Presentation Group 4

Mike Selkirk and Sean Holmes

December 8, 2022

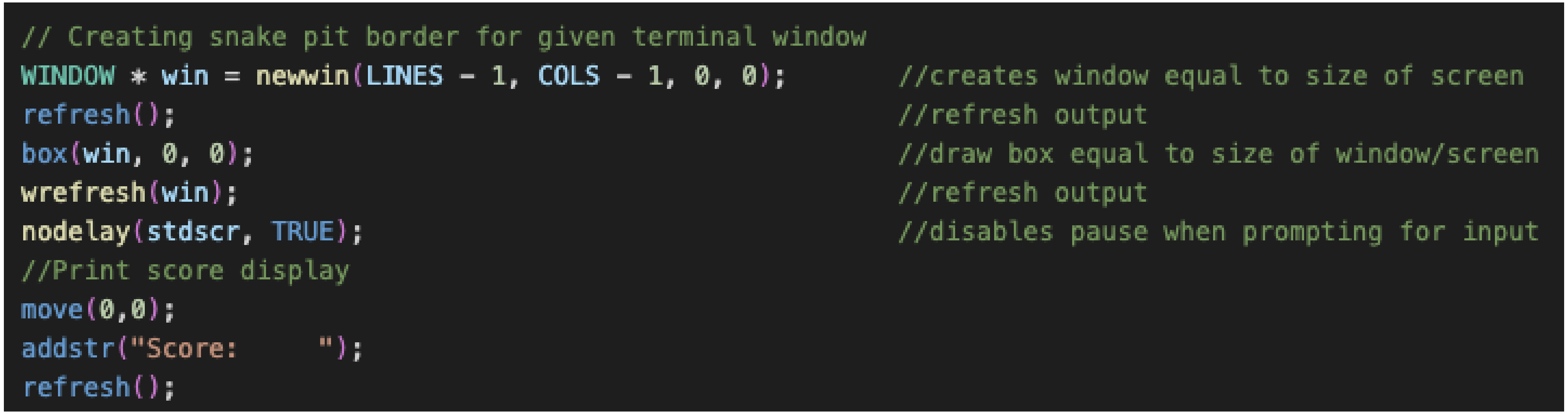
Outline

- Snake Pit
- Initializing the Snake
- Snake Movement
- Adjusting Snake Speed
- Trophy Display
- Trophy Duration
- Trophy Eating
- Win/Lose Game States

Team Member Tasks

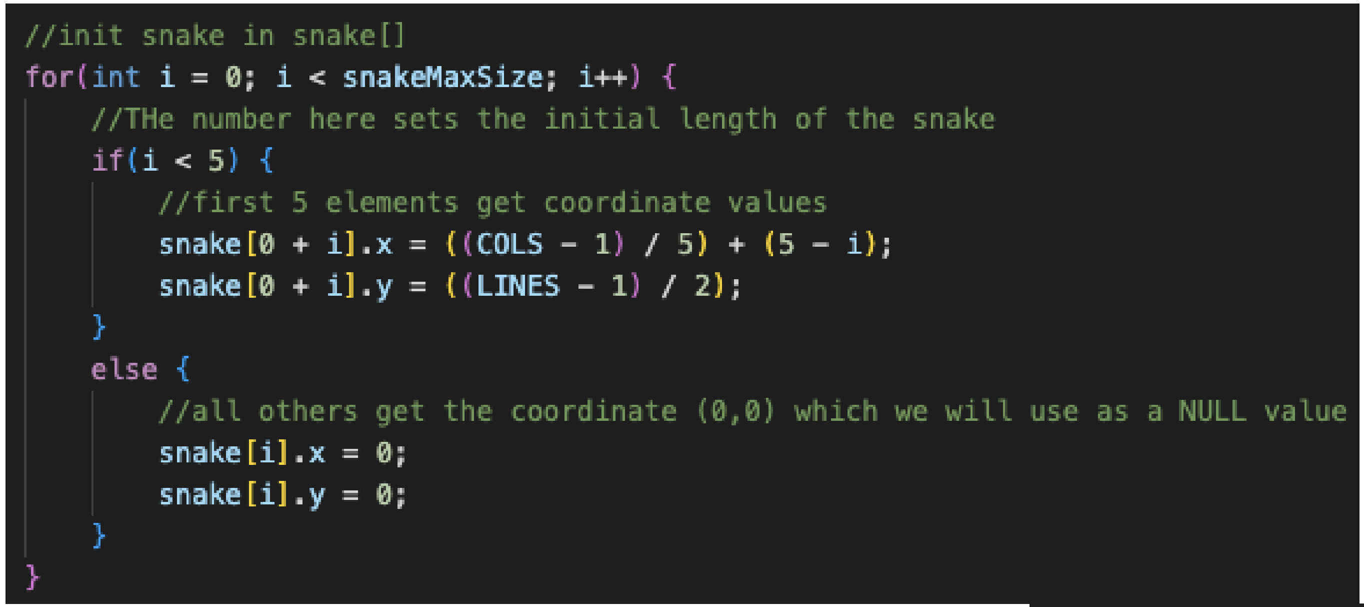
Tasks	Sean	Mike
Snake Pit	✓	
Init the Snake		✓
Movement	✓	✓
Adjusting Speed	✓	
Display Trophies		✓
Trophy Expiration	✓	✓
Eating Trophies	✓	✓
WIN/LOSE States		✓
Themes	✓	
This Table	✓	✓

The Snake Pit

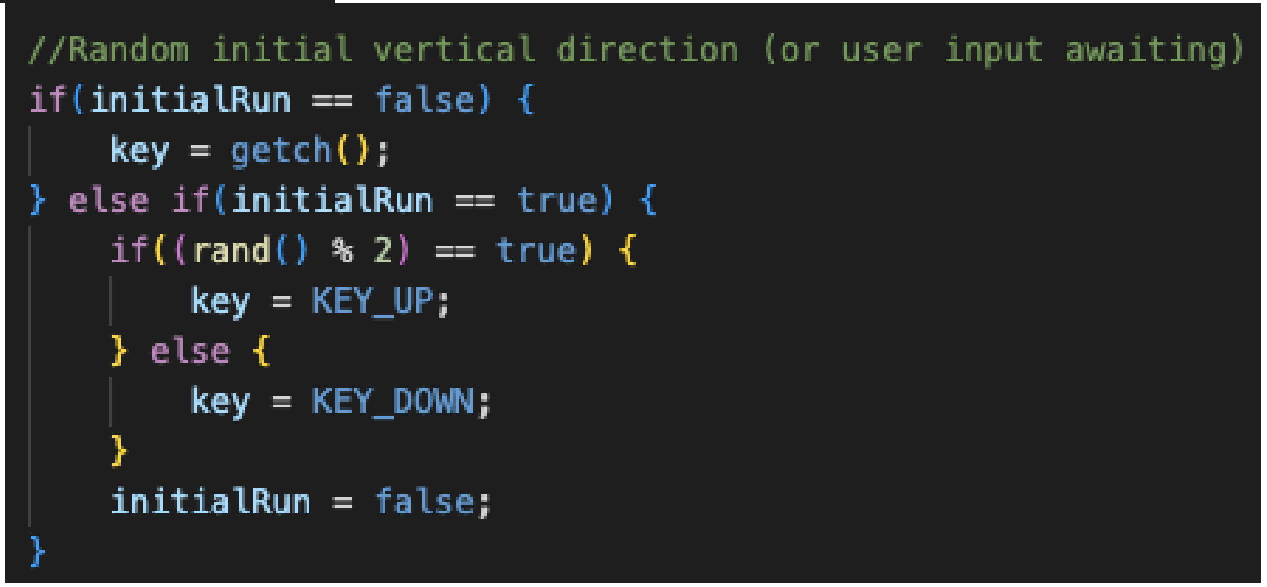


```
// Creating snake pit border for given terminal window
WINDOW * win = newwin(LINES - 1, COLS - 1, 0, 0);      //creates window equal to size of screen
refresh();                                             //refresh output
box(win, 0, 0);                                        //draw box equal to size of window/screen
wrefresh(win);                                         //refresh output
nodelay(stdscr, TRUE);                                 //disables pause when prompting for input
//Print score display
move(0,0);
addstr("Score:     ");
refresh();
```

