Relazione progetto “Uno e Mezzo”

La presente relazione descrive lo sviluppo di un progetto ispirato al celebre gioco di carte “UNO”. Il nostro obiettivo è stato quello di riprodurre le dinamiche del gioco in un ambiente digitale, implementando le regole fondamentali che determinano lo svolgimento delle partite e la gestione delle mosse dei giocatori.

Nel corso del progetto, abbiamo definito i meccanismi per la distribuzione delle carte, la validazione delle mosse dei giocatori, l’applicazione degli effetti speciali delle carte e la determinazione delle condizioni che portano ad un’eventuale vittoria o sconfitta. Inoltre, abbiamo affrontato diverse sfide, tra cui la gestione delle regole di gioco in modo chiaro ed efficiente, garantendo che ogni fase della partita fosse correttamente rappresentata e implementata nella finestra di gioco.

Nei paragrafi illustreremo le nostre scelte progettuali, le principali meccaniche implementate e le difficoltà incontrate durante lo sviluppo, seguite da una riflessione sui possibili miglioramenti futuri.

Flusso del codice

Partiremo spiegando come prima cosa la parte logica del programma, per poi vedere come viene mostrata nella finestra di gioco.

- *Carte:* Per ogni colore possibile (blu, giallo, rosso, verde), viene creata una carta di ogni valore, da *0* a *9* e le carte speciali *+2* e *stop.* In più, vengono aggiunte anche le carte speciali *cambio colore* e *+4-cambio colore*.

```
% Definizione carte
card(1,red).
card(2, red).
card(3, red).
card(4, red).
card(5, red).
card(6, red).
card(7, red).
card(8, red).
card(9, red).
card(0, red).
card(+2, red).
card(stop, red).
```

- *Mazzo:* Usiamo un predicato per trovare tutte le carte definite in precedenza e creiamo una lista che le contiene.

```
lista_carte(CardList) :-
    findall(card(Value, Color), (value(Value), color(Color)), NormalCards),
    %lista con i +4
    SpecialCards1 = [card(+4, cambio), card(+4, cambio), card(+4, cambio), card(+4, cambio)]
,
    %lista con i cambiocolore
    SpecialCards2 = [card(cambio, cambio), card(cambio, cambio), card(cambio, cambio), card(
cambio, cambio)],
    append(NormalCards,SpecialCards1,CardList1),
    append(CardList1,SpecialCards2,CardList).
```

- *Mischiare il mazzo:* Usiamo un predicato per prendere la lista di tutte le carte e randomizzare la posizione di ogni carta al suo interno. Questa nuova disposizione viene salvata in un'altra lista, che sarà il mazzo che useremo per giocare.

```
% Funzione per ottenere una lista randomizzata di carte.
lista_carte_randomizzata(Mazzo) :-
    lista_carte(ListaCarte),
    random_permutation(ListaCarte, Mazzo).
```

- *Distribuzione carte:* Con questo predicato, riusciamo a estrarre dal mazzo le prime cinque carte, che vengono assegnate alla mano del primo giocatore, le cinque carte successive da mettere nella mano dell'IA e un'altra carta da mettere al centro del tavolo.

```
% Funzione per prendere le prime N carte dal Mazzo.
prendi_prime_n_carte(N, Mazzo, PrimeCarteSenzaSuffisso, Rimanenti) :-
    length(PrimeCarte, N),  % Crea una lista di lunghezza N
    append(PrimeCarte, Rimanenti, Mazzo),  % Dividi la lista in PrimeCarte e Rimanenti
    maplist(rimuovi_suffisso_carta, PrimeCarte, PrimeCarteSenzaSuffisso).

rimuovi_suffisso_carta(card(Valore, Colore), card(Valore, ColoreSenzaSuffisso)) :-
    rimuovi_suffisso_2(Colore, ColoreSenzaSuffisso).

% Verifica che la carta non sia una cambio altrimenti ne cerca un altra.
prendi_carta_valida([Carta | Resto], [Carta], Resto) :-
    Carta \= card(cambio, cambio),
    Carta \= card(+4, cambio).
prendi_carta_valida([Carta | Resto], Carte_Giocate, NuoveRimanenti) :-
    (Carta = card(cambio, cambio); Carta = card(+4, cambio)),
    prendi_carta_valida(Resto, Carte_Giocate, NuoveRimanenti).


% Funzione per distribuire le carte ai giocatori.
distribuisci_carte(ManoGiocatore1, ManoGiocatore2,Carte_Giocate) :-
    mazzo(Mazzo),
    % Prendi le prime 5 carte per il Giocatore 1
    prendi_prime_n_carte(5, Mazzo, ManoGiocatore1, Rimanenti),
    % Prendi le prime 5 carte per il Giocatore 2
    prendi_prime_n_carte(5, Rimanenti, ManoGiocatore2,  NuoveRimanenti),
    % Metti la prima carta del mazzo nelle carte giocate
    prendi_carta_valida(NuoveRimanenti, Carte_Giocate, RimanentiFinali),
    retract(mazzo(Mazzo)),
    assertz(mazzo(RimanentiFinali)).
```

