Gioco di carte con intelligenza artificiale

1 Introduzione 3

1.1 Informazioni sul progetto 3

1.2 Abstract 3

1.3 Scopo 3

2 Analisi 4

2.1 Analisi del dominio 4

2.2 Analisi dei costi 4

2.3 Analisi e specifica dei requisiti 4

2.4 Use case 6

2.5 Pianificazione 7

2.6 Analisi dei mezzi 8

2.6.1 Software 8

2.6.2 Hardware 8

3 Progettazione 8

3.1 Design delle interfacce 8

3.1.1 Design scelta della telecamera 8

3.1.2 Design della scelta delle prime carte 9

3.1.3 Design della pagina di gioco dell’IA 9

3.1.4 Design pagina di gioco del giocatore 10

3.1.5 Design pagina delle impostazioni 10

3.2 Design delle cartelle 11

3.3 Design delle classi 12

4 Implementazione 13

4.1 Intelligenza artificiale 13

4.1.1 Ciclo 13

4.1.2 Condizione 14

4.1.3 Salvataggio combinazione di carte 16

4.1.4 Aggiunta tabulazione 16

5 Test 17

5.1 Protocollo di test 17

5.2 Risultati test 17

6 Consuntivo 17

7 Conclusioni 17

7.1 Sviluppi futuri 17

7.2 Considerazioni personali 18

8 Bibliografia 18

8.1 Sitografia 18

9 Allegati 18

# Introduzione

## Informazioni sul progetto

* Alessandro Colugnat – Allievo
* Ugo Bernasconi – Docente Responsabile
* Scuola Arti e Mestieri Trevano, Informatica
* Data inizio: 08.01.2019
* Data fine: 10.04.2019

## Abstract

The job of this project is to create a robotic arm who plays card game called UNO and he plays with its own intelligence.

## Scopo

Lo scopo del progetto è quello di creare un’intelligenza artificiale che gioca a UNO e avrà una intelligenza che apprende autonomamente in maniera dinamica ciò servirà a far migliorare la IA ogni partita che verrà svolta, si deve implementare anche una videocamera che permette di riconoscere le carte di gioco e verificare che tutte le regole siano state applicate correttamente, per fare in modo che quando si gioca il computer riconosce le carte ed elabora la mossa che permette di battere l’avversario, verrà anche implementato un braccio robotico per poter fare in modo che il computer riesca a giocare le proprie carte a livello fisico.

Questa documentazione serve a introdurre gli utenti alla creazione del programma insieme anche al suo scopo del motivo per cui è stato creato.

Serve anche a introdurre su come è stata programmata l’intelligenza artificiale e come la macchina riesca ad auto apprendere le varie mosse che sono migliori per vincere e come fa a scegliere autonomamente quale strategia utilizzare durante la partita, la documentazione mostra anche come è stato reso possibile il riconoscimento delle carte tramite computer e come verificare che entrambi i giocatori riescano a seguire le regole impostate nel computer, fa vedere anche come la macchina riesca a gestire un braccio robotico che gestisce i movimenti scelti dalla intelligenza artificiale.

# Analisi

## Analisi del dominio

Questo progetto verrà utilizzato ogni volta che si vuole fare una partita a carte, un prodotto del genere esiste già e sono dei robot che giocano a scacchi oppure qualche altro gioco di carte, ma su internet non ho visto nessuno che ha costruito un robot con intelligenza artificiale che gioca a UNO.

