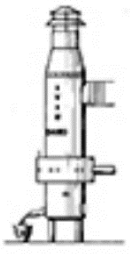
2022



ISTITUTO TECNICO INDUSTRIALE "G. B. BOSCO LUCARELLI" BENEVENTO

NEGOZIO

MASONE ALESSANDRO

RAGIONAMENTO LOGICO

Presi i dati in input ed effettuato il controllo su di essi, decido dove posizionare le entrate ed uscite, successivamente genero i clienti con i seguenti passi:

1. Nascita del cliente
2. Attesa del cliente che si rechi al negozio
3. Ricerca della coda più corta (in maniera approssimata per via dell'errore umano)
4. Presa di posto all'interno della coda minore partendo dall'inizio e scalando fino a trovare il posto disponibile
5. Controllo che la persona davanti a lui se ne vada, fino ad arrivare in prima posizione utile per l'entrata nel negozio
6. Controllo di un pass disponibile per l'accesso al negozio
7. Cammino all'interno del negozio
8. Decisione di uscita
9. Intraprendere la strada più breve per l'uscita
10. Rilasciare il pass
11. Abbandonare il negozio

DESCRIZIONE CODICE

```
/// Parser delle informazini nel file TXT
/// @param vettore vettore dopo immettere i dati
void Funzione::inputdati(int vettore[]) {
    char s[65000] = {};
    FILE *fp;
    char *line = NULL;
    size_t len = 0;
    ssize_t read;
    int init_size = 0;
    char *ptr;
    int i = 0;
    int j = 0;
    fp = fopen(FILENAME, "r");
    while ((read = getline(&line, &len, fp)) != -1) {
        init_size = (int)strlen(line);
        ptr = strtok(line, ":");
        while (ptr != NULL) {
            if (i %2 != 0) {
                strcpy(s, ptr);
                s[strlen(s)] = '\0';
                vettore[j] = atoi(s);
                j++;
            }
            ptr = strtok(NULL, ":");
            i++;
        }
    }
    contaclienti = 0;
}
```

Effettua il parser sul file txt per immettere i dati all'interno di un vettore che poi verrà utilizzato per associare i valori base

```
/// Assegnazione decisione uscita
/// @param valore valore da assegnare alla variabile della decisione
void Funzione::assegnadecisioneuscita(int valore) {
    decisione_uscita = valore;
}
/// Salva i dati della velocità del movimento
/// @param min valore minimo
/// @param max valore massimo
void Funzione::assegnarangetempo(int min, int max) {
    velocita_minima = min;
    velocita_massima = max;
}
/// Assegnazione tempo di arrivo del cliente
/// @param valore valore da assegnare
void Funzione::assegnatempoarrivo(int valore) {
    tempo_arrivo = valore;
}
/// Assegnazione numero clientie
/// @param valore valore da assegnare
void Funzione::assegnanumeroclienti(int valore) {
    numero_clienti = valore;
}
/// Assegnazioni delle dimensioni dei lati
/// @param x valore X
/// @param y valore Y
void Funzione::assegnadimensioni(int x, int y) {
    dimensionelato.assegnax(x);
    dimensionelato.assegnay(y);
}
/// Presa del numero di entrate
/// @param valore_numero_entrate numero di entrate
void Funzione::assegnanumeroentrate(int valore_numero_entrate) {
    numero_entrate = valore_numero_entrate;
}
/// Presa del numero di uscite
/// @param valore_numero_uscite numero di uscite
void Funzione::assegnanumerouscite(int valore_numero_uscite) {
    numero_uscite = valore_numero_uscite;
}
```