- *Gestione dei turni:* Attraverso due predicati, riusciamo a bloccare il turno dell'avversario per dare il tempo necessario per giocare e, una volta scelta la propria mossa, il turno viene passato all'altro giocatore, creando un ciclo che non si fermerà finché non si raggiunge la *win/loss condition*.

```
turno_giocatore :-
    retractall(giocatore_attivo(_)),
    assertz(giocatore_attivo(1)),
    (   gioco_finito(si)
    ->  true
    ;   turno_bloccato(si)
    ->
        retractall(turno_bloccato(_)),
        assertz(turno_bloccato(no)),
        turno_ia
    ;
        controlla_vittoria,
        controllo_mazzo,
        turno_ia
    ).
% contrario del truno giocatore.
 turno_ia :-
    retractall(giocatore_attivo(_)),
    assertz(giocatore_attivo(2)),
    (   gioco_finito(si)
    ->  true
    ;   turno_bloccato(si)
    ->  writeln('Il turno dell\'IA è bloccato!'),
        retractall(turno_bloccato(_)),
        assertz(turno_bloccato(no)),
        turno_giocatore
    ;   writeln('Turno dell\'IA...'),
        gioca_carta_ia,
        controlla_vittoria,
        controllo_mazzo,
        turno_giocatore
    ).
```

- Gioco carta giocatore (dalla mano del giocatore, viene controllata la validità della carta rispetto a quella al centro; se verificata, gioca la carta): Questo predicato viene chiamato ogni volta che il giocatore clicca su una carta della propria mano sulla finestra di gioco e controlla se la carta scelta si può giocare, ovvero se ha stesso colore o valore della carta al centro; quindi, la carta scelta viene giocata.

```
gioca_carta(card(Valore,Colore)) :-
    carte_giocate(CarteGiocate),
    mano_giocatore1(ManoGiocatore1),
    Carta = card(Valore,Colore),
    NuoveCarteGiocate = [Carta, CarteGiocate],
    select(Carta, ManoGiocatore1, NuovaMano),
    retract(mano_giocatore1(ManoGiocatore1)),
    assertz(mano_giocatore1(NuovaMano)),
    retract(carte_giocate(CarteGiocate)),
    assertz(carte_giocate(NuoveCarteGiocate)),
    attiva_effetto(Carta),
    crea_carta_giocata.
```

- Pescaggio carte: Questo predicato aggiunge la carta in cima alla lista del mazzo, nella mano del giocatore che pesca, che sia perché non ha carte da giocare, o ha ricevuto un *+2* o *+4,* oppure viene penalizzato in altri modi.

```
pesca_carte(N,P) :-
    mazzo(Mazzo),
    mano_giocatore1(ManoGiocatore1),
    mano_giocatore2(ManoGiocatore2),

    prendi_prime_n_carte(N, Mazzo, PrimeCarte, Rimanenti),
    (   P = 2
    ->  append(ManoGiocatore2,PrimeCarte,NuovaMano),
        retract(mazzo(Mazzo)),
        assertz(mazzo(Rimanenti)),
        retract(mano_giocatore2(ManoGiocatore2)),
        assertz(mano_giocatore2(NuovaMano)),
        setta_mano_IA
    ;
    append(ManoGiocatore1,PrimeCarte,NuovaMano),
        retract(mazzo(Mazzo)),
        assertz(mazzo(Rimanenti)),
        retract(mano_giocatore1(ManoGiocatore1)),
        assertz(mano_giocatore1(NuovaMano)),
        setta_mano_giocatore
    ).
bottone_pesca :-
    retractall(giocatore_attivo(_)),
    assertz(giocatore_attivo(2)),
    pesca_carte(1,1),
    gioca_carta_ia,
    crea_carta_giocata,
    setta_mano_IA,
    controlla_vittoria.
```