Initial Snake Length and Initial Movement

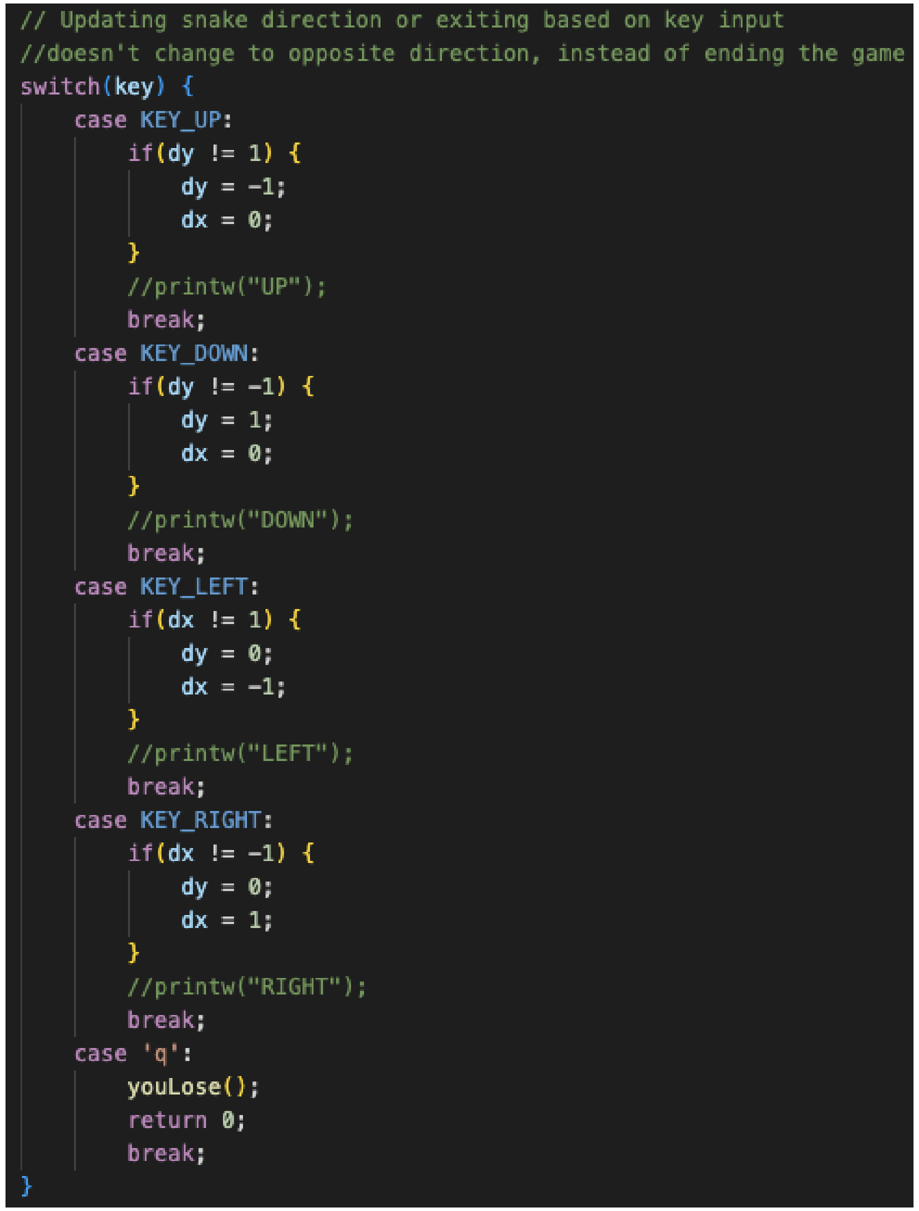


```
//init snake in snake[]
for(int i = 0; i < snakeMaxSize; i++) {
    //THe number here sets the initial length of the snake
    if(i < 5) {
        //first 5 elements get coordinate values
        snake[0 + i].x = ((COLS - 1) / 5) + (5 - i);
        snake[0 + i].y = ((LINES - 1) / 2);
    }
    else {
        //all others get the coordinate (0,0) which we will use as a NULL value
        snake[i].x = 0;
        snake[i].y = 0;
    }
}
```

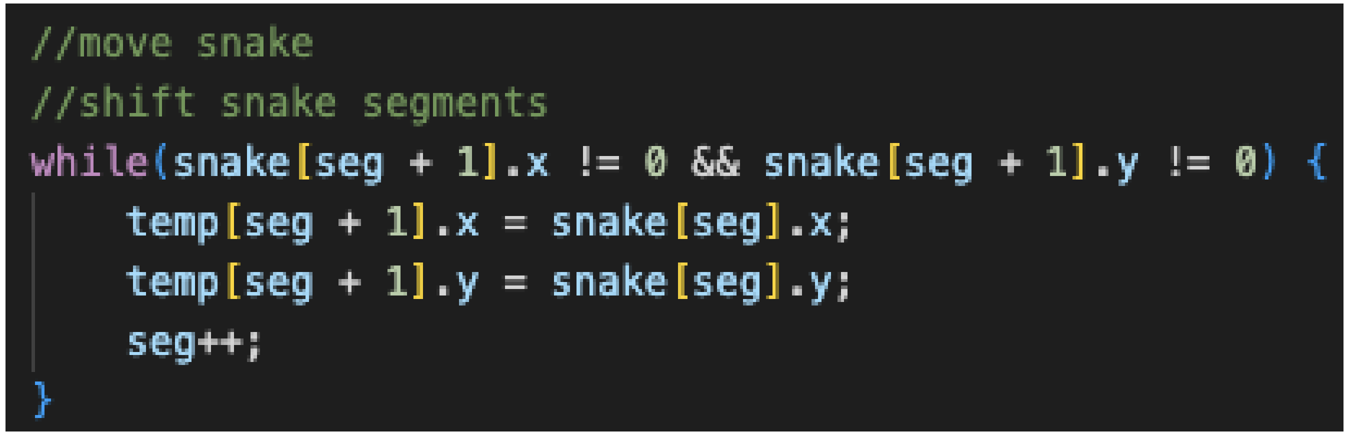


```
//Random initial vertical direction (or user input awaiting)
if(initialRun == false) {
    key = getch();
} else if(initialRun == true) {
    if((rand() % 2) == true) {
        key = KEY_UP;
    } else {
        key = KEY_DOWN;
    }
    initialRun = false;
}
```