Assegna i valori all’interno della zona privata della classe delle funzionalità per mantenerne traccia

```
/// Inizializzazione dei passi e sblocco posti
/// @param valore numero di pass da creare
void Funzione::inizializzapass(int valore) {
    semaforo.resize(valore);
    for (int i =  0; i < semaforo.size(); i++) {
        semaforo[i].utente = -1;
        semaforo[i].pass.unlock();
    }
}
/// Associaizone valore base dei clienti
/// @param persona vettore delle persone
void Funzione::inizializzaclienti(vector<Persona> &persona) {
    srand((unsigned int)time(NULL));
    persona.resize(numero_clienti);
    for (int i = 0; i < persona.size(); i++) {
        persona[i].assegnaidentificativo(i);
        persona[i].movimentofermo();
        persona[i].inizializzaspostamenti();
        persona[i].assegnavelocita(rand() % velocita_massima + velocita_minima);
    }
}
/// Assegnazione valori base delle code
/// @param coda vettore code
/// @param mattonella base negozio
/// @param valore lunghezza massima della coda
void Funzione::inizializzacoda(vector<Coda> &coda, vector<vector<Mattonella>> &mattonella, int
valore) {
    coda.resize(numero_entrate);
    for (int i = 0; i < coda.size(); i++) {
        coda[i].assegnaidentificativo(i);
        coda[i].assegnaposti(numero_clienti);
        for (int j = 0; j < numero_clienti; j++) {
            coda[i].assegnaidentificativoutente(-1, j);
            coda[i].sbloccaposto(j);
        }
    }
    int z = 0;
    for (int x = 0; x < dimensionelato.dammix(); x++) {
        for (int y = 0; y < dimensionelato.dammiy(); y++) {
            if (mattonella[x][y].dammitipo() == 1) {
                coda[z].assegnacoordinateentrata(x, y);
                z++;
            }
        }
    }
}
/// Assegnazione dimensioni al negozio
/// @param mattonella base del negozio
void Funzione::inizializzadimensioninegozio(vector<vector<Mattonella>> &mattonella) {
    mattonella.resize(dimensionelato.dammiy());
    for (int x = 0; x < dimensionelato.dammix(); x++) {
        mattonella[x].resize(dimensionelato.dammiy());
    }
    for (int x = 0; x < mattonella.size(); x++) {
        for (int y = 0; y < mattonella[x].size(); y++) {
            mattonella[x][y].assegnautente(-1);
        }
    }
}
```

Inizializzazione dei requisiti base per le varie strutture

```
/// RItorno numero di clienti
int Funzione::damminumeroclienti() {
    return numero_clienti;
}
/// Ritorno del perimetro del negozio
int Funzione::calcolaperimetro() {
    return ((dimensionelato.dammix()-1)+(dimensionelato.dammiy()-1)+(dimensionelato.dammix()-
1)+(dimensionelato.dammiy()-1));
}
```

Return dei valori indicati all’interno della funzione

```
/// Assegnazione delle uscite lungo il periemtro
/// @param mattonella base del negozio
void Funzione::assegnauscite(vector<vector<Mattonella>> &mattonella) {
    int lunghezzabordi = calcolaperimetro();
    vector<int> a(numero_uscite);
    randomizzavettore(a, lunghezzabordi);
    int numeroassegnamenti = 0;
    lunghezzabordi = 0;
    for (int x = 0; x < dimensionelato.dammix(); x++) {
        for (int y = 0; y < dimensionelato.dammiy(); y++) {
            if (x == 0 || x == dimensionelato.dammix()-1) {
                if (mattonella[x][y].dammitipo() == 0){
                for (int z = 0; z < a.size() ; z++) {
                    if (lunghezzabordi == a[z]){
                        mattonella[x][y].assegnauscita();
                        numeroassegnamenti++;
                        }
                    }
                    lunghezzabordi++;
                }
            }
            else if (y == 0 || y == dimensionelato.dammiy()-1) {
                if (mattonella[x][y].dammitipo() == 0){
                    for (int z = 0; z < a.size() ; z++) {
                        if (lunghezzabordi == a[z]){
                            mattonella[x][y].assegnauscita();
                            numeroassegnamenti++;
                        }
                    }
                    lunghezzabordi++;
                }
            }
        }
    }
}
/// Asswegnazione entrate lungo il perimetro
/// @param mattonella base del negozio
void Funzione::assegnaentrate(vector<vector<Mattonella>> &mattonella) {
    int lunghezzabordi = calcolaperimetro();
    vector<int> a(numero_entrate);
    randomizzavettore(a, lunghezzabordi);
    int numeroassegnamenti = 0;
    lunghezzabordi = 0;
    for (int x = 0; x < dimensionelato.dammix(); x++) {
        for (int y = 0; y < dimensionelato.dammiy(); y++) {
            if (x == 0 || x == dimensionelato.dammix()-1) {
                if (mattonella[x][y].dammitipo() == 0){
                for (int z = 0; z < a.size() ; z++) {
                    if (lunghezzabordi == a[z]){
                        mattonella[x][y].assegnaentrata();
                        numeroassegnamenti++;
                        }
                    }
                    lunghezzabordi++;
                }
            }
            else if (y == 0 || y == dimensionelato.dammiy()-1) {
                if (mattonella[x][y].dammitipo() == 0){
                    for (int z = 0; z < a.size() ; z++) {
                        if (lunghezzabordi == a[z]){
                            mattonella[x][y].assegnaentrata();
                            numeroassegnamenti++;
                        }
                    }
                    lunghezzabordi++;
                }
            }
        }
    }
}
```

