THE ULTIMATE SHOOTING GAME

Name | Object Oriented Analysis and Design | 22/11/17

TABLE OF CONTENTS

System Requirement

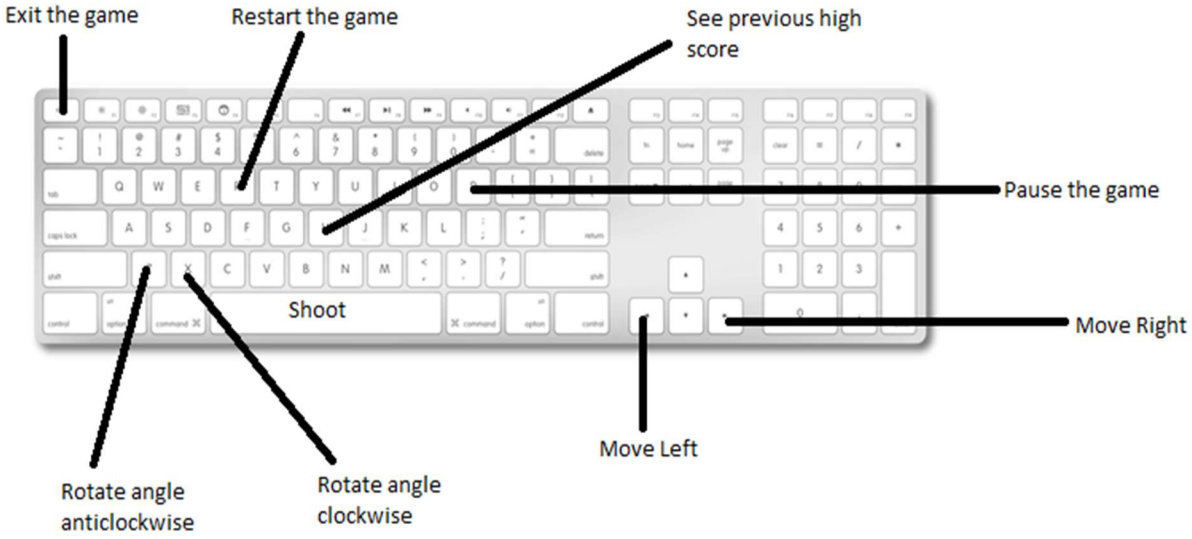
- To Compile the project you need include graphics.h and and conio.h (if compiling with g++ compiler).

How to start the game

- **Windows Users :** Simply double click the .exe file.

- **Linux Users :** You need to install windows campatiblity in linux i.e **wine** and then Start the .exe file by writing "wine Ultimate_shooting_game.exe" in the directory of the game (in terminal). To compile you have to write g++ ultimate_shooting_game.cpp a -lgraph. and then /a