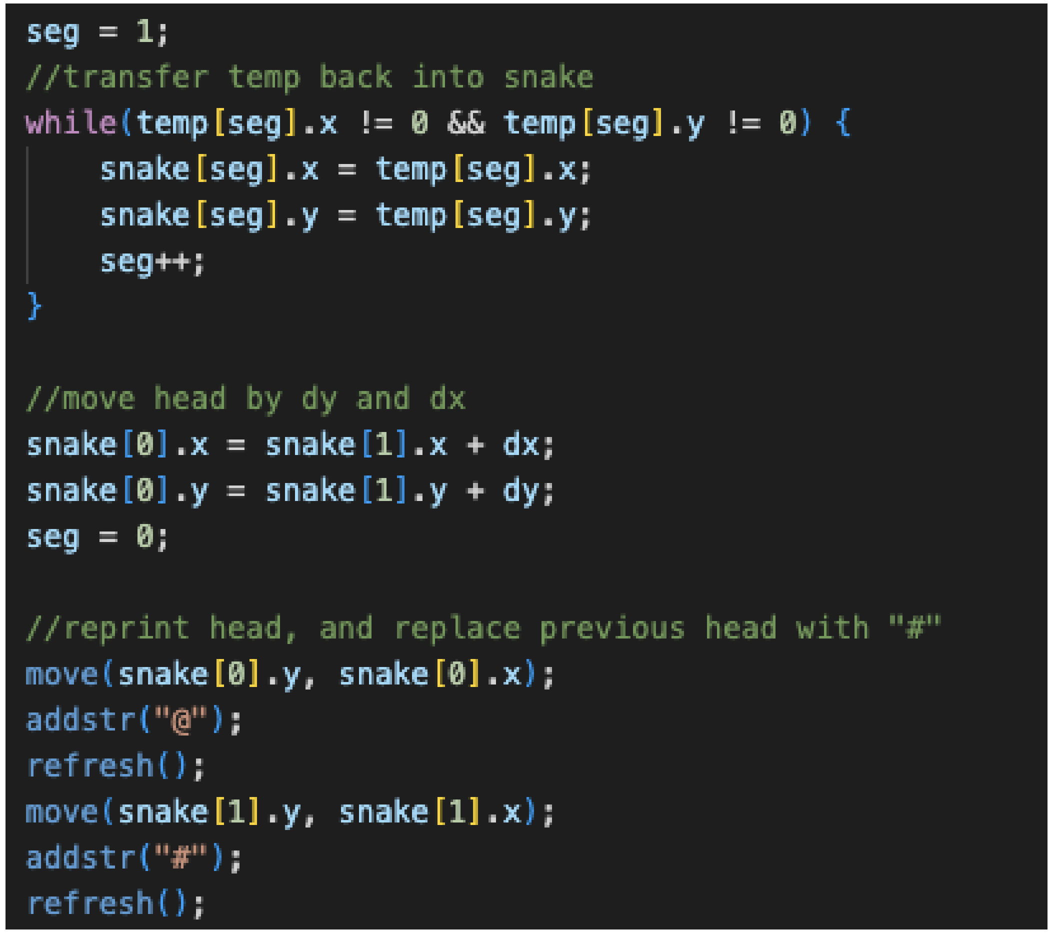
Snake Movement



```
// Updating snake direction or exiting based on key input
//doesn't change to opposite direction, instead of ending the game
switch(key) {
    case KEY_UP:
        if(dy != 1) {
            dy = -1;
            dx = 0;
        }
        //printw("UP");
        break;
    case KEY_DOWN:
        if(dy != -1) {
            dy = 1;
            dx = 0;
        }
        //printw("DOWN");
        break;
    case KEY_LEFT:
        if(dx != 1) {
            dy = 0;
            dx = -1;
        }
        //printw("LEFT");
        break;
    case KEY_RIGHT:
        if(dx != -1) {
            dy = 0;
            dx = 1;
        }
        //printw("RIGHT");
        break;
    case 'q':
        youLose();
        return 0;
        break;
}
```

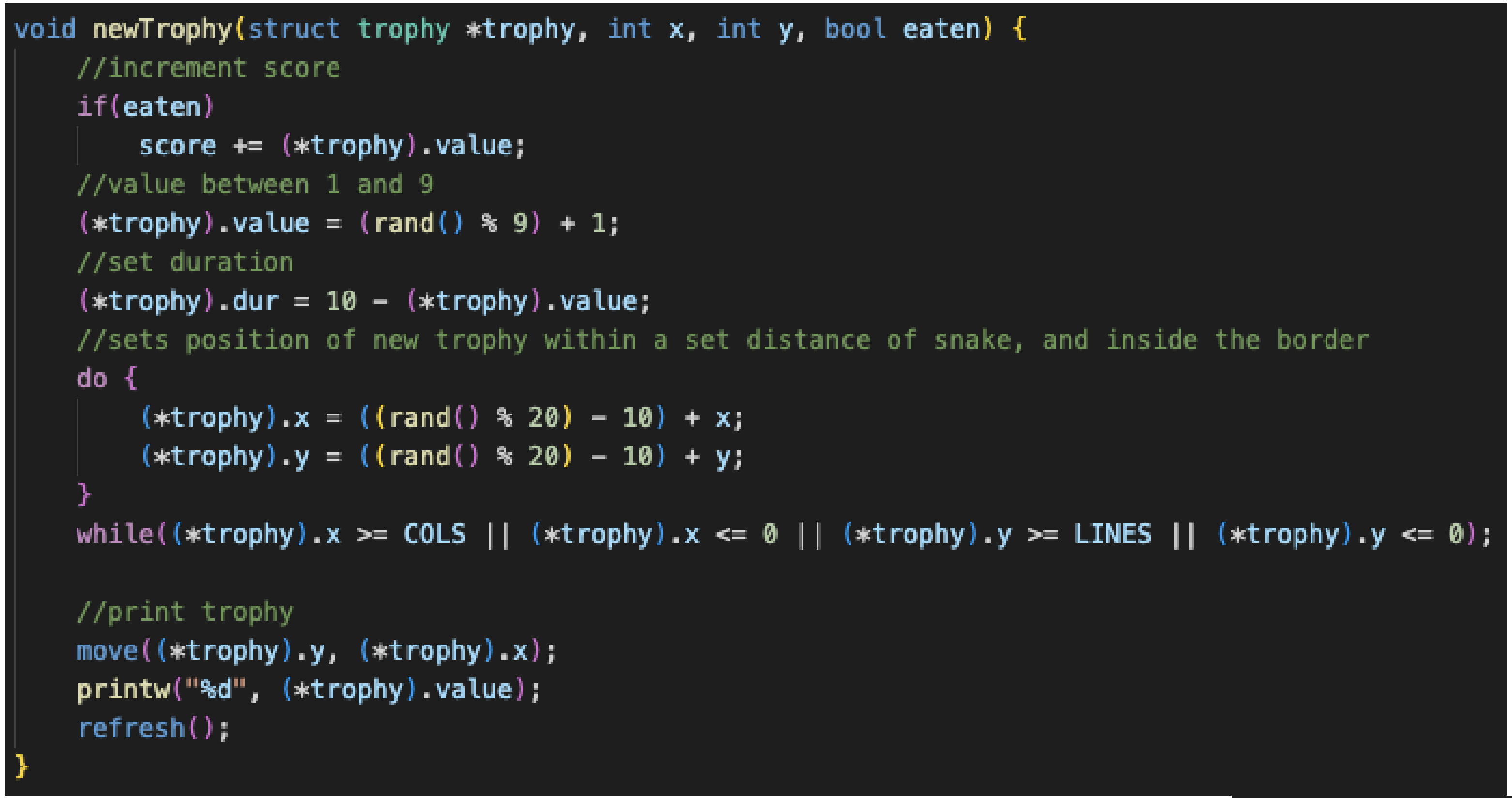


```
//move snake
//shift snake segments
while(snake[seg + 1].x != 0 && snake[seg + 1].y != 0) {
    temp[seg + 1].x = snake[seg].x;
    temp[seg + 1].y = snake[seg].y;
    seg++;
}
```