Assegnazione di entrate ed uscite alla matrice del negozio

```
/// Gestisce la vita del cliente e crea tutte le sue azioni
/// @param identificativo numero della persona
/// @param mattonella base del negozio
/// @param coda code da analizzare
/// @param persona tutte le persone presenti
/// @param type valore del debug
void Funzione::azionicliente(int identificativo, vector<vector<Mattonella>> &mattonella,
vector<Coda> &coda, vector<Persona> &persona, int type) {
    ///Attesa arrivo cliente al negozio
    int e = rand()% tempo_arrivo + 1;
    if (type == 2) {
        stampa.lock();
        cout << identificativo << ": " << "Tempo di attesa per l'arrivo " << e << endl;
        stampa.unlock();
    }
    sleep(e);
```

Parte dell'attesa dell'arrivo del cliente al negozio per poi prendere la coda minore

```
    ///Calcolo lunghezza delle varie code
    if (type == 2) {
        stampa.lock();
        cout << identificativo << ": " << "Calcolo lunghezza delle code " << endl;
        stampa.unlock();
    }
    vector<int> lunghezzacoda(coda.size());
    for (int i = 0; i < coda.size(); i++) {
        lunghezzacoda[i] = 0;
        for (int j = 0; j < persona.size(); j++) {
            if (coda[i].dammiidentificatoreutente(j) != -1) {
                lunghezzacoda[i]++;
            }
        }
    }
```

Parte per il calcolo delle persone all'interno delle varie code (con errore umano)

```
    ///Sort per la coda più piccola
    int codaminore = 0;
    for (int i = 1; i < coda.size(); i++) {
        if (lunghezzacoda[i] < lunghezzacoda[codaminore]) {
            codaminore = i;
        }
    }
    if (type == 2) {
        stampa.lock();
        cout << identificativo << ": " << "Coda minore " << codaminore << endl;
        stampa.unlock();
    }
    persona[identificativo].assegnacoda(codaminore);
```

Sorte tra tutte le code dimensioni delle code per trovare quella minore

```
    ///Presa di posizione all'interno della coda minore
    int posizionecoda;
    for (int j = 0; ; j++) {
        if (coda[codaminore].provabloccare(j) != 0) {
            posizionecoda = j;
            coda[codaminore].assegnaidentificativoutente(identificativo, j);
            if (type == 2) {
                stampa.lock();
                cout << identificativo << ": " << "Poszione nella coda " << j << endl;
                stampa.unlock();
            }
            break;
        }
    }
```