- *Gestione delle carte speciali:* Ogni volta che viene giocata una carta, si controlla il suo valore e, a seconda di esso, viene attivato un effetto. Come da regole del gioco originale, una carta numero non attiva nessun effetto speciale. Una carta *+2* assegna due carte all'avversario. Una carta *+4-cambio* assegna quattro carte all'avversario e permette al giocatore di cambiare il colore. Una carta *stop* ferma il turno dell'avversario e permette al giocatore di giocare un'altra carta. Una carta *cambio colore* permette al giocatore di determinare il colore della carta che dovrà essere giocata.

```
    % Funzione che attiva l'effetto della carta giocata.
attiva_effetto(card(ValoreGiocato, ColoreGiocato)) :-
    giocatore_attivo(GiocatoreAttivo),
    (   ValoreGiocato == +2 ->
        (   GiocatoreAttivo = 1 ->
            pesca_carte(2, 2)
        ;
            pesca_carte(2, 1)
        )
    ;   ValoreGiocato == +4 ->
        (   GiocatoreAttivo = 1 ->
            cambia_colore(card(ValoreGiocato, ColoreGiocato)),
            pesca_carte(4, 2)
        ;
            cambia_colore(card(ValoreGiocato, ColoreGiocato)),
            pesca_carte(4, 1)
        )
    ;   ValoreGiocato == stop ->
        (   retractall(turno_bloccato(_)),
            assertz(turno_bloccato(si))
        )
    ;   ValoreGiocato == cambio ->
        (   GiocatoreAttivo = 1 ->
            cambia_colore(card(ValoreGiocato, ColoreGiocato))
        ;
            cambia_colore(card(ValoreGiocato, ColoreGiocato))
        )
    ;
    true
    ).
```

- *Gestione effetto cambia colore:* Prima che si gioca una carta *cambio colore* o una carta *+4-cambio*, bisogna selezionare il colore nuovo nella finestra di gioco e solo dopo cliccare sulla carta per giocarla. Il predicato aggiunge una carta del colore scelto al centro, in modo che l'avversario sia costretto a giocare una carta del nuovo colore.

```
% Funzione per scegliere il colore.
scegli_colore(card(_, ColoreScelto),NuovoColore) :-
    scegli_colore_aux(ColoreScelto,NuovoColore).

% Richiesta effettiva del colore.
scegli_colore_aux(_,ColoreScelto) :-
    writeln('Scegli un colore (red, green, blue, yellow):'),
    %read(ColoreInserito),
    (color(ColoreInserito)
    ->
    writeln('Colore valido'),
    ColoreScelto = ColoreInserito
    ;
    writeln('Colore non valido, riprova.'),
     scegli_colore_aux(_,ColoreScelto)
    ).

% Funzione per cambiare il colore della carta.
cambia_colore(card(ValoreGiocato, ColoreScelto)) :-
    giocatore_attivo(Giocatore),
    mano_giocatore2(ManoGiocatore2),
    (
        Giocatore = 1
        ->
        colore(NuovoColore),
        carte_giocate(CarteGiocate),
    % Crea una lista con solo la nuova carta e la appende alla nuova lista di carte giocate
e poi aggiorna la variabile dinamica
        CartaNuova = [card(ValoreGiocato,NuovoColore)],
        CartaVecchia = [card(ValoreGiocato,ColoreScelto)], % dovuto mette una variabile perc
hè prolog va schifo
        append(CartaVecchia,CarteSenzaCambio,CarteGiocate), % per levare la vecchia cambioco
lore
        append(CartaNuova,CarteSenzaCambio,NuoveCarteGiocate),
        retractall(carte_giocate(_)),
        assertz(carte_giocate(NuoveCarteGiocate))
        ;
        colore_massimo(ManoGiocatore2,NuovoColore2),
        carte_giocate(CarteGiocate),
    % Crea una lista con solo la nuova carta e la appende alla nuova lista di carte giocate
e poi aggiorna la variabile dinamica
        CartaNuova = [card(ValoreGiocato,NuovoColore2)],
        CartaVecchia = [card(ValoreGiocato,ColoreScelto)],
        append(CartaVecchia,CarteSenzaCambio,CarteGiocate), % per levare la vecchia cambioco
lore
        append(CartaNuova,CarteSenzaCambio,NuoveCarteGiocate),
        retractall(carte_giocate(_)),
        assertz(carte_giocate(NuoveCarteGiocate))
    ).
```

