Contents

1 Introduction

This project report is for centipede game which is different from the original one - ***Atari Centipede***. However, I try to make the game more like its original.
Attentions: This game requires ncurses library, which only could be used in Unix OS. Before I start, I suggest you to install this library by the following command in Debian/Ubuntu Linux:

```
sudo apt-get install libncurses5-dev libncursesw5-dev
//libncurses5-dev : Developer's libraries for ncurses
//libncursesw5-dev : Developer's libraries for ncursesw
```

1.1 Restatement

Based on the arcade original, I rewrite the rules of my own game. In order to make my game more stable, clear and smooth, I make some essential assumptions.

1.1.1 Game Regulations

For master:
I call the player as 'Master', Master could turn left, right, up and down by direction key. In addition, if player wants to fire a bullet, press space key can easily make it work. Just remember master only has 4 lives, be careful!

For mushroom:
It is the main obstacle for players to move and fire. Mushrooms can be destroyed after taking four shots or eating by spider.

For spider:
It is one kind of the headstrong guy in this game. It could either move up or down or move around. The moving tracks are similar to 'W' and 'I'. In addition, it is also some people tern a 'frienemy'- If it hits mushroom, mushroom will be destroyed. Conversely, master will die once spider hits him. For the sake of security, I strongly advise players to avoid it unless they could shoot it.

For sea monster:
In ***Atari Centipede***, it has scorpion. But here, I have sea monster because I think it is cool. It is another headstrong guy in this game. Compared to spider, it is totally a bad guy who is always loafing around horizontally and doing nothing but hitting master and making it die.

For centipede:
It is the major target of players. If nothing happens, centipede just walks line by line. Every time player shoots the centipede, the shot segment becomes a mushroom. Then if the shot segment is no in the tail or the body centipede only leaves, the centipede will split into two, gain a new head and both descend towards the player but ascend towards the player if it is in the floor. Besides, if centipede hits the wall or mushroom, it will try to turn back and descend towards the player if

no other mushroom and centipede in where it will descend. Keep yourself away from centipede in case of hitting, which gonna takes you life!

To make above game regulations more easier to understand, I create table 1:

Table 1: The result of each collision relationship

Hit	Master	bullets&Points	Mushroom	Centipede
Spider	Master die lose one life	(1 bullet, 600 points) Spider die	Mushroom die mushrooms- -	Ignore
Sea monster	Master die lose one life	(1 bullet, 600 points) Sea monster die	Ignore	Ignore

Centipede	Master die
	lose one life
		If bullet hits its head
		{
		100 points
		}
		else
		{
		10 points;
		}
			If no other mushrooms
			under it
			{
			turn back and descend;
			}
			else if its head is between
			mushroom and wall
			{
			descend;
			}
			else
			{
			turn back;
			}
				One of them
				turns back

Mushroom	Mushroom die mushroom- -	(4 bullet, 1 point)	Ignore	No kind of this case

Further, I set different levels for this game. The initial lengths of centipede in different levels are the same but this doesn't mean easy. The more mushrooms there are, the harder it is for the player to shoot the centipede. Needless to say spider and sea monster.

1.1.2 Assumption

While I ran my own game, I met some unexpected circumstances. Therefore, I list some necessary assumptions:

- At the beginning of each level, If mushroom position coincides with the initial position of the centipede, then player could believe that these mushrooms have been 'eaten' by centipede.
- The spider only appears in the lower part of the game window. Conversely, the sea monster only appears in the upper part of the game window.
- The player's scope of activity should not exceed the boundary of the window. In addition, Master couldn't hit mushroom but bullet could.
- The shot segments become mushrooms, these mushrooms will always be there whichever level is being played or whatever win or lose unless player use 4 bullets to destroyed them.

2 Game Flow Chart

2.1 Computer Running Game Flow

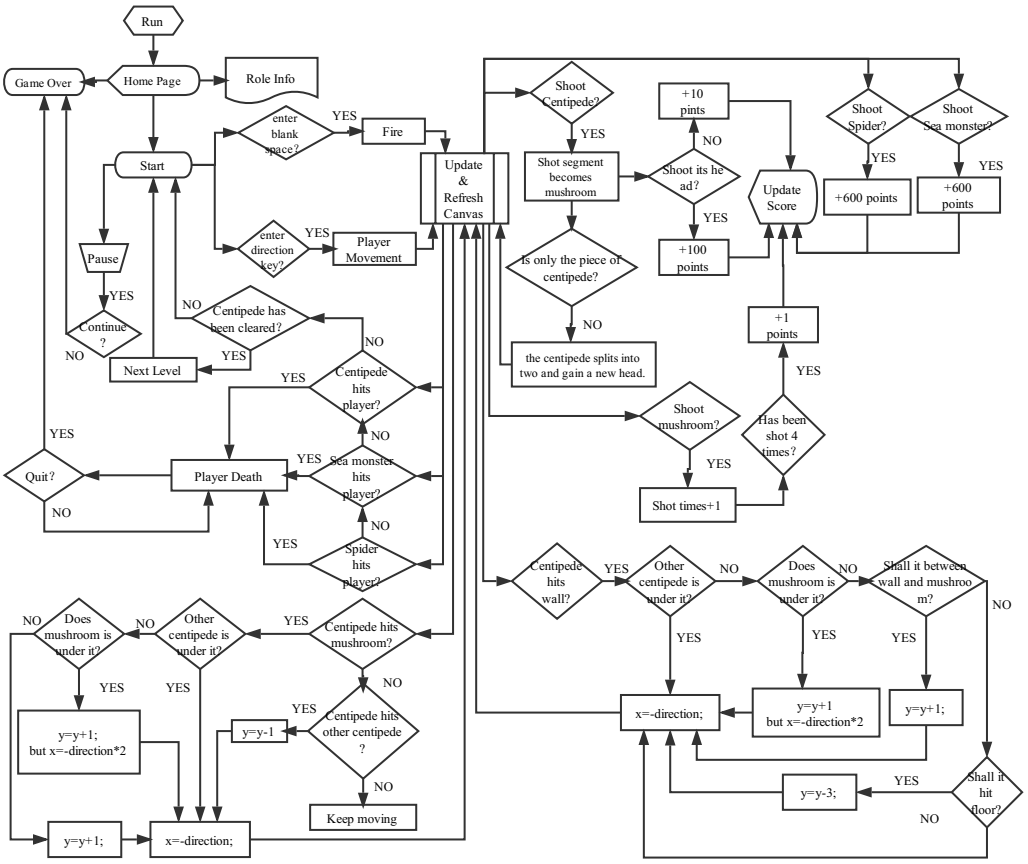


Figure 1: Game Flow Chart

Figure 1 shows all the processes of the game in detail, including the collision results, score updates and characters movements, etc. In this process, each time computer needs to get order from I/O and determine the collision judgment whether is satisfied or not, and then, refresh the canvas and update the scores.

If you're unfamiliar with this process, you can see my game screen recorder in YouTube:

`https://www.youtube.com/watch?v=iQRfnJUAYQs`

Or go to my ProcessOn website:

`https://www.processon.com/view/link/5b040ea4e4b05f5d6b641243`

2.2 Function Implementations

2.2.1 Start Menu

Firstly, I want to create a home page menu to give a short introduction about the game rule. This roll-up menu have 2 choices: play or not.

Home Page Text:

```
char *startMenu[] =
{
  "Start", "Exit",
};
char *readme[] =
{
        "*Centipede Game*",
        " ",
        "This game is written by Miao Cai, Lancaster University.",
        "Before the game, you'd better recognize these following characters.",
        " ",
        "*Character*",
        " ",
        "_^_",
        "|H|  Master",
        "___",
        " ",
        "This role is for you! you can move it by pressing keyboard:",
        "KEY_LEFT, KEY_RIGHT, KEY_UP and KEY_DOWN.",
        "Besides, you can fire bullets by pressing the blank space key.",
        " ",
        "*********O Centipede",
        " ",
        "This is the major character you need to shoot.",
        "Once it hits you, you lose your life.",
        "You only have 4 lives, be careful!",
        " ",
        "& Mushroom",
        " ",
        "This is a barrier, you cannot walk through it unless you destory it.",
        "Once you destory one of them, you will lost 4 bullets.",
        "Don't worry, you have tons of bullets.",
        " ",
        "_$ Sea monster",
        " ",
        "This character walks line by line. You can shoot it and get bonus.",
        "However, once it hits you, you lose your life.",
        " ",
        "^W^ Spider",
        " ",
        "This character crawls all the time.",
        "Spider could either help you by eating mushrooms or hitting you.",
        " ",
```

```
        "*Tips*",
        " ",
        "Press \'P\' or \'p\' to pause.",
        "Press \'C\' or \'c\' to continue.",
        "Press \'Q\' or \'q\' to quit or replay.",
        " ",
        "All in all, be careful! Hope you enjoy the game!",
        " ",
};
```

After that, I need to create a window to show this menu. Because I couldn't find a way to make my text scroll, so I just put them all into menu but only give my text scroll attribution.