Partendo dall'inizio vede in che posizione c'è il posto disponibile per lui

```
    ///Scala di posto fino all'arrivo del prima posto utile per acceidere al pass/entrata negozio
    while (true) {
        if (posizionecoda > 0) {
            if (coda[codaminore].provabloccare(posizionecoda-1) != 0) {
                if (type == 2) {
                    stampa.lock();
                    cout << identificativo << ": " << "Posizione nella coda " << posizionecoda <<
endl;
                    stampa.unlock();
                }
```

```
                coda[codaminore].sbloccaposto(posizionecoda);
                coda[codaminore].assegnaidentificativoutente(-1, posizionecoda);
                posizionecoda--;
                coda[codaminore].assegnaidentificativoutente(identificativo, posizionecoda);
            }
        }
        else {
            break;
        }
    }
```

Questo rappresenta il cammino all'interno della coda per arrivare in prima posizione

```
    ///Presa del pass per il cliente
    int pass = -1;
    while (true) {
        for (int i = 0; i < semaforo.size(); i++) {
            if (semaforo[i].pass.try_lock()) {
                pass = i;
                semaforo[pass].utente = identificativo;
                coda[codaminore].sbloccaposto(posizionecoda);

mattonella[coda[codaminore].dammicoordinateentrata('x')][coda[codaminore].dammicoordinateentrata(
'y')].bloccamattonella();

mattonella[coda[codaminore].dammicoordinateentrata('x')][coda[codaminore].dammicoordinateentrata(
'y')].assegnautente(identificativo);

mattonella[coda[codaminore].dammicoordinateentrata('x')][coda[codaminore].dammicoordinateentrata(
'y')].assegnautente(identificativo);

persona[identificativo].assegnaentrata(coda[codaminore].dammicoordinateentrata('x'),
coda[codaminore].dammicoordinateentrata('y'));

persona[identificativo].assegnaposizione(coda[codaminore].dammicoordinateentrata('x'),
coda[codaminore].dammicoordinateentrata('y'));
                calc.lock();
                contaclienti++;
                calc.unlock();
                if (type == 2) {
                    stampa.lock();
                    cout << identificativo << ": " << "Presa del pass numero " << pass << endl;
                    stampa.unlock();
                    stampa.lock();
                    cout << "Clienti dentro " << contaclienti << endl;
                    stampa.unlock();
                }
                break;
            }
        }
        if (pass != -1) {
            break;
        }
    }
```