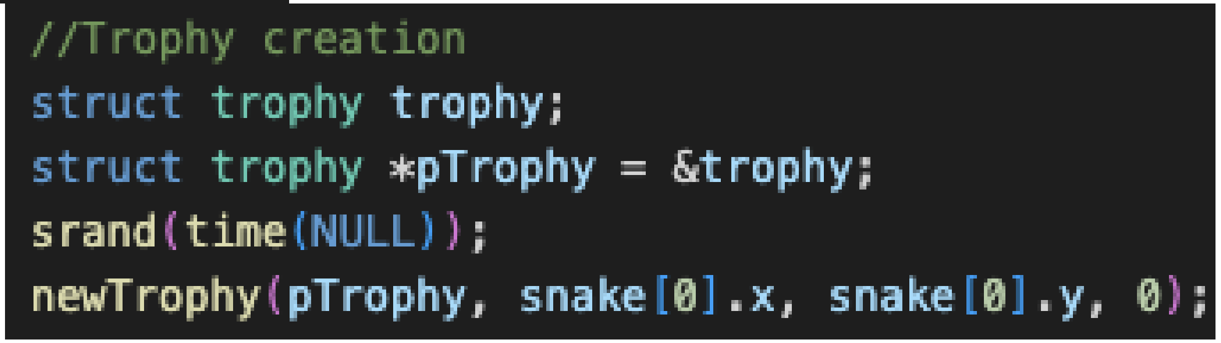


```
seg = 1;
//transfer temp back into snake
while(temp[seg].x != 0 && temp[seg].y != 0) {
    snake[seg].x = temp[seg].x;
    snake[seg].y = temp[seg].y;
    seg++;
}

//move head by dy and dx
snake[0].x = snake[1].x + dx;
snake[0].y = snake[1].y + dy;
seg = 0;

//reprint head, and replace previous head with "#"
move(snake[0].y, snake[0].x);
addstr("@");
refresh();
move(snake[1].y, snake[1].x);
addstr("#");
refresh();
```

Trophy Display and Movement

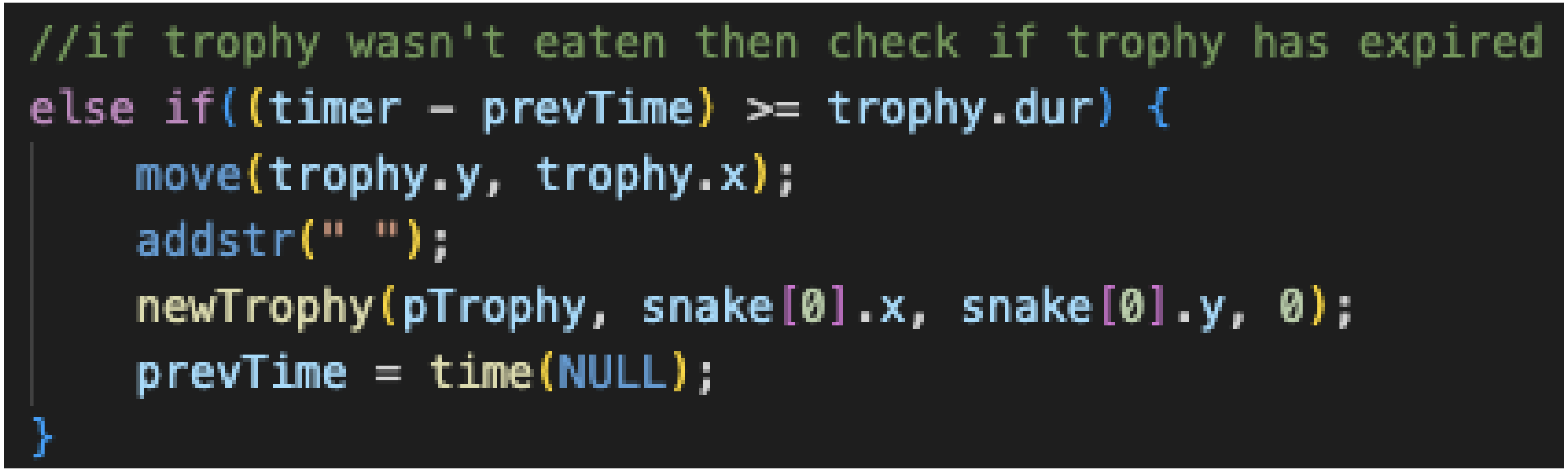


```c
void newTrophy(struct trophy *trophy, int x, int y, bool eaten) {
    //increment score
    if(eaten)
        score += (*trophy).value;
    //value between 1 and 9
    (*trophy).value = (rand() % 9) + 1;
    //set duration
    (*trophy).dur = 10 - (*trophy).value;
    //sets position of new trophy within a set distance of snake, and inside the border
    do {
        (*trophy).x = ((rand() % 20) - 10) + x;
        (*trophy).y = ((rand() % 20) - 10) + y;
    }
    while((*trophy).x >= COLS || (*trophy).x <= 0 || (*trophy).y >= LINES || (*trophy).y <= 0);

    //print trophy
    move((*trophy).y, (*trophy).x);
    printw("%d", (*trophy).value);
    refresh();
}
```



