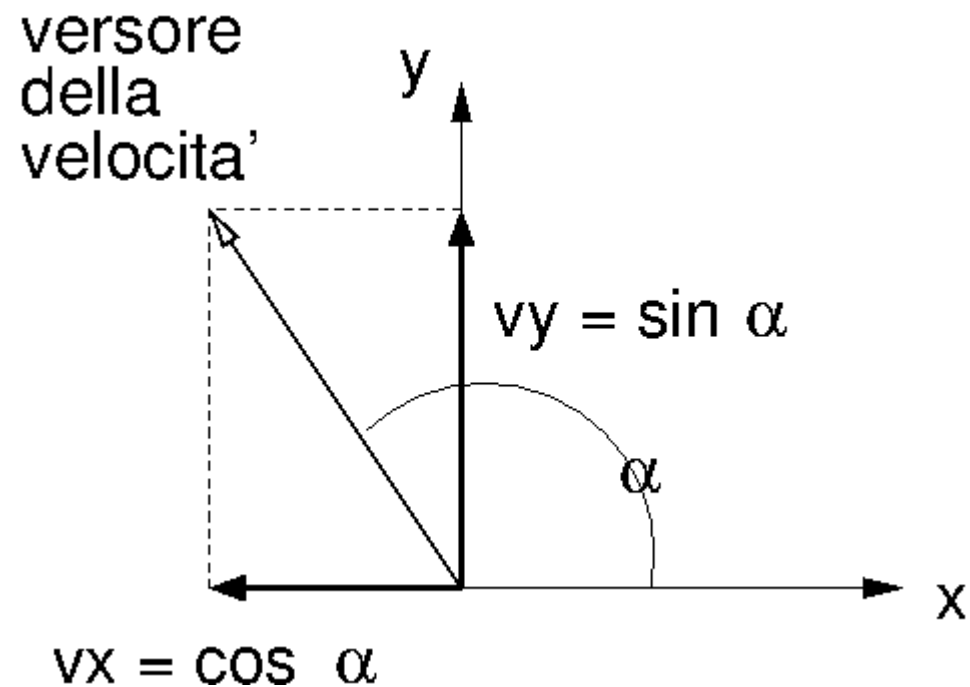
# Modello matematico (semplificato)

Velocita' della boccia = **modulo**  $|v|$  e **versore**  $(v_x, v_y)$ .

Viene fornita come modulo e angolo  $\alpha$  .

Le componenti del versore si ricavano:

- $v_x = \cos \alpha$
- $v_y = \sin \alpha$



# Durante uno scatto

La boccia passa da posizione P0 a posizione P1

$$P1 = P0 + \Delta |v| (v_x, v_y)$$

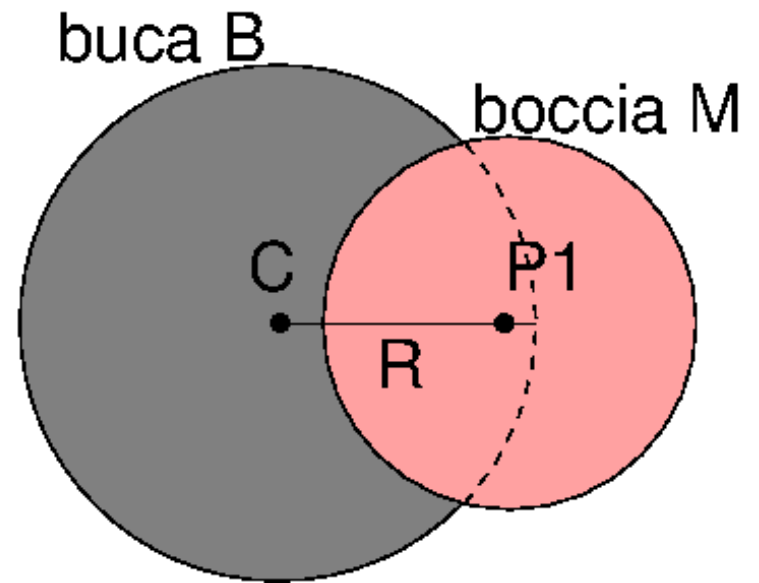
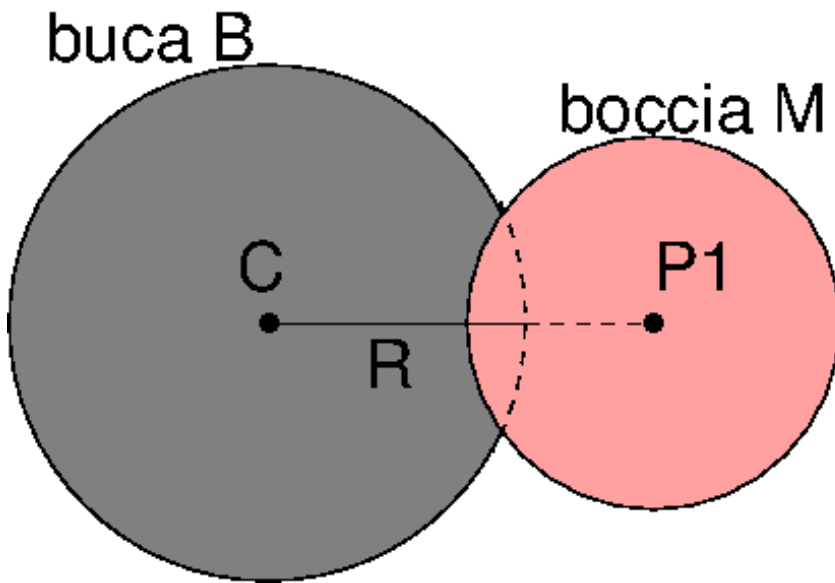
Nella nuova posizione potrebbe:

- cadere in buca
- urtare una sponda
- urtare un'altra boccia

# Cadere in buca

Se il centro P1 capita dentro la buca, cioè  
se P1 dista dal centro della buca meno del raggio della buca

$$\text{dist}(P1, C) \leq R$$

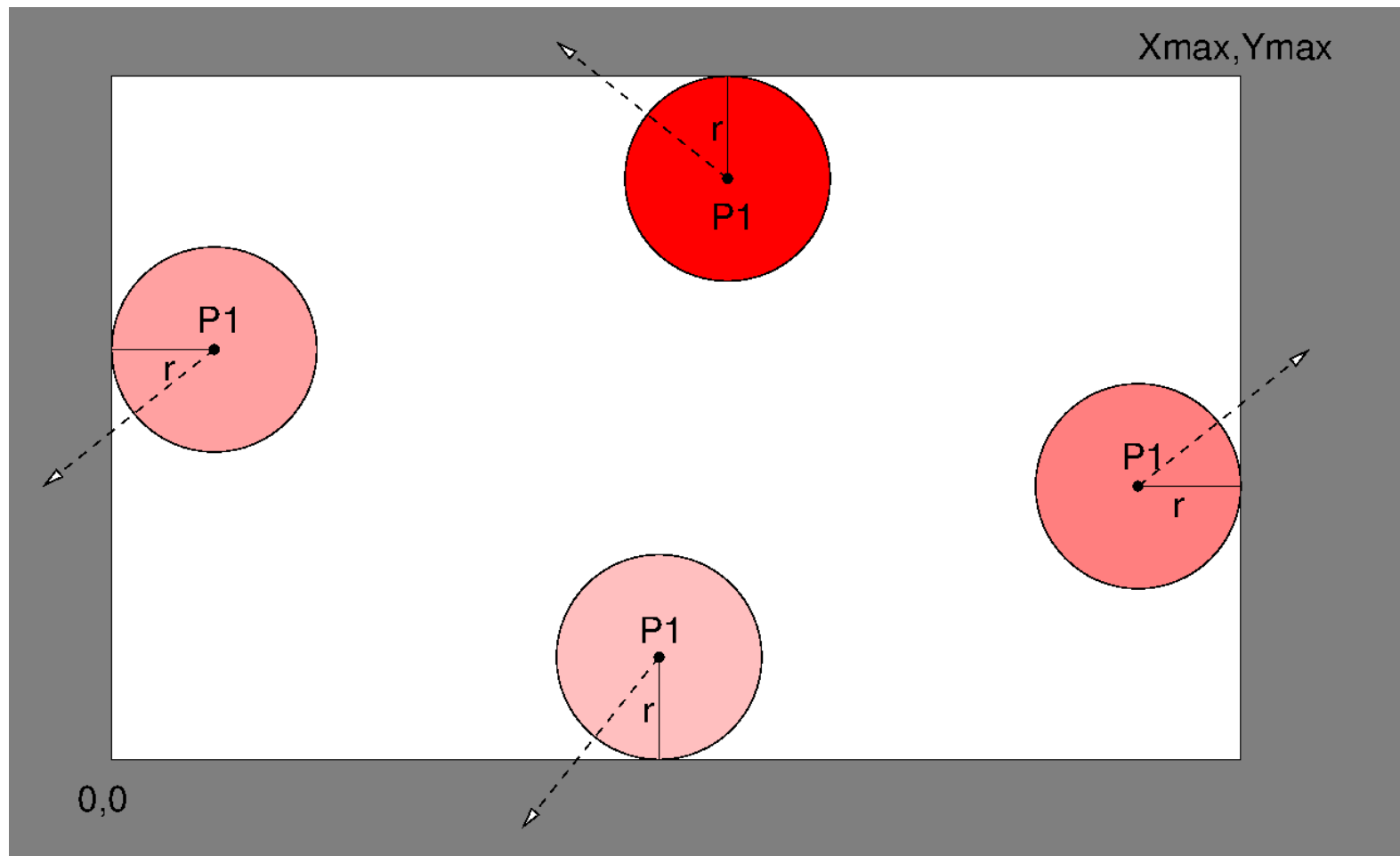


Se la boccia cade in buca, viene eliminata.