Arrivato in prima posizione verifica la disponibilità del pass per poi poter entrare

```
    ///Movimento e decisione di uscite del cliente
    while (true) {
        int decisione = 0;
        int direzione = FERMO;
        direzione = rand() % 4 + 1;
        if (type == 2) {
            stampa.lock();
            cout << identificativo << ": " << "Posizione attuale " <<
persona[identificativo].dammicoordinate('x') << "-" <<
persona[identificativo].dammicoordinate('y') << endl;
            stampa.unlock();
        }
        switch (direzione) {
            case SOPRA:
                if (persona[identificativo].dammicoordinate('y') +1 > dimensionelato.dammiy()-1)
{
                }
                else {
```

```
                    if
(mattonella[persona[identificativo].dammicoordinate('x')][persona[identificativo].dammicoordinate
('y')+1].provabloccamattonella()) {

mattonella[persona[identificativo].dammicoordinate('x')][persona[identificativo].dammicoordinate(
'y')].assegnautente(-1);

mattonella[persona[identificativo].dammicoordinate('x')][persona[identificativo].dammicoordinate(
'y')].sbloccamattonella();
                        persona[identificativo].incrementay();
                        persona[identificativo].incrementaspostamenti();

mattonella[persona[identificativo].dammicoordinate('x')][persona[identificativo].dammicoordinate(
'y')].assegnautente(identificativo);
                    }
                }
                break;
            case SOTTO:
                if (persona[identificativo].dammicoordinate('y') -1 < 0) {
                }
                else {
                    if
(mattonella[persona[identificativo].dammicoordinate('x')][persona[identificativo].dammicoordinate
('y')-1].provabloccamattonella()) {

mattonella[persona[identificativo].dammicoordinate('x')][persona[identificativo].dammicoordinate(
'y')].assegnautente(-1);

mattonella[persona[identificativo].dammicoordinate('x')][persona[identificativo].dammicoordinate(
'y')].sbloccamattonella();
                        persona[identificativo].decrementay();
                        persona[identificativo].incrementaspostamenti();

mattonella[persona[identificativo].dammicoordinate('x')][persona[identificativo].dammicoordinate(
'y')].assegnautente(identificativo);
                    }
                }
                break;
            case DESTRA:
                if (persona[identificativo].dammicoordinate('x') +1 > dimensionelato.dammix()-1)
{
                }
                else {
                    if
(mattonella[persona[identificativo].dammicoordinate('x')+1][persona[identificativo].dammicoordina
te('y')].provabloccamattonella()) {

mattonella[persona[identificativo].dammicoordinate('x')][persona[identificativo].dammicoordinate(
'y')].assegnautente(-1);

mattonella[persona[identificativo].dammicoordinate('x')][persona[identificativo].dammicoordinate(
'y')].sbloccamattonella();
                        persona[identificativo].incrementax();
                        persona[identificativo].incrementaspostamenti();

mattonella[persona[identificativo].dammicoordinate('x')][persona[identificativo].dammicoordinate(
'y')].assegnautente(identificativo);
                    }
                }
                break;
            case SINISTRA:
                if (persona[identificativo].dammicoordinate('x') -1 < 0) {
                }
                else {
                    if (mattonella[persona[identificativo].dammicoordinate('x')-
1][persona[identificativo].dammicoordinate('y')].provabloccamattonella()) {

mattonella[persona[identificativo].dammicoordinate('x')][persona[identificativo].dammicoordinate(
'y')].assegnautente(-1);

mattonella[persona[identificativo].dammicoordinate('x')][persona[identificativo].dammicoordinate(
'y')].sbloccamattonella();
                        persona[identificativo].decrementax();
                        persona[identificativo].incrementaspostamenti();
```

```
mattonella[persona[identificativo].dammicoordinate('x')][persona[identificativo].dammicoordinate(
'y')].assegnautente(identificativo);
                    }
                }
                break;
            default:
                break;
        }
        usleep(persona[identificativo].dammivelocita());
        decisione = rand() % decisione_uscita;
        if (decisione == 1) {
            Coordinate vecchie;
            vecchie.assegnax(persona[identificativo].dammicoordinate('x'));
            vecchie.assegnay(persona[identificativo].dammicoordinate('y'));
            if (type == 2) {
                stampa.lock();
                cout << identificativo << ": " << "Decisione uscita" << endl;
                stampa.unlock();
            }
            int decisione_uscita = 0;
            int temp = 0;
            decisione_uscita = rand() % numero_uscite;
            for (int x = 0; x < mattonella.size(); x++) {
                for (int y = 0; y < mattonella[x].size(); y++) {
                    if (mattonella[x][y].dammitipo() == USCITA) {
                        if (decisione_uscita == temp) {
                            persona[identificativo].assegnauscita(x, y);
                        }
                        temp++;
                    }
                }
            }
            int temp2 = 0;
            int temp3 = 0;
            int fermo = 0;
            while (true) {
                if (type == 2) {
                    stampa.lock();
                    cout << identificativo << ": " << "Posizione attuale " <<
persona[identificativo].dammicoordinate('x') << "-" <<
persona[identificativo].dammicoordinate('y') << endl;
                    stampa.unlock();
                }
               if (fermo == 1) {
                    direzione = rand() % 4 + 1;
                    if (temp3 > 2) {
                        fermo = 0;
                        temp3 = 0;
                    }
                    else {
                        temp3++;
                    }
                }
                else {
                    e = rand() % 2;
                    if (e == 0) {
                        if (persona[identificativo].dammiuscita('x') >
persona[identificativo].dammicoordinate('x')) {
                            direzione = DESTRA;
                        }
                        else if (persona[identificativo].dammiuscita('x') <