CONTROLS

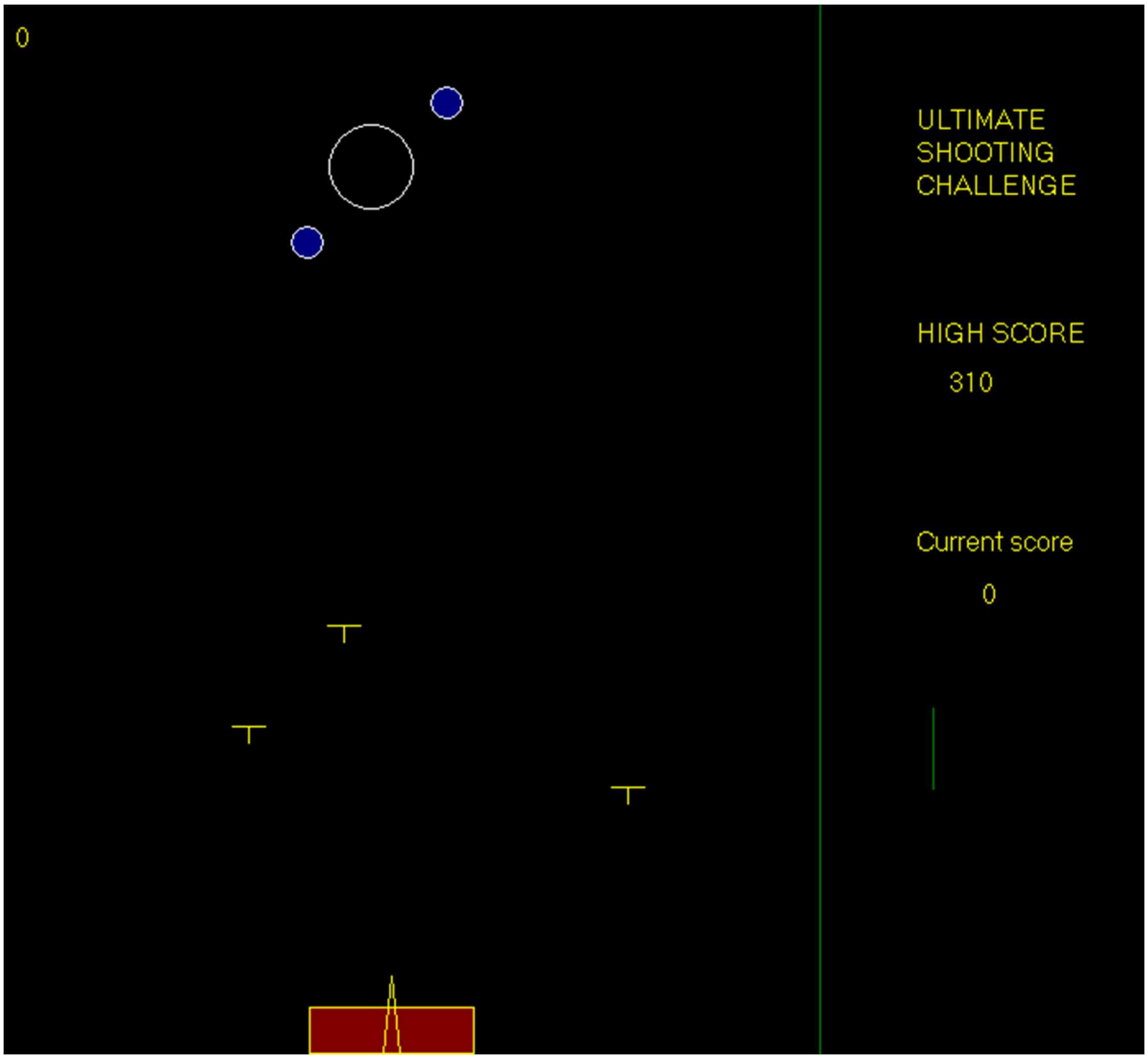


<- (left arrow key)	Move left
-> (right arrow key)	Move right
¤ Z	Rotate angle anti-clock wise
X	Rotate angle clock wise
Space	Shoots bullets
Esc	Exit the game
P	Pause the game
R	Restart's the game

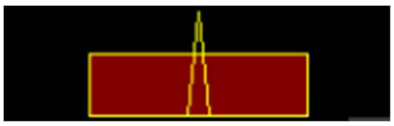
STARING THE GAME

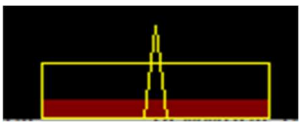
- First screen will be this you have to press space to go to next screen

- Next screen will contain the instructions of the game.
- After that you have to enter your name so that if you break any previous high score we will save your high score in our database.
- After this screen your game will be started.



- Health Bar & the Shooter tank

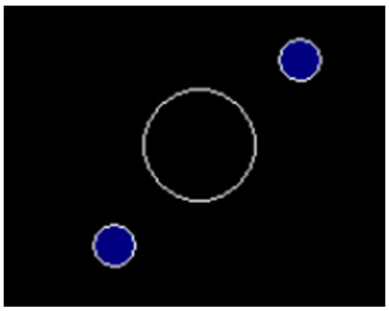


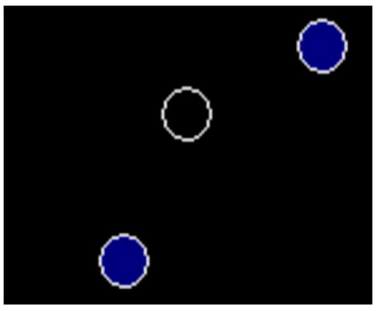


1. This Tank is actually you, you have to move this and protect this.
2. The triangle represents your gun you can rotate it as you like you have to shoot at the circles genrating randomly on the screen using this gun.
3. The red bar represets your health
4. Initially you have 7 health means you can take upto 7 damage from bullets.
5. After getting certain amount of score level will be changed and you health bar will get full and then you can get upto 14 damage from the bullets.
6. As the damage increases your red bar will Starts decreasing.