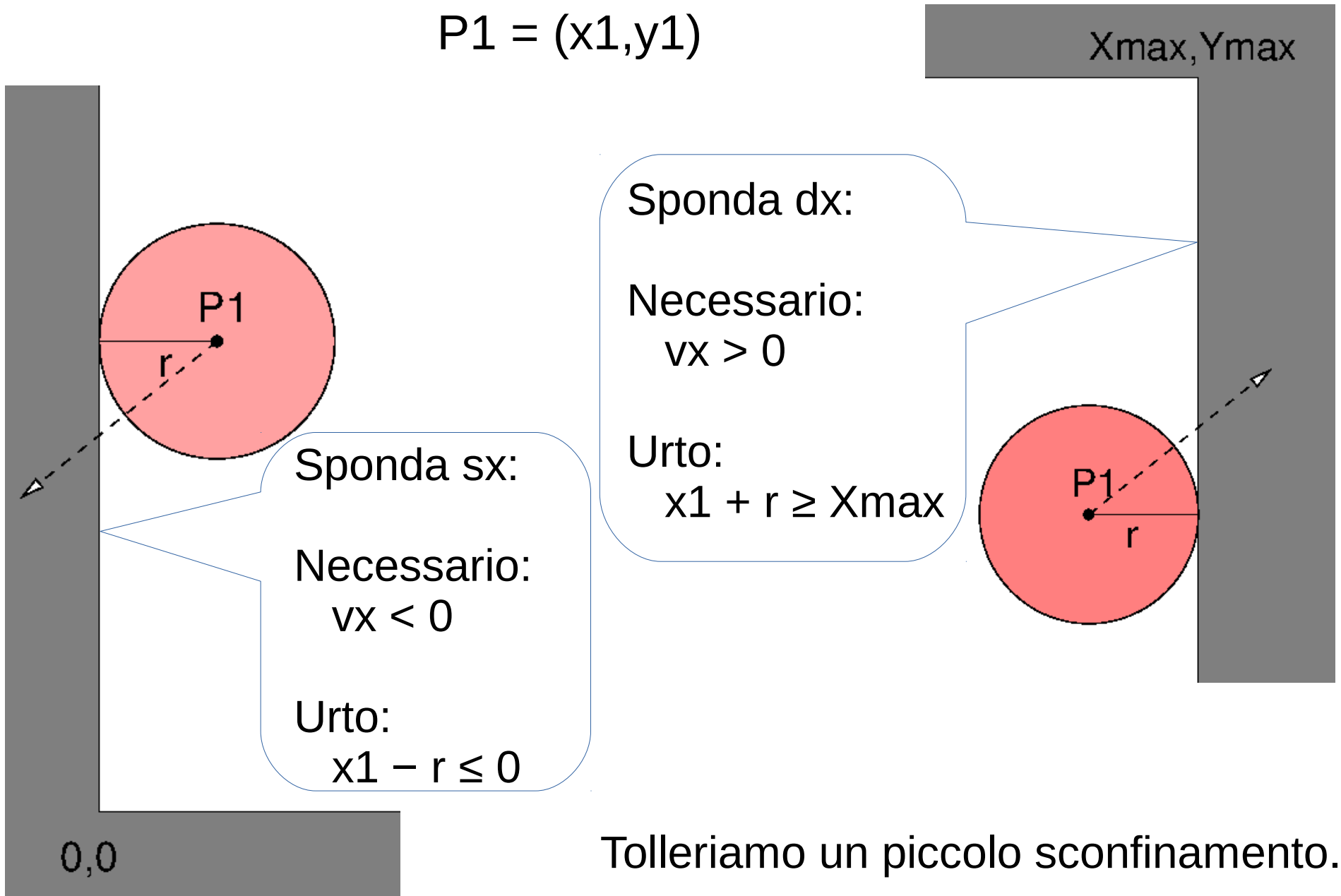
# Urtare una sponda

Se il centro P1 dista dalla sponda meno del raggio della boccia  
(le sponde sono parallele agli assi)