persona[identificativo].dammicoordinate('x')) {
                            direzione = SINISTRA;
                        }
                        else {
                            direzione = FERMO;
                        }
                    }
                    else {
                        if (persona[identificativo].dammiuscita('y') >
persona[identificativo].dammicoordinate('y')) {
                            direzione = SOPRA;
                        }
                        else if (persona[identificativo].dammiuscita('y') <
persona[identificativo].dammicoordinate('y')) {
```

```
                            direzione = SOTTO;
                        }
                        else {
                            direzione = FERMO;
                        }
                    }
                }
                switch (direzione) {
                    case SOPRA:
                        if (persona[identificativo].dammicoordinate('y') +1 >
dimensionelato.dammiy()-1) {
                        }
                        else {
                            if
(mattonella[persona[identificativo].dammicoordinate('x')][persona[identificativo].dammicoordinate
('y')+1].provabloccamattonella()) {

mattonella[persona[identificativo].dammicoordinate('x')][persona[identificativo].dammicoordinate(
'y')].assegnautente(-1);

mattonella[persona[identificativo].dammicoordinate('x')][persona[identificativo].dammicoordinate(
'y')].sbloccamattonella();
                                persona[identificativo].incrementay();
                                persona[identificativo].incrementaspostamenti();

mattonella[persona[identificativo].dammicoordinate('x')][persona[identificativo].dammicoordinate(
'y')].assegnautente(identificativo);
                            }
                        }
                        break;
                    case SOTTO:
                        if (persona[identificativo].dammicoordinate('y') -1 < 0) {
                        }
                        else {
                            if
(mattonella[persona[identificativo].dammicoordinate('x')][persona[identificativo].dammicoordinate
('y')-1].provabloccamattonella()) {

mattonella[persona[identificativo].dammicoordinate('x')][persona[identificativo].dammicoordinate(
'y')].assegnautente(-1);

mattonella[persona[identificativo].dammicoordinate('x')][persona[identificativo].dammicoordinate(
'y')].sbloccamattonella();
                                persona[identificativo].decrementay();
                                persona[identificativo].incrementaspostamenti();

mattonella[persona[identificativo].dammicoordinate('x')][persona[identificativo].dammicoordinate(
'y')].assegnautente(identificativo);
                            }
                        }
                        break;
                    case DESTRA:
                        if (persona[identificativo].dammicoordinate('x') +1 >
dimensionelato.dammix()-1) {
                        }
                        else {
                            if
(mattonella[persona[identificativo].dammicoordinate('x')+1][persona[identificativo].dammicoordina
te('y')].provabloccamattonella()) {

mattonella[persona[identificativo].dammicoordinate('x')][persona[identificativo].dammicoordinate(
'y')].assegnautente(-1);

mattonella[persona[identificativo].dammicoordinate('x')][persona[identificativo].dammicoordinate(
'y')].sbloccamattonella();
                                persona[identificativo].incrementax();
                                persona[identificativo].incrementaspostamenti();

mattonella[persona[identificativo].dammicoordinate('x')][persona[identificativo].dammicoordinate(
'y')].assegnautente(identificativo);
                            }
                        }
                        break;
                    case SINISTRA:
                        if (persona[identificativo].dammicoordinate('x') -1 < 0) {
                        }
```

```
                        else {
                            if (mattonella[persona[identificativo].dammicoordinate('x')-
1][persona[identificativo].dammicoordinate('y')].provabloccamattonella()) {

mattonella[persona[identificativo].dammicoordinate('x')][persona[identificativo].dammicoordinate(
'y')].assegnautente(-1);

mattonella[persona[identificativo].dammicoordinate('x')][persona[identificativo].dammicoordinate(
'y')].sbloccamattonella();
                                persona[identificativo].decrementax();
                                persona[identificativo].incrementaspostamenti();

mattonella[persona[identificativo].dammicoordinate('x')][persona[identificativo].dammicoordinate(
'y')].assegnautente(identificativo);
                            }
                        }
                        break;
                    default:
                        break;
                }
                if (persona[identificativo].dammiuscita('x') ==
persona[identificativo].dammicoordinate('x') && persona[identificativo].dammiuscita('y') ==
persona[identificativo].dammicoordinate('y')) {

mattonella[persona[identificativo].dammicoordinate('x')][persona[identificativo].dammicoordinate(
'y')].assegnautente(-1);

mattonella[persona[identificativo].dammicoordinate('x')][persona[identificativo].dammicoordinate(
'y')].sbloccamattonella();
                    if (type == 2) {
                        stampa.lock();
                        cout << identificativo << ": " << "Uscita presa" << endl;
                        stampa.unlock();
                    }
                    break;
                }
                if (persona[identificativo].dammicoordinate('x') == vecchie.dammix() &&
persona[identificativo].dammicoordinate('y') == vecchie.dammiy()) {
                    temp2++;
                    if (temp2 > 2) {

                        fermo = 1;
                    }
                    else {
                        fermo = 0;
                        }
                }
                else {
                    vecchie.assegnax(persona[identificativo].dammicoordinate('x'));
                        vecchie.assegnay(persona[identificativo].dammicoordinate('y'));
                }
                usleep(persona[identificativo].dammivelocita());
            }
            break;
        }
    }
```

