GOMOKU - czyli kółko krzyżyk dla profesjonalistów

Agnieszka Rabiej

10 lutego 2020

Streszczenie

Celem projektu było napisanie gry komputerowej w jęzuku C++ z wykorzystaniem poznanych na zajęciach elementów GUI oraz techniki programowania obiektowego.

1 Zasady gry

Popularna gra GOMOKU to tak naprawdę rozbudowana wersja jeszcze popularniejszej gry kółko krzyżyk. W rozgrywce biorą udział dwaj gracze, jeden z nich posługuje się pionkiem 'x', a drugi 'o'. Gracze na przemian umieszczają swój pionek na planszy, a raz ustawiony pionek nie może już zostać przestawiony. Aby zapewnić sobie zwyciestwo, jedyne co należy zrobić to ustawić pięc swoich pionków w rędzie pionowo, poziomo lub na ukos. Z pozoru wydawać by się mogło to banalnie proste, ale czy na pewno? Jaką tatktykę powinien obrać gracz by przechytrzyć przeciwnika, zagrać ofensywnie czy może defensywnie? W przypadku, gdy żadnemu z graczy nie uda się doprowadzić do wygranej, a wszystkie pola na planszy są już zajęte, gra kończy się remisem.

2 Projekt

Projekt składa się z czterech oddzielnym klas, a każda z nich ma w nim swoje zadania.

- Klasa tic tac toe gui jest odpowiedzialna za wszystko to co użytkowanik widzi na ekranie, w niej obsługiwane jest kliknięcie myszką, aby wstawić pionek na planszę, wyświetlanie informacji na temat rezultatu rozgrywki oraz wybór czy jako przeciwnika wybieramy drugiego gracza czy może komputer.
- w przypadku kiedy wybraliśmy wersję gry z komputerem z pomocą przychodzi nam klasa tic tac toe engine. To tutaj została zaimplementowana cała obsługa ruchów ktore wykonuje nasz komputerowy przeciwnik.
- klasa board - to nasza plansza, potrafi ona rozpoznać czy wybrane przez gracza miejsce na planszy jest już zajęte przez przeciwnika, a przede wszystkim czy ta lokalizaja została określona jako lokalizaja naszej planszy.
- klasa paw, czyli pionek, którym posługuje się gracz. Obiekt tej klasy przechowuje informację na temat tego, do którego gracza należy oraz gdzie został umiejscowiony

2.1 Zasada działania programu

Na poczatku gracz wybiera jedną z dwóch opcji, poprzez naciśnięcie na klawiaturze przycisku odpowiednio t(two gamers) aby grać z przeciwnikiem w postaci drugiej fizycznej osoby, bądź klawisz k - aby grać z komputerem. Zaczyna się gra, na zmiane gracze wykonują ruchy, aż któryś z nich wygra lub plansza będzie zapełniona. Najciekawszą częścią programu jest sposób wyboru ruchu dla komputera, dlatego właśnie skupię się na tej części programu.

2.1.1 Jak komputer decyduje gdzie postawić pionek

W celu podjęcia decyzji, gdzie postawić pionek w swoim ruchu komputer tworzy sobie dwie tablice pomocnicze. Wymiary tych tablic są takie same jak wymiary planszy gry. Każda z nich przechowuje informacje na temat 'wagi' danego pola. Jedna tablica służy do ocenienia wag dla przeciwnika, a druga dla samego siebie. Przyjęłam, że poszczególne ustawienia pionków mają ustalone wagi, im wartość jest wyższa tym większe szanse na wygraną przy postawieniu w tym miejscu swojego pionka - w przypadku tablicy wag wyliczanej dla samego siebie. Im wyższa waga w tablicy wyliczonej u przeciwnika tym bardziej komputer powinien blokować to pole, żeby nie przegrać. W programie odróżniamy następujące wagi:

- 5 - cztery pionki tego samego gracza pod rząd
- 4 - jeden pionek z jedej strony pola i dwa z drugiej (razem z polem w środku tworzą 4)
- 3 - miejsce w którym po dodaniu pionka tworzymy 3 trójki, ponieważ z dwóch stron mamy dwa pod rząd pionki, nie bierzemy tu pod uwagę takich stron, które po wstawieniu w srodek pionka stworzą 5 pod rząd, ponieważ jest to już sprawdzane wcześniej i posiada wyższą wagę
- 3 - trójka 'krytyczna', czyli taka, którą jeśli nie zastawimy z którejś strony to przeciwnik będzie mógł ustawić czwarty pionek pod rząd i wtedy doprowadzi to do porażki ponieważ taka czwórka nie będzie zastawiona z żadnej strony
- 2 - 'normalna' 3 - trzy pionki pod rząd, ale z jednej strony zastawione pionkiem przeciwnika lub dwa pionki pod rząd
- 1 - jeden pionek

Program przechowuje informacje o najwyższej wadze w danej tablicy i jeśli tablica wag przeciwnika pokazuje wartość wyższą lub równą 3 komputer i jeżeli najwyższa waga przeciwnika jest wyższa od najwyższej wagi komputera, to komputer blokuje ruch przeciwnika. W przeciwnym razie komputer wykonuje taki ruch by powiększyć swoje szanse na wygraną, a nie tylko się bronić.

3 Opis działania programu

Program kompilujemy z konsoli linuxowej, za pomocą komendy g++ main.cpp tic_tac_toe_gui.cpp board.cpp paw.cpp tic_tac_toe_engine.cpp -std=c++11 -lcurses oraz standardowo uruchamiamy komendą ./a.out.

W funkcji main, inicjalizujemy nową grę - czyli tworzymy obiekt klasy tic_tac_toe_gui oraz wywołujemy na nim metode build_gui_game().

3.1 Klasa tic_tac_toe_gui

W klasie tic_tac_toe_gui znajduje się cała obsługa programu związana z częścią gui'ową. Deklaracja klasy w pliku nagłówkowym ".h"wygląda następująco:

```
class tic_tac_toe_gui{
    public:
        board current_board;
        tic_tac_toe_engine engine;
        bool two_gamer;
        char current_gamer = 'x';
    public:
        void init();
        void build_gui_game();
        void refresh_board(WINDOW* okno);
        void chose_gamer(int we);
        void clean_board(WINDOW* okno);
        void print_winner(WINDOW* okno, char winner);
        bool check_remis(WINDOW* okno);
};
```

gdzie current_board reprezentuje obiekt klasy board, określa nam aktualną tablice do gry, engine - obiekt klasy tic_tac_toe_engine - silnik naszej gry odpowiedzialny głównie za wyliczanie ruchów wykonywanych przez kompyter, zmienną typu bool - two_gamer - określającą czy gracz wybrał wersję gry z komputerem czy z drugim graczem - człowiekiem oraz zmienną typu char - current_gamer - która określa nam czy ruch wykonuje gracz 'x' czy 'o'.

3.1.1 Funkcja init()

```
 void tic_tac_toe_gui::init(){
    board c_board;
    c_board.new_board();
    tic_tac_toe_engine eng;
    this->current_board = c_board;
    this->engine = eng;
};
```

Tworzymy obiekt klasy board oraz obiekt klasy engine i przypisujemy je do odpowiednich slotów w funkcji tic_tac_toe_gui.