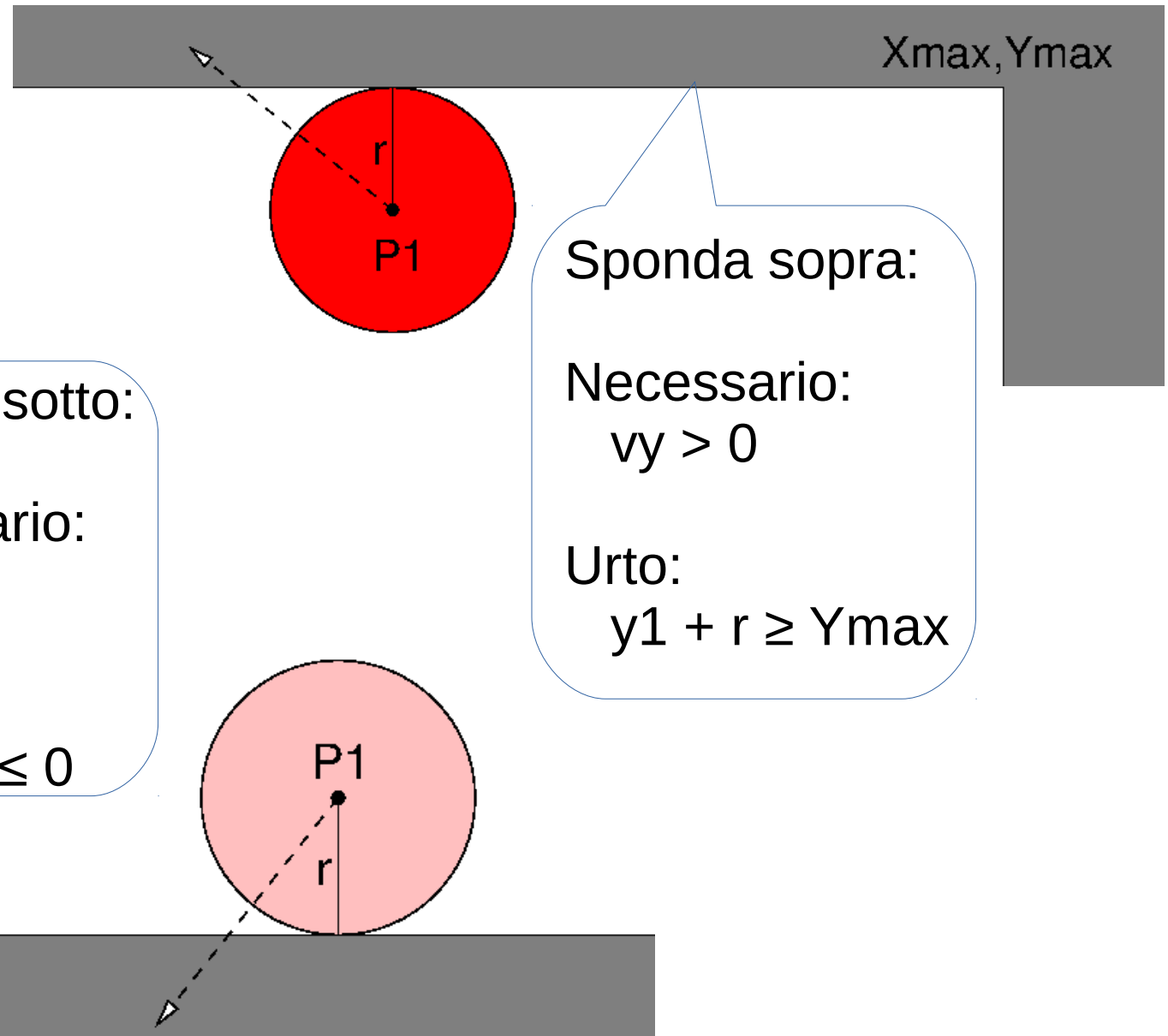
# Sponde verticali

$$P1 = (x1, y1)$$



# Sponde orizzontali

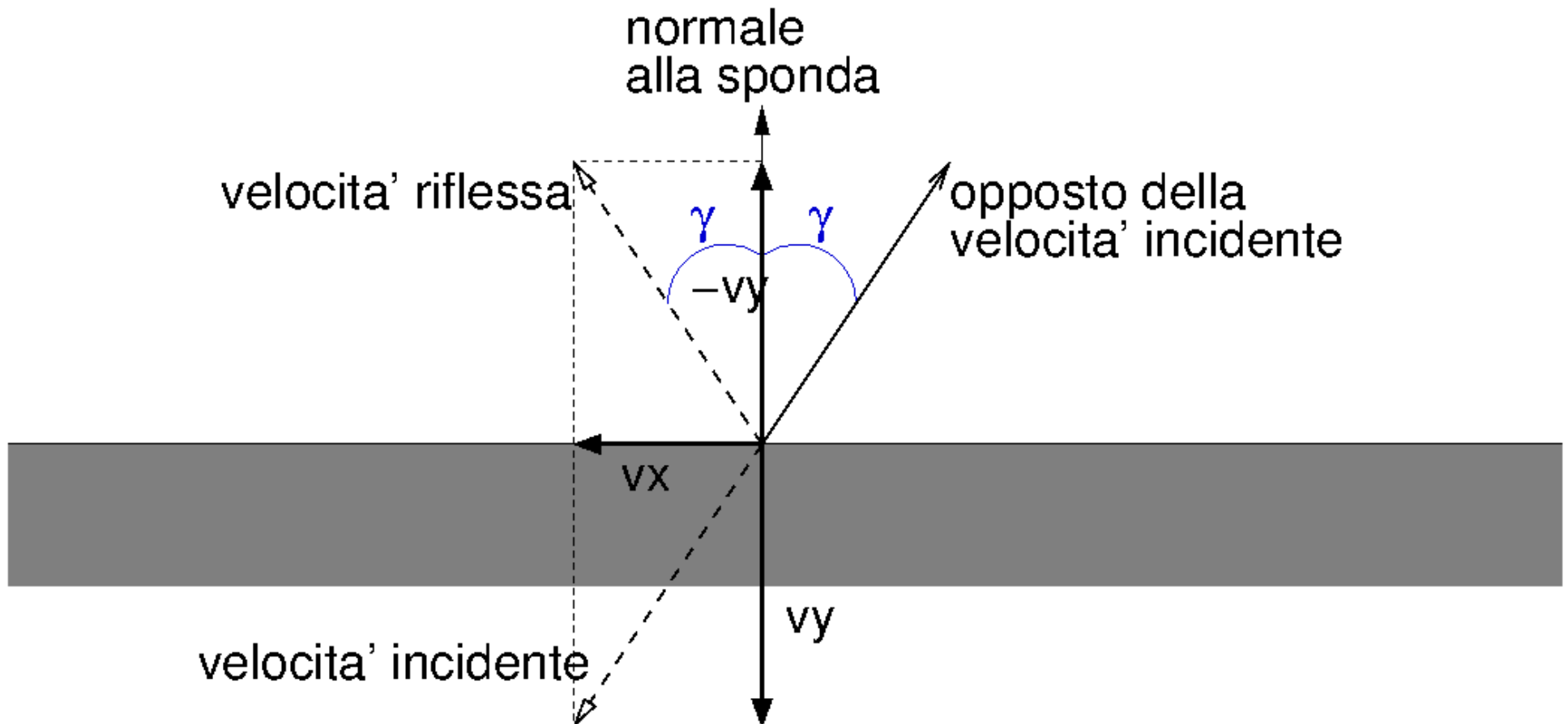
$$P1 = (x1, y1)$$



# Rimbalzo

Riflessione simmetrica