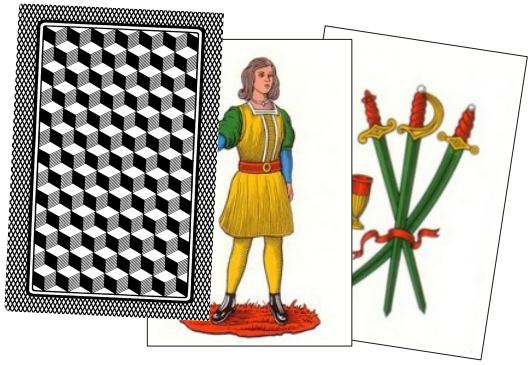
Laboratorio di Applicazioni Mobili

Applicazione in ambiente iOS

Ivan Heibi
Ivan.heibi@studio.unibo.it

iOS Application



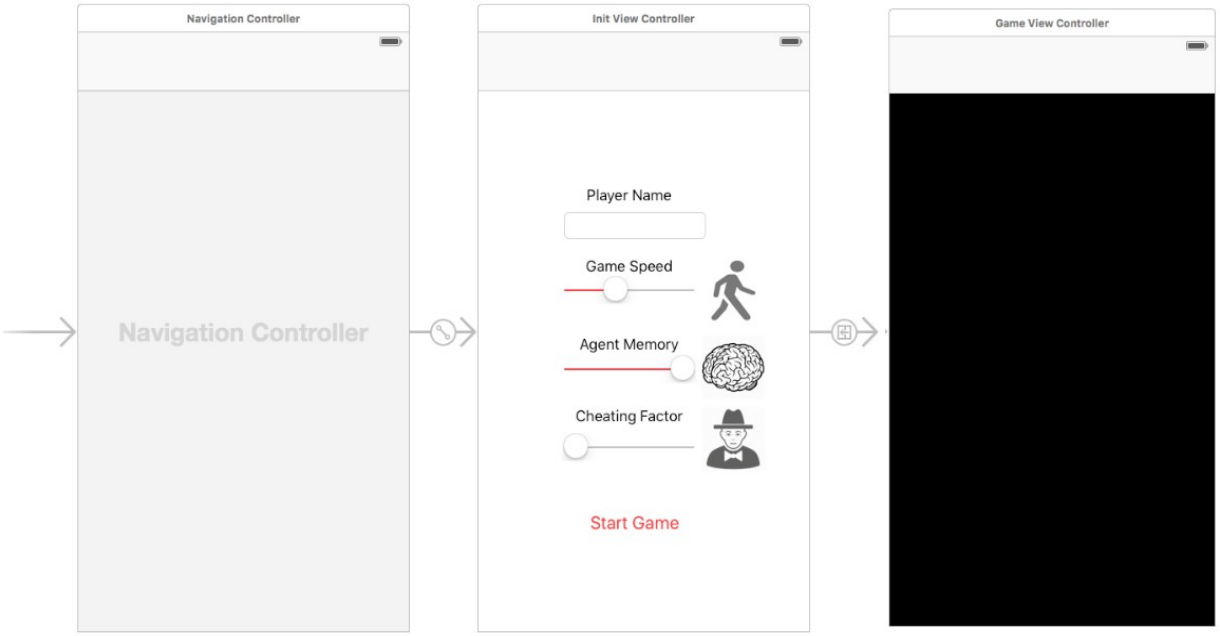
La Briscola

Apple's IDE
Xcode

Swift programming language

Architettura Generale

- Il pattern utilizzato è MVC (Model View Controller)
- Perfetto per separare la logica applicativa dalla user interface (UI)
- Il gioco è composto da 2 view
- Utilizziamo un Navigation Controller per spostarci



Le Classi

- Card : Le carte da gioco sono istanze di questa classe
- Table : Il tavolo da gioco, informazioni comuni a tutti
- ComputerMind : L'utente PC (agente intelligente)