Introduction Menu Source:

```
void start()
{
    ITEM **start_items;
    int c;
    MENU *menu;
    WINDOW *start_menu;
    int n_startMenu, item;
    int n_readme;
    /* Curses initialization */
    initscr();
    start_color();
    cbreak();
    noecho();
    init_pair(1, COLOR_WHITE, COLOR_BLACK);
    init_pair(2, COLOR_RED, COLOR_YELLOW);
    /* Create introduction menu items.
     * The following operations mainly aim to implement the following requirement:
     * 1. Readme only shows words and could scroll.
     * 2. Highlight all choices of startMenu.
     */
    n_startMenu = ARRAY_SIZE(startMenu);
    n_readme=ARRAY_SIZE(readme);
    start_items = (ITEM **)calloc(n_startMenu+n_readme+1, sizeof(ITEM *));
    for(item = 0;item<n_readme;item++)
    {
        start_items[item] = new_item(readme[item], readme[item]);
        item_opts_off(start_items[item], O_SELECTABLE);
        set_item_userptr(start_items[item], Click);
    }
    int cho=0;
    for(;item<n_readme+n_startMenu;item++)
    {
        start_items[item] = new_item(startMenu[cho], startMenu[cho]);
        /* Set the user pointer, even in readme*/
        set_item_userptr(start_items[item], Click);
        cho++;
    }
    /*Set the last one is NULL*/
    start_items[n_readme+n_startMenu] = (ITEM *)NULL;
    /* Crate menu */
```

```
    menu = new_menu((ITEM **)start_items);
    /* This color will be used in startMenu's choices*/
    set_menu_fore(menu, COLOR_PAIR(2));
    set_menu_back(menu, COLOR_PAIR(1));
    set_menu_grey(menu, COLOR_PAIR(1));
    menu_opts_off(menu, O_SHOWDESC);
    /* Create the window and make it at center*/
    int max_y, max_x;
    getmaxyx(stdscr, max_y, max_x);
    int x_position=(max_x-box_length)/2;
    int y_position=(max_y-box_width)/2;
    start_menu= newwin(box_width, box_length, y_position, x_position);
    keypad(start_menu, TRUE);//To listen keyboard
    /* Set main window and sub window */
    set_menu_win(menu, start_menu);
    set_menu_sub(menu, derwin(start_menu, box_width-1, box_length-2, 1,
        1));
    set_menu_format(menu, 18, 1);// Set 18 lines inside the box and one
        choices each line
    /*Create borders around the windows*/
    box(start_menu, 0, 0);
    refresh();
    /* Post the menu */
    post_menu(menu);
    wrefresh(start_menu);
    while((c = wgetch(start_menu)) != KEY_END)
    {
        switch(c)
        { case KEY_DOWN:
                menu_driver(menu, REQ_DOWN_ITEM);
                break;
            case KEY_UP:
                menu_driver(menu, REQ_UP_ITEM);
                break;
            case KEY_NPAGE:
                menu_driver(menu, REQ_SCR_DPAGE);
                break;
            case KEY_PPAGE:
                menu_driver(menu, REQ_SCR_UPAGE);
                break;
            case 10: /* This is Enter key*/
            {
                /*To point to current choice*/
                ITEM *cur;
                void (*p)(char *);
                cur = current_item(menu);
                p = item_userptr(cur);
                p((char *)item_name(cur));
                pos_menu_cursor(menu);
                break;
            }
        }
        wrefresh(start_menu);
    }
    /* Unpost and free all the memory taken up */
    unpost_menu(menu);
    for(int i = 0; i < n_startMenu+n_readme; ++i)
```

```
96          free_item(start_items[i]);
97      free_menu(menu);
98      /*Quit this window and wait for another*/
99      endwin();
100 }
101 /*Filter choices*/
102 void Click(char *choice)
103 {
104   if (choice == "Exit")
105   {
106     endwin();
107     exit(0);
108   }
109   if (choice == "Start")
110   {
111     clear();
112     refresh();
113     Initializition();
114     GameInterface();
115   }
116 }
```

From line 27 to 40 I divided menu items into two parts, the first part is text, I don't want computer do more thing for it. But in the second part, I set user pointers to allow player make a choice. In order to remind player to choose, I highlight the background of choices (see line 47-49). The running result of above codes is shown in figure 2.

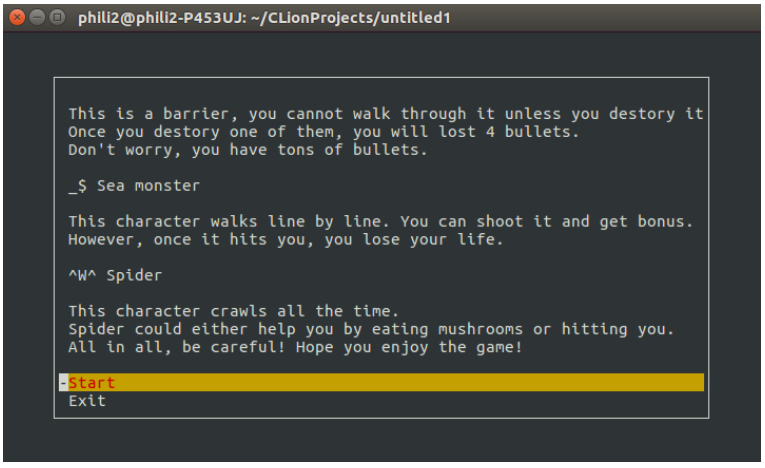


Figure 2: Home Page

2.2.2 Display Roles

Roles Initialization Source:

```
1 /*
2  * Produce mushroom
3  * The following operations mainly aim to implement the following
     requirements:
4  * 1. The mushroom position is randomly generated.
5  * 2. However, the last 3 lines and 1st line cannot be used because of
     master and centipede.
```

```c
 */
void MushroomProduce(int y,int x)
{
    int i=0;
    mushLength=20;
    srand(time(NULL));
    do {
        mushroom[i].x = rand() % (x-3)+1;
        mushroom[i].y =rand() % (y-4)+1;
        for (int j = 0; j < mushLength; j++) {
            if (mushroom[i].x == mushroom[j].x && mushroom[i].y == mushroom[j].y && i!=j)
            {
                mushroom[i].x = rand() % (x-2);
                mushroom[i].y =rand() % (y-4)+1;
            }
        }
        i++;
    }while (i<mushLength);/*Until no overlap*/
    for(int i=0;i<mushLength;i++)
    {
        mushroom[i].mush_record=4;
    }
}

/*
 * Produce master (player)
 * The following operations mainly aim to implement the following requirements:
 * 1. Make player central at the bottom of the screen.
 */
void MasterProduce(WINDOW *win)
{
    int win_y;
    int win_x;
    getmaxyx(win, win_y, win_x);
    master_1_y=win_y-3;
    master_1_x=(win_x-master_length)/2;
    master_2_y=win_y-2;
    master_2_x=(win_x-master_length)/2;
    master_3_y=win_y-1;
    master_3_x=(win_x-master_length)/2;
    curr_bullet_y = master_1_y;
    curr_bullet_x = master_1_x + 1;
    /*Draw in window*/
    mvwprintw(win,master_1_y,master_1_x,"_");
    mvwprintw(win,master_1_y,master_1_x+1,"^");
    mvwprintw(win,master_1_y,master_1_x+2,"_");
    mvwprintw(win,master_2_y,master_2_x,"|");
    mvwprintw(win,master_2_y,master_2_x+1,"H");
    mvwprintw(win,master_2_y,master_2_x+2,"|");
    mvwprintw(win,master_3_y,master_3_x,"-");
    mvwprintw(win,master_3_y,master_3_x+1,"-");
    mvwprintw(win,master_3_y,master_3_x+2,"-");
}
/*
 * Produce centipede
```

```
 * Start with the tail
 */
void CentipedeProduce(int x)
{
    int i;
    Length = 10;
    int start_x=(x-Length)/2;
    for (i = Length - 1; i >=0; i--) {
        Centipede[i].x = start_x;
        Centipede[i].y = 0;
        start_x++;
    }
    for(int i=0;i<Length;i++)
    {
        for(int j=0;j<mushLength;j++)
        {
            if(Centipede[i].x==mushroom[j].x&&Centipede[i].y==mushroom[j].y)
            {
                j++;
                while (j < mushLength) {
                    mushroom[j - 1].x = mushroom[j].x;
                    mushroom[j - 1].y = mushroom[j].y;
                    mushroom[j-1].mush_record = mushroom[j].mush_record;
                    j++;
                }
                mushLength -= 1;
            }
        }
        Centipede[i].head=-1;
        Centipede[i].Clear=-1;
        Centipede[i].x_direction=1;
        Centipede[i].y_direction=1;
    }
    Centipede[0].head=0;
}
/*
 * Produce sea monster
 * The following operations mainly aim to implement the following requirements:
 * Make sea monster appear randomly.
 */
void Sea_MonsterProduce(int y,int x)
{
    srand(time(NULL));
    int start_y=rand() % y/2+1;
    int num=rand() % 2;
    if(num==0)
    {
        Sea_Monster[0].x =1;
        Sea_Monster[0].y = start_y;
        Sea_Monster[1].x =0;
        Sea_Monster[1].y = start_y;
        Sea_Monster[0].x_direction=1;
    }
    if(num==1)
```

```
    {
        Sea_Monster[0].x =x-2;
        Sea_Monster[0].y = start_y;
        Sea_Monster[1].x =x-1;
        Sea_Monster[1].y = start_y;
        Sea_Monster[0].x_direction=-1;
    }
}
/*
 * Produce spider
 * The following operations mainly aim to implement the following requirements:
 * Make spider appear randomly.
 */
void SpiderProduce(int y,int x)
{
    int start_y=y-(rand() % y/2+1);
    if(num==1)
    {
        int start_x=0;
        for(int i=0;i<3;i++)
        {
            Spider[i].x =start_x;
            Spider[i].y = start_y;
            start_x++;
            Spider[i].x_direction=1;
            Spider[i].y_direction=1;
        }
    }
    if(num==-1)
    {
        int start_x=x-2;
        for(int i=0;i<3;i++)
        {
            Spider[i].x =start_x;
            Spider[i].y = start_y;
            start_x--;
            Spider[i].x_direction=-1;
            Spider[i].y_direction=1;
        }
    }
}
```