rispetto alla normale alla sponda

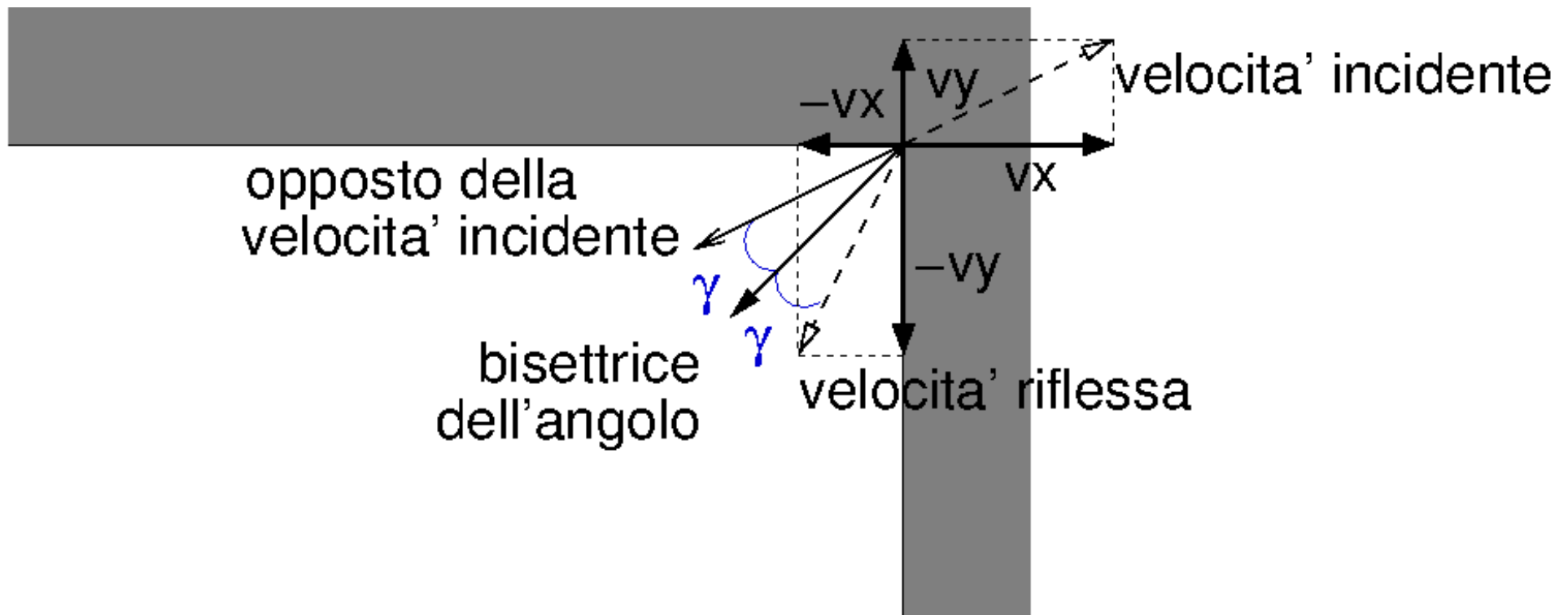
- Urto con sponda orizzontale: cambia il segno di  $v_y$
- Urto con sponda verticale: cambia il segno di  $v_x$