## Analisi dei costi

|  |  |
| --- | --- |
| **Componenti** | **Prezzo** |
| 1 Lavoratore | 62 CHF/ ora |

Prezzo totale per 160 ore: 9920 CHF

## Analisi e specifica dei requisiti

|  |  |
| --- | --- |
| **ID: REQ-01** | |
| **Nome** | Creazione riconoscimento telecamera per carta in gioco |
| **Priorità** | 1 |
| **Note** | Si necessità di una buona videocamera con alta risoluzione per avere migliori prestazioni. |
| **Sotto requisiti** | |
| **001** | Si deve collegare la telecamera al computer |
| **002** | Prendere la carta e metterla davanti alla telecamera |
| **003** | Riconoscere il colore della carta |
| **004** | Riconoscere il numero della carta |

|  |  |
| --- | --- |
| **ID: REQ-02** | |
| **Nome** | Creazione riconoscimento telecamera per carta in possesso |
| **Priorità** | 1 |
| **Note** | Si necessità di una buona videocamera con alta risoluzione per avere migliori prestazioni. |
| **Sotto requisiti** | |
| **001** | Si deve collegare la telecamera al computer (oppure la telecamera del pc) |
| **002** | Vedere il colore della carta pescata |
| **003** | Vedere il numero della carta pescata |
| **004** | Salvare la posizione della carta |

|  |  |
| --- | --- |
| **ID: REQ-03** | |
| **Nome** | Creazione IA |
| **Priorità** | 1 |
| **Sotto requisiti** | |
| **001** | Controllare numero o colore delle carte IA siano uguale alla carta in gioco |
| **002** | Controllare se si possono fare altre combinazioni con le carte rimanenti |
| **003** | Scelta della mossa migliore tramite una percentuale di successo |
| **004** | Eseguire la mossa |

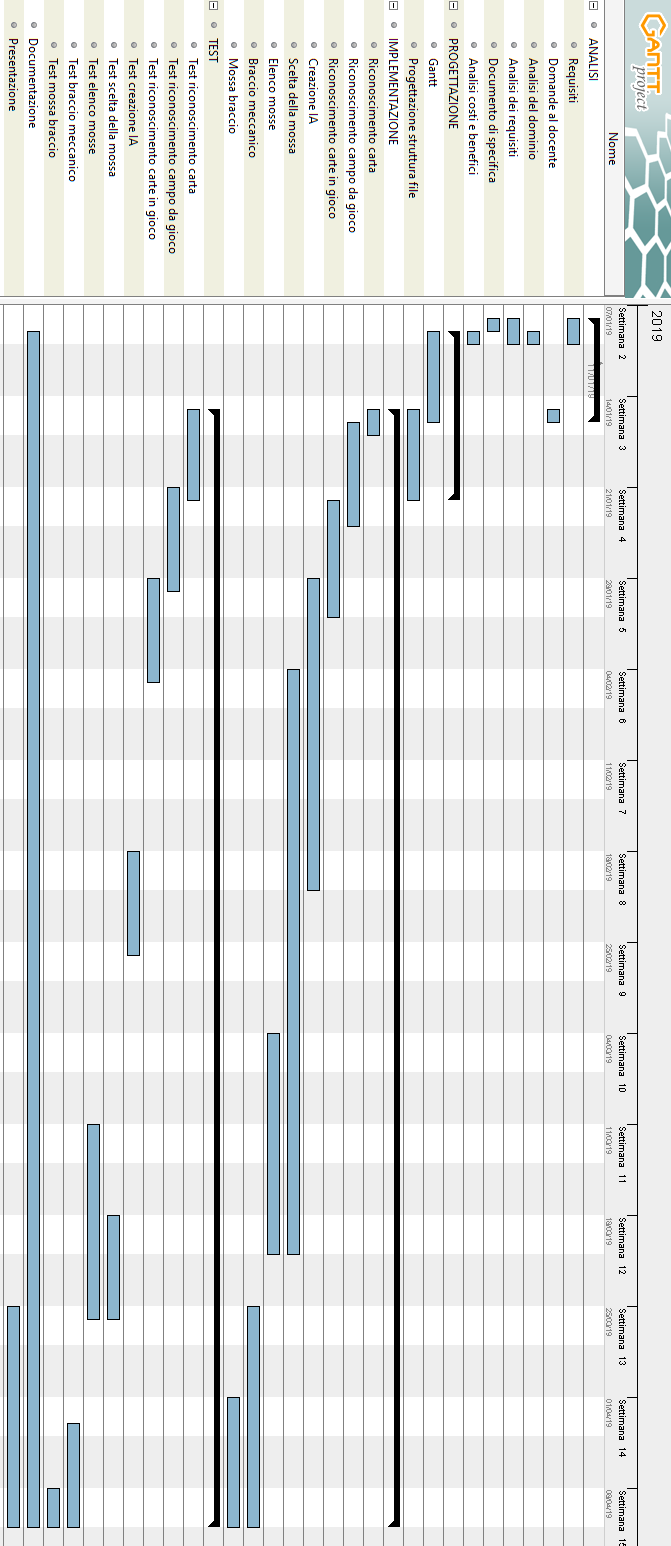
|  |  |
| --- | --- |
| **ID: REQ-04** | |
| **Nome** | Movimento braccio elettronico per mossa della carta |
| **Priorità** | 1 |
| **Note** | Si necessità di un braccio con minimo 5DOF |
| **Sotto requisiti** | |
| **001** | Aspettare la scelta della mossa dell’IA |
| **002** | Avere le coordinate della posizione della carta |
| **003** | Muovere il braccio nella posizione corretta e prendere la carta |
| **004** | Trasportare carta nella sezione della carte in gioco |
| **005** | Lasciare la carta e tornare in posizione iniziale |