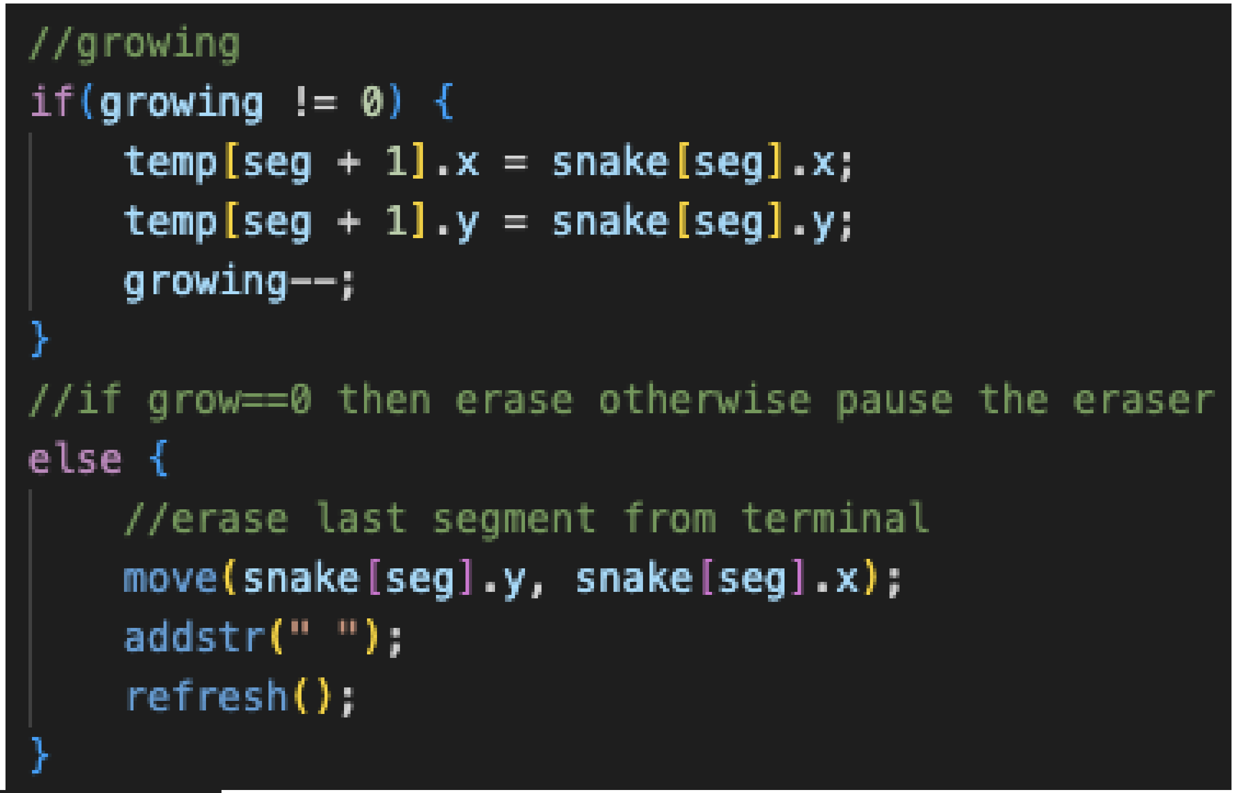
```c
//Trophy creation
struct trophy trophy;
struct trophy *pTrophy = &trophy;
srand(time(NULL));
newTrophy(pTrophy, snake[0].x, snake[0].y, 0);
```

Trophy Expiration

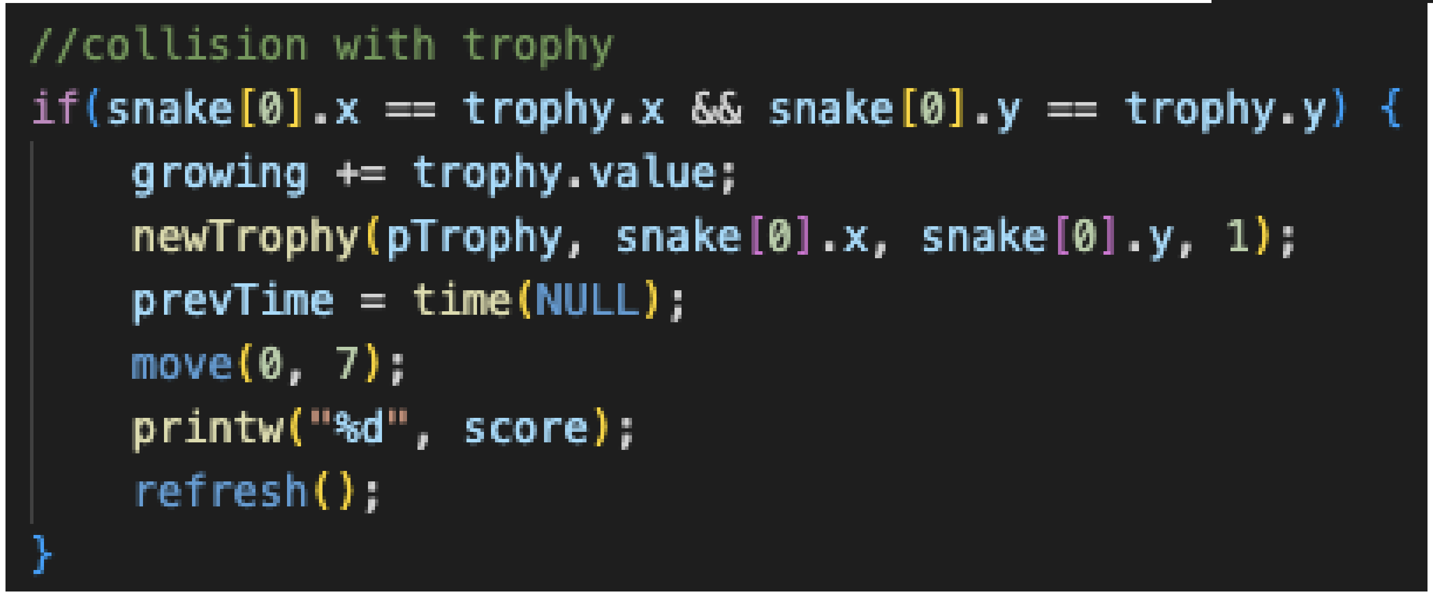


```c
//if trophy wasn't eaten then check if trophy has expired
else if((timer - prevTime) >= trophy.dur) {
    move(trophy.y, trophy.x);
    addstr(" ");
    newTrophy(pTrophy, snake[0].x, snake[0].y, 0);
    prevTime = time(NULL);
}
```

Trophy Eating

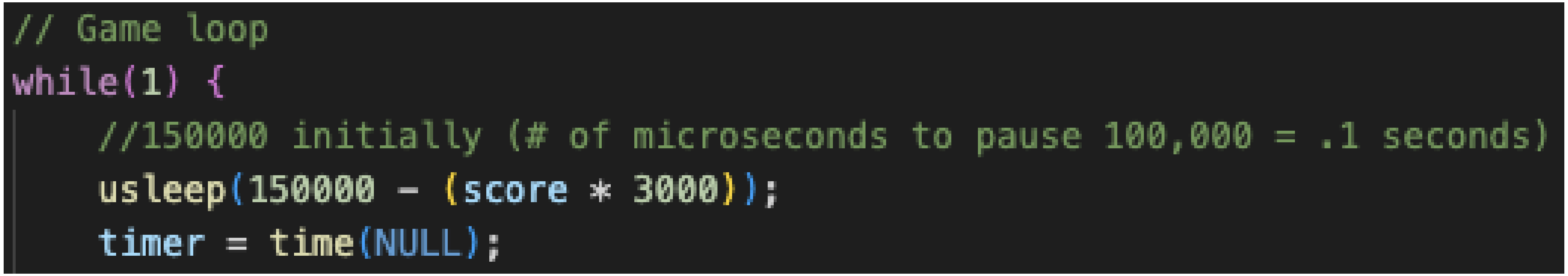


```
//growing
if(growing != 0) {
    temp[seg + 1].x = snake[seg].x;
    temp[seg + 1].y = snake[seg].y;
    growing--;
}
//if grow==0 then erase otherwise pause the eraser
else {
    //erase last segment from terminal
    move(snake[seg].y, snake[seg].x);
    addstr(" ");
    refresh();
}
```