RANDOMLY GENERATING CIRCLES

- This object will be generated on your screen randomly anywhere you have to shoot the inner bigger circle before it disappears
- Radius of the inner circle will be decreasing at some rate after which it will disappear and your life will decrease.





- Those 2 blue circles will be rotating in opposite direction to each other around the inner circle and these two will be an obstacle for you to shoot the inner circle.

PLAYER'S LIFE

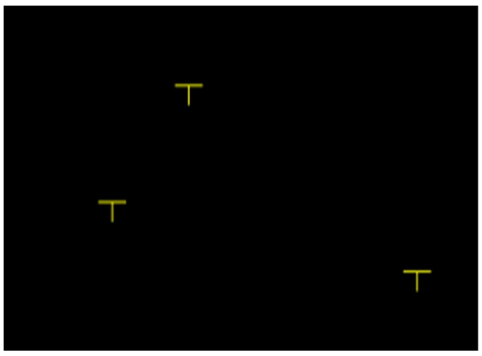
- Initially player will have 3 lifes as you as game goes if you are unable to shoot the target circle your life will decrease.
- If you miss more than 3 circles you will not have any life lift so you will lose.

LEVEL CHANGE

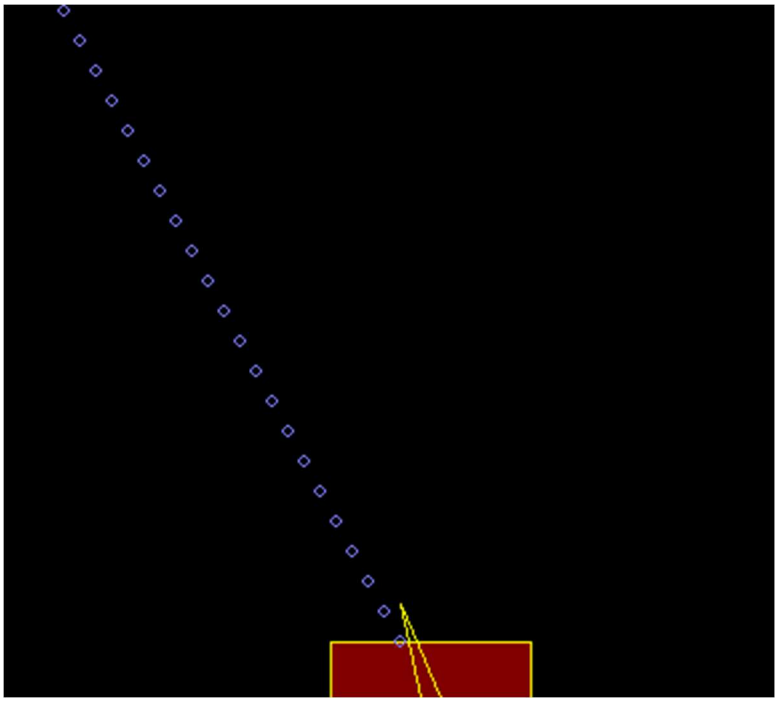
- As you break certain amount of score the level will get change speed of bullets falling from about will increase and color of the bullets will change from yellow to red and your health will increase from 7 to 14.

FALLING BULLETS

- These Bullets are the real enemy of you in this in this game.
- If you got hit by them them your health will decrease.
- As the level change speed of their falling will increase
- At a time, maximum 4 bullets can fall.



THE GUN



- The weapon you have in this game is this gun. It shoots circle in range from 0 to 180 degree.
- You have to shoot the randomly genrating target(circles) with this gun.

PAUSE:

To pause the game at any movement press ‘p’ and to resume press ‘p’ again.