# Urto con due sponde che fanno angolo

Riflessione simmetrica rispetto alla bisettrice dell'angolo

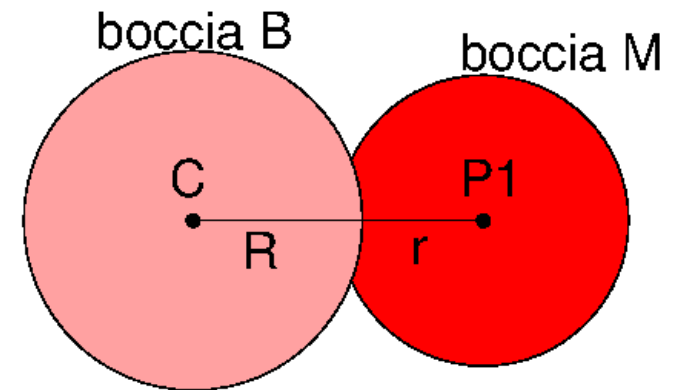
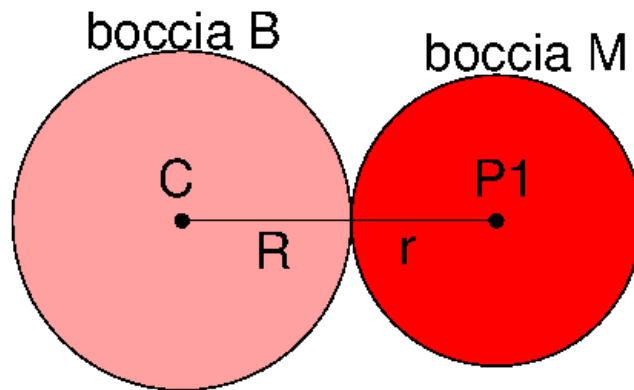
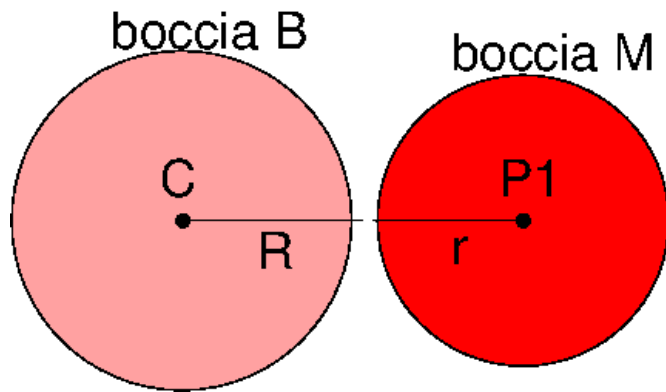
- cambia il segno di entrambe le componenti e si scambiano
- $v_x' = -v_y$  e  $v_y' = -v_x$



# Urtare un'altra boccia

Se i due centri distano fra loro meno della somma dei raggi

$$\text{dist}(P1, C) \leq r + R$$



Tolleriamo una piccola sovrapposizione...

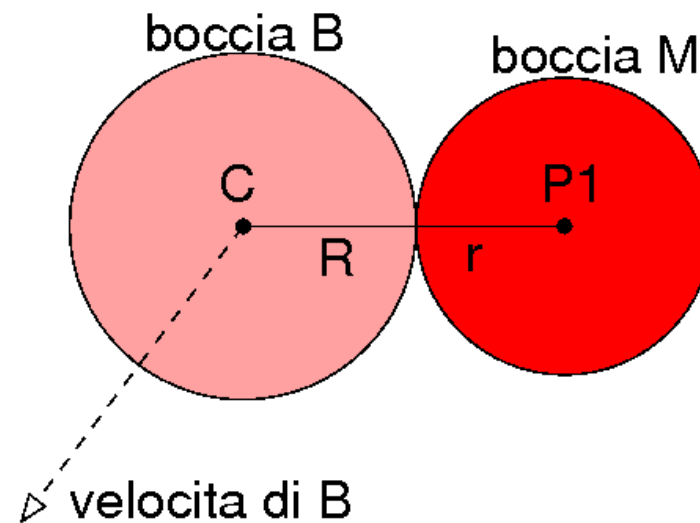
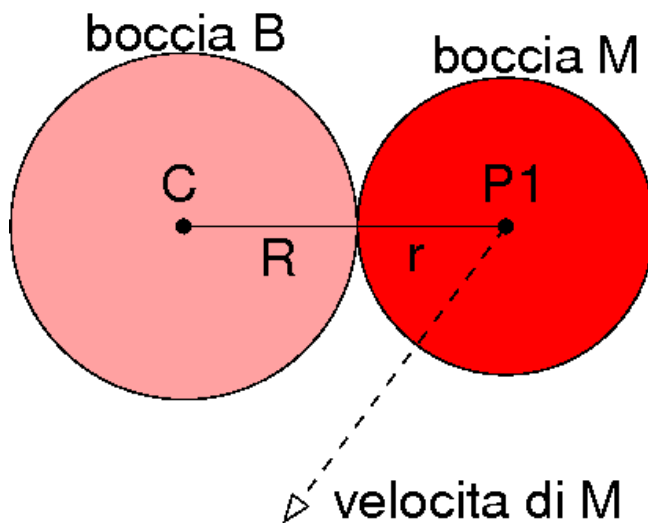
# Urtare un'altra boccia

Se i due centri distano fra loro meno della somma dei raggi

$$\text{dist}(P1, C) \leq r + R$$

In caso di urto:

- tutto il moto si trasferisce alla boccia urtata, che acquista la stessa velocita', la boccia urtante si ferma



Questo modello non corrisponde alla realta', ma a noi basta!

# Eventi molto improbabili

Urto con due bocce:

- Consideriamo come se urtasse una sola delle due.