mushrooms are placed in random but their initial positions cannot be the same as master and centipede (see line 12-13), I use for loop in line 14 to create new random position of mushroom in case of duplication.

From line 34 to 40, **master** is placed in the bottom but center by using getmaxyx() function then I use 3 rows and 3 columns to store it.

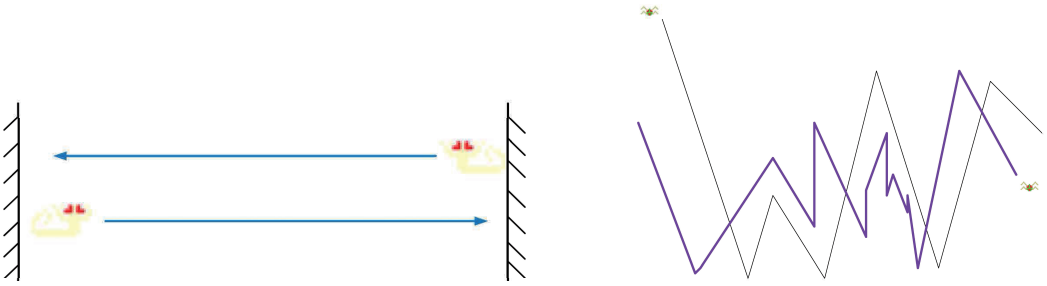
From line 61 to 66, **centipede** is placed in the top but center by using getmaxyx() function and I assigned positions(x,y) from the tail to the head.

From line 91 to 145 are about **spider** and **sea monster**. However, I don't want to see them always appear in the same position, so I use variable named "num" to decide whether they appears from the left or right of the wall. In addition, I used another random numbers to decide whether to let them appear (see below codes).

```
srand(time(NULL));
    int ran_sea=rand()%10;
    if(ran_sea==1)
    {
        sea_appear=1;
    }
    int ran_spider=rand()%5;
    if(ran_spider==1)
    {
        spider_appear=1;
    }
```

2.2.3 Enemies movement

These enemies are sea monster, spider and centipede. The first two of them are simpler than the last one.



(a) Sea monster (b) Spider

Figure 3: Trajectory diagram of sea monster and spider

As figure 3.a shown, sea monster walks horizontally, later if it hits the wall, computer will give a default position for it and make it disappear in the canvas.
Figure 3.b shows that spider walks at will and we can't find any rule. However, I could achieve more complex trajectory than the ***Atari Centipede*** by using 2 random numbers on both x and y direction. This might causes spider walks the same path (see figure 4) but it will finally walk from one side to another side because random x and y are non-negative number.

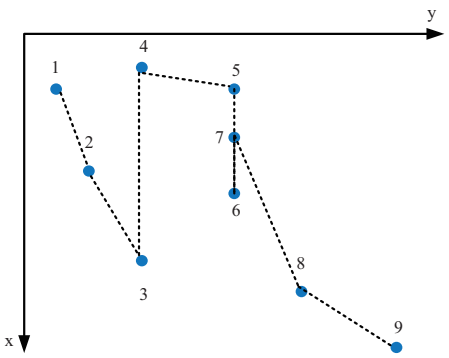


Figure 4: Spider's trace

Besides, I give spider x and y boundaries conditions in order that it will never hit its partner-sea monster. In addition, I set a switch to delay the walk speed of spider (not in following code) and make it start from left to right and then from

right to left by turns.

Sea Monster Movement Source:

```
void sea_monsterMove(int x)
{
        Sea_Monster[1].x=Sea_Monster[0].x;
        Sea_Monster[0].next_x=Sea_Monster[0].x + Sea_Monster[0].
            x_direction;
        /* If sea monster will hit the wall, make it disappear*/
        if (Sea_Monster[0].next_x > x || Sea_Monster[0].next_x < 0)
        {
            Sea_Monster[0].x=-1;
            Sea_Monster[1].x=-1;
            sea_appear=0;
        }
        else
        {
            Sea_Monster[0].x += Sea_Monster[0].x_direction;
        }
}
```

Spider Movement Source:

```
void SpiderMove(int x,int y)
{
    srand(time(NULL));
    int ran_y=y-(rand() % y/2+3);
    int ran_x=rand()%2;
    for(int j=0;j<3;j++)
    {
        Spider[j].next_x=Spider[j].x + ran_x*Spider[j].x_direction;
        Spider[j].next_y=Spider[j].y + Spider[j].y_direction;
        /* If spider will hit the wall, make it disappear and change
            its initial position*/
        if (Spider[j].next_x > x|| Spider[j].next_x < 0)
        {
            Spider[0].x=-1;
            Spider[1].x=-1;
            Spider[2].x=-1;
            spider_appear=0;
            num*=-1;
            break;
        }
        /* If spider will hit the floor, change its y direction*/
        if(Spider[j].next_y > y-1 || Spider[j].next_y < ran_y)
        {
            Spider[j].y_direction*=-1;
        }
            Spider[j].x+=ran_x*Spider[j].x_direction;
            Spider[j].y+=Spider[j].y_direction;
    }
}
```

As for centipede, it is the most complex case in my game. All special cases I could image are listed in figure 5 and figure 6. The major method I use is struct and fixed array. Array is thought to be stupid than linked list but I think it is helpful in collision because I could record each node's situation (see figure 10). Then each

time in movement, I only deal with the issue of their heads, later make their bodies follow their head by assigning previous node's positions. Finally, repaint the canvas. Specific operations you could see in following source code.