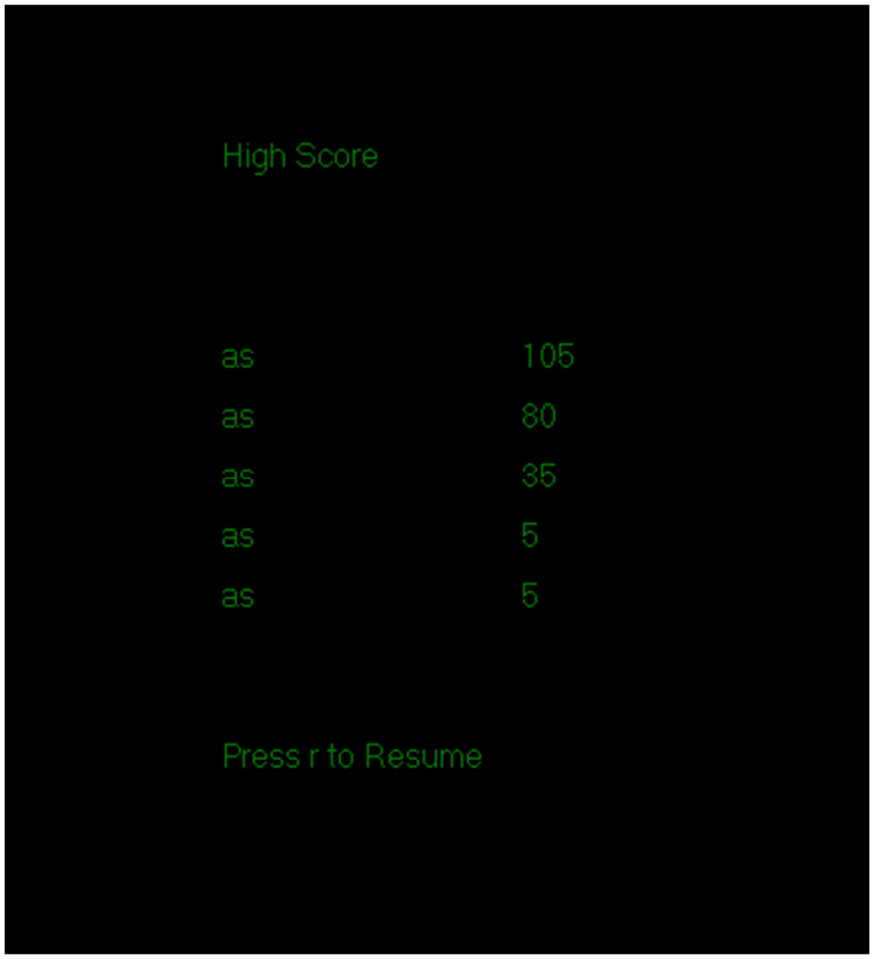
GAME OVER

- To restart the game press ‘r’ and to see high score press h and to exit press esc.

HIGH SCORE

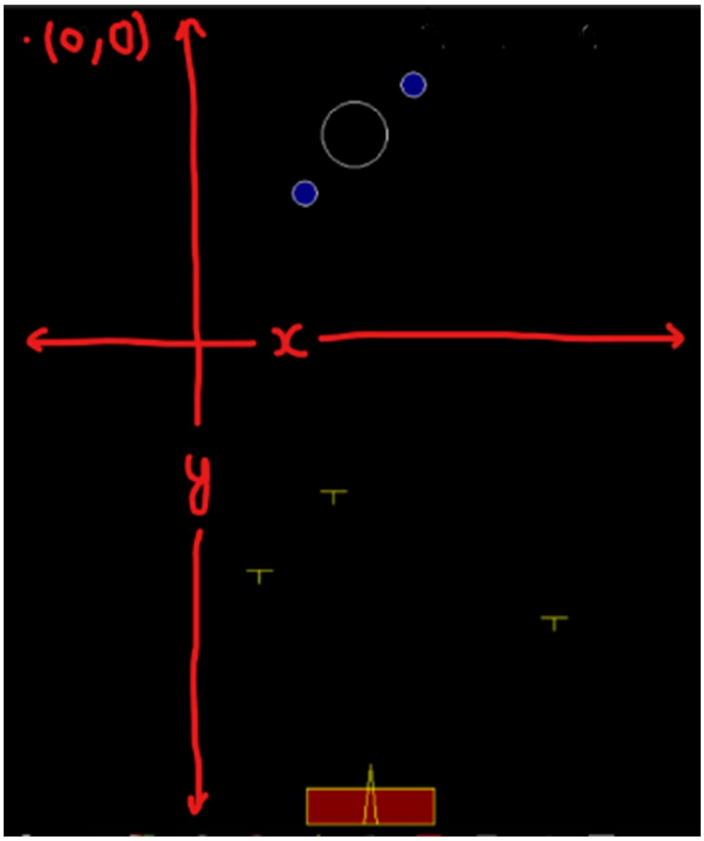
- To see your previous high score you can either press h at the starting screen or you can pause the game and then see your high score or you can see it when your game is over by pressing h.

