시스템프로그래밍 텀프로젝트 보고서

소속: IT대학 컴퓨터학부 심화컴퓨터전공

학번: 2020111854

이름: 신찬규

프로젝트 명: gomoku

1. 개요

오목은 오목판과 오목알을 사용해 흑백이 번갈아 한 수씩 두어 가로, 세로, 대각선 중 한 방향으로 같은 색 돌을 다섯 개 먼저 늘어놓으면 승리하는 보드 게임이다. 본 프로그램은 오목을 구현한 프로그램으로써 2명의 사용자가 서로 1:1 게임을 진행할 수 있다. 사용자가 번갈아 가면서 표준 입력을 통해 돌을 착수할 수 있으며, 제한 시간 30초 내에 착수를 해야 한다. 게임 진행 중 사용자는 `Ctrl+C`로 중도 포기를 할 수 있다.

2. 상세 구조 및 동작

전역 변수 및 상수

- `BLACK`: 흑돌을 나타내는 상수. 값은 1
- `WHITE`: 백돌을 나타내는 상수. 값은 2
- `gomoku_board[19][19]`: 오목판을 나타내는 배열. 빈 칸은 0, 흑돌이 착수하면 `BLACK`, 백돌이 착수하면 `WHITE` 값을 가진다.
- `player_1[4]`: 흑돌 플레이어의 이니셜을 저장하는 배열
- `player_2[4]`: 백돌 플레이어의 이니셜을 저장하는 배열
- `player_now_placement`: 현재 착수할 차례인 플레이어를 나타낸다. 흑돌 차례의 경우 `BLACK`, 백돌 차례의 경우 `WHITE` 값을 가진다.

함수

프로그램을 구성하는 주요 함수들에 대해서만 기술하였다.

1. `main()`

프로그램을 실행하면 가장 먼저 `main()`이 실행된다. `main()`은 플레이어가 메뉴를 선택하도록 하며, 1을 입력하면 먼저 signal handler를 등록한다. `give_up()`은 플레이어가 `Ctrl+C`를 입력하면 호출되며, `time_out()`은 `SIGALRM`이 발생하면 호출된다. 그리고 게임을 초기화하며 먼저 흑돌이 착수를 하도록 설정하고 게임을 시작한다.

```
int main() {
  ...
  while (1) {
    printf("%s", gomoku_logo);
    printf("\nPlay a gomoku with your friend!\n");

    puts("1) Play game");
    puts("2) Scoreboard");
    puts("3) Rule");
    puts("4) Exit");
    printf(": ");
    scanf("%d", &option);
    clear_read_buffer();

    switch (option) {
      case 1:
        // when the player inputs Ctrl + C, pause the game
        signal(SIGINT, give_up);
        // when the player doesn't input anything, end the game
        signal(SIGALRM, time_out);
        init_game();
        player_now_placement = BLACK;
        play_gomoku();
        break;
      case 2:
        ...
    }
  }
}
```

2. `play_gomoku()`

`play_gomoku()`는 프로그램의 주요 함수로 다음과 같은 구조를 가지고 있다.

```
// start the game
void play_gomoku() {
  print_gomoku_board();

  while (1) {
    placement();
```

```
    print_gomoku_board();
    if (is_game_end()) {
      end_gomoku(player_now_placement);
      break;
    }
    switch (player_now_placement) {
      case BLACK:
        player_now_placement = WHITE;
        break;

      case WHITE:
        player_now_placement = BLACK;
        break;
    }
  }
}
```

먼저 오목판을 출력한다. 그 다음 착수를 하고, 오목판을 출력한 다음 승리 조건이 만족됐는지 판단한다. 만약 승리 조건이 만족되면 게임을 끝낸다. 그렇지 않다면 다음 플레이어가 착수를 하게 된다. 이를 승리 조건이 만족될 때까지 무한 반복한다.