Urto con una boccia e una sponda:

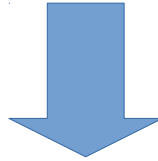
- Consideriamo solo l'urto con la sponda.
- L'urto con la boccia sara' eventualmente processato nel prossimo scatto.



# Se non ci sono stati urti

Il modulo della velocità diminuisce di un decremento fissato

- se diventa 0 allora la palla si ferma
- se diventa  $< 0$ , si considera 0



Dopo un certo numero di scatti, le palle saranno di nuovo tutte ferme.

# Riepilogo di uno scatto della simulazione

- La boccia in moto avanza dalla posizione P0 alla posizione P1
- Ciclo sulle buche
  - se la boccia M cade in una buca, allora viene eliminata e stop
- Ciclo sulle sponde
  - se la boccia urta una sponda o due sponde, allora rimbalza e stop
- Ciclo sulle bocce
  - se la boccia urta un'altra boccia, il moto si trasferisce all'altra boccia e stop
- Se il flusso d'esecuzione arriva qui, si applica il decremento alla velocita' della boccia

# L'interfaccia Gioco

- E' dato il file **Gioco.java** contenente l'interfaccia che dovete implementate.
- I commenti sono predisposti per generare la documentazione con javadoc.
- **Leggete attentamente la documentazione prima di cominciare!**
- L'interfaccia contiene le costanti:
  - DELTA\_T intervallo di tempo tra due scatti
  - DECREMENTO decremento della velocita' dopo ogni scatto
- e definisce le funzioni pubbliche che dovete implementare per supportare il gioco.

# Le funzioni del gioco

void **inizializzaLeggendo** (String nome) throws Exception

Inizializza il gioco leggendo da file

- dimensioni del campo
- numero di bocce e di buche
- diametri delle bocce e delle buche

Le posizioni  
delle bocce e  
delle buche le  
decidete voi!

void **cambiaNumeroTiri** (int n) throws Exception

Stabilisce il numero di tiri a  
disposizione del giocatore

# Le funzioni del gioco

int **numeroBuche()**

int **numeroBocce()**

double **campoX()**

double **campoY()**

Ritornano il numero  
di bocce e di buche

Ritornano  
le  
dimensioni  
del campo

double **bocciaX**(int indiceBoccia)

double **bocciaY**(int indiceBoccia)

double **bucaX**(int indiceBuca)

double **bucaY**(int indiceBuca)

double **diametroBoccia**(int indiceBoccia)

double **diametroBuca**(int indiceBuca)

Ritornano la posizione  
e il diametro di  
ciascuna boccia e di  
ciascuna buca

gli indici  
partono da 0

# Le funzioni del gioco

int **indiceBoccino** ( )

Ritorna quale  
boccia e' il  
boccino

boolean **caduta** (int indiceBoccia)

Controlla se  
una boccia e'  
in buca

int **numeroCadute** ( )

Ritorna il numero di bocce  
cadute in buca

# Le funzioni del gioco

boolean **giocoFinito** ( )

Controlla se il gioco e' finito

boolean **bocceFerme** ( )

Controlla se c'è una  
boccia in movimento

int **numeroTiri** ( )

Ritorna quanti tiri sono  
rimasti disponibili

int **punti** ( )

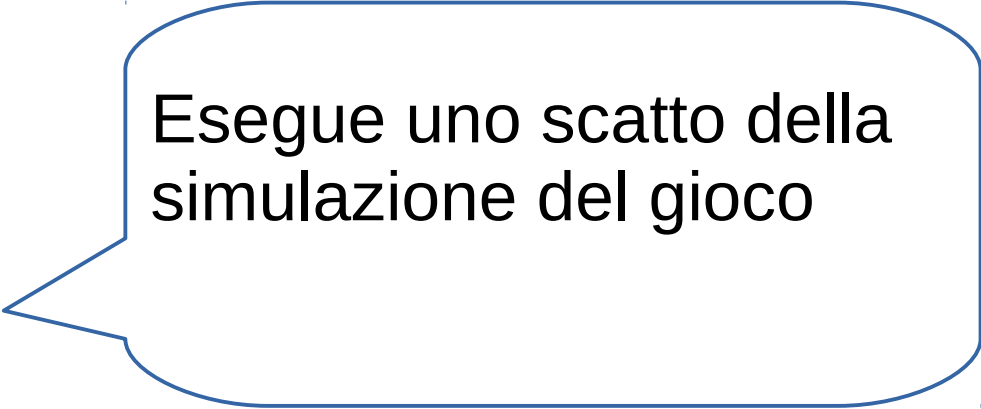
Ritorna il punteggio corrente

# Le funzioni del gioco

void **preparaBoccino**(double intensita, double angoloDirezione)



Stabilisce la velocita' iniziale del boccino per il prossimo tiro



Esegue uno scatto della simulazione del gioco

boolean **evoluzioneDeltaT** ( )