- *Gestione bottone Uno!:* Questo predicato gestisce quello che succede quando si clicca il bottone *UNO!*. Se viene cliccato quando non si ha una sola carta, al giocatore verranno fatte pescare due carte. Se viene cliccato al momento giusto, il giocatore può giocare la sua ultima carta in mano. Se non viene cliccato al momento giusto, il giocatore deve pescare due carte.

```
% Funzione che cambia il valore della variabile dinamica.
bottone_uno :-
    writeln('Hai detto uno'),
    retractall(detto_uno(_)),
    assertz(detto_uno(si)).

% Funzione che controlla le varie casistiche di UNO
controllo_uno:-
    mano_giocatore1(ManoGiocatore1),
    detto_uno(Uno),
    length(ManoGiocatore1,Lunghezza),
    (
     Lunghezza = 1
        ->
        (   Uno = si
        ->
        writeln('UNO!')
        ;
        writeln('NON HAI DETTO UNO!'),
            pesca_carte(2,1)
        )
    ;
        (   Uno = si
        ->
        writeln('HAI DETTO UNO ANCHE SE NON AVEVI UNA SOLA CARTA!'),
            pesca_carte(2,1)
        ;
        writeln('')
        )
    ),
    retractall(detto_uno(_)),
    assertz(detto_uno(no)).
```

- *gioca carta IA (come sceglie qual è la carta migliore da giocare):* Il predicato inizia con il richiamo del predicato *controllo_uno*, che controlla se il giocatore ha cliccato il bottone al momento giusto, poi prende la mano dell'*IA* e l'ultima carta giocata, richiama il predicato *miglior_carta_da_giocare_aux* che richiama a sua volta il predicato *miglior_carta_da_giocare*, il predicato è richiamato con \+ davanti perché esso ritornerà sempre *false* (come da scelta del programmatore) e facendo così il *false* diventa *true* non bloccando il proseguimento del codice. Questo predicato fa il giro di tutto il mazzo e controlla se ogni carta è giocabile e la aggiunge alla lista *carte_giocabili_ia*, con annesso un peso settato a *0* se è una carta numero, e *1* se è una carta cambio colore. Questa lista verrà poi riordinata tramite la funzione *sort*, utilizzando *sort/4* sul parametro peso; successivamente, si prenderà la prima carta di questa lista, così l'*IA* giocherà prima le carte valore e poi le carte cambio colore.

Dopo aver selezionato la carta migliore, essa viene rimossa dalla mano dell'*IA* e messa in cima al mazzo delle carte giocate, dove ne verrà attivato l'effetto.