3. `placement()`

`placement()`는 플레이어가 돌을 착수하는 함수다. 플레이어는 A10과 같이 입력을 통해 착수를 할 수 있다. 만약 범위 내 값을 입력하지 않으면, 이를 무시하고 새로 입력을 받는다. 정상 범위 내의 값을 입력하면`gomoku_board[row][col] = player_now_placement`를 한다. 30초 내에 아무 입력을 하지 않으면 패배를 하도록 `alarm(30)`으로 30초 뒤 `time_up()`을 실행하도록 설정하고, 시간 내에 입력을 하면 `alarm(0)`을 한다.

```
// place the stone at the gomoku board
void placement() {
  int row;
  char col;

  printf("where (%s): ", player_now_placement == BLACK ? "black" : "white");
  fflush(stdout);
  alarm(30);

  while (1) {
    row = 0;
    col = 0;

    scanf("%c%d", &col, &row);
    getchar();

    col = toupper(col);
    col -= 'A';

    if (gomoku_board[row][col] == 0) {
      gomoku_board[row][col] = player_now_placement;
```

```
      alarm(0);
      break;
    }
  }
}
```

4. `is_game_end()`

`is_game_end()`은 승리 조건을 판단하는 함수다. 승리 조건이 충족되면 1을, 그렇지 않으면 0을 반환한다. 전체 오목판의 칸마다 위, 아래, 양 옆, 대각선으로 연속해서 5개의 돌이 있는지 탐색한다. 연속해서 5개의 돌이 존재하면 승리 조건을 만족하고, 전체 오목판에 연속해서 5개의 돌이 존재하지 않으면 만족하지 않는다.

```
// if the game ends return 1, or not, return 0
int is_game_end() {
  int row;
  int col;
  int rock;
  int check;

  check = 0;

  for (row = 0; row < 19; row++) {
    for (col= 0; col < 19; col++) {
      if (gomoku_board[row][col] == player_now_placement) {
        check++;

        for (rock = 1; rock < 5; rock++) {
          if (col + rock >= 19) {
            break;
          } else if (gomoku_board[row][col + rock] == player_now_placement) {
            check++;
            continue;
          } else {
            check = 1;
            break;
          }
        }

        if (check == 5) {
          return 1;
        }

        for (rock = 1; rock < 5; rock++) {
          if (col - rock <= -1) {
            break;
          } else if (gomoku_board[row][col - rock] == player_now_placement) {
            check++;
            continue;
          } else {
            check = 1;
```

```
            break;
          }
        }

        if (check == 5) {
          return 1;
        }

        for (rock = 1; rock < 5; rock++) {
          if (row + rock >= 19) {
            break;
          } else if (gomoku_board[row + rock][col] == player_now_placement) {
            check++;
            continue;
          } else {
            check = 1;
            break;
          }
        }

        if (check == 5) {
          return 1;
        }

        for (rock = 1; rock < 5; rock++) {
          if (row + rock <=-1) {
            break;
          } else if (gomoku_board[row + rock][col] == player_now_placement) {
            check++;
            continue;
          } else {
            check = 1;
            break;
          }
        }

        if (check == 5) {
          return 1;
        }

        for (rock = 1; rock < 5; rock++) {
          if (row + rock >= 19 || col + rock >= 19) {
            break;
          } else if (gomoku_board[row + rock][col + rock] == player_now_placement) {
            check++;
            continue;
          } else {
            check = 1;
            break;
          }
        }

        if (check == 5) {
          return 1;
        }

        for (rock = 1; rock < 5; rock++) {
          if (row + rock >= 19 || col - rock <= -1) {
```

```
            break;
          } else if (gomoku_board[row + rock][col - rock] == player_now_placement) {
            check++;
            continue;
          } else {
            check = 1;
            break;
          }
        }

        if (check == 5) {
          return 1;
        }

        for (rock = 1; rock < 5; rock++) {
          if (row - rock <= -1 || col + rock >= 19) {
            break;
          } else if (gomoku_board[row - rock][col + rock] == player_now_placement) {
            check++;
            continue;
          } else {
            check = 1;
            break;
          }
        }

        if (check == 5) {
          return 1;
        }

        for (rock = 1; rock < 5; rock++) {
          if (row - rock <= -1 || col - rock <= -1) {
            break;
          } else if (gomoku_board[row + rock][col] == player_now_placement) {
            check++;
            continue;
          } else {
            check = 1;
            break;
          }
        }

        if (check == 5) {
          return 1;
        }
      }
    }
  }

  return 0;
}
```