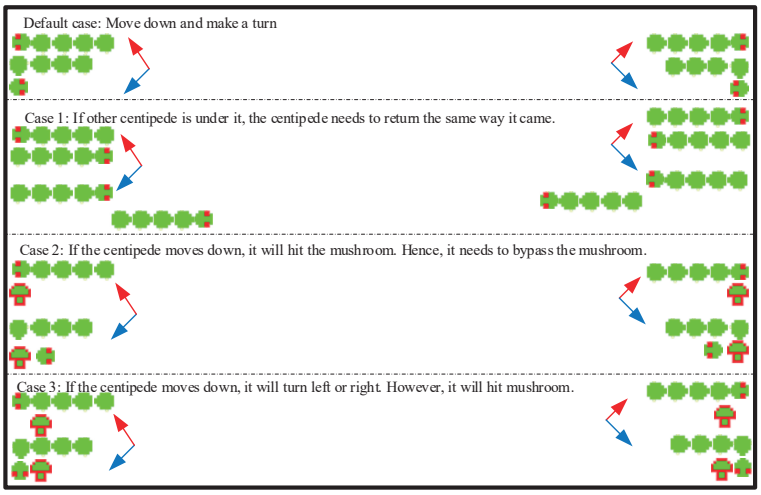


Figure 5: Centipede Movement Case 1

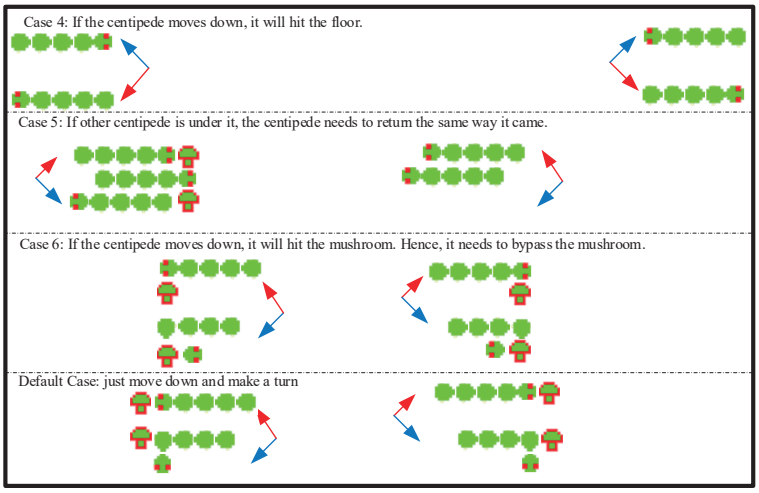


Figure 6: Centipede Movement Case 2

Centipede Movement Source:

```
void CentipedeMove(int x,int y)
{
    for(int i=0;i<Length;i++)
    {
        int skip=0;//This is used to mark
        /*Computer only moves the position of centipede which has not
            yet been shot*/
        if(Centipede[i].head>=0&&Centipede[i].Clear<0)
        {
            int j=i;
            int k=i+1;
            /*For multiple centipedes, computer recognize their head,
                and make their own bodies follow them*/
            while(k<=Length-1&&Centipede[k].head<0&&Centipede[k].Clear
                <0)
```

```
            {
                j++;
                k++;
            }
            while(j>i)
            {
                Centipede[j].x = Centipede[j-1].x;
                Centipede[j].y = Centipede[j-1].y;
                Centipede[j].x_direction = Centipede[j-1].x_direction;
                j--;
            }
            Centipede[i].next_x=Centipede[i].x + Centipede[i].x_direction;
            Centipede[i].next_y=Centipede[i].y + Centipede[i].y_direction;
            /* If the centipede head will hit the wall, it will move down and make a turn. However:
             * 1. If other centipede is under it, the centipede needs to return the same way it came.
             * 2. If the centipede moves down, it will hit the mushroom. Hence, it needs to bypass the mushroom.
             * 3. If the centipede moves down, it will turn left or right. However, it will hit mushroom.
             *    In this case, it should not turn left or right, just move down.
             * 4. If the centipede moves down, it will hit the floor. It needs to make a turn and turn up.
             */
            if (Centipede[i].next_x > x || Centipede[i].next_x < 0)
            {
                /*For case 1*/
                for(int k=0;k<Length;k++)
                {
                    if(Centipede[k].Clear<0&&(k<i||k>j))
                    {
                        if(Centipede[i].next_y==Centipede[k].y&&Centipede[i].x==Centipede[k].x)
                        {
                            Centipede[i].x_direction*= -1;
                            skip=1;
                        }
                    }
                }

                for(int j=0;j<mushLength;j++)
                {
                    /*For case 2*/
                    if(Centipede[i].next_y==mushroom[j].y&& Centipede[i].x==mushroom[j].x)
                    {
                        Centipede[i].x_direction*= -1;
                        Centipede[i].y+=1;
                        skip=1;
                        Centipede[i].x += Centipede[i].x_direction;
                    }
                    /*For case 3*/
                    if(Centipede[i].next_y==mushroom[j].y&& Centipede[i
```

```
                        ].x-1==mushroom[j].x||Centipede[i].next_y==mushroom[j].y&& Centipede[i].x+1==mushroom[j].x)
                    {
                        Centipede[i].y+=1;
                        skip=1;
                    }
                }
                /*For case 4*/
                if (Centipede[i].next_y > y)
                {
                    Centipede[i].y-=3;
                    Centipede[i].x_direction*= -1;
                    skip=1;
                }
                /*Default case, just move down and make a turn*/
                if(!skip)
                {
                    Centipede[i].x_direction*= -1;
                    Centipede[i].y+=1;
                }
            }
            else
            {
                /* If the centipede head will hit the mushroom, it will move down and make a turn. However:
                 * 5. If other centipede is under it, the centipede needs to return the same way it came.
                 * 6. If the centipede moves down, it will hit the mushroom. Hence, it needs to bypass the mushroom.
                 */
                for(int j=0;j<mushLength;j++)
                {
                    if(Centipede[i].next_x==mushroom[j].x&& Centipede[i].y==mushroom[j].y)
                    {
                        /*For case 5*/
                        for(int k=0;k<Length;k++)
                        {
                            if(Centipede[k].Clear<0&&(k<i||k>j))
                            {
                                if(Centipede[i].next_y==Centipede[k].y && Centipede[i].x==Centipede[k].x)
                                {
                                    Centipede[i].x_direction*= -1;
                                    skip=1;
                                }
                            }
                        }
                        /*For case 6*/
                        for(int k=0;k<mushLength;k++)
                        {
                            if(Centipede[i].next_y==mushroom[k].y&& Centipede[i].x==mushroom[k].x&&j!=k)
                            {
                                Centipede[i].x_direction*= -1;
                                Centipede[i].y+=1;
```

```
                                skip=1;
                                Centipede[i].x += Centipede[i].x_direction;
                            }
                        }
                        /*Default case, just move down and make a turn */
                        if(!skip)
                        {
                            Centipede[i].x_direction*= -1;
                            Centipede[i].y+=1;
                            skip=1;
                        }
                    }
                }
                /* For case 7
                 * If the centipede head will hit other centipedes in the same y direction, then:
                 * It will move down and make a turn.
                 */
                for(int k=0;k<Length;k++)
                {
                    if(Centipede[k].Clear<0&&(k<i||k>j))
                    {
                        if(Centipede[i].next_x==Centipede[k].x&&Centipede[i].y==Centipede[k].y)
                        {
                                Centipede[i].x_direction*= -1;
                                Centipede[i].y+=1;
                                skip=1;
                        }
                     }
                }
                /*If not above case happens, just move in x direction*/
                if(skip==0)
                {
                    Centipede[i].x += Centipede[i].x_direction;
                }
            }
        }
    }
}
```

2.2.4 Master Movement

I use getch() funtion to listen keyboard information. However, this is not enough because we need to listen all the time, which means that I couldn't directly put it because input/output protocol is synchronous. I need to make it asynchronous, hence I use another thread to receive keyboard information. There are few things to discuss, just make sure master cannot rush across the wall, mushroom, ceiling and floor. Here are source code:

Master Movement Source:

```
/*
 * Key Listener
```

```
 * The following operations mainly implement the following requirements:
 * 1. master could move left, right, up and down.
 * 2. master could fire by press blank space.
 * 3. If master will hit the mushroom, stop it.
 * 4. If master will hit the screen edge, stop it.
 * 5. When a bullet is fired, bullet's position should be where the current master is.
 */
 /*
 * This function is used to receive key from key board, all the time!
 */
void * waitForKey() {
    while (1) {
        usleep(10);//In case of macroblocking
        ch = getch();
    }
}
//Call thread in canvas function
    int listener;
    listener = pthread_create(& work, NULL, waitForKey, NULL);//Create thread for master
    if (listener != 0) {
        exit(1);
    }
void getOrder(WINDOW *win,int x,int y)
{
    int skip=0;
    switch(ch)
    {
        case KEY_LEFT:
            for(int i=0;i<mushLength;i++)
            {
                if(master_1_x==0||(master_1_y==mushroom[i].y&&master_1_x==mushroom[i].x+1)||(master_2_y==mushroom[i].y&&master_2_x==mushroom[i].x+1)||(master_3_y==mushroom[i].y&&master_3_x==mushroom[i].x+1))//on the  left hand of master
                {
                    //do nothing
                    skip=1;
                }
            }
            if(!skip)
            {
                master_1_x -= 1;
                master_2_x -= 1;
                master_3_x -= 1;
                curr_bullet_y = master_1_y;
                curr_bullet_x = master_1_x + 1;
            }
            break;
        case KEY_RIGHT:
            for(int i=0;i<mushLength;i++)
            {
                if ((master_1_x + 2) == x - 1 || (master_1_y == mushroom[i].y && master_1_x + 2 == mushroom[i].x -
```

```
                    1) ||
                    (master_2_y == mushroom[i].y && master_2_x + 2 ==
                        mushroom[i].x - 1) ||
                    (master_3_y == mushroom[i].y && master_3_x + 2 ==
                        mushroom[i].x - 1))
                {
                    //do nothing
                    skip=1;
                }
            }
            if(!skip)
            {
                master_1_x += 1;
                master_2_x += 1;
                master_3_x += 1;
                curr_bullet_y = master_1_y;
                curr_bullet_x = master_1_x + 1;
            }
            break;
        case KEY_UP:
            for(int i=0;i<mushLength;i++)
            {
                if ((master_1_y==mushroom[i].y+1&&master_1_x==mushroom[
                    i].x)||(master_1_y==mushroom[i].y+1&&master_1_x+1==
                    mushroom[i].x)||(master_1_y==mushroom[i].y+1&&
                    master_1_x+2==mushroom[i].x))
                {
                    //do nothing
                    skip=1;
                }
            }
            if(!skip)
            {
                master_1_y -= 1;
                master_2_y -= 1;
                master_3_y -= 1;
                curr_bullet_y = master_1_y;
                curr_bullet_x = master_1_x + 1;
            }
            break;
        case KEY_DOWN:
            for(int i=0;i<mushLength;i++)
            {
                if (master_3_y==y-1||(master_3_y==mushroom[i].y-1&&
                    master_3_x==mushroom[i].x)||(master_3_y==mushroom[i
                    ].y-1&&master_3_x+1==mushroom[i].x)||(master_3_y==
                    mushroom[i].y-1&&master_3_x+2==mushroom[i].x))
                {
                    //do nothing
                    skip=1;
                }
            }
            if(!skip)
            {
                master_1_y += 1;
                master_2_y += 1;
                master_3_y += 1;
```

```
                curr_bullet_y = master_1_y;
                curr_bullet_x = master_1_x + 1;
            }
            break;
        case ' ':
            bullet_x=curr_bullet_x;
            bullet_y=curr_bullet_y;
            Fire=1;
            break;
    }
    ch='0';
    mvwprintw(win,master_1_y,master_1_x,"_");
    mvwprintw(win,master_1_y,master_1_x+1,"^");
    mvwprintw(win,master_1_y,master_1_x+2,"_");
    mvwprintw(win,master_2_y,master_2_x,"|");
    mvwprintw(win,master_2_y,master_2_x+1,"H");
    mvwprintw(win,master_2_y,master_2_x+2,"|");
    mvwprintw(win,master_3_y,master_3_x,"—");
    mvwprintw(win,master_3_y,master_3_x+1,"—");
    mvwprintw(win,master_3_y,master_3_x+2,"—");
}
```

2.2.5 Menu Operation

I create 3 options in game interface. If player press 'P' or 'p', this button will disappear. All operation are stopped and "Continue" button appears (see figure 7.a). Inversely, If player press 'C' or 'c', this button will disappear. All operation keep working and "Pause" button appears (see figure 7.b). Besides, player could quit from game whenever they want, just press "Q" or 'q' and choose "Yes" (see figure 8).

(a) Pause Operation (b) Continue Operation

Figure 7: Pause and Continue Interface

Figure 8: Quit Interface

Menu Operation Source:

```c
/*This struct is used to achieve "pause" and "continue"*/
typedef struct PANEL_HIDE {
    int hide;
}Panel;
/*
 * Key Listener
 * The following operations mainly implement the following requirements:
 * 1. If I press "Continue", then it will disappear but "Pause" appear.
 * 2. If I press "Pause", then it will disappear but "Continue" appear.
 */
void getInt(PANEL *Con,PANEL *Pau,int x,int y)
{
    Panel *temp;
    switch(ch)
    {
        case 'c':
        case 'C':
            temp = (Panel *)panel_userptr(Con);
            hide_panel(Con);
            temp->hide = TRUE;
            if(temp->hide == TRUE)
            {
                temp = (Panel *) panel_userptr(Pau);
                show_panel(Pau);
                temp->hide = FALSE;
            }
            stop=0;
                interrupt_end = time(NULL);
            break;
        case 'p':
        case 'P':
            temp = (Panel *)panel_userptr(Pau);
            hide_panel(Pau);
            temp->hide = TRUE;
            temp = (Panel *)panel_userptr(Con);
            if(temp->hide == TRUE)
            { show_panel(Con);
                temp->hide = FALSE;
            }
            stop=1;
            if(interrupt_begin==0)
            {
                interrupt_begin=time(NULL);
            }

            break;
        case 'Q':
        case 'q':
            QuitMenu(x,y);
            break;
        case KEY_END:
            endwin();
            exit(0);
    }
```

```c
}
/*These choices are used in the quit menu*/
char *quitMenu[]={
        "No",
        "Yes",
        "Replay",
};
/*
 * Quit Menu
 * The following operations mainly implement the following requirements:
 * 1. Create a window that could cover the previous one.
 * 2. However, after clicking "No", the current window disappears and previous windows continue work.
 * 3. If I choose "Yes", end all windows and return to terminal.
 * 4. Otherwise, I could replay the game.
 */
void QuitMenu(PANEL *Pau,int x,int y)
{
    ITEM **my_items;
    MENU *my_menu;
    WINDOW *my_menu_win;
    Panel *temp;
    int n_quitMenu, i;
    /* Curses initialization */
    initscr();
    start_color();
    cbreak();
    noecho();
    keypad(stdscr, TRUE);
    init_pair(1, COLOR_WHITE, COLOR_BLACK);
    init_pair(2, COLOR_RED, COLOR_YELLOW);
    /* Create items and menu*/
    n_quitMenu = ARRAY_SIZE(quitMenu);
    my_items = (ITEM **)calloc(n_quitMenu+1, sizeof(ITEM *));
    for(i = 0; i < n_quitMenu; i++)
        my_items[i] = new_item(quitMenu[i], quitMenu[i]);
    my_items[n_quitMenu] = (ITEM *)NULL;
    my_menu = new_menu(my_items);
    /* Make quit window at center */
    menu_opts_off(my_menu, O_SHOWDESC);
    int lent=(x-quit_length)/2;
    int widt=(y-quit_width)/2;
    my_menu_win = newwin(8, 35, widt, lent);
    keypad(my_menu_win, TRUE);
    /* Set main window and suitable sub window */
    set_menu_win(my_menu, my_menu_win);
    set_menu_sub(my_menu, derwin(my_menu_win,2, 26, 4, 6));
    set_menu_format(my_menu, 1, 3);
    set_menu_mark(my_menu, ">");
    box(my_menu_win, 0, 0);
    wattron(my_menu_win,COLOR_PAIR(2));
    mvwprintw(my_menu_win,1,1 ,"    Are you really want to quit? ");
    wattroff(my_menu_win,COLOR_PAIR(2));
    post_menu(my_menu);
    wrefresh(my_menu_win);
    int check=0;
```

```
    while(check!= 1)
    {
        switch(ch)
        {
            /*Each time after ch=getch(), we need to give ch a default value. Otherwise, it will report error.*/
            case KEY_LEFT:
                menu_driver(my_menu, REQ_PREV_ITEM);
                ch=0;
                break;
            case KEY_RIGHT:
                menu_driver(my_menu, REQ_NEXT_ITEM);
                ch=0;
                break;
            case 10:  /* This is Enter key*/
            {
                if(item_name(current_item(my_menu))=="Yes")
                {
                    ch=0;
                    wclear(my_menu_win);
                    endwin();
                    delwin(subwin(my_menu_win, 2, 26, 4, 6));
                    delwin(my_menu_win);
                    exit(0);
                }
                /*clear quit menu and return to previous one*/
                if(item_name(current_item(my_menu))=="No")
                {
                    wclear(my_menu_win);
                    endwin();
                    delwin(subwin(my_menu_win, 2, 26, 4, 6));
                    delwin(my_menu_win);
                    check=1;
                    ch=0;
                    break;
                }
                /*clear quit menu and into new game*/
                if(item_name(current_item(my_menu))=="Replay")
                {
                    check=1;
                    temp = (Panel *)panel_userptr(Pau);
                    hide_panel(Pau);
                    temp->hide = TRUE;
                    endwin();
                    clear();
                    Initialization();
                    ReGameInterface();
                    ch=0;
                    break;
                }
            }
        }
        wrefresh(my_menu_win);
    }
    unpost_menu(my_menu);
    free_menu(my_menu);
    for(i = 0; i < n_quitMenu; ++i)
```

```
        free_item(my_items[i]);
}
```

2.2.6 Collision Problem

I divide collision problems as 2 parts. The first part is that bullet hits enemies: one bullet could make spider, sea monster and one part of centipede die but mushroom needs 4 bullets. However, some situations shooting centipede could gain higher score but sometimes not. What's more, centipede sometimes could be tricky by gaining new heads, I have listed all the situations about centipede that I could image in figure 9. If master shoot all parts of centipede successfully, then it will go into next level.