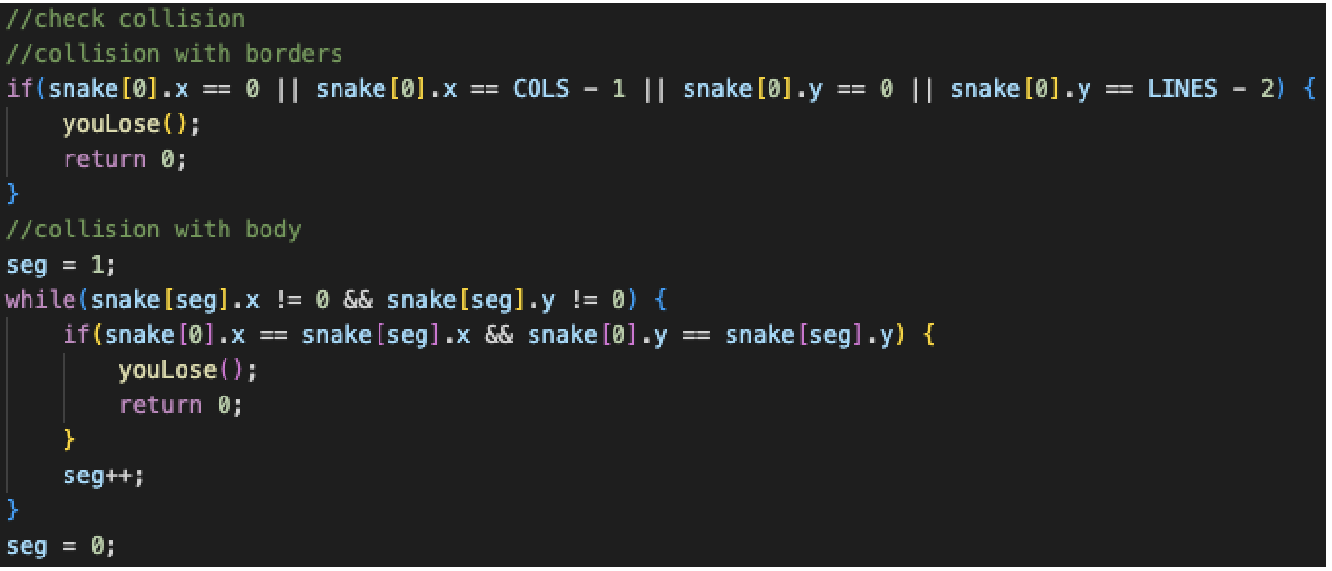


```
//collision with trophy
if(snake[0].x == trophy.x && snake[0].y == trophy.y) {
    growing += trophy.value;
    newTrophy(pTrophy, snake[0].x, snake[0].y, 1);
    prevTime = time(NULL);
    move(0, 7);
    printw("%d", score);
    refresh();
}
```

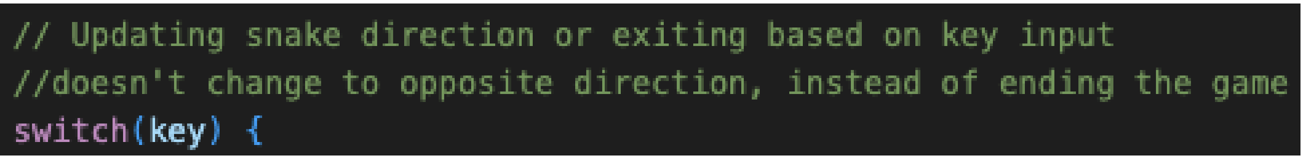
Snake Speed


```
// Game loop
while(1) {
    //150000 initially (# of microseconds to pause 100,000 = .1 seconds)
    usleep(150000 - (score * 3000));
    timer = time(NULL);
```

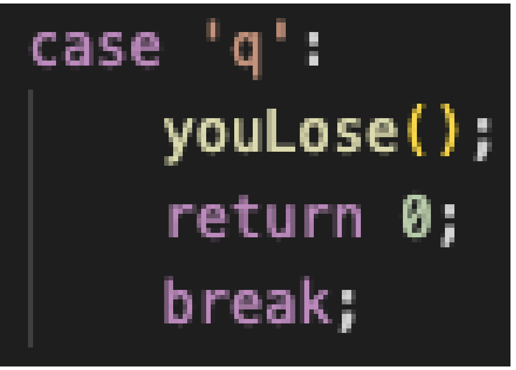
Game End Conditions



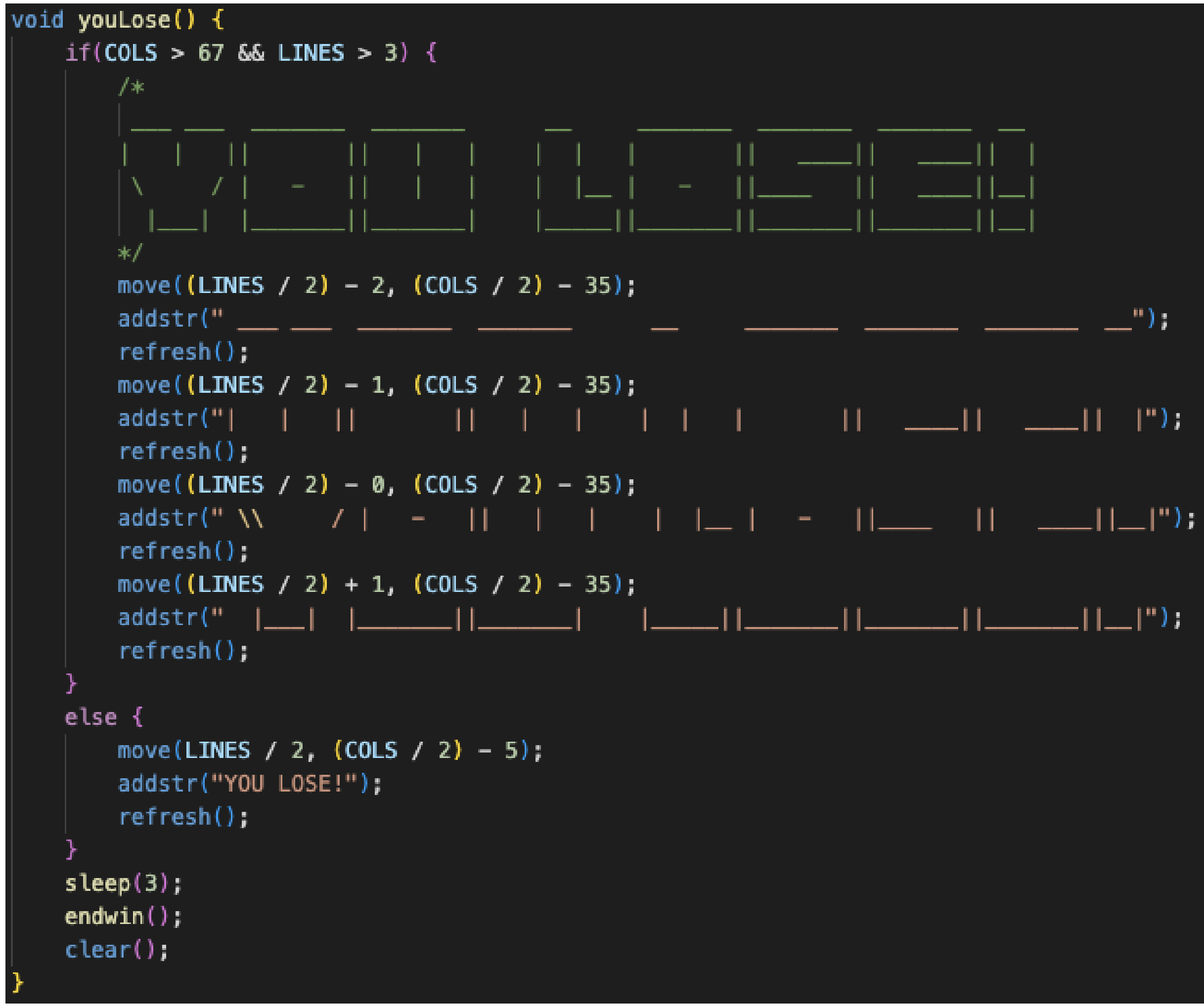
```c
//check collision
//collision with borders
if(snake[0].x == 0 || snake[0].x == COLS - 1 || snake[0].y == 0 || snake[0].y == LINES - 2) {
    youLose();
    return 0;
}
//collision with body
seg = 1;
while(snake[seg].x != 0 && snake[seg].y != 0) {
    if(snake[0].x == snake[seg].x && snake[0].y == snake[seg].y) {
        youLose();
        return 0;
    }
    seg++;
}
seg = 0;
```



```c
// Updating snake direction or exiting based on key input
//doesn't change to opposite direction, instead of ending the game
switch(key) {
```