5. `end_gomoku()`

승리 조건을 만족해 게임이 종료되면 `end_gomoku()` 함수가 호출된다. 게임이 종료되었으므로 signal을 무시하도록 설정하고 게임 결과를 기록을 할지 말지 플레이어가 선택하도록 한다. 기록을 한다고 선택할 경우 플레이어의 이니셜 3글자를 입력받아 `record_game()`을 호출하고, 그렇지 않으면 프로그램을 종료한다.

```
// end the game, if want, record the result of the game
void end_gomoku(int winner) {
  int option;

  // the game finish, so ignore the signal
  signal(SIGINT, SIG_IGN);
  signal(SIGALRM, SIG_IGN);

  if (winner == BLACK) {
    printf("Black is winner!\n");
  } else if (winner == WHITE) {
    printf("White is winner!\n");
  }

  puts("Do you want to record the result of this game at the scoreboard?");
  puts("1) yes");
  puts("2) no");
  printf(": ");
  scanf("%d", &option);

  if (option == 2) {
    puts("GG!");
    exit(0);
  }

  puts("--------------------------------------");
  puts("| Input your name within 3 letters. |");
  puts("--------------------------------------");

  printf("Player 1 (Black): ");
  scanf("%s", player_1);

  printf("Player 2 (White): ");
  scanf("%s", player_2);

  record_game(winner);

  puts("GG!");

  exit(0);
}
```

6. `record_game()`

`record_game()`은 실제로 바이너리 파일에 결과를 기록하는 함수다. 빠른 기록을 위해 병렬 처리로 구현했다. `fork()`를 통해 자식 프로세스를 만든 후, 부모 프로세스가 기록할 문자열을 pipe를 통해 자식 프로세스에게 보내주면 자식 프로세스가 이를 `scoreboard`에 기록한다.

```
// record the result of the game
void record_game(int winner) {
  FILE *scoreboard_fp;
  int thepipe[2];
  char buf[BUFSIZ];

  if (pipe(thepipe) == -1) {
    perror("pipe");
  }

  // parallel processing of the recording using fork() and pipe()
  switch (fork()) {
    case -1:
      perror("fork");
      exit(1);
    // child process records the result
    case 0:
      close(thepipe[1]);
      read(thepipe[0], buf, BUFSIZ);

      scoreboard_fp = fopen("./bin/scoreboard", "a+");

      if (scoreboard_fp == NULL) {
        perror("scoreboard_fp fopen");
        exit(1);
      }

      fprintf(scoreboard_fp, "%s", buf);
      fflush(scoreboard_fp);

      exit(0);
    // parent process sends the result to the child process
    default:
      close(thepipe[0]);

      if (winner == BLACK) {
        sprintf(buf, "%s %s %s %ld\n", player_1, player_2, player_1, time(NULL));
      } else if (winner == WHITE) {
        sprintf(buf, "%s %s %s %ld\n", player_1, player_2, player_2, time(NULL));
      }

      if (write(thepipe[1], buf, strlen(buf)) != strlen(buf)) {
        perror("write");
        exit(1);
      }

      wait(NULL);

      puts("Record completed.");

      break;
```

```
  }
}
```

7. `give_up()`

`give_up()`는 플레이어가 `Ctrl+C`를 입력하면 실행되는 signal handler이다. 1을 입력하면 포기를 한 플레이어가 패배하게 되고, 2를 입력하면 다시 게임을 재개한다.

```
// pause the game when the player inputs Ctrl + C
void give_up() {
  int option;

  alarm(0);
  system("clear");

  puts("Do you really want to give up this game?");
  puts("1) yes");
  puts("2) no");
  printf(": ");
  scanf("%d", &option);
  clear_read_buffer();

  if (option == 1) {
    switch (player_now_placement) {
      case BLACK:
        end_gomoku(WHITE);
        break;
      case WHITE:
        end_gomoku(BLACK);
        break;
    }
  } else if (option == 2) {
    play_gomoku();
  }
}
```

8. `time_out()`

`time_out()`은 `placement()`에서 `alarm(30)`을 통해 30초 뒤에 `SIGALRM`이 오도록 설정한 후 플레이어가 아무 입력을 하지 않아 `SIGALRM`이 오게 되면 실행되는 signal handler이다. 아무 입력을 하지 않은 플레이어를 패배 처리 한다.

```
// end the game when the player doesn't input anything during 30 seconds
void time_out() {
  puts("30 seconds have passed.");
  switch (player_now_placement) {
```

```
    case BLACK:
      end_gomoku(WHITE);
      break;
    case WHITE:
      end_gomoku(BLACK);
      break;
  }
}
```