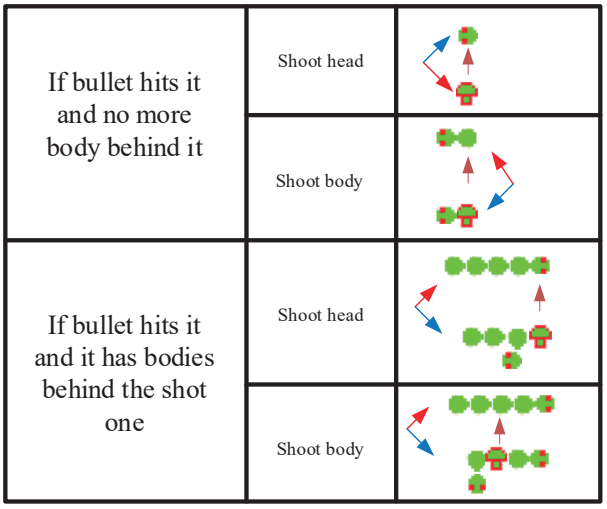
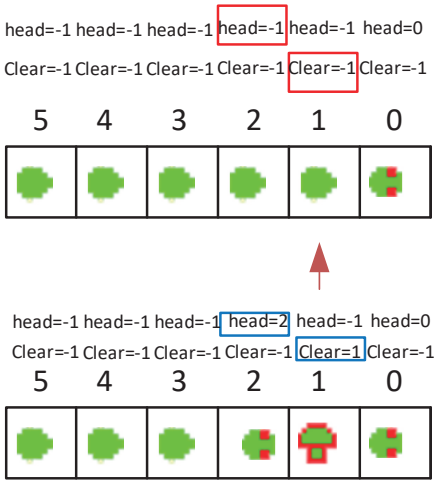


Figure 9: Centipede Issue

The second part is that once enemies hit master (player), master will lose one life until master doesn't have any life then game over. After being hitting, if master still has one more lives, player could play this level again, however, each mushroom has 4 lives, including which has being hit. All enemies return to their initial position waiting for orders.

You might have a question about how I achieve multiple centipede after splitting and make them move in the same rule. Well, I define an int type "Clear" in struct so that I could trace all nodes of the original centipede. I will explain this combined with figure 10. For example, if I shoot the *ith* node of it, I record this node by writing i into Centipede[i].Clear, the original Clear value of its bodies are both -1. After that, we need to traverse all centipedes whose Clear value is -1 and head value is greater than or equal to 0 (which means they have not yet been shot and they are the first node of each centipede). I don't care about the rest bodies of each centipede, if they have not yet been shot, I will make them follow their heads.



Mushroom[new].x=Centipede[1].x;
Mushroom[new].y=Centipede[1].y;