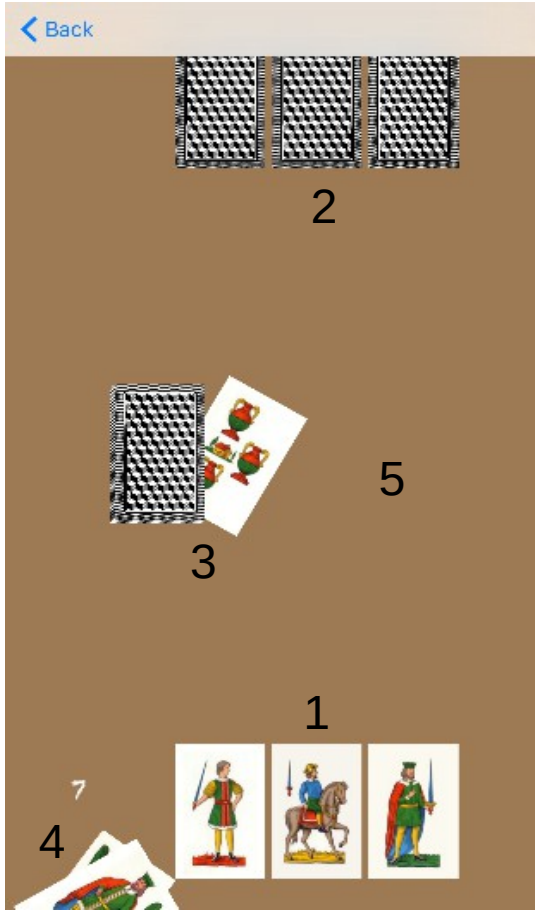
Views :

- initViewController : inizializza i parametri iniziali della partita
- gameViewController : la schermata del gioco

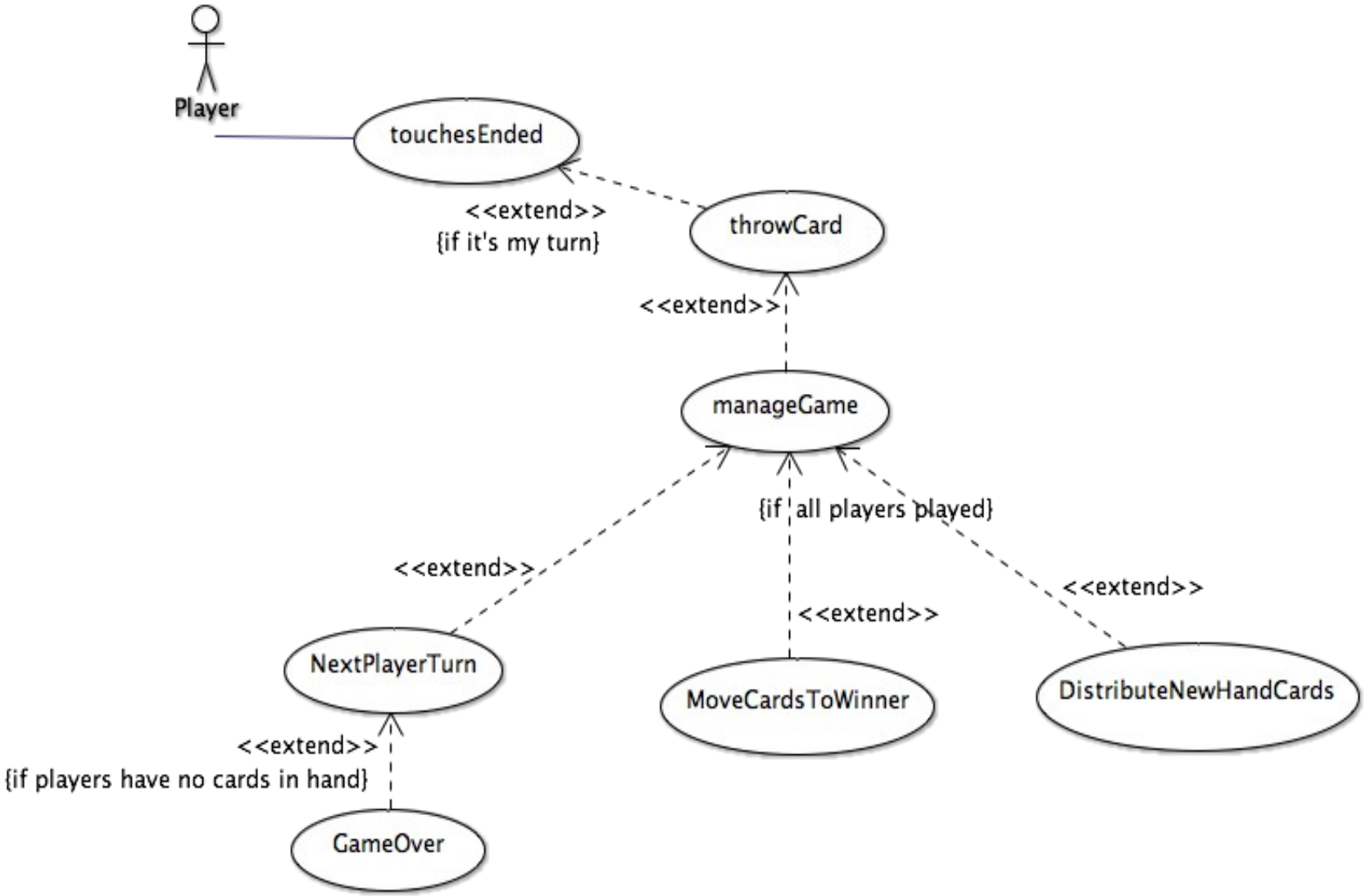
- GameScene : creazione della scena e parte concettuale del gioco