|  |  |
| --- | --- |
| **ID: REQ-05** | |
| **Nome** | Movimento braccio elettronico per pescare la carta |
| **Priorità** | 1 |
| **Note** | Si necessità di un braccio con minimo 5DOF |
| **Sotto requisiti** | |
| **001** | Prendere le coordinate del mazzo di carte |
| **002** | Avvicinare il braccio elettronico al mazzo |
| **003** | Appiccicare la carta alla mano |
| **004** | Riconoscere la carta con la telecamera |
| **005** | Mettere la carta in posizione |
| **006** | Salvare le coordinate della carta nuova |

## Use case

I casi d’uso rappresentano l’interazione tra i vari attori e le funzionalità del prodotto.

## Pianificazione



Nella parte dell’implementazione ho scelto di mettere più importanza al riconoscimento delle carte e della intelligenza artificiale perché sono le parti principali del progetto, invece il braccio meccanico è secondario.

## Analisi dei mezzi

### Software

* Visual studio 2017
* EmGu CV / AForge.NET

### Hardware

* 1 PC portatile (Windows 10)
* Braccio con 5 DOF Arduino
* Raspberry 3B
* Telecamera

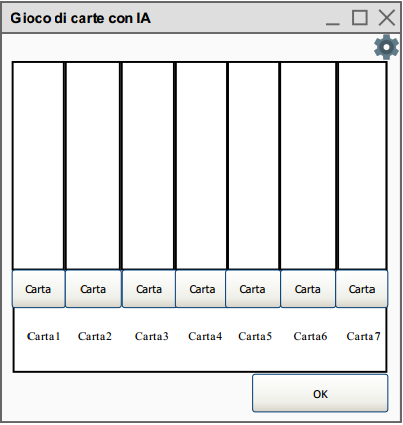
# Progettazione

## Design delle interfacce

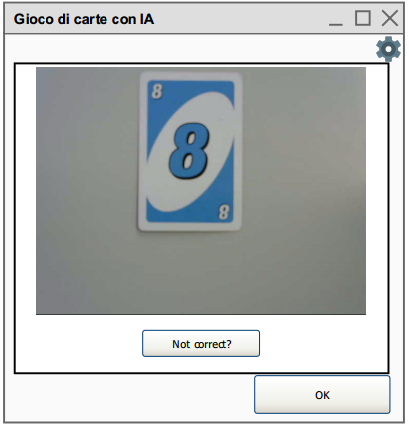
### Design scelta della telecamera



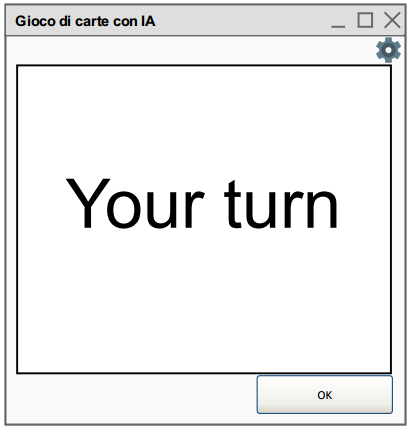
### Design della scelta delle prime carte