Figure 10: Centipede Issue 2

Collision Problem Source:

```
void CollisionCheck(WINDOW *win,WINDOW *win0,int boundary,PANEL *Pau)
{
    /*The bullet hits the mushroom, Mushrooms can be destroyed and
        disappear only when taking 4 shoots*/
    for(int i=0;i<mushLength;i++) {
        if (bullet_x == mushroom[i].x && bullet_y == mushroom[i].y)
        {
            mushroom[i].mush_record -=1;
            score += 0;//do nothing
            mvwprintw(win, 1, 1, "Score: %d", score);
            if (mushroom[i].mush_record == 0) {
                i++;
                while (i < mushLength) {
                    mushroom[i - 1].x = mushroom[i].x;
                    mushroom[i - 1].y = mushroom[i].y;
                    mushroom[i-1].mush_record = mushroom[i].mush_record
                        ;
                    i++;
                }
                mushLength -= 1;
                score += 1;//If mushroom is destroyed, take 1 point
                mvwprintw(win, 1, 1, "Score: %d", score);
            }
            Fire = 0;
        }
    }
    /*When spider hits mushroom, mushroom will be eaten and disappear*/
    for(int i=0;i<mushLength;i++) {
        if (Spider[1].x== mushroom[i].x && Spider[1].y== mushroom[i].y)
        {
            mushroom[i].mush_record =0;
            i++;
            while (i < mushLength)
            {
                mushroom[i - 1].x = mushroom[i].x;
                mushroom[i - 1].y = mushroom[i].y;
```

```
                mushroom[i-1].mush_record = mushroom[i].mush_record;
                i++;
            }
            mushLength -= 1;
        }
    }
    /*Once spider, sea monster hits master, master will die*/
    for(int i=0;i<3;i++)
    {
        if((Spider[i].x==master_1_x && Spider[i].y==master_1_y)||(Spider[i].x==master_1_x+1 && Spider[i].y==master_1_y)||(Spider[i].x==master_1_x+2 && Spider[i].y==master_1_y)||
           (Spider[i].x==master_2_x && Spider[i].y==master_2_y)||(Spider[i].x==master_2_x+2 && Spider[i].y==master_2_y)||
           (Spider[i].x==master_3_x && Spider[i].y==master_3_y)||(Spider[i].x==master_3_x+1 && Spider[i].y==master_3_y)||(Spider[i].x==master_3_x+2 && Spider[i].y==master_3_y))
        {
            end(win0,boundary,Pau);
        }
    }
    for(int i=0;i<2;i++)
    {
        if((Sea_Monster[i].x==master_1_x && Sea_Monster[i].y==master_1_y)||(Sea_Monster[i].x==master_1_x+1 && Sea_Monster[i].y==master_1_y)||(Sea_Monster[i].x==master_1_x+2 && Sea_Monster[i].y==master_1_y)||
           (Sea_Monster[i].x==master_2_x && Sea_Monster[i].y==master_2_y)||(Sea_Monster[i].x==master_2_x+2 && Sea_Monster[i].y==master_2_y)||
           (Sea_Monster[i].x==master_3_x && Sea_Monster[i].y==master_3_y)||(Sea_Monster[i].x==master_3_x+1 && Sea_Monster[i].y==master_3_y)||(Sea_Monster[i].x==master_3_x+2 && Sea_Monster[i].y==master_3_y))
        {
            end(win0,boundary,Pau);
        }
    }
    /*Once bullet hits sea monster or spider, they will be destroyed and disappear*/
    for(int i=0;i<2;i++) {
        if (bullet_x == Sea_Monster[i].x && bullet_y == Sea_Monster[i].y)
        {
            score += 600;//This shoot take 600 points bonus
            mvwprintw(win, 1, 1, "Score: %d", score);
            Sea_Monster[0].x=-1;
            Sea_Monster[1].x=-1;
            sea_appear=0;
            Fire = 0;
        }
    }

    for(int i=0;i<3;i++) {
        if (bullet_x == Spider[i].x && bullet_y == Spider[i].y)
        {
            score += 600;//This shoot take 600 points bonus
```

```
            mvwprintw(win, 1, 1, "Score: %d", score);
            Spider[0].x=-1;
            Spider[1].x=-1;
            Spider[2].x=-1;
            spider_appear=0;
            Fire = 0;
        }
    }
    /*Once bullet hits centipede, computer need to discuss the situations separately*/
    for(int i=0;i<Length;i++)
    {
        if(bullet_x==Centipede[i].x && bullet_y==Centipede[i].y)
        {
            /*If the component of centipede has not yet been hit*/
            if(Centipede[i].Clear<0)
            {
                /*All these situations have common result: the shot segment becomes a mushroom*/
                mushLength += 1;
                mushroom[mushLength - 1].x = Centipede[i].x;
                mushroom[mushLength - 1].y = Centipede[i].y;
                mushroom[mushLength - 1].mush_record=4;
                Centipede[i].Clear=i;
                /*If bullet hits its body and no more body behind it*/
                if(Centipede[i+1].Clear>=0&&i+1<Length)
                {
                    /*If bullet hits its head*/
                    if (i == Centipede[i].head)
                    {
                        score += 100;//This shoot take 100 points
                        mvwprintw(win, 1, 1, "Score: %d",score);
                    }
                    else
                    {
                        score += 10;//This shoot take only 10 points
                        mvwprintw(win, 1, 1, "Score: %d",score);
                    }
                    Fire=0;
                    break;
                }
                /*If bullet hits its body and it has bodies behind the shoot one*/
                if(Centipede[i+1].Clear<0&&i+1<Length)
                {
                    //the centipede splits into two, gaining a new head
                    Centipede[i+1].head = i + 1;
                    /*If bullet hits its head*/
                    if (i == Centipede[i].head)
                    {
                        score += 100;//This shoot take 100 points
                        mvwprintw(win, 1, 1, "Score: %d",score);
                    }
                    else
                    {
                        score += 10;//This shoot take only 10 points
                        mvwprintw(win, 1, 1, "Score: %d",score);
```

```
                    }
                    Fire=0;
                    break;
                }
            }
        }
        /*Once centipede hits master, computer need to discuss the situations separately*/
        if(Centipede[i].Clear<0)
        {
            if((Centipede[i].x==master_1_x && Centipede[i].y==master_1_y)||(Centipede[i].x==master_1_x+1 && Centipede[i].y==master_1_y)||(Centipede[i].x==master_1_x+2 && Centipede[i].y==master_1_y)||
               (Centipede[i].x==master_2_x && Centipede[i].y==master_2_y)||(Centipede[i].x==master_2_x+2 && Centipede[i].y==master_2_y)||
               (Centipede[i].x==master_3_x && Centipede[i].y==master_3_y)||(Centipede[i].x==master_3_x+1 && Centipede[i].y==master_3_y)||(Centipede[i].x==master_3_x+2 && Centipede[i].y==master_3_y))
            {
                end(win0,boundary,Pau);
                break;

            }
        }
    }
}
void end(WINDOW* win,int boundary,PANEL *Pau) {
    int x;
    int y;
    int start_x;
    int start_y;
    int start1_x;
    int start2_x;
    life--;
    if(life>0)
    {
        getmaxyx(win, y, x);
        start_x=(x-fake_lost)/2;
        start1_x=(x-fake_lost_w)/2;
        start2_x=(x-chance_left)/2;
        start_y=(y-3)/2;
        int Times=3;
        interrupt_begin=time(NULL);
        while(Times>0)
        {
            wclear(win);
            mvwprintw(win,start_y, start_x,"You lost!");
            mvwprintw(win,start_y+1, start1_x,"%d second later you will try again!",Times);
            mvwprintw(win,start_y+2, start2_x,"Chance left: %d",life);
            wrefresh(win);
            usleep(SECOND);
            Times--;
        }
```

```
        interrupt_end=time(NULL);
        wclear(win);
        Fire=0;
        /*Reset all the roles, including the lives of mushroom*/
        MasterProduce(win);
        Reset_Sea();
        Reset_Spider();
        CentipedeProduce(boundary);
        Reset_Mushroom(win);
    }
    else
    {
        getmaxyx(win, y, x);
        start_x=(x-lost)/2;
        start_y=(y-3)/2;
        wclear(win);
        mvwprintw(win,start_y, start_x,"Game Over! you lost!");
        mvwprintw(win,start_y+1, start_x,"Your score: %d",score);
        mvwprintw(win,start_y+2, start_x,"Playtime: %d m %d s",min,sec);
        wrefresh(win);
        while(ch!='Q'||ch!='q')
        {
            switch(ch)
            {
                case 'Q':
                case 'q':
                    QuitMenu(Pau,x,y);
                    ch=0;
                    break;
            }
            mvwprintw(win,start_y, start_x,"Game Over! you lost!");
            mvwprintw(win,start_y+1, start_x,"Your score: %d",score);
            mvwprintw(win,start_y+2, start_x,"Playtime: %d m %d s",min,sec);
            wrefresh(win);
        }
    }
}
int success(WINDOW* win,int boundary,PANEL *Pau)
{
    int count=0;
    int x;
    int y;
    int start_x;
    int start_y;
    int start1_x;

    for(int i=0;i<Length;i++)
    {
        if(Centipede[i].Clear>=0)
        {
            count++;
        }
    }
    if(count==Length)
    {
```

```
        Level+=1;
        /*I don't want players spend much time on this game, it's enough if they have won 5 times.*/
        if(Level==6)
        {
            getmaxyx(win, y, x);
            start_x=(x-WIN)/2;
            start_y=(y-2)/2;
            wclear(win);
            mvwprintw(win,start_y, start_x,"You win!");
            mvwprintw(win,start_y+1, start_x,"Your score: %d",score);
            mvwprintw(win,start_y+2, start_x,"Playtime: %d m %d s",min, sec);
            wrefresh(win);
            while(ch!='Q'||ch!='q')
            {
                switch(ch)
                {
                    case 'Q':
                    case 'q':
                        QuitMenu(Pau,x,y);
                        break;
                }
    mvwprintw(win, start_y, start_x, "You win!");
    mvwprintw(win, start_y + 1, start_x, "Your score: %d", score);
    mvwprintw(win, start_y + 2, start_x, "Playtime: %d m %d s", min, sec);
    wrefresh(win);
            }
        }
        else
        {
            getmaxyx(win, y, x);
            start_x=(x-Congrat)/2;
            start1_x=(x-Congrat_w)/2;
            start_y=(y-2)/2;
            int Times=3;
            interrupt_begin=time(NULL);
            while(Times>0)
            {
                wclear(win);
                mvwprintw(win,start_y, start_x,"Congratulations!");
                mvwprintw(win,start_y+1, start1_x,"%d second later you will go to level %d",Times,Level);
                wrefresh(win);
                usleep(SECOND);
                Times--;
            }
            interrupt_end=time(NULL);
            /*
             * If play wins, change color of roles and increase the length of centipede
             * Reset the position of roles
             */
            wclear(win);
            Fire=0;
            changeColor();
```

```
            MasterProduce(win);
            Reset_Sea();
            Reset_Spider();
            CentipedeProduce(boundary);
            wrefresh(win);
        }
    }
    /*If only one segment that has not yet been hit, increase the speed of that one*/
    else if(count==Length-1)
    {
        usleep(SHORT_DELAY);
    }
    else
    {
        usleep(DELAY);
    }
}
```

2.2.7 Canvas Refreshment

I debugged many times until I found the ideal steps of canvas refreshment (see figure 11). I fix out the problem which could make spider walk slow by adding new int type variable "delay_spider", like switch I could turn on or turn off. I do this process when repainting canvas. There is nothing else worth saying, the process is clear in my flow chart.