Interfaccia del gioco

- 1: Le mie carte
- 2: Le carte dell'avversario PC
- 3: Il mazzo di carte
- 4: Le carte vinte e lo score
- 5: Il tavolo da gioco



Casi d'uso lato utente



Il PC: avversario intelligente

- Un 'Agente Intelligente: Percepisco → Agisco → effettuo un cambiamento nel mio ambiente
- Percezioni: i movimenti della partita
- Azioni: muovere le carte
- Ambiente: il tavolo da gioco
- Sensori: la simulazione della vista

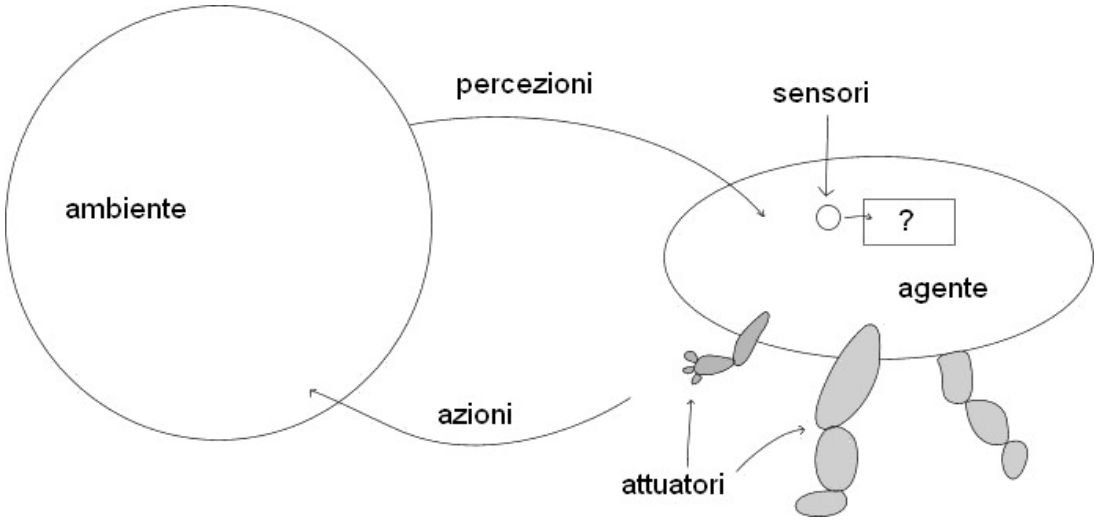


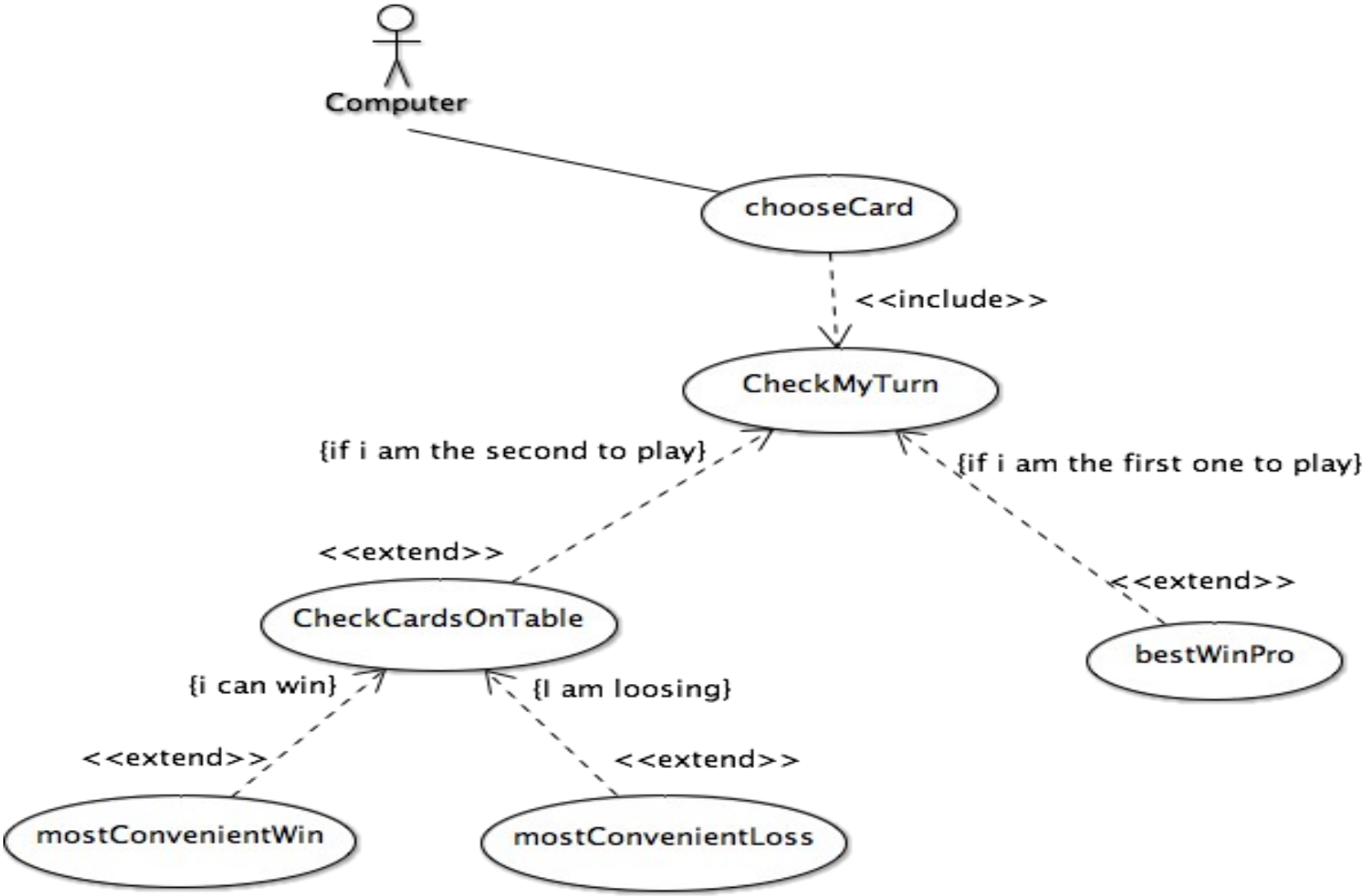
*Foto da Wikipedia.org

Le capacità del PC

- **Memoria**: ricordare le carte che sono già state giocate
  +Capacità di memorizzazione alta → numero di carte maggiore ricordate

- **Fattore di scorrettezza**: sbirciare in maniera furbesca per vedere le carte in mano nostra.
  +Fattore di scorrettezza alto → probabilità maggiore di vedere le nostre carte