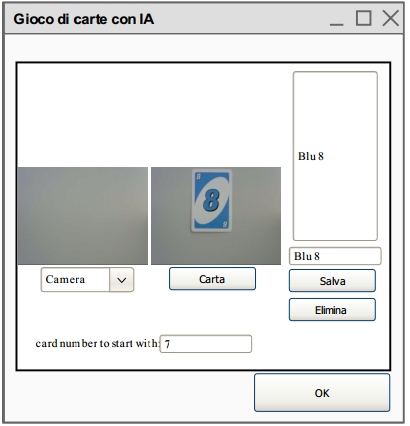
### Design della pagina di gioco dell’IA



### Design pagina di gioco del giocatore

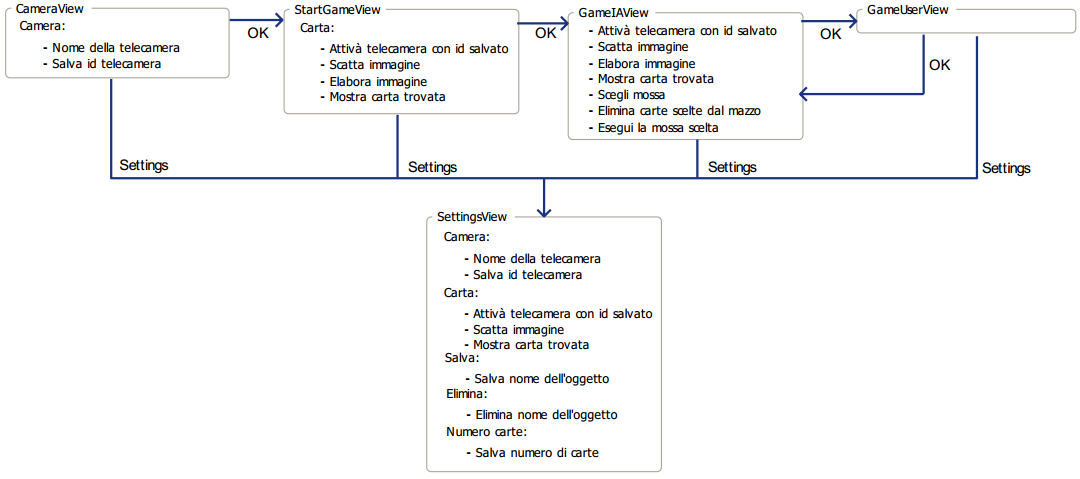


### Design pagina delle impostazioni



## Design delle cartelle

## Design delle classi



Questo rappresenta lo schema di funzionamento delle varie classi e ciò che devono fare, viene indicato anche quale passaggio si deve fare per proseguire di classe, le frecce indicano dei pulsanti, le immagini sono la parte fondamentale per il funzionamento totale del progetto.

# Implementazione

## Intelligenza artificiale

Come prima parte dell’implementazione verrà spiegata la logica di come funziona l’intero programma dell’intelligenza artificiale, il codice è corto ma da comprendere è molto complesso perché per fare questo pezzo di codice serviva pensare a ciò che fa il codice nei vari possibili casi, ho creato un algoritmo che permette di vedere tutte le possibili mosse:

private void SelectGame(String selectedCard, String selectedNum, bool nextIsMy)

{

for(int i = 0; i < cardAICopy.Length; i++)

{

if(((cardAICopy[i][0] == selectedCard) && !nextIsMy) || (cardAICopy[i][1] == selectedNum))