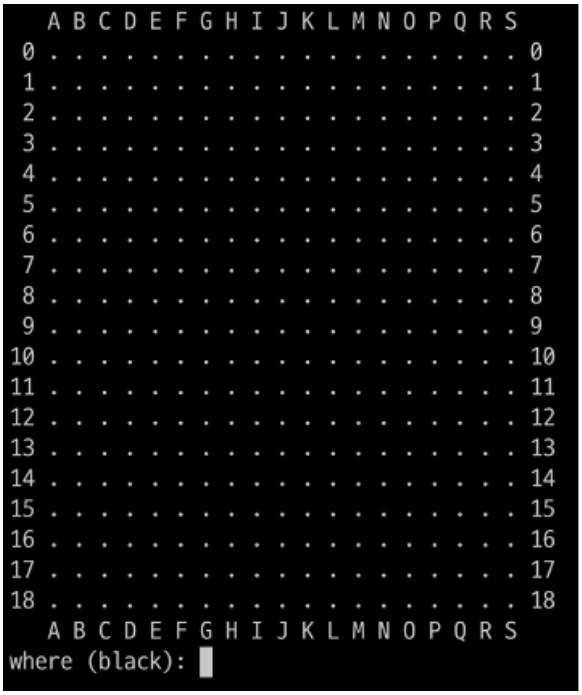
3. 결과 및 사용법

다음 과정을 통해 프로그램을 다운로드, 실행한다.

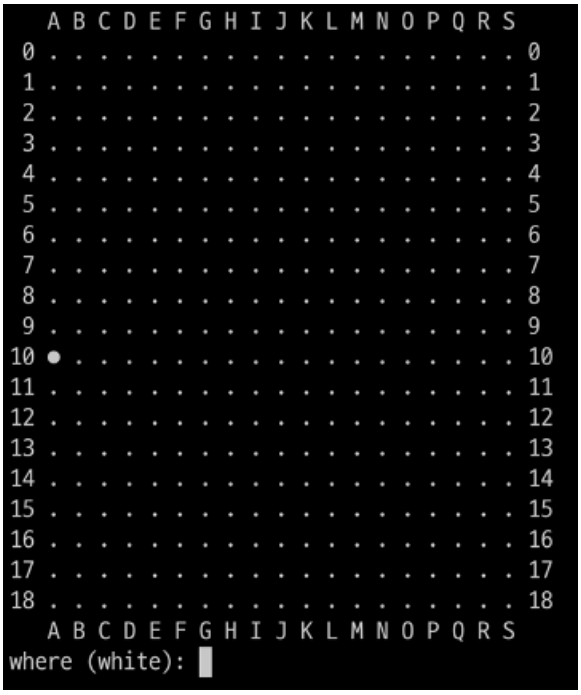
1. `git clone https://github.com/scv1702/gomoku`
2. `cd ./gomoku`
3. `./gomoku.sh`

그러면 다음과 같이 프로그램이 실행된다. 숫자를 입력해 메뉴를 선택할 수 있다.

1. 게임을 실행한다.
2. 점수판을 출력한다.
3. 오목 규칙을 출력한다.
4. 프로그램을 종료한다.