Rappresenta tutti i movienti della persone compresi quelli quando decide di uscire

```
    ///Rilascio risorse utili
    if (type == 2) {
        stampa.lock();
        cout << identificativo << ": " << "Rilascio del pass " << pass << endl;
        stampa.unlock();
    }
    semaforo[pass].utente = -1;
    semaforo[pass].pass.unlock();
    calc.lock();
    contaclienti--;
    if (type == 2) {
        stampa.lock();
        cout << "Clienti dentro " << contaclienti << endl;
        stampa.unlock();
    }
    calc.unlock();
}
```

Rilascio del pass per poi uscire

```
/// Stampa del negozio
/// @param mattonella base del negozio
/// @param semaforo pass utenti
/// @param fuori se la stampa si deve fermare
/// @param type debug
void negoziostart(vector<vector<Mattonella>> &mattonella, vector<passaport> *semaforo, bool
&fuori, int type) {
    if (type == 1) {
        while (1) {
            initscr();
            refresh();
            move(0,0);
            for (int i = 0; i < mattonella.size(); i++) {
                for (int j = 0; j < mattonella[i].size(); j++) {
                    if (mattonella[i][j].dammitipo() == 1) {
                        if (mattonella[i][j].dammiutente() != -1) {
                            for (int k = 0; k < semaforo->size(); k++) {
                                if (mattonella[i][j].dammiutente() == semaforo->at(k).utente) {
                                    printw("[%i]", k+1);
                                }
                            }
                        }
                        else {
                            printw("[E]");
                        }
                    }
                    else if (mattonella[i][j].dammitipo() == 2) {
                        if (mattonella[i][j].dammiutente() != -1) {
                            for (int k = 0; k < semaforo->size(); k++) {
                                if (mattonella[i][j].dammiutente() == semaforo->at(k).utente) {
                                    printw("[%i]", k+1);
                                }
                            }
                        }
                        else {
                            printw("[U]");
                        }
                    }
                    else {
                        if (mattonella[i][j].dammiutente() == -1) {
                            printw("[ ]");
                        }
                        else {
                            for (int k = 0; k < semaforo->size(); k++) {
                                if (mattonella[i][j].dammiutente() == semaforo->at(k).utente) {
                                    printw("[%i]", k+1);
                                }
                            }
                        }
                    }
                }
                printw("\n");
            }
            for (int f = 0; f < semaforo->size(); f++) {
                if (semaforo->at(f).utente == -1) {
                    printw("PASS %i NON ASSEGNATO\n", f+1);
                }
                else {
                    printw("PASS %i IN MANO AL CLIENTE %i\n", f+1, semaforo->at(f).utente);
                }
            }
            if (fuori) {
                break;
            }
        }
        endwin();
    }
    if (type == 0){
        while (1) {
            initscr();
            refresh();
            start_color();
            init_pair(1, COLOR_RED, COLOR_WHITE);
            init_pair(2, COLOR_GREEN, COLOR_WHITE);
            init_pair(3, COLOR_BLUE, COLOR_WHITE);
            init_pair(4, COLOR_WHITE, COLOR_WHITE);
```

```
            move(0,0);
            for (int i = 0; i < mattonella.size(); i++) {
                for (int j = 0; j < mattonella[i].size(); j++) {
                    if (mattonella[i][j].dammitipo() == ENTRATA) {
                        if (mattonella[i][j].dammiutente() != -1) {
                            attron(COLOR_PAIR(1));
                            for (int k = 0; k < semaforo->size(); k++) {
                                if (mattonella[i][j].dammiutente() == semaforo->at(k).utente) {
                                    printw(" %i ", k+1);
                                }
                            }
                            attroff(COLOR_PAIR(1));
                        }
                        else {
                            attron(COLOR_PAIR(2));
                            printw(" E ");
                            attroff(COLOR_PAIR(2));
                        }
                    }
                    else if (mattonella[i][j].dammitipo() == USCITA) {
                        if (mattonella[i][j].dammiutente() != -1) {
                            attron(COLOR_PAIR(1));
                            for (int k = 0; k < semaforo->size(); k++) {
                                if (mattonella[i][j].dammiutente() == semaforo->at(k).utente) {
                                    printw(" %i ", k+1);
                                }
                            }
                            attroff(COLOR_PAIR(1));
                        }
                        else {
                            attron(COLOR_PAIR(3));
                            printw(" U ");
                            attroff(COLOR_PAIR(3));
                        }
                    }
                    else {
                        if (mattonella[i][j].dammiutente() == -1) {
                            attron(COLOR_PAIR(4));
                            printw("[ ]");
                            attroff(COLOR_PAIR(4));
                        }
                        else {
                            attron(COLOR_PAIR(1));
                            for (int k = 0; k < semaforo->size(); k++) {
                                if (mattonella[i][j].dammiutente() == semaforo->at(k).utente) {
                                    printw(" %i ", k+1);
                                }
                            }
                            attroff(COLOR_PAIR(1));
                        }
                    }
                }
                printw("\n");
            }
            for (int f = 0; f < semaforo->size(); f++) {
                if (semaforo->at(f).utente == -1) {
                    printw("PASS %i NON ASSEGNATO\n", f+1);
                }
                else {
                    printw("PASS %i IN MANO AL CLIENTE %i\n", f+1, semaforo->at(f).utente);
                }
            }
            if (fuori) {
                break;
            }
        }
        endwin();
    }
}
```