{

Console.WriteLine(AddTab(true)+cardAICopy[i][0]+" "+cardAICopy[i][1]+Environment.NewLine);

cardInGameCopy[0] = cardAICopy[i][0];

cardInGameCopy[1] = cardAICopy[i][1];

combinations.Add(cardInGameCopy[0] + " " + cardInGameCopy[1]);

cardAICopy[i][0] = "";

cardAICopy[i][1] = "";

SelectGame(cardInGameCopy[0], cardInGameCopy[1], true);

cardAICopy[i][0] = cardAI[i][0];

cardAICopy[i][1] = cardAI[i][1];

AddTab(false);

}

}

Save();

combinations.RemoveRange(0, combinations.Count);

}

Questo mostrato qui sopra è l’intero codice che permette di far mostrare tutte le mosse possibili con le carte che si ha in mano. Ora spiegherò la logica del funzionamento del codice.

Ci sono delle variabile che sono degli Array multidimensionali che vengono utilizzati in questo modo:

**cardAICopy[][]** 🡪 la prima parte dell’array e utilizzato per la carta, invece il secondo contiene il colore nell’indice 0 e invece il numero nell’indice 1.

Ci sono anche degli Array:

**cardInGameCopy[]** 🡪 nel primo indice si trova il colore della carta e invece nel secondo indice si trovava il numero della carta.

Ci sono anche delle variabili all’interno del codice:

**selectedCard** 🡪 il colore della carta.

**selectedNum** 🡪 il numero della carta.

**nextIsMy** 🡪 una variabile booleana.

Tutti queste variabili verranno spiegate nella documentazione sottostante e del motivo per qui sono state create.

### Ciclo

La prima parte che voglio introdurre è ii ciclo for, viene utilizzato semplicemente per far passare tutte le carte al controllo che verrà usato nel passaggio successivo, ho preferito utilizzare il ciclo for perché come ciclo verrà creata la variabile i che sarà molto importante alla fine del ciclo, il for verrà richiamato più volte a se stesso perché come verrà mostrato nella documentazione ci sarà il metodo stesso che verrà richiamato, quindi tutto ciò sarà necessario per controllare delle carte del proprio mazzo con altre carte che provengono dallo stesso mazzo.

### Condizione

Nella condizione subito dopo il for vengono controllate due condizioni molto importanti, che sono quelli del numero della carta e del colore della carta che viene utilizzata nel primo turno, tutto questo viene fatto per ogni carta che si possiede, tutto questo grazie al ciclo, e molto importante perché se non si gestisce bene questa condizione si rischia di creare della chiamate infinite ed inutili a se stesso e facendo così si creerebbe un ciclo infinito da bloccare il programma e far lavorare il computer in maniera inutile e pesante, ma con l’if che ho pensato riesce a gestire questi problemi e far riuscire a trovare una soluzione per il programma. Nelle condizione si può prendere il primo pezzo:

(cardAICopy[i][0] == selectedCard)

In questo pezzo di codice si fa il controllo delle due carte che siano dello stesso colore, questa regola del colore può essere applicata solamente con la carta che si trova in mezzo al campo. Infatti davanti a questo codice si può vedere che c’è stata l’aggiunta di una condizione molto importante:

((cardAICopy[i][0] == selectedCard) && !nextIsMy)

Si può notare la variabile che si chiama nextIsMy questa variabile booleana significa che la prossima carta è la mia, per spiegare meglio significa che se la carta in gioco è quella della IA, nella prossima mossa può decidere di fare un’altra mossa ma senza l’utilizzo del colore, perché secondo le regole, se vuoi mettere più di una carta per terra si dovrà prendere solamente il segno uguale e non il colore, e questa variabile fa questo tipo di controllo, infatti nella prossima condizione non si verrà aggiunta nessuna variabile nextIsMy:

(cardAICopy[i][1] == selectedNum)