# Cave canem

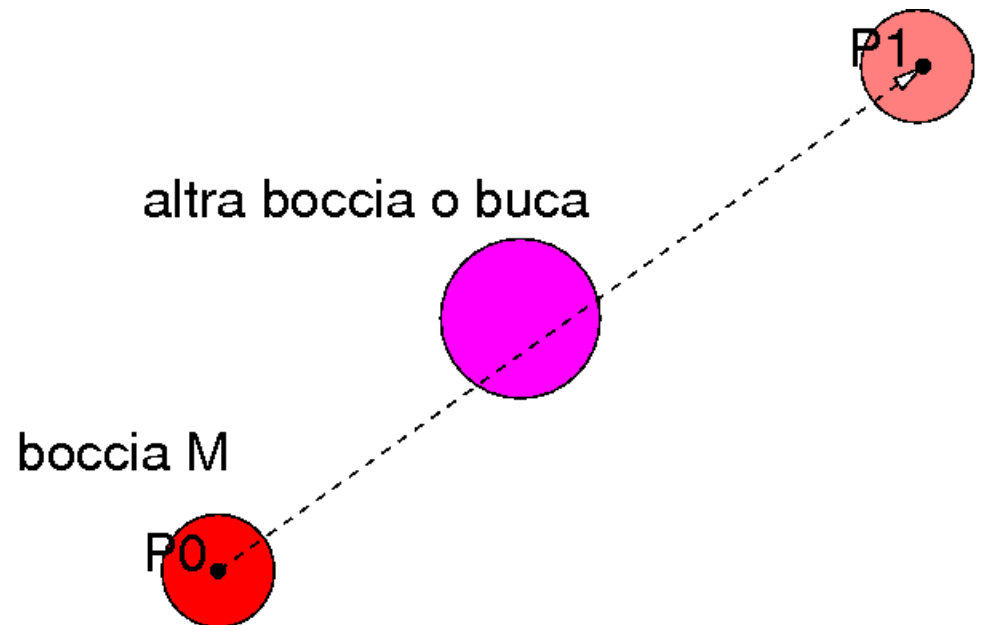
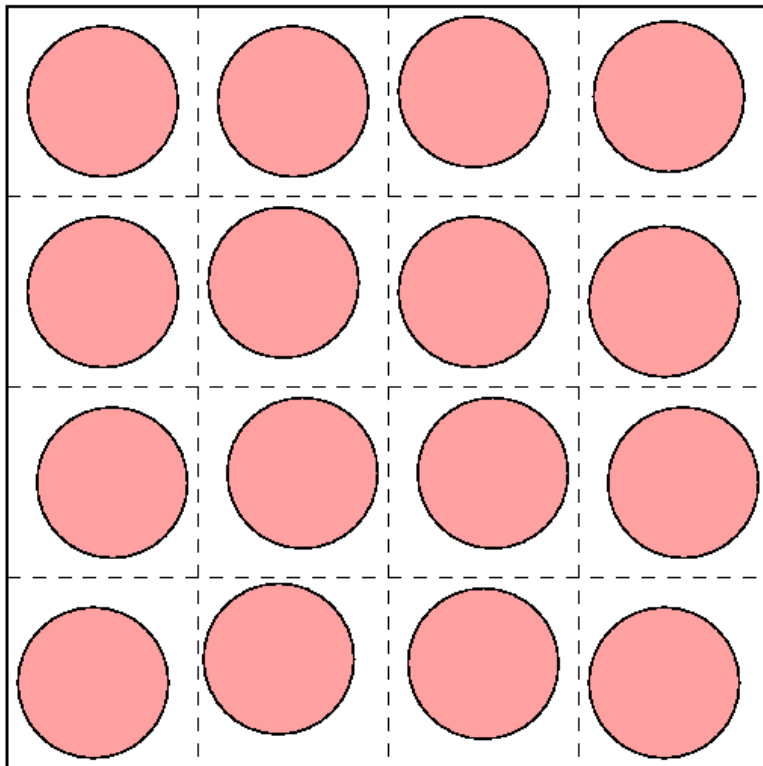
Modi in cui potreste pensare di complicarvi la vita:

- Non reinventare le regole per urtare, rimbalzare... e le operazioni da fare in uno scatto di simulazione.
- Le sponde del campo sono parallele agli assi cartesiani. Non trattare inclinazione generica.

# Cave canem

Gli ordini di grandezza sono quelli, perciò:

- e' sempre possibile trovare una configurazione valida per collocare inizialmente le buche e le bocce
- la granularita' della simulazione e' sufficiente a rilevare gli urti



# Cave canem

- Riciclare cose viste a lezione e fatte nei laboratori.
- Se qualche classe predefinita di Java vi è utile, usatela.
- Il modello matematico e' molto semplificato: va bene.

**Se la strada che avete intrapreso sta diventando troppo complicata, fermatevi e contattatemi.**  
Probabilmente state facendo una cosa non necessaria.

# Cosa fare per adesso

- Pensare all'organizzazione del progetto: classi, contenuto di ogni classe, relazioni tra le classi...
- Una settimana di tempo circa
- Ogni gruppo mi spieghera' la sua idea
- Dopo potrete iniziare a scrivere codice
- Se proprio volete iniziare subito, potete fare la lettura del file di input (legge gestendo gli errori e ristampa cio' che ha letto)

**Leggete bene le specifiche!**

(documentazione dell'interfaccia Gioco)