UNIVERSITÀ
DEGLI STUDI
DI BERGAMO

Quantum IDE: Manuale Utente

Maatoug Sabrin, mat.1065576

November 8, 2025

Contents

1 Installazione e Avvio **3**
- 1.1 Prerequisiti . . . 3
- 1.2 Intallazione . . . 3
- 1.3 Avvio dell'applicazione . . . 3
- 1.4 Vincoli e Limitazioni . . . 3

2 Funzionalità **4**
- 2.1 Scrittura o importazione del codice . . . 4
- 2.2 Esecuzione del codice . . . 4
- 2.3 Visualizzazione del circuito . . . 5
- 2.4 Risultati della simulazione . . . 5
- 2.5 Salvataggio del codice . . . 6

3 Grammatica Utente del Linguaggio **6**
- 3.1 Dichiarazioni . . . 6
- 3.2 Gate . . . 6
- 3.3 Definizione di Gate e Funzioni da parte dell'utente . . . 6
- 3.4 Misura e Reset . . . 7
- 3.5 Controllo di flusso . . . 7
- 3.6 Funzioni matematiche . . . 7
- 3.7 Operatori e letterali . . . 7

4 Esempi di Utilizzo **8**

5 Esempi di Errori Comuni **9**
- 5.1 Errori Lessicali . . . 10
- 5.2 Errori Sintattici . . . 10
- 5.3 Errori Semantici . . . 11

1 Installazione e Avvio

1.1 Prerequisiti

Per utilizzare **Quantum IDE** è necessario:

- **Java Runtime Environment (JRE) 17+** installato sul sistema.
- **Python 3** con la libreria `Qiskit` (`pip install qiskit`).
- Un terminale funzionante (cmd, PowerShell o simili su Windows).

1.2 Intallazione

Il progetto può essere scaricato come file JAR precompilato direttamente dal repository GitHub:

- Repository: https://github.com/sabrin99m/quantum-ide
- Scaricare il file `quantum-ide.jar` dalla sezione `Releases` oppure compilare il progetto localmente.

1.3 Avvio dell'applicazione

1. Aprire il terminale nella cartella dove si trova il file `QuantumIDE.jar`.
2. Eseguire il comando:

```
java -jar QuantumIDE.jar
```

3. L'applicazione si avvierà e aprirà l'interfaccia grafica.

1.4 Vincoli e Limitazioni

- L'applicazione è stata testata su **Windows**. L'esecuzione su altri sistemi non è garantita.
- La gestione dei percorsi e dei file dipende dalla cartella di avvio del terminale.
- Attualmente l'IDE può eseguire codice solo attraverso il backend Python con Qiskit installato.

2 Funzionalità

2.1 Scrittura o importazione del codice

L'utente può scrivere il codice quantistico direttamente nell'**Editor Panel** oppure importare un file esistente:

1. Per scrivere manualmente, digitare le istruzioni nella finestra dell'editor.
2. Per importare un file, cliccare sul pulsante **Apri** nella toolbar e selezionare il file.

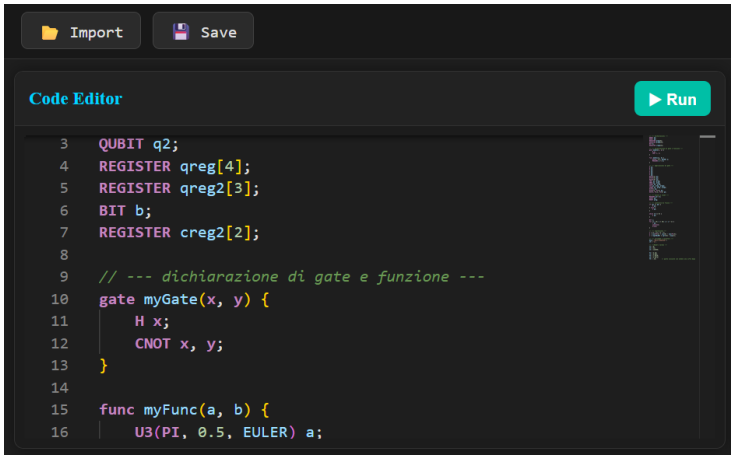

Figure 1: Editor Panel: scrittura o importazione del codice quantistico.

Eventuali errori lessicali vengono evidenziati immediatamente nel codice tramite sottolineature rosse.

2.2 Esecuzione del codice

Dopo aver scritto o importato il codice:

1. Cliccare sul pulsante **Run** nella toolbar.
2. Il codice viene inviato al backend per la compilazione.
3. Eventuali errori di sintassi o semantica vengono mostrati nel panel di output.