Dentro il predicato *attiva_effetto* si controlla il tipo di carta giocata: nel caso che l'*IA* giochi una carta cambio verrà richiamato il predicato *cambia_colore,* che a sua volta chiamerà il predicato *colore_massimo* che calcola il colore più favorevole da selezionare, creando una lista composta da *colore-numeroCarteDiQuelColore* che controlla tutta la mano dell'*IA* e restituisce il numero di carte di ogni colore. Se invece l'*IA* gioca una carta stop, essendo il turno del giocatore bloccato, il predicato si chiamerà ricorsivamente, così da permettere al giocatore di giocare un'altra carta.

```
% Fa giocare l'ia:
% 1. controlla la miglior carta da giocare,
% 2. modifica la mano e le carte giocate,
% 3. se non puo giocare pesca una carta.
gioca_carta_ia :-
    non_mio_turno,
    controllo_uno,
    mano_giocatore2(ManoGiocatore2),
    carte_giocate(Carte_Giocate),
    carte_giocate([PrimaCarta|_]),
        miglior_carta_da_giocare_aux(Carte_Giocate, ManoGiocatore2, _),
        carta_da_giocare_ia(CartaGiocata),
        nonvar(CartaGiocata), % controllo che la carta esite altrimenti esce
        carta_valida(CartaGiocata,PrimaCarta),
        select(CartaGiocata, ManoGiocatore2, NuovaMano2),
        NuoveCarteGiocate = [CartaGiocata | Carte_Giocate],
        retract(mano_giocatore2(ManoGiocatore2)),
        assertz(mano_giocatore2(NuovaMano2)),
        retract(carte_giocate(Carte_Giocate)),
        assertz(carte_giocate(NuoveCarteGiocate)),
        sleep(1),
        attiva_effetto(CartaGiocata),
        retractall(carte_giocabili_ia(_)),
        assertz(carte_giocabili_ia([])),
        retract(carta_da_giocare_ia(CartaGiocata)),
        assertz(carta_da_giocare_ia(_)),
        crea_carta_giocata,
        setta_mano_IA,
        controlla_vittoria,
        turno_bloccato(ControlloBloccoTurno),
        (    ControlloBloccoTurno = si
        ->
        retractall(turno_bloccato(_)),
        assertz(turno_bloccato(no)),
            gioca_carta_ia
        ;
        true
        ),
        retractall(giocatore_attivo(_)),
        assertz(giocatore_attivo(1)),
        mio_turno.

gioca_carta_ia :-
    pesca_carte(1, 2),
    retractall(giocatore_attivo(_)),
    assertz(giocatore_attivo(1)),
    mio_turno.
```

- *Win/Loss condition:* Ogni volta che un giocatore gioca una carta si richiama il predicato *controllo_vittoria* che controlla le mani dei giocatori e, a sua volta, richiama *win_condition* se il giocatore non ha più carte in mano, altrimenti richiama *loss_condition* se l'*IA* ha giocato tutte le sue carte.

```
win_condition :-
    send(@device, clear),
    free(@device),
    new(@device, device),
    send(@dialog, display, @device, point(0,0)),

    %sfondo
    source_file(win_condition, File1),
    rimuovi_file_da_percorso(File1, Risultato1),
    directory_file_path(Risultato1, '/Immagini/win.jpg', NuovoPercorso1),
    new(Imagefile3, image(NuovoPercorso1)),
    new(Bitmap1, bitmap(Imagefile3)),
    send(@device, display, Bitmap1),
    send(Bitmap1, center, @device?center),

    %box replay
    free(@bottoneReplay),
    new(@bottoneReplay, box(300,75)),
    send(@bottoneReplay, fill_pattern, colour(black)),
    send(@bottoneReplay, radius, 10),
    send(@bottoneReplay, colour, white),
    send(@dialog, display, @bottoneReplay, point(200,400)),
    send(@bottoneReplay, recogniser, click_gesture(left, '', single,
                message(@prolog, start_the_game))),

    %testo replay
    free(@testoReplay),
    new(@testoReplay, text('Replay')),
    send(@testoReplay, font, font(helvetica, bold, 30)),
    send(@testoReplay, colour, colour(white)),
    get(@testoReplay, width, TestoWidth),
    get(@testoReplay, height, TestoHeight),
    TestoX is 200 + (300 - TestoWidth) / 2,
    TestoY is 400 + (75 - TestoHeight) / 2,
    send(@dialog, display, @testoReplay, point(TestoX, TestoY)),

    false.
```

Per quanto riguarda la parte di gestione della grafica:

- Come prima cosa, quando viene avviato il gioco, viene creata una finestra a schermo, che ha uno sfondo e due bottoni: uno per aprire un'altra finestra, con all'interno scritte le regole fondamentali del gioco, e l'altro per passare alla schermata effettiva del gioco.

```
start_game :-
    free(@dialog), % @ prende indirizzo di memoria
    % creo finestra

    new(@dialog, dialog('Uno e mezzo')),
    send(@dialog, size, size(700,700)),
    send(@dialog, background, white),

     % Device.
    free(@device),
    new(@device, device),
    send(@dialog, display, @device, point(0,0)),

    % inserisco l'immagine come sfondo
    source_file(start_game, File),
    rimuovi_file_da_percorso(File, Risultato),
    directory_file_path(Risultato, '/Immagini/schermatainiziale.jpg', NuovoPercorso),
    new(@imagefile, image(NuovoPercorso)),
    new(Bitmap, bitmap(@imagefile)),
    send(@device, display, Bitmap),
    send(Bitmap, center, @device?center),


    %box play
    free(@bottonePlay),
    new(@bottonePlay, box(300,75)),
    send(@bottonePlay, fill_pattern, colour(black)),
    send(@bottonePlay, radius, 10),
    send(@bottonePlay, colour, white),
    send(@dialog, display, @bottonePlay, point(200,350)),
    send(@bottonePlay, recogniser, click_gesture(left, '', single,
                message(@prolog, start_the_game))),

    %testo play
    free(@testoPlay),
    new(@testoPlay, text('Gioca')),
    send(@testoPlay, font, font(helvetica, bold, 36)),
    send(@testoPlay, colour, colour(white)),
    get(@testoPlay, width, TestoWidth),
    get(@testoPlay, height, TestoHeight),
    TestoX is 200 + (300 - TestoWidth) / 2,
    TestoY is 350 + (75 - TestoHeight) / 2,
    send(@dialog, display, @testoPlay, point(TestoX, TestoY)),

    %box regole
    free(@bottoneRegole),
    new(@bottoneRegole, box(300,75)),
    send(@bottoneRegole, fill_pattern, colour(black)),
    send(@bottoneRegole, radius, 10),
    send(@bottoneRegole, colour, white),
    send(@dialog, display, @bottoneRegole, point(200,475)),
    send(@bottoneRegole, recogniser, click_gesture(left, '', single,
                message(@prolog, regole))),

    %testo play
    free(@testoRegole),
    new(@testoRegole, text('Regole')),
    send(@testoRegole, font, font(helvetica, bold, 36)),
    send(@testoRegole, colour, colour(white)),
    get(@testoRegole, width, TestoWidthR),
    get(@testoRegole, height, TestoHeightR),
    TestoRX is 200 + (300 - TestoWidthR) / 2,
    TestoRY is 475 + (75 - TestoHeightR) / 2,
    send(@dialog, display, @testoRegole, point(TestoRX, TestoRY)),

    send(@dialog, open).
```