In questo pezzo di codice viene fatto solamente il controllo dei numeri della carta in gioco con la carta posseduta. Tutto questo viene sommato con una porta OR per fare in modo che nel caso nextIsMy sia a false, la condizione possa comunque venire assolta grazie al numero della carta:

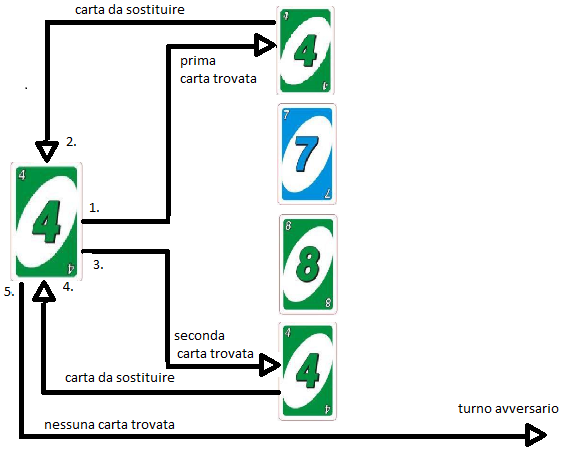
if(((cardAICopy[i][0] == selectedCard) && !nextIsMy) || (cardAICopy[i][1] == selectedNum))

#### Condizione falsa

Nel caso in cui i requisiti non soddisfacessero la condizione del metodo, non sarà causato alcun cambiamento nel metodo, ma passera alla carta successiva, fino a quando non ci saranno più carte da scegliere e in quel caso, finirà il ciclo delle carte.

#### Condizione vera

Se la condizione soddisfa i criteri che sono stati spiegati soprastante, allora inizierà, l’operazione più impegnativa che è quella di mostrare la carta scelta e richiamare il metodo stesso per fare di nuovo il controllo se si possono aggiungere nuove carte alla mossa. La logica del codice permette alla IA di poter scegliere tutte le mosse possibili, così facendo potrà scegliere la mossa migliore per fare in modo che l’avversario rimanga bloccato per poi essere sconfitto, nel codice viene richiamata lo stesso metodo che permette di controllare più volte la carta giocata, potrebbe crearsi un loop infinito se non si gestisce bene questa chiamata del metodo, in parte il problema è risolto grazie alla condizione ma il problema e soprattutto risolto grazie alla rimozione della carta trovata, così facendo non si creano continuamente chiamate del metodo con la stessa carta, per fare in questo modo, si deve togliere la carta prima della chiamata del metodo. Per questo motivo vengono utilizzate delle variabili che hanno copiato i dati dalla variabile madre, per fare in modo che venga cancellata la carta sul momento, ma il mazzo originale non viene toccato, appena si richiama il metodo vengono richieste tre valori, il colore, il numero e se la carta è quella che gioca la IA, così cambia la condizione dell’if. La logica del metodo è spiegata in maniera semplificata nello schema riportato qui sotto:



In questo schema viene mostrato il caso in cui ci fossero dei numeri uguali nella propria mano, si potrebbe fare la combinazione con il numero 8 verde, ma non ci sarebbe la possibilità di aggiungere il 4, per fare una combinazione maggiore.

### Salvataggio combinazione di carte

Appena finisce il ciclo delle carte, verrà richiamato un metodo che si chiama **Save()**, questo metodo permette di salvare le combinazioni trovate precedentemente, contiene queste informazioni:

private void Save()

{

if(combinations.Count != 0)

{

addCard += 1;

saveCombinations[addCard - 1] = combinations.ToList();

}

}

Questo metodo è molto semplice, perché viene aggiunta una combinazione ogni volta che finisce il metodo **SelectGame()**, perché se il metodo finisce significa che la combinazione è arrivata al termine, nel codice si può trovare la variabile **combinations** che serve a salvare la combinazione ogni volta che se ne trova una, e alla fine verrà salvata in un array, che poi sarà utilizzata in futuro per scegliere la mossa in maniera completamente casuale. Viene fatto un controllo nel metodo perché nel caso non ci fossero combinazione non si dovrebbe riempire inutilmente l’intero array. Alla fine la lista verrà completamente pulita per poi contenere una nuova combinazione:

combinations.RemoveRange(0, combinations.Count);