Casi d'uso lato PC



bestWinPro()

- PC → chooseCard() → è il primo a giocare
- Guardo le carte in mano → scelgo quella con probabilità maggiore di non essere battuta

- Le carte che devo considerare nel calcolo della probabilità :
  - + Quelle del mio avversario (se in mia conoscenza)
  - + Quelle che ricordo non essere ancora giocate

- Strategia che minimizza la perdità momentanea ma non tiene conto del risultato su scala più grande

mostConvenientWin()

• PC → il secondo a giocare → controllo le carte sul tavolo → se posso vincere → scelgo la carta che mi da la vincità più conveniente

• Considero le carte in mio possesso che possono vincere
• Le mie priorità sono:
    1) Vinco senza una briscola e guadagno punti
    2) Vinco con la briscola meno potente

• Nel caso in cui la vincità è pari a 0 punti → quantifico la miglior perdità per le carte in mano → se uguale a 0 punti → gioco quest'ultima carta

mostConvenientLoss()

- PC → il secondo a giocare → controllo le carte sul tavolo → se non posso vincere → scelgo la carta che minimizza la mia perdità

- Prendo in considerazione tutte le carte in mio possesso
- Scelgo la carta con potenza minore

- La potenza di una carta = card.rank + 10(se è briscola)

- Nota*: preferisco perdere 10 punti invece di una briscola da 2 punti

Aspetti Implementativi - Grafica

- Abbiamo bisogno di oggetti di tipo SKNode (Sprite Kit Node)
- Oggetti fatti per poter essere animati se messi all'interno di una scena SKScene
- Una view che mostra una SKScene deve essere di tipo SKView

- GameScene : SKScene
- Card : SKNode

- Una animazione viene fatta con oggetti di tipo SKAction (applicazione di una azione ad un SKNode)

Aspetti Implementativi – Logica del gioco (1)

- Il modello del gioco è implementato in GameScene

- Strutture dati :

  + Mazzo di carte = [Card] (count: 40)
  (10 carte per ognuno dei 4 semi)

  + Carte di ogni giocatore = [[Card]] (count: 4 repeatedValue: [Card](count:3))
  (3 carte per ogni giocatore, max numero di giocatori possibili è 4)

Aspetti Implementativi – Logica del gioco (2)

- Fase di inizializzazione :
  + Costruzione del mazzo di carte: riempire l'array con oggetti Card appropriati

  + Mescolare il mazzo: dealCards() prende il mazzo e sposta ogni carta in una posizione casuale

  + Distribuzione delle carte: distributeCards(), dato il mazzo mescolato si danno 3 carte ad ogni giocatore

  + Impostare il tavolo di gioco Table (Turni e briscola)

  + Creazione dei giocatori di tipo PC (ComputerMind)

Aspetti Implementativi – Logica del gioco (3)

- La fase di gioco:
  + Tutte le nostre mosse sono eventi UIEvent di tipo UItouches
  + Gli oggetti che possono essere “touched” sono le Card
  + Non tutti possono essere spostati → dipende dalla posizione ed in quale momento
  + I nostri movimenti sono “trigger” a delle mosse automatiche da parte del PC

  + Tutte le azioni sono delle modifiche ai vettori
  + La partita finisce quando i vettori del gioco sono tutti vuoti

Sviluppi futuri (1)

- **Numero di giocatori :**

  + Il gioco in questa versione è una partita tra 2 giocatori

  + Il codice è preimpostato per essere ampliato, date le strrutture dati utilizzate

  + Più giocatori nella partita → difficoltà nella loro gestione su una sola macchina

  + Valutare la possibilità di un online-multiplayer

Sviluppi futuri (2)

- **Strategia di gioco del PC :**

  + Perfezionare la sua strategia attuale:
    - Quando è il primo a giocare (attualmente gioca la carta con probabilità di vincita maggiore)
    - Quando deve sceglire la carta da perdere

  + Poter scegliere tra un numero di strategie di gioco diverse da assegnare all'avversario

  + Il PC sceglie la strategia più adeguata in base al nostro gioco

Grazie per l'attenzione

Ivan Heibi
Ivan.heibi@studio.unibo.it