- Una volta iniziato il gioco, vengono rimossi tutti gli elementi nella finestra e vengono sostituiti con il layout del gioco: uno sfondo, uno spazio in alto riservato alle carte dell'*IA*, uno spazio in basso riservato alle carte del giocatore, al centro invece si trovano il mazzo di carte giocate e il mazzo da cui pescare, affiancati a sinistra da alcuni bottoni per scegliere il colore e a destra dal bottone *Uno!*. Inoltre, è presente una piccola scritta al centro per segnalare al giocatore se è il suo turno oppure no.

```
campo_inizio :-
    Yoffset is 105,

    PositionXmazzo is 355,
    PositionYmazzo is 350-Yoffset/2,

    % Carta Mazzo
    free(@cartamazzo),
    new(@cartamazzo, box(68,100)),
    send(@cartamazzo, fill_pattern, colour(black)),
    send(@cartamazzo, radius, 7),
    send(@cartamazzo, colour, white),

    send(@dialog, display, @cartamazzo, point(PositionXmazzo,PositionYmazzo)),
    send(@cartamazzo, recogniser, click_gesture(left, '', single,
                message(@prolog, bottone_pesca))),

    % Testo Carta Mazzo
    free(@testoMazzo),
    new(@testoMazzo, text('UNO')),
    send(@testoMazzo, font, font(helvetica, bold, 20)),
    send(@testoMazzo, colour, colour(yellow)),
    get(@testoMazzo, width, TestoMazzoWidth),
    get(@testoMazzo, height, TestoMazzoHeight),
    TestoMazzoX is PositionXmazzo + (68 - TestoMazzoWidth) / 2,
    TestoMazzoY is PositionYmazzo + (100 - TestoMazzoHeight) / 2,
    send(@dialog, display, @testoMazzo, point(TestoMazzoX, TestoMazzoY)),


    % Bottone Uno!
    free(@bottoneUno),
    new(@bottoneUno, box(150, 90)),
    send(@bottoneUno, fill_pattern, colour(yellow)),
    send(@bottoneUno, radius, 20),
    send(@bottoneUno, pen, 3),
    send(@bottoneUno, colour, red),
    send(@dialog, display, @bottoneUno, point(480,350)),
    send(@bottoneUno, recogniser, click_gesture(left, '', single,
                message(@prolog, bottone_uno))),

    % Testo bottone Uno!
    free(@testoUno),
    new(@testoUno, text('UNO!')),
    send(@testoUno, font, font(helvetica, bold, 36)),
    send(@testoUno, colour, colour(red)),
    get(@testoUno, width, TestoWidth),
    get(@testoUno, height, TestoHeight),
    TestoX is 480 + (150 - TestoWidth) / 2,
    TestoY is 350 + (90 - TestoHeight) / 2,
    send(@dialog, display, @testoUno, point(TestoX, TestoY)),
```

- Ad ogni carta creata attraverso il predicato *crea_carta_giocata*, viene assegnato un oggetto che permette al box carta di chiamare un predicato ogni volta che ci si clicca sopra con il mouse. Ad ogni clic, si verifica se la carta è valida da giocare e, se si, viene rimossa dalla mano del giocatore e spostata al centro del tavolo. Se invece si clicca la carta mazzo al centro del tavolo, il giocatore pesca una carta.

```
gestisci_click(Carta) :-
    giocatore_attivo(GiocatoreAttivo),
    boxes_giocatore(ListaBox),
    carte_giocate([PrimaCarta|_]),
    cerca_carta(Carta, ListaBox,Valore,Colore),

    (   GiocatoreAttivo = 1
        ->
                        (
                            carta_valida(card(Valore,Colore),PrimaCarta)
                            ->
                            gioca_carta(card(Valore,Colore)),
                            cerca_e_rimuovi_carta(Carta, ListaBox, NuovaListaBox),
                            retract(boxes_giocatore(_)),
                            assertz(boxes_giocatore(NuovaListaBox)),
                            setta_mano_giocatore,
                            retractall(giocatore_attivo(_)),
                            assertz(giocatore_attivo(2)),
                            controlla_vittoria,
                            crea_carta_giocata,
                            (   Valore \= stop
                            ->
                            gioca_carta_ia
                            ;
                            retractall(giocatore_attivo(_)),
                            assertz(giocatore_attivo(1))
                            )
                            ;
                            writeln('carta non valida')
                        )
        ;
        writeln('Non è il tuo turno!')
    ).

crea_carte(Valore, Colore, X, Y) :-
    new(Carta, box(68,100)),

    (   Colore = yellow
    ->
    send(Carta, fill_pattern, colour(orange))
    ;
    (
        Colore = cambio
    ->
        send(Carta, fill_pattern, colour(black))
    )
    ;
    send(Carta, fill_pattern, colour(Colore))
    ),

    send(Carta, colour, white),
    send(Carta, radius, 7),
    send(@dialog, display, Carta, point(X,Y)),

    send(Carta, recogniser, click_gesture(left, '', single,
                message(@prolog, gestisci_click, Carta))),

    term_string(Valore, String),
    new(Testo, text(String)),
    (   String == "cambio"
    ->  send(Testo, string, 'C')
    ;
    true),
    send(Testo, font, font(helvetica, bold, 20)),
    send(Testo, colour, colour(white)),
    get(Testo, width, TestoWidth),
    get(Testo, height, TestoHeight),
    TestoX is X + (68 - TestoWidth) / 2,
    TestoY is Y + (100 - TestoHeight) / 2,
    send(@dialog, display, Testo, point(TestoX, TestoY)),
    boxes_giocatore(ListaBox),
    Box = [Carta,Testo,Valore,Colore],
    append([Box],ListaBox,NuovaListaBox),
    retract(boxes_giocatore(_)),
    assertz(boxes_giocatore(NuovaListaBox)).
```