All’interno della lista verrà salvata come stringa, per poi alla fine quando si dovrà svolgere la mossa, si dovrà dividere la stringa e prendere i due valori principali che sono il colore e il numero:

combinations.Add(cardInGameCopy[0] + " " + cardInGameCopy[1]);

Una volta che sono state salvate tutte le combinazioni, dovrà essere scelte una di quelle mosse per farlo ho scelto di fare un numero casuale per scegliere la mossa così potrà fare tutti i tipi di mossa è non solo quelle in cui puoi mettere più carte alla volta:

Random num = new Random();

int value = num.Next(0, saveCombinations.Length);

saveCombinations[num.Next(0, saveCombinations.Length)];

Quando si trova la mossa verrà fatto in modo che le carte scelte vengano eliminate dall’array che contiene tutte le carte possedute dalla IA. Si utilizza un numero casuale che si trova dallo 0 fino al numero masso delle combinazioni che si possono fare. Infine la lista che contiene tutte le combinazioni verrà pulita per fare in modo che potrà venire utilizzato un’altra volta:

saveCombinations = null;

### Aggiunta tabulazione

Per semplicità, nel codice viene aggiunta una tabulazione per vedere in maniera più veloce le varie combinazione trovate dalla IA, questa parte serve ai programmatori per vedere se il codice funziona come si deve. I dati vengono stampati a terminale:

Console.WriteLine(AddTab(true)+cardAICopy[i][0]+" "+cardAICopy[i][1]+Environment.NewLine);

Nel caso in cui il richiamo del metodo ha come valore true, verrà aggiunta una tabulazione, invece se è false verrà tolta. Così non verrà aggiunta continuamente una tabulazione. Il metodo è composto in questo modo:

private String AddTab(bool remove)

{

tab = "";

if (remove)

countTab += 1;

else

countTab -= 1;

for(int i = 0; i < countTab; i++)

{

tab += " L ";

}

return tab;

}

### Eliminare carte utilizzate

Dopo che la mossa è stata eseguita si possono eliminare le carte che erano nel mazzo della IA:

private void DeleteCard(string delete)

{

for(int i = 0; i > Carta.CardAI.Length; i++)

{

if ((Carta.CardAI[i][0] + " " + Carta.CardAI[i][1]) == delete)

{

Carta.CardAI[i][0] = "";

Carta.CardAI[i][1] = "";

Carta.NumCard -= 1;

break;

}

}

}

Il codice è molto semplice perché serve solamente a cancellare il dato all’interno dell’array che contiene tutte le carte che possiede l’intelligenza artificiale. Alla fine rimarrà un buco nell’array che questo permetterà di essere riempito di nuovo nel caso in cui si pesca una nuova carta. Nel for viene passato come lunghezza del ciclo e la grandezza dell’array di CardAI che contiene 76 spazi, il motivo per cui si hanno 76 spazi all’interno dell’array e per fare in modo che in ogni caso si possano tenere tutte le carte del mazzo in mano e sono state tolte le carte speciali perché il programma gestisce soltanto i numeri. Quindi il for farà 76 cicli per controllare ogni possibilità che si abbia una carta, nel ciclo verrà fatto un controllo che permette di vedere se la carta selezionata nel ciclo sia identica alla carta che si deve eliminare. Appena si entra nella condizione si verranno cancellati i dati all’interno dell’array e decrementare una variabile che si spiegherà più in avanti. E alla fine si esce dal ciclo perché è inutile continuare a cercare altre carte dello stesso valore se la carta giocata è una soltanto.