3.1.2 Funkcja build_gui_game()

```
void tic_tac_toe_gui::build_gui_game(){
    WINDOW* okno = initscr();
    noecho();
    curs_set(0);
    nodelay(okno, 1);
    mmask_t mmask;
    MEVENT mysz;
    keypad(stdscr, TRUE);
    mousemask(ALL_MOUSE_EVENTS | REPORT_MOUSE_POSITION,NULL);
    int we = 0;
    bool h = false;
    chose_gamer(we);
    refresh_board(okno);

    while(true){
        we = getch();
        if(we == KEY_MOUSE){
            if(getmouse(&mysz) == OK){
                if(this->current_board.check_click(mysz.y,mysz.x, this->current_gamer)){
                    if(this->current_gamer == 'x'){
                        this->current_gamer = 'o';
                    }else this->current_gamer = 'x';

                    vector <int> coordinate;
                    h = this->engine.check_winner(this->current_board,mysz.y,mysz.x);
                    refresh_board(okno);
                    if(this->two_gamer == true){
                        coordinate = this->engine.count_movement(this->current_board,
 this->current_gamer);
                        vector <int> coordinate_computer;
                        coordinate_computer = this->current_board.add_computer_movement(coordinate
  this->current_gamer);

                        if(this->current_gamer == 'x'){
                            this->current_gamer = 'o';
                        }else this->current_gamer = 'x';
```

```
                    h = this->engine.check_winner(this->current_board,coordinate_computer[0],
      coordinate_computer[1]);
                    if(check_remis(okno)){
                        print_winner(okno, 'r');
                        refresh();
                        sleep(3);
                        clean_board(okno);
                        refresh();
                        sleep(3);
                    }
                }
            refresh_board(okno);
            if(h == true ){
                print_winner(okno, this->current_gamer);
                refresh();
                sleep(3);
                clean_board(okno);
                refresh();
                sleep(3);
                break;
            }
            }
        }
    }else{
        if(we == 'q') break;
    }refresh();
}
endwin();

};
```

Inicjalizujemy okno oraz włączamy obsługę myszy i wybieramy liczbe graczy, a następnie jeśli zostanie naciśniety klawisz myszy sprawdzamy czy znajdujemy się w zakresie planszy, zmieniamy gracza, sprawdzamy czy nie nastąpił koniec gry, i w zależności od tego czy gramy z komputerem czy z drugim graczem, następuje kolejny ruch, ponowna zmiana gracza oraz sprawdzenie zwyciezcy. Ruchy graczy wykonywane są w pętli i jeśli nastąpi zakonczenie gry - wygrana lub remis, to wychodzimy z petli i wyswietlana jest informacja o zwyciezcy gry. Program kończy działanie.

3.1.3 Funkcja check_remis

```
bool tic_tac_toe_gui::check_remis(WINDOW* okno){
    bool remis = false;
    for(int i=0; i<=board_size; i++){
        for(int j=0; j<=board_size; j++){
            paw current_paw;
            current_paw = this->current_board.get_element_to_draw(i,j);
            if(current_paw.name[0] != '*'){
                remis = true;
            }else return false;
        }
    }
    return remis;
}
```

Sprawdzamy po kolei wszystkie pola na planszy i jeśli któreś jest równe '*' zwracamy false, czyli nie nastąpił remis, w przeciwnym razie funkcja zwróci true - remis.

3.1.4 Funkcja clean_board

```
void tic_tac_toe_gui::clean_board(WINDOW* okno){
    for(int i=0; i<=board_size; i++){
        for(int j=0; j<=board_size; j++){
            paw current_paw;
            current_paw = this->current_board.get_element_to_draw(i,j);
            mvprintw(current_paw.y, current_paw.x, "-");
        }
    }
};
```

Funkcja czyści tablice - służy do wizualizacji zakończenia działania programu poprzez wypełnienie pól znakami '-'.

3.1.5 Funkcja print_winner

```
void tic_tac_toe_gui::print_winner(WINDOW* okno, char winner){
    if(winner == 'o'){
        mvprintw(0, 0, "wygral x");
    }else if(winner == 'x'){
        mvprintw(0, 0, "wygral o");
    }else mvprintw(0, 0, "remis");
};
```

Wyświetla komunikat o zwyciezcy rozgrywki, bądź o remisie.

3.2 Funkcja chose_gamer

```
void tic_tac_toe_gui::chose_gamer(int we){
    while(true){
        we = getch();
        if(we== 'k'){
            this->two_gamer = true;
            break;
        }else if(we == 't'){
            this->two_gamer = false;
            break;
        }
    }
};
```

Poprzez naciśniecie klawisza decydujemy czy gramy z 'k' - komputerem lub 't' - drugim graczem. Wartość przechowywana jest przez egzemplar klasy.