2.3 Visualizzazione del circuito

Il circuito quantistico generato viene visualizzato in ASCII nel **Circuit Panel**.

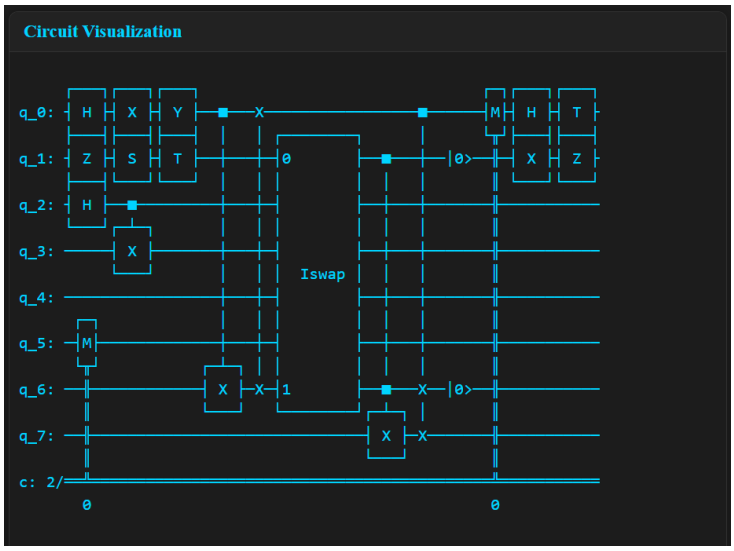

Figure 2: Circuit Panel: visualizzazione del circuito quantistico in formato ASCII.

2.4 Risultati della simulazione

I risultati della simulazione sono mostrati nel **Simulation Panel**, che riporta:

- La distribuzione delle probabilità dei bit classici (numero di misurazioni sul totale).
- Gli stati finali del circuito.

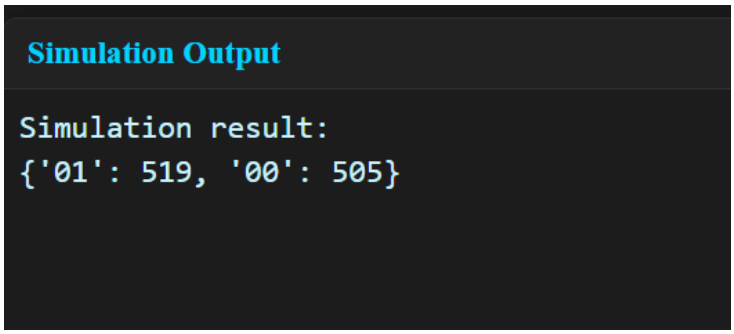

Figure 3: Simulation Panel: risultati della simulazione dei circuiti.

2.5 Salvataggio del codice

L'utente può salvare il codice corrente premendo il pulsante **Salva** nella toolbar e scegliendo il percorso del file. In questo modo è possibile riprendere il lavoro successivamente.

3 Grammatica Utente del Linguaggio

Di seguito sono riportate le principali regole sintattiche che l'utente può utilizzare per scrivere codice nell'IDE Quantum.

3.1 Dichiarazioni

- **Qubit singolo:** `QUBIT q0`
- **Registro di qubit:** `REGISTER q[3]`
- **Bit singolo:** `BIT c0`
- **Registro di bit:** `REGISTER c[3]`

3.2 Gate

- Gate standard: `H <qubit>`, `X <qubit>`, `Y <qubit>`, `Z <qubit>`
- Gate a due o tre qubit: `CX <qubit1> <qubit2>`, `CCX <qubit1> <qubit2> <qubit3>`
- Gate parametrizzati: `RX(<angolo>) <qubit>`, `U1(<param>) <qubit>`, ...

3.3 Definizione di Gate e Funzioni da parte dell'utente

- Definizione di un nuovo gate:

```
GATE myGate(q0, q1) {
    H q0
    CX q0 q1
}
```

- Definizione di una nuova funzione:

```
FUNC myFunction(q0, q1) {
    RX(1.57) q0
    CCX q0 q1 q2
}
```

3.4 Misura e Reset

- `MEASURE <qubit>` → `<bit_classico>`
- `RESET <qubit>`

3.5 Controllo di flusso

- `if (<condizione>) { ... } [else { ... }]`
- `while (<condizione>) { ... }`
- `for (<inizializzazione>; <condizione>; <incremento>) { ... }`
- `break;` e `continue;`