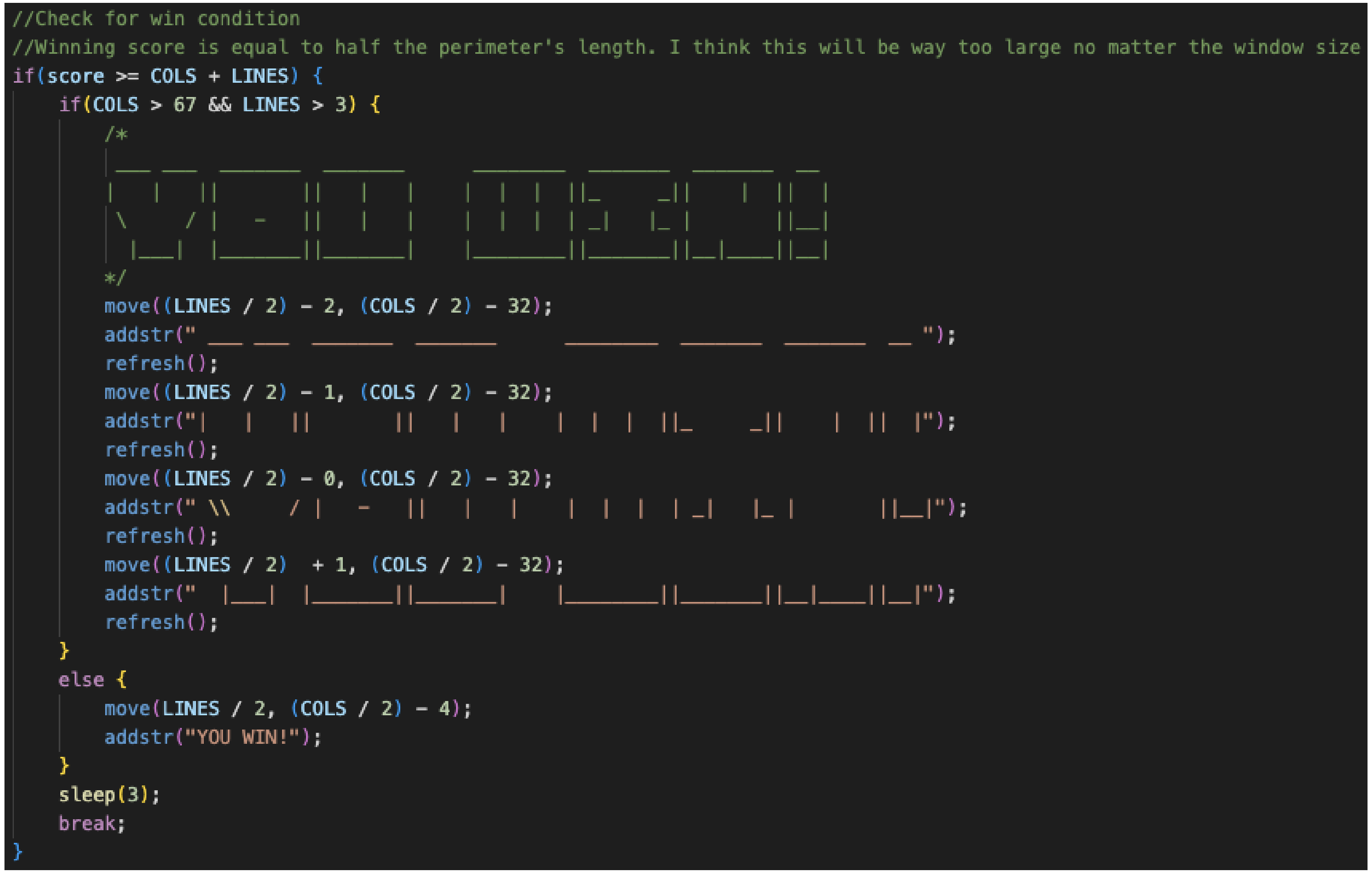


```c
case 'q':
    youLose();
    return 0;
    break;
```