La variabile NumCard serve a vedere quante carte rimangono all’intelligenza artificiale per fare in modo di vedere se si deve chiamare UNO oppure vedere se si ha vinto:

foreach (string comb in saveCombinations[num.Next(0, saveCombinations.Length)])

{

Thread.Sleep(1000);

Move.Content = comb;

DeleteCard(comb);

if (Carta.NumCard == 1)

{

Thread.Sleep(2000);

Move.Content = "UNO!";

}

else if (Carta.NumCard == 0)

{

Move.Content = "!!!!!! I WIN !!!!!!";

}

}

In questo pezzo di codice si vede l’utilità della variabile NumCard e si vede anche che processi si fanno per quando si avvia una mossa. Come primo passaggio si deve fare un foreach che fa passare tutte le carte che sono state trovate per la mossa che si è scelto dall’algoritmo descritto in precedenza. Come secondo passo ho fatto in modo che il programma venga fermato per 1 secondo, perché ogni volta che fa il foreach si deve mostrare quale carta si utilizza per la mossa, dopo si chiama il metodo che permette di eliminare la carta dall’array, dopo che si è eliminata la carta si controlla che la prossima carta sia la penultima e nel caso si scrive UNO! oppure se è l’ultima si deve scrivere !!!!!! I WIN !!!!!! queste informazioni servono per notificare l’avversario su quante carte possiede ancora la IA.

### Pescare una nuova carta

## Riconoscimento delle carte

### Utilizzo videocamera

### Comandi per salvare carta da riconoscere

### Riconoscere una carta

## Utilizzo delle interfacce

### Passaggio dei dati

### Controlli per cambiare pagina

# Test

## Protocollo di test

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | Primo | **Nome:** | Funzionamento python |
| **Descrizione:** | Provare a testare le funzioni e metodi di python | | |
| **Prerequisiti:** | Per far funzionare python si deve avere una piattaforma per far compilare il codice di python | | |
| **Procedura:** | 1. Provare a far stampare una scritta 2. Testare una somma di numeri o di testo 3. Creare un’array e verificare i possibili funzionamenti 4. Provare a mettere i tipi di costruttori condizionali 5. Testare un ciclo 6. Provare a creare funzioni | | |
| **Risultati attesi:** | Ogni singolo punto ha funzionato ci sono stati errori ma li sono risolti | | |

## Risultati test

|  |  |
| --- | --- |
| **Test** | **Risultati** |
| Provare a far stampare una scritta | Provato a stampare “Hello world” all’inizio non andava perché non avevamo messo le parentesi ma poi ci siamo corretti e abbiamo messo le parentesi e tutto ha funzionato |
| Testare una somma di numeri o di testo | Non ci sono stati problemi nella somma |
| Creare un’array e verificare i possibili funzionamenti | È andato tutto bene nel funzionamento degli array |
| Provare a mettere i tipi di costruttori condizionali | Non ci sono stati problemi |
| Testare un ciclo | Il ciclo ha avuto un problema perché eravamo abituati in un altro linguaggio ma poi abbiamo risolto il problema facendo una ricerca su internet |

# Consuntivo

Consuntivo del tempo di lavoro effettivo e considerazioni riguardo le differenze rispetto alla pianificazione (cap 1.7) (ad esempio Gannt consuntivo).

# Conclusioni

Quali sono le implicazioni della mia soluzione? Che impatto avrà? Cambierà il mondo? È un successo importante? È solo un’aggiunta marginale o è semplicemente servita per scoprire che questo percorso è stato una perdita di tempo? I risultati ottenuti sono generali, facilmente generalizzabili o sono specifici di un caso particolare? ecc

## Sviluppi futuri

Migliorie o estensioni che possono essere sviluppate sul prodotto.

## Considerazioni personali

Cosa ho imparato in questo progetto? ecc

# Bibliografia

## Sitografia

* <https://www.codeproject.com/Articles/125478/Versatile-WebCam-C-library> 16.01.2019
* <http://hemant-srivastava.blogspot.com/2012/11/image-color-detector-in-c.html> 16.01.2019

# Allegati

* Diari di lavoro