3.6 Funzioni matematiche

`sin(<expr>)`, `cos(<expr>)`, `tan(<expr>)`, `log(<expr>)`, `sqrt(<expr>)`, `exp(<expr>)`

3.7 Operatori e letterali

- Operatori aritmetici: `+ - * / $$%`
- Operatori logici: `&& || !`
- Confronto: `== != < <= > >=`
- Letterali: `0 1 true false PI E SQRT2`

4 Esempi di Utilizzo

Di seguito è riportato un esempio completo che mostra tutte le funzionalità del linguaggio Quantum: dichiarazioni di qubit e bit, definizione di gate e funzioni, applicazioni di gate, misure, reset, controllo di flusso, espressioni matematiche e gestione di stringhe e numeri.

```
// --- dichiarazioni ---
QUBIT q1;
QUBIT q2;
REGISTER qreg[4];
REGISTER qreg2[3];
BIT b;
REGISTER creg2[2];

// --- dichiarazione di gate e funzione ---
gate myGate(x, y) {
    H x;
    CNOT x, y;
}

func myFunc(a, b) {
    U3(PI, 0.5, EULER) a;
    MEASURE a -> b;
}

// --- applicazioni di gate ---
H q1;
X q1;
Y q1;
Z q2;
S q2;
T q2;
RX(PI/2) q1;
RY(PI/4) q2;
RZ(PI/8) q2;
CNOT q1, qreg;
SWAP q1, qreg;
ISWAP q2, qreg;
CCX q2, qreg, qreg2;
CSWAP q1, qreg, qreg2;
U1(PI) q1;
U2(PI/2, PI/4) q2;
U3(PI, PI/2, PI/3) q2;

// --- misura e reset ---
MEASURE q1 -> b;
```

```
RESET q2;
RESET qreg;

// --- controllo di flusso ---
if (q1 == q2) {
    H q1;
} else {
    X q2;
}

while (b == 0) {
    Z q2;
}

BIT i;
for (i = 0; i < 10; i = i + 1) {
    T q1;
    continue;
    break;
}

// --- espressioni ---
x = (1 + 2) * 3 ^ 2;
y = sin(PI/2) + cos(0) - tan(PI/4);
z = log(EULER) * sqrt(4) / exp(1);

// --- stringhe e caratteri ---
msg = "Hello␣Quantum!";
ch = 'a';

// --- numeri validi ---
i1 = 0;
i2 = 42;
i3 = 123456;

f1 = 3.14;
f2 = 0.001;
f3 = 1.0e10;
f4 = 2.5E-3;
f5 = .25;
```

5 Esempi di Errori Comuni

Di seguito sono riportati esempi di codice che generano errori di vario tipo. Gli errori sono divisi in tre categorie: lessicali, sintattici e semantici.

5.1 Errori Lessicali

```
// Caratteri non previsti
QUBIT q$;
H q#;
X y@;

// Numeri mal formattati
123.a
.456E+
1e

// Stringhe o char non chiusi
'c
"stringa
```

5.2 Errori Sintattici

```
// Dichiarazioni senza punto e virgola
QUBIT q
BIT b

// Registro senza parentesi quadre corrette
REGISTER r1 5;

// Gate sconosciuto
Hadamard q;

// Gate con sintassi errata
CNOT(q q2;

// Funzione senza parentesi chiusa
func myFunc(a, b {
    H a;
}

// Applicazioni gate con errori
SWAP q r1;
CCX q, r1 q2;
U3(PI, 0.5, EULER a;

// Misura e reset errati
MEASURE ;
MEASURE q b;
RESET ;

// Controllo di flusso con errori
```

```
if (q == q1 {
    H q;
}
while q < q2 {
    X q;
}
for (i = 0 i < 10 i++) {
    Y q;
}
break
continue

// Espressioni malformate
x = 1 + 2
y = sin(PI/2) + cos(0) - tan(PI/4)
z = (1 + 2 * 3;
f1 = 10.20.30;
f2 = 20a;
```

5.3 Errori Semantici

```
// Variabile gia dichiarata
QUBIT q1;
QUBIT q1;

// Variabile non dichiarata
H undeclaredQubit;

// Gate applicato a variabile non qubit
BIT b1;
H b1;

// Gate multi-qubit con tipo sbagliato
QUBIT q2;
CNOT q2, b1;
SWAP q2, b1;
ISWAP q2, b1;
CCX q2, b1, q2;
CSWAP q2, b1, q2;

// Misura con destinazione errata
MEASURE q2 -> q2;

// Assegnazione a variabile non dichiarata
undeclared = 42;
```