Stampa del negozio

```
/// Attesa dei thread
/// @param persona vettore contenete le persone
void Funzione::uscitaclienti(vector<Persona> &persona) {
    for (int i = 0; i < persona.size(); i++) {
        persona[i].threadw.join();
    }
}
/// Generazione entità attive dei clienti
/// @param mattonella base del negozio
/// @param coda code da analizzare
/// @param persona tutte le persone presenti
/// @param type valore debug
void Funzione::generaclienti(vector<vector<Mattonella>> &mattonella, vector<Coda> &coda,
vector<Persona> &persona, int type) {
    for (int i = 0; i < persona.size(); i++) {
        persona[i].threadw = thread(&Funzione::azionicliente, this,
persona[i].dammiidentificativo(), ref(mattonella), ref(coda), ref(persona), type);
    }
}
/// Randomizzazzione del vettore passato
/// @param a vettore da randomizzare
/// @param max valore massimo del rand
void Funzione::randomizzavettore(vector<int> &a, int max) {
    srand((unsigned int)time(NULL));
    for (int i = 0; i < a.size(); i++) {
        a.at(i) = rand() % max;
        for (int j = 0; j < i; j++) {
            if (a.at(i) == a.at(j)) {
                i--;
                break;
            }
        }
    }
}
/// Ritorno del puntatore al vettore dei pass per passarlo successivamente al thread della stampa
vector<passaport> *Funzione::dammipassaporto() {
    return (&semaforo);
}
```

Altre funzioni di attesa del cliente, generazione dei clienti, generazione di un vettore casuale e restituzione di un puntare di un vettore