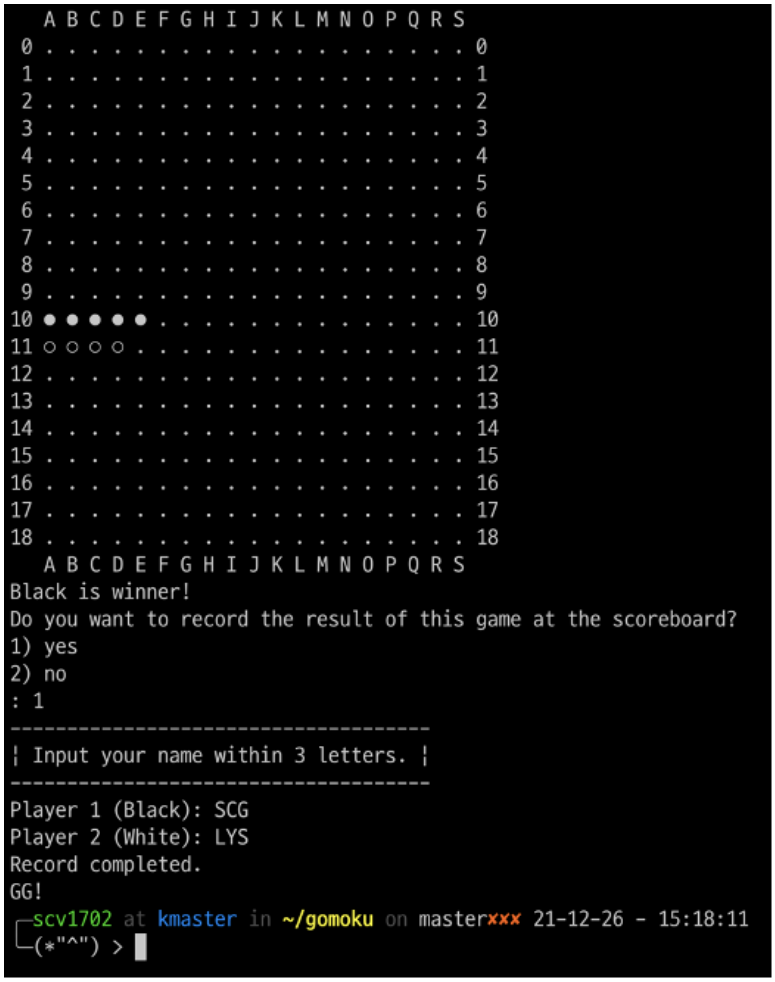


1번을 선택하면 다음과 같이 오목판이 출력되며 어디에 착수할지 입력을 할 수 있다.

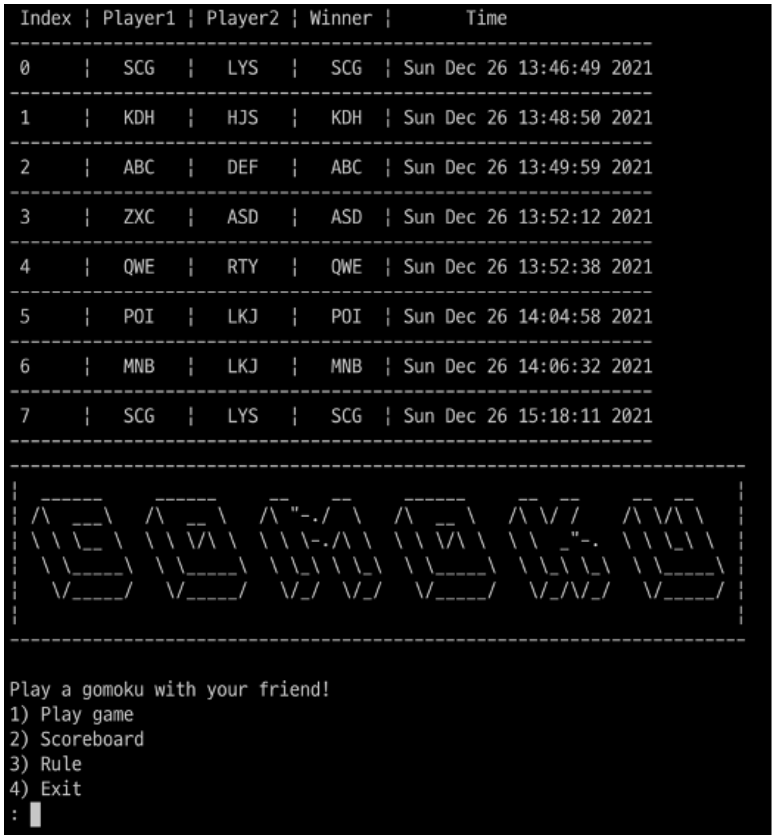


흑돌이 먼저 착수하며, A10과 같이 입력을 통해 착수할 수 있다.