3.2.1 Funkcja refresh_board

```
void tic_tac_toe_gui::refresh_board(WINDOW* okno){
    for(int i=0; i<=board_size; i++){
        for(int j=0; j<=board_size; j++){
            paw current_paw;
            current_paw = this->current_board.get_element_to_draw(i,j);
            mvprintw(current_paw.y, current_paw.x, current_paw.name);
        }
```

```
    }
};
```

Służy do wyświetlenia aktualnego stanu gry.

3.3 Klasa tic_tac_toe_engine

```
class tic_tac_toe_engine{
    public:
        bool check_winner(board board_t, int y, int x);
        vector <int> count_movement(board board_t, char name);
        int count_state(board * board_t, int i, int j, char name);
};
```

Klasa w głównej mierze służy do obsługi ruchów komputera. Wyżej został opisany dokładnie sposób realizajci obsługi zdarzeń. Ściślej ujmując w poszczególnych metodach mamy:

3.3.1 Funkcja count_movement

Tutaj wyliczany jest dokładny ruch komputera na podstawie przygotowanych tablic z wagami poszczególnych ruchów (wyliczane w funkcji count_state). Komputer podejmuje decyzje czy grać ofensywnie czy defensywnie

3.3.2 Funkcja count_state

Wylicza wagi ruchów dla tablic.

3.3.3 Funkcja check_winner

Sprawdza czy na planszy nastąpiła wygrana któregoś z graczy.

3.4 Klasa board

```
 class board{
    public:
        paw tab[board_size+1][board_size+1];
    public:
        void new_board();
        paw get_element_to_draw(int i, int j);
        bool check_click(int y, int x, char current_gamer);
        vector <int> add_computer_movement(vector <int> tab, char current_gamer);
};
```

Klasa przechowuje tablice pionków - jako planszę gry oraz posiada metody:

3.4.1 Funkcja new_board

```
void board::new_board(){

    for(int i=0; i<=board_size; i++){
        for(int j=0; j<=board_size; j++){
            paw new_paw_element;
            new_paw_element.new_paw('*', i+3, j+7);
            this->tab[i][j] = new_paw_element;
        }
```

```
    }
};
```

Tworzy nową tablicę - czyli tablicę pionków w odpowiednim miejscu na ekranie.

3.4.2 Funkcja get_element_to_draw

```
paw board::get_element_to_draw(int i, int j){
    paw n;
    n = this->tab[i][j];
    return n;
};
```

Zwraca obiekt klasy paw - żeby można go było narysować w klasie tic_tac_toe_gui.

3.4.3 Funkcja add_computer_movement

Dodaje do tablicy ruch komputera.

3.4.4 Funkcja check_click

```
bool board::check_click(int y, int x, char current_gamer){
    if(y>=this->tab[0][0].y && y<=this->tab[board_size][board_size].y &&
x>=this->tab[0][0].x && x<=this->tab[board_size][board_size].x){
        for(int i=0; i<=board_size; i++){
            for(int j=0; j<=board_size; j++){
                if(y == this->tab[i][j].y && x == this->tab[i][j].x &&
   this->tab[i][j].name[0] == '*'){
                    this->tab[i][j].name[0] = current_gamer;
                    return true;
                }
            }
        }
    }
    return false;
}
```

Sprawdzaczy ruch gracza jest możliwy - czyli czy znajduje się w obrębie planszy oraz czy dane pole nie jest już zajęte.

3.5 Klasa paw

```
 class paw{
    public:
        char name[2];
        int y;
        int x;
    public:
        void new_paw(char name, int y, int x);
};
```

Klasa posiada pola przechowujące nazwę gracza - 'o', 'x', '*' - wolne pole oraz wartości x i y wskazujące na umiejscowienie pionka.