- Per quanto riguarda il modo in cui vengono create le carte delle mani dei giocatori, ci sono quattro liste di coordinate *x* e *y*, che segnano le posizioni che avranno le carte all'interno della finestra. A ogni turno, le carte vengono rimosse e una volta aggiornata la mano vengono ricreate nella posizione giusta.

```
setta_mano_giocatore:-
    rimuovi_tutte_carte,
    mano_giocatore1(ManoGiocatore1),
    length(ManoGiocatore1, IndiceMax),
    findall(Valori,
             member(card(Valori, _), ManoGiocatore1),
            ListaValori),
    findall(Colori,
             (member(card(_, ColoreConSuff),ManoGiocatore1),
                rimuovi_suffisso_2(ColoreConSuff,Colori)),
            ListaColori),

    lista_X(ListaX),
    lista_Y(ListaY),


    IndiceMaxNuovo is IndiceMax-1,
    forall(
        between(0, IndiceMaxNuovo, Indice),
        (
            nth0(Indice, ListaValori, Valore),
            nth0(Indice, ListaColori, Colore),
            nth0(Indice, ListaX, PosizioneX),
            nth0(Indice, ListaY, PosizioneY),
            (crea_carte(Valore, Colore, PosizioneX, PosizioneY),
             flush_output))
    ).
```

- Una volta raggiunta la fine del gioco, viene cambiato lo sfondo, aggiornandolo con una scritta che segnala l'esito della partita, e viene mostrato un bottone per permettere di giocare ancora.

```
loss_condition :-
    send(@device, clear),
    free(@device),
    new(@device, device),
    send(@dialog, display, @device, point(0,0)),

    %sfondo
    source_file(win_condition, File1),
    rimuovi_file_da_percorso(File1, Risultato1),
    directory_file_path(Risultato1, '/Immagini/loss.jpg', NuovoPercorso1),
    new(Imagefile3, image(NuovoPercorso1)),
    new(Bitmap1, bitmap(Imagefile3)),
    send(@device, display, Bitmap1),
    send(Bitmap1, center, @device?center),

    %box replay
    free(@bottoneReplay),
    new(@bottoneReplay, box(300,75)),
    send(@bottoneReplay, fill_pattern, colour(black)),
    send(@bottoneReplay, radius, 10),
    send(@bottoneReplay, colour, white),
    send(@dialog, display, @bottoneReplay, point(200,400)),
    send(@bottoneReplay, recogniser, click_gesture(left, '', single,
                message(@prolog, start_the_game))),

    %testo replay
    free(@testoReplay),
    new(@testoReplay, text('Replay')),
    send(@testoReplay, font, font(helvetica, bold, 30)),
    send(@testoReplay, colour, colour(white)),
    get(@testoReplay, width, TestoWidth),
    get(@testoReplay, height, TestoHeight),
    TestoX is 200 + (300 - TestoWidth) / 2,
    TestoY is 400 + (75 - TestoHeight) / 2,
    send(@dialog, display, @testoReplay, point(TestoX, TestoY)),

    false.
```

- *Sviluppi futuri:* Per quanto riguarda gli sviluppi futuri, una delle possibili strade che si potrebbero prendere è quella di rivolgersi a *python* per gestire l'intera parte grafica del progetto, in quanto è molto più adatto per questo tipo di lavoro e offre una gamma di librerie e funzionalità molto ampia.

- *Github:* Per facilitare la collaborazione durante lo svolgimento del progetto, abbiamo deciso di creare una repository su Github, condivisa tra Diego e Lucia, in modo tale da riuscire ad aggiornare i file e i progressi in maniera efficiente e, in caso di problemi, usare i commit fatti nel tempo come fossero dei backup files.

Progetto svolto da:

- Diego Alejandro Salazar (1109929)
- Lucia Ugolini (1109538)