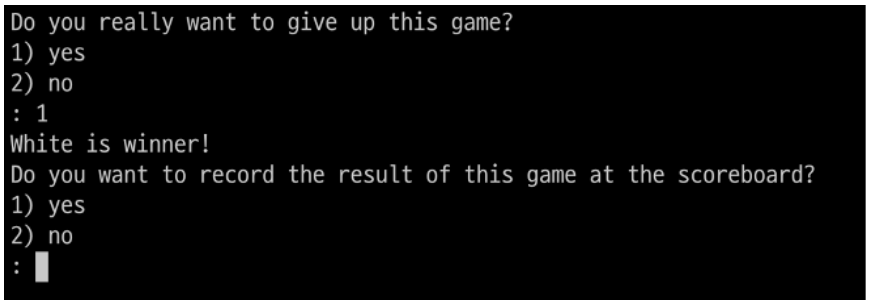
흑돌과 백돌이 번갈아 착수를 하며, 승리 조건인 돌이 연속으로 5개 놓이게 되면 게임이 종료되면서 게임 결과를 기록할 지 선택할 수 있다.



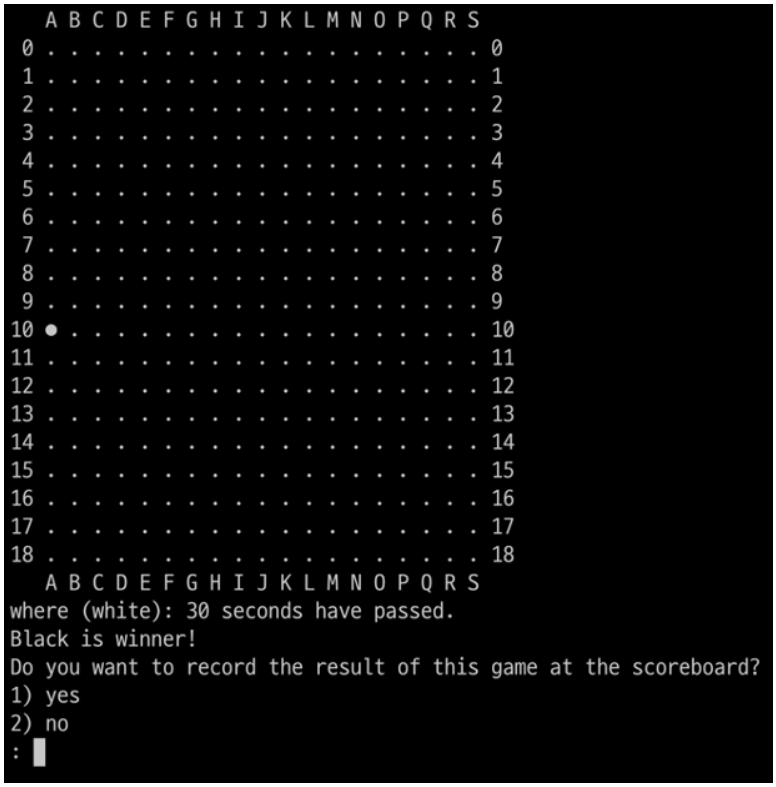
1을 입력해 게임 결과를 기록한다고 선택하면, 플레이어의 이니셜 3글자를 입력해 게임 기록을 남길 수 있다. 기록이 완료되면, 프로그램은 종료된다.



기록된 결과는 다음과 같다.



게임 도중 포기를 하고 싶으면 `Ctrl+C`를 입력해 게임을 포기할 수 있다. 이때 1번을 선택하면 포기한 플레이어가 패배하게 되고, 2번을 선택하면 게임이 재개된다.



플레이어가 30초 동안 아무 것도 입력하지 않으면, 해당 플레이어는 패배하게 된다.

4. 고찰

지금까지 구현한 오목은 플레이어 두 명에서 하나의 컴퓨터를 통해 게임을 진행한다. 하지만 socket 등을 이용해 두 컴퓨터 간 통신을 할 수 있게 한다면, 멀티 플레이 게임 또한 가능할 것이다.

또한 현재 흑돌과 백돌 상관 없이 아무나 연속해서 5개의 돌을 놓으면 승리 조건이 만족하게 되는데, 흑돌이 먼저 착수할 수 있기 때문에 흑돌이 매우 유리한 조건이다. 이를 해결하기 위한 렌주룰, 오프닝룰 등 다양한 오목 규칙들이 존재하는데 본 프로그램은 이를 구현하지 못했다. 향후 추가적인 개발을 통해 이를 보완할 수 있을 것이다.

5. 참고 자료

Wiki Pedia, 『Gomoku』, https://en.wikipedia.org/wiki/Gomoku