HIGH SCORE SCREEN WILL LOOK LIKE THIS..



How the game works?...

- At first there exists a base class that is x_y_coordinates, which contains and maintains all the x and y coordinates of the game.
- Origin of the screen is at top left point of the screen.



```
class x_y_coordinates{
public:
    x_y_coordinates(int ii=0,int jj=0);
    void draw(int o,int i=14,int j=10);
    void set_coordinate(int ii,int jj);
    int get_x_coordinate()const;
    int get_y_coordinate()const;
    ~x_y_coordinates(){}
protected:
    int x,y;
};
```

BASIC FUNCTIONS WHICH WE HAVE USED FROM GRAPHCS.H

1. outtextxy(x_coordinate,y_coordinate,char* textstring): It silmpy prints the given string to to given coordinates.
2. settextstyle(int font,int direction,int charsize): It decides to font of the string which we like to print.

Name	Value	Description
DEFAULT_FONT	0	8x8 bit-mapped font
TRIPLEX_FONT	1	Stroked triplex font
SMALL_FONT	2	Stroked small font
SANS_SERIF_FONT	3	Stroked sans-serif font
GOTHIC_FONT	4	Stroked gothic font
SCRIPT_FONT	5	Stroked script font
SIMPLEX_FONT	6	Stroked triplex script font
TRIPLEX_SCR_FONT	7	Stroked triplex script font
COMPLEX_FONT	8	Stroked complex font
EUROPEAN_FONT	9	Stroked European font
BOLD_FONT	10	Stroked bold font

3. setcolor(color name) : It decide the color of the whatever printing after this function line.

4. circle(x_center_coordinate,y_center_coordinate,radius): make circle at at provided center.
5. line(x1,y1,x2,y2): make line from (x1,y1) to (x2,y2).
6. delay(time(ms)): stop the code execution for that much milli seconds.
7. getch(): saves char in its stream.
8. cleardevice(): clear the whole screen .
9. initwindow(): creates a window on which we work in graphics.

WHY WE HAVE USED CONIO.H?

we include conio.h just to use only one of its basic function and that is kbhit() and its work is to return true if any key is pressed.

```
if(kbhit()){
    key_stroke = getch();
    if(key_stroke=='r')break;
    if(key_stroke==27)goto end;
    check(key_stroke,machine,target,rotating_circle);
    if(key_stroke=='p'){
        setcolor(GREEN);
        settextstyle(SANS_SERIF_FONT ,HORIZ_DIR,3);
        print_at_x_y(250,300,"To resume press p");
        char c=getch();
        if(c=='p'){continue;}
    }
}else{
    target--;
    delay(50);
    cleardevice();
 }flag=1;
```

HOW THE SHOOTER TANK IS MOVING HORIZONTALLY AND SHOOTING BULLETS AT DIFFERENT ANGELS

```
class shooter:public x_y_coordinates{
public:
    shooter(int,int,int,char,char,char,char,char);
    void draw();
//    void set_coordinate(int,int);

    void set_keys(char,char);
    void set_s_keys(char,char,char);

    char get_left_key()const;
    char get_right_key()const;

    char get_s_left_key()const;
    char get_s_right_key()const;
    char get_s_shoot_key()const;
    int get_h_coordinate()const;

//to move shooter machine use       : postfix operato
//to change angel of shooter use    : prefix operator
    void operator--(int);
    void operator++(int);
    void operator--();
    void operator++();
private:
    int h;
    char left_key,right_key;
    char s_left_key,s_right_key;
    char shoot_key;

};
```

- Here shooter tank is derived class of x_y_coordinate as all the x,y coordinates are maintained by x_y_coordinate class.
- set_s_keys and set_keys set the gun and tank movement keys.
- get.. returns left ,right keys of the tank.
- get_s.. returns left and right of gun.
- get_h_key return x coordinate of top of the gun.
- Here prefix overloading is for angle movement.
- Here postfix overloading is for shooter tank moving horizontally.

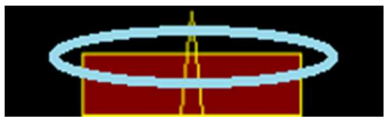
HOW BULLETS ARE FALLING?

```
class Bullet_fall{
public:
    Bullet_fall(int);   //constructor
    void fall(const shooter&);
    friend bool fall_hit_shooter(const x_y_coordinates&,const shooter&);
    void draw();
private:
    short number;   //number of objects are falls
    short speed;
    x_y_coordinates *b;
    int o;
};
```

- here fall() is responsible for bullets to fall and for checking simultaneously that the tank is hitted by the bullet or not by calling the friend function fall_hit_shooter.
- draw() make bullets.
- We can also control the speed of the bullets falling.