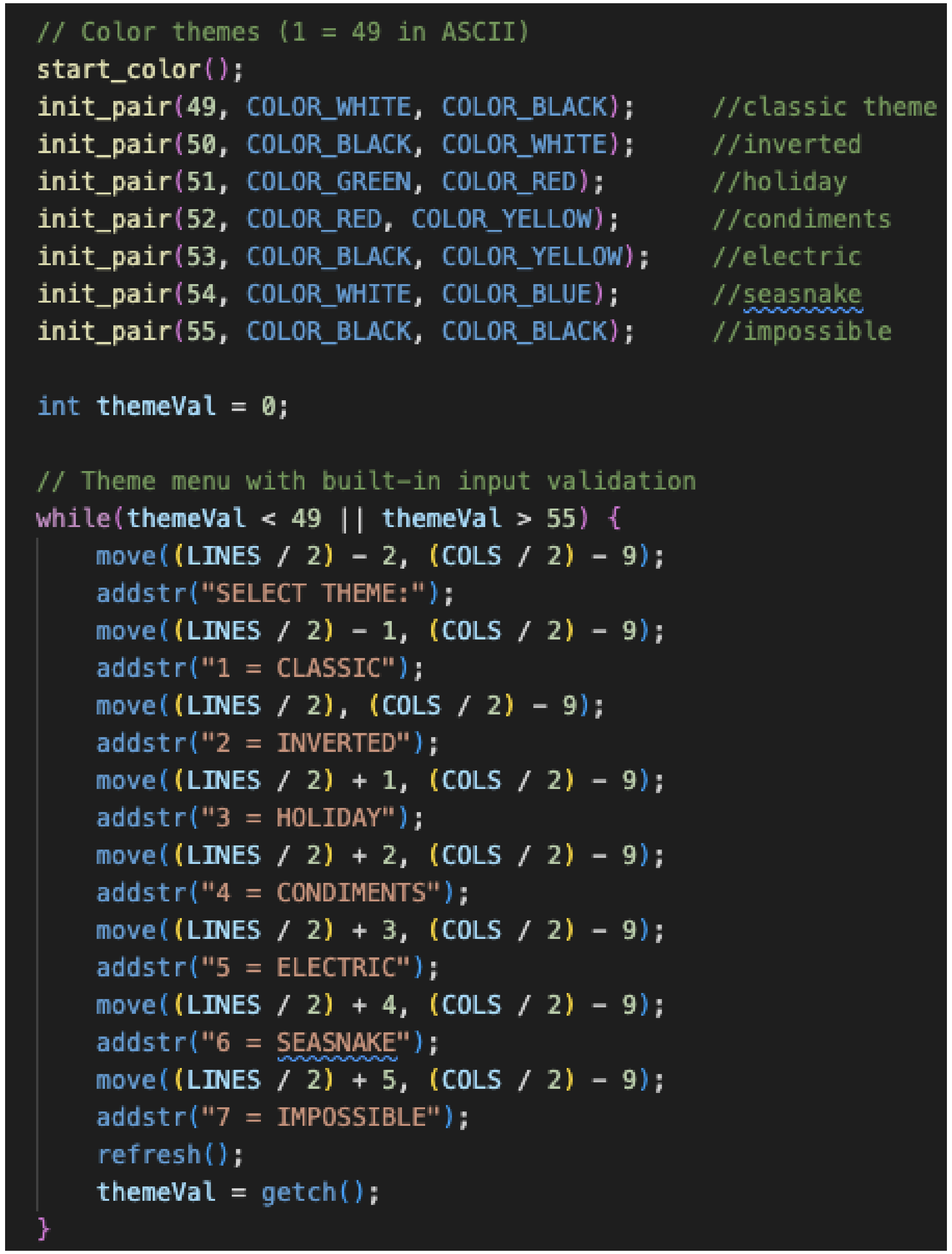
```c
void youLose() {
    if(COLS > 67 && LINES > 3) {
        /*
         ___ ___  _______  _______      __     _______  _______  _______  __
        |   |   ||       ||   |   |    |  |   |       ||    ___||    ___||  |
         \     / |   -   ||   |   |    |  |_  |   -   ||___    ||    ___||__|
          |___|  |_______||_______|    |_____||_______||_______||_______||__|
        */
        move((LINES / 2) - 2, (COLS / 2) - 35);
        addstr(" ___ ___  _______  _______      __     _______  _______  _______  __");
        refresh();
        move((LINES / 2) - 1, (COLS / 2) - 35);
        addstr("|   |   ||       ||   |   |    |  |   |       ||    ___||    ___||  |");
        refresh();
        move((LINES / 2) - 0, (COLS / 2) - 35);
        addstr(" \\     / |   -   ||   |   |    |  |_  |   -   ||___    ||    ___||__|");
        refresh();
        move((LINES / 2) + 1, (COLS / 2) - 35);
        addstr("  |___|  |_______||_______|    |_____||_______||_______||_______||__|");
        refresh();
    }
    else {
        move(LINES / 2, (COLS / 2) - 5);
        addstr("YOU LOSE!");
        refresh();
    }
    sleep(3);
    endwin();
    clear();
}
```

Game Win Condition

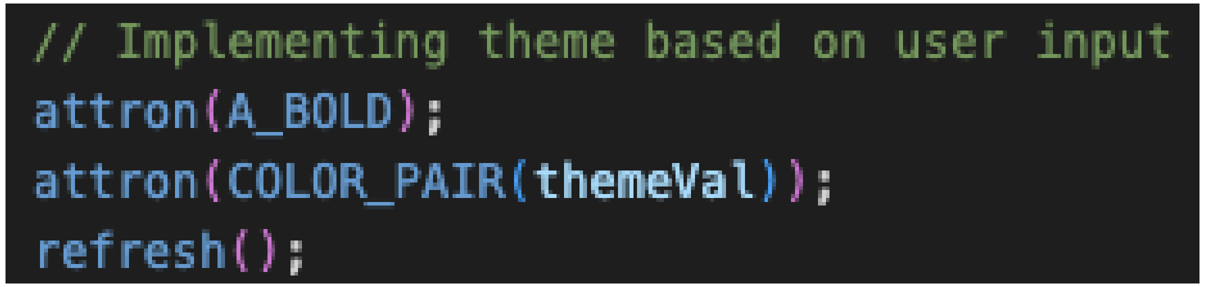


```c
//Check for win condition
//Winning score is equal to half the perimeter's length. I think this will be way too large no matter the window size
if(score >= COLS + LINES) {
    if(COLS > 67 && LINES > 3) {
        /*
         ___ ___  _______  _______      ________  _______  _______  __ 
        |   |   ||       ||   |   |    |  |  |  ||_     _||    |  ||  |
         \     / |   -   ||   |   |    |  |  |  | _|   |_ |       ||__|
          |___|  |_______||_______|    |________||_______||__|____||__|
        */
        move((LINES / 2) - 2, (COLS / 2) - 32);
        addstr(" ___ ___  _______  _______      ________  _______  _______  __ ");
        refresh();
        move((LINES / 2) - 1, (COLS / 2) - 32);
        addstr("|   |   ||       ||   |   |    |  |  |  ||_     _||    |  ||  |");
        refresh();
        move((LINES / 2) - 0, (COLS / 2) - 32);
        addstr(" \\     / |   -   ||   |   |    |  |  |  | _|   |_ |       ||__|");
        refresh();
        move((LINES / 2)  + 1, (COLS / 2) - 32);
        addstr("  |___|  |_______||_______|    |________||_______||__|____||__|");
        refresh();
    }
    else {
        move(LINES / 2, (COLS / 2) - 4);
        addstr("YOU WIN!");
    }
    sleep(3);
    break;
}
```

Bonus Features



```
// Color themes (1 = 49 in ASCII)
start_color();
init_pair(49, COLOR_WHITE, COLOR_BLACK);      //classic theme
init_pair(50, COLOR_BLACK, COLOR_WHITE);      //inverted
init_pair(51, COLOR_GREEN, COLOR_RED);        //holiday
init_pair(52, COLOR_RED, COLOR_YELLOW);       //condiments
init_pair(53, COLOR_BLACK, COLOR_YELLOW);     //electric
init_pair(54, COLOR_WHITE, COLOR_BLUE);       //seasnake
init_pair(55, COLOR_BLACK, COLOR_BLACK);      //impossible

int themeVal = 0;

// Theme menu with built-in input validation
while(themeVal < 49 || themeVal > 55) {
    move((LINES / 2) - 2, (COLS / 2) - 9);
    addstr("SELECT THEME:");
    move((LINES / 2) - 1, (COLS / 2) - 9);
    addstr("1 = CLASSIC");
    move((LINES / 2), (COLS / 2) - 9);
    addstr("2 = INVERTED");
    move((LINES / 2) + 1, (COLS / 2) - 9);
    addstr("3 = HOLIDAY");
    move((LINES / 2) + 2, (COLS / 2) - 9);
    addstr("4 = CONDIMENTS");
    move((LINES / 2) + 3, (COLS / 2) - 9);
    addstr("5 = ELECTRIC");
    move((LINES / 2) + 4, (COLS / 2) - 9);
    addstr("6 = SEASNAKE");
    move((LINES / 2) + 5, (COLS / 2) - 9);
    addstr("7 = IMPOSSIBLE");
    refresh();
    themeVal = getch();
}
```



```
// Implementing theme based on user input
attron(A_BOLD);
attron(COLOR_PAIR(themeVal));
refresh();
```

Game Demo Video

Questions?