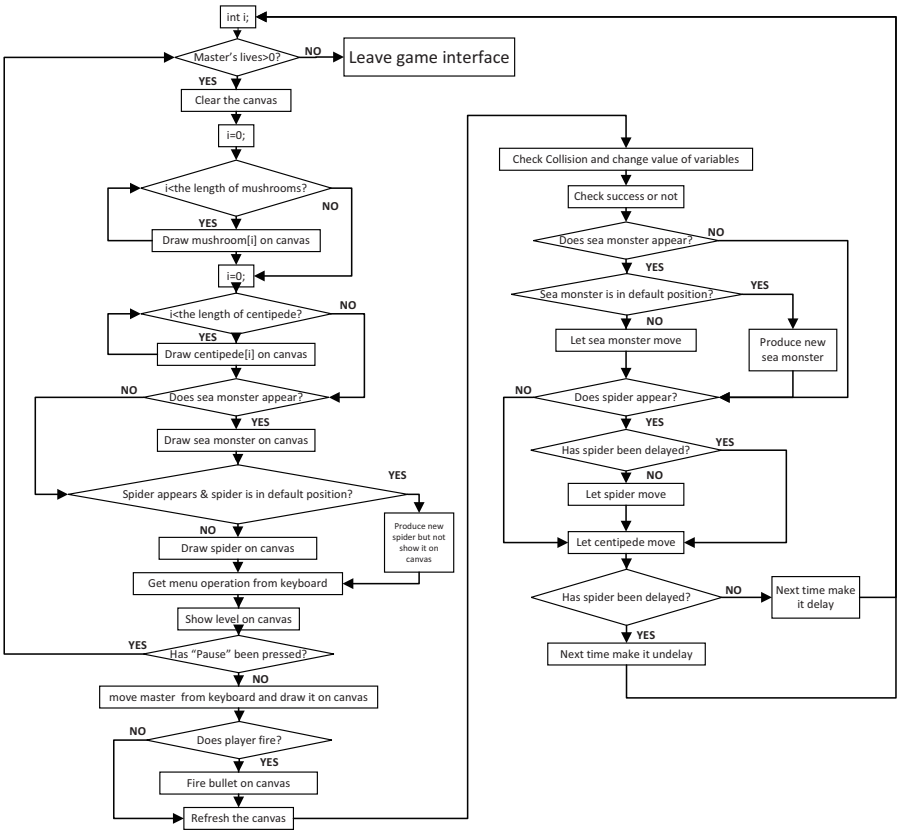


Figure 11: Canvas Refreshment

Game Interface Source:

```
/*This function is used to create game interface*/
void GameInterface()
{
    WINDOW *my_wins[8];
    PANEL  *my_panels[6];
    Panel panel_datas[6];
    int i=0,max_y=0, max_x=0,win_6_x,win_6_y;
    score=0;
    min=0;
    sec=0;
    interrupt=0;
    interrupt_begin=0;
    interrupt_end=0;
    begin=0;
    over=0;
    /* Curses initialization */
    initscr();
    noecho();//don't display input character
    start_color();
    init_pair(8, COLOR_MAGENTA, COLOR_BLACK);
    init_pair(7, COLOR_CYAN, COLOR_BLACK);
    init_pair(6, COLOR_BLUE, COLOR_BLACK);
    init_pair(5, COLOR_YELLOW, COLOR_BLACK);
    init_pair(4, COLOR_GREEN, COLOR_BLACK);
    init_pair(3, COLOR_RED, COLOR_BLACK);
    curs_set(FALSE);

    int listener;
    listener = pthread_create(& work, NULL, waitForKey, NULL);//Create thread for master
    if (listener != 0) {
        exit(1);
    }
    /* The following operations mainly aim to implement the following requirements:
     * 1. Create 7 windows and ball window will cover the big window.
     * 2. Besides, other windows are used for showing score, pause, continue and quit.
     * 3. Except for big windows, each window need a panel.
     * 4. Only the panel of continue need to hide at beginning.
     */
    getmaxyx(stdscr, max_y, max_x);
    /* Create windows for the panels */
    my_wins[0] = newwin(max_y-score_line, max_x, 0, 0);//big window
    my_wins[1] = newwin(score_line, max_x-quit-level-Pause-Continue-Time, max_y-score_line, 0);//score
    mvwprintw(my_wins[1], 1, 1, "Score: %d", score);
    my_wins[7]= newwin(score_line,Time,max_y-score_line,max_x-quit-level-Pause-Continue-Time);//time
    mvwprintw(my_wins[7], 1, 1, "Time: ");
    my_wins[2] = newwin(score_line, level, max_y-score_line, max_x-quit-level-Pause-Continue);//level
    my_wins[3] = newwin(score_line, Pause, max_y-score_line,max_x-quit-Pause-Continue);//Pause
    mvwprintw(my_wins[3], 1, 1, "\"P\"ause");//Pause label
```

```
    my_wins[4] = newwin(score_line, Continue, max_y-score_line, max_x-quit-Continue);//Replay
    mvwprintw(my_wins[4], 1, 1, "\"C\"ontinue"); //Continue label
    my_wins[5] = newwin(score_line, quit, max_y-score_line, max_x-quit);//quit
    mvwprintw(my_wins[5], 1, 1, "\"Q\"uit");//quit label
    my_wins[6] = newwin(max_y-score_line-2, max_x-2, 1, 1);// ball
    getmaxyx(my_wins[6], win_6_y, win_6_x);
    keypad(stdscr, TRUE);
    for(i = 0; i < 6; i++)
    {
        box(my_wins[i], 0, 0);
    }
    for(i=0;i<6;i++)
    {
        my_panels[i] = new_panel(my_wins[i]);
    }
    my_panels[6] = new_panel(my_wins[7]);
    for(i=0;i<6;i++)
    {
        panel_datas[i].hide = FALSE;
        set_panel_userptr(my_panels[i], &panel_datas[i]);
    }
    panel_datas[6].hide = FALSE;
    set_panel_userptr(my_panels[6], &panel_datas[7]);
    panel_datas[4].hide = TRUE;
    hide_panel(my_panels[4]);
    /* Show it on the screen */
    doupdate();
    /*Produce mushrooms, a master and a centipede at first*/
    MushroomProduce(win_6_y,win_6_x-1);
    MasterProduce(my_wins[6]);
    CentipedeProduce(win_6_x);
    /*If player do not lose all lives, the game will continue. Else, quit*/
    begin=time(NULL);
    while(life>0)
    {
        wclear(my_wins[6]);
        for(i=0;i<mushLength;i++)
        {
            wattron(my_wins[6],COLOR_PAIR(mushroom_color));
            mvwprintw(my_wins[6],mushroom[i].y,mushroom[i].x,"&");
            wattroff(my_wins[6],COLOR_PAIR(mushroom_color));
        }
        wattron(my_wins[6],COLOR_PAIR(centipede_color));
        for(i=0;i<Length;i++)
        {
            if(Centipede[i].Clear<0)
            {

                if(i==Centipede[i].head)
                {
                    mvwprintw(my_wins[6],Centipede[i].y,Centipede[i].x,"O");
                }
                else
```

```
                {
                    mvwprintw(my_wins[6],Centipede[i].y,Centipede[i].x, "*");
                }

            }
        }
        wattroff(my_wins[6],COLOR_PAIR(centipede_color));
        if(sea_appear)
        {
            wattron(my_wins[6],COLOR_PAIR(sea_color));
            mvwprintw(my_wins[6],Sea_Monster[0].y, Sea_Monster[0].x, "$");
            mvwprintw(my_wins[6],Sea_Monster[1].y, Sea_Monster[1].x, "_");
            wattroff(my_wins[6],COLOR_PAIR(sea_color));
        }
        if(Spider[1].x!=-1&spider_appear)
        {
                wattron(my_wins[6],COLOR_PAIR(spider_color));
                mvwprintw(my_wins[6],Spider[0].y, Spider[0].x, "^");
                mvwprintw(my_wins[6],Spider[1].y, Spider[1].x, "W");
                mvwprintw(my_wins[6],Spider[2].y, Spider[2].x, "^");
                wattroff(my_wins[6],COLOR_PAIR(spider_color));
        }
        else
        {
                SpiderProduce(win_6_y,win_6_x);
        }
        mvwprintw(my_wins[2], 1, 1, "Level: %d",Level);
        getInt(my_panels[4],my_panels[3],win_6_x,win_6_y,my_wins[6]);
        if(stop<1)
        {
            wattron(my_wins[6],COLOR_PAIR(master_color));
            getOrder(my_wins[6],win_6_x,win_6_y);
            wattroff(my_wins[6],COLOR_PAIR(master_color));
            wattron(my_wins[6],COLOR_PAIR(bullet_color));
            if(Fire)
            {
                bullet_y-=1;
                mvwprintw(my_wins[6],bullet_y,bullet_x,"|");
            }
            else
            {
                bullet_y=master_1_y;
                bullet_x=master_1_x+1;
            }
            wattroff(my_wins[6],COLOR_PAIR(bullet_color));
            wrefresh(my_wins[2]);
            wrefresh(my_wins[1]);
            wrefresh(my_wins[6]);
            CollisionCheck(my_wins[1],my_wins[6],win_6_x,my_panels[3]);
            success(my_wins[6],win_6_x,my_panels[3]);
            if(sea_appear)
            {
                if(Sea_Monster[0].x!=-1)
                {
```

```
                    sea_monsterMove(win_6_x);
                }
                else
                {
                    Sea_MonsterProduce(win_6_y,win_6_x);

                }
            }
            if(spider_appear)
            {
                if(delay_spider==1)
                {
                    SpiderMove(win_6_x,win_6_y);
                }
            }
            CentipedeMove(win_6_x-1,win_6_y-1);
            delay_spider*=-1;
            //timer
            over = time(NULL);
            double seconds=difftime(over,begin);
            if(difftime(interrupt_end,interrupt_begin)>0)
            {
                interrupt+=difftime(interrupt_end,interrupt_begin);
                interrupt_begin=0;
                interrupt_end=0;
            }
            seconds-=interrupt;
            if(seconds>=60)
            {
                min=((int)seconds)/60;
                sec=((int)seconds)-min*60;
            }
            else
            {
                sec=(int)seconds;
            }
            wclear(my_wins[7]);
            box(my_wins[7], 0, 0);
            mvwprintw(my_wins[7], 1, 1, "Time: %d m %d s",min,sec);
            wrefresh(my_wins[7]);
        }
        update_panels();
        doupdate();
    };
    endwin();
}
```