HOW WE ARE CHECKING DOES THE BULLETS HIT THE TANK OR NOT?

We simply check whether x coordinate of bullet is in range of length of x coordinate of the tank and greater than y coordinate of the top of tank.



HOW CIRCLES ARE GENERATING AT RANDOM PLACE AND HOW GUN IS SHOOTING AND THOSE ROTATING CIRCLES?

```
class circles:public x_y_coordinates{                    //for
public:
    circles(int ii=50,int jj=50,int kk=50,int decay_rate=0);
    void draw(int i);                                      //crea
    void setdecayrate(int);                              //change t
    int get_radius()const;                               //return r
    void set_radius(int);                                //change t
    void operator--(int);                                //to reduc
    void set_circle(int,int,int);
    ~circles();                                          //destruct
    void draw_rotator(int);
private:
    int r;                                               //centre c
    int decayrate;                                       //rate at
};
```

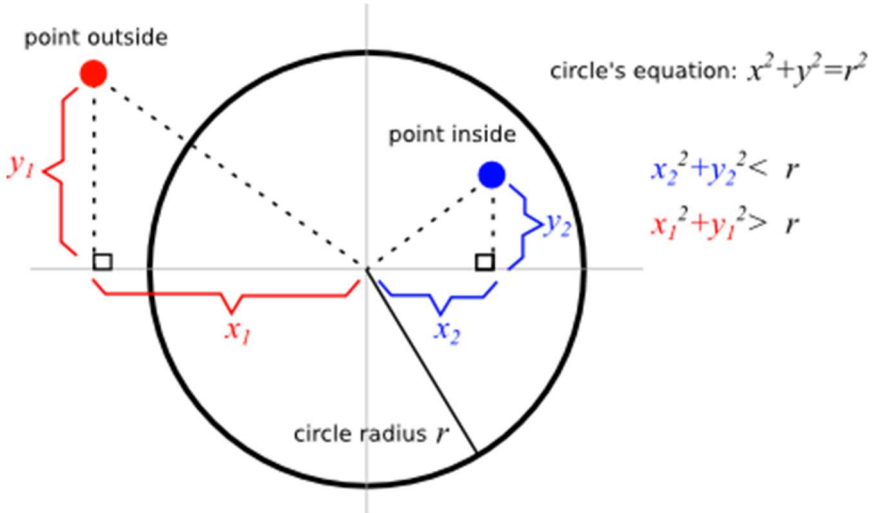
- As discussed above all the x,y coordinates are maintained by x_y_coordinates class that's why circles is also derived class of this base class.
- Here we can set decay rate of the circle using setdecayrate() i.e how fast circle radius will decrease.
- get_radius returns current radius of the circle.
- set_radius sets radius to a circle.
- Postfix – operater is overloaded to reduce the radius of the circle.
- Set_circle sets coordinates to circle i.e random coordinates
- ~circles() is distructor of the circle it makes radius of the circle to 0
- draw_rotator() draws the rotating circles, takes int l in argument to continuously rotate and increase the angle of rotation.
- draw draws the circles and gun also use this same function to shoot the circles.

```
void circles::draw(int i){
    setcolor(i);
    circle(x,y,r);
}
void circles::draw_rotator(int l){
    setcolor(WHITE);
    setfillstyle(SOLID_FILL,BLUE);
    circle(x+(60*cos(l*0.05)),y+(60*sin(l*0.05)),10);
    floodfill(x+(60*cos(l*0.05)),y+(60*sin(l*0.05)),WHITE);
    setfillstyle(SOLID_FILL,BLUE);
    circle(x+(60*sin(l*0.05)),y+(60*cos(l*0.05)),10);
    floodfill(x+(60*sin(l*0.05)),y+(60*cos(l*0.05)),WHITE);

}
```

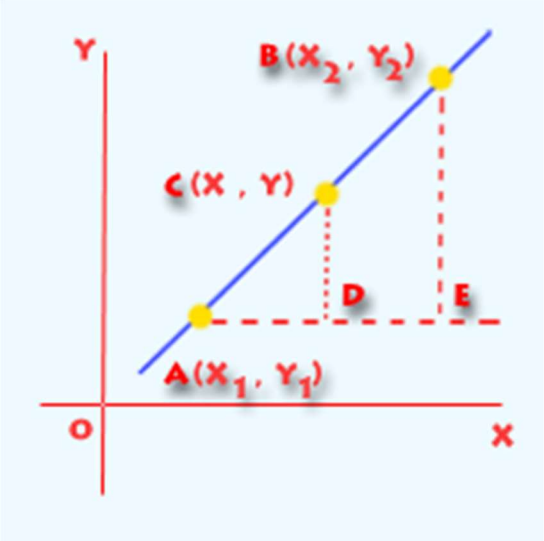
HOW WE ARE CHECKING GUN HAS SHOT THE TARGET CIRCLE OR NOT?

- Shooter is actually shooting circle whose center coordinates are increasing in straight line.
- So if distance from centre of these shoots circles to the target circle is less than that radius of the target circle than the shooted circle lies inside that of target circle so in other way target has been shoot so in that case destructor of the target circle has been called and score will increase we will see later how score is changing.



- Same concept has been used when the gun shoots the rotating circle but in this case destructor of the shooting circle has been called because these rotating circles made to be solid shooting circle can't pass these rotating circles.

HOW WE ARE SHOOTING AT DIFFERENT ANGLES?



In our case y2=y1 and we are only changing x2 with help of postfix ++ and – overloading in shooter class we have discussed earliear.

So by using 2 point form we get a direction for shooting

2 point form:

(y-y1)=((y2-y1)/(x2-x1))(x-x1)

HOW LEVEL OF THE GAME IS CHANGING?

- Run time polymorphism has been used to do this.
- Here we have made 2 classes to this.

```
class player{
   | int number_laser;
    protected:
    int velocity;
    public:
    enum player_type{B,P};
    player_type type;
    player(int laser):number_laser(laser),velocity(5){type=B;}
    player* promote();
    virtual int increase_speed(){return velocity;}
    virtual int health_bar(){return 4;}

};
class pro_player:public player{
    int accerleration;
    public:
    pro_player():player(2){type=P;accerleration=7;};
    int increase_speed(){return accerleration+velocity;}
    int health_bar(){return 2;}
    void drop_spl_obj();
};
```

- At the beginning of the game we have pointer p to the player class or we call that player initially a beginner.
- So initially all the function like speed, health bar are used of that of player class as pointer p is pointing to player classs.
- As player breaks certain amount of score pointer p will start pointing to an object of pro_player class or player has become a pro-player.
- So now pointer p will start using functions of pro_player class so speed of the bullets wil increase and health bar will also change according to what we want to set.

HOW SCORE, HEALTH AND LIFE HAS BEEN MAINTAINED?

```
class Game_Status{
public:
    Game_Status();
    int get_life() const;
    void set_life(int);
    int get_score()const;
    void update_c_score();
    void display();
    void reset();
    void display(int i);
    void display_life();                    // dis
    void save_h_score();
    void update_h_score();
    void enter_name();
    void sort_score();
    void reduce_health();
    void pro_health();
    bool check_health()const;
    short get_health()const;
    int curr_score(){return current_score;}
private:
    short health;
    int life,current_score;
    //char name[100];
    struct high{
        char user_name[100];
        int s;
        high(){
            user_name[0]='\0';
            s=0;
        }
    }high_score[6];
}life_count;
```

- get_life(), set_life() maintains life of the player.
- Reduce_health(), pro_health(),check_health(),get_health() maintains life bar of the player.
- Update_c_score() updates current score continuously.
- Update_h_score() updates high score if current score>top score till now.
- Sort_score sort the scores in the file.
- Save_h_score() save the score into the file
- Enter() name used for entering the name of the player at the beginning of the game.

HOW SCORE IS SAVED INTO THE FILE?

```
void Game_Status::save_h_score(){
    if(current_score>high_score[4].s){
        high_score[4]=high_score[5];
        high_score[4].s = current_score;
        life_count.sort_score();
    }
    for(int ii=0;ii<5;ii++){
        cout<<high_score[ii].user_name<<" "<<high_score[ii].s<<" "<<endl;
    }

    FILE* file = fopen("save","w");
    for(int i=0;i<5;i++)
        fprintf(file,"%s \t %d \n",high_score[i].user_name,high_score[i].s);
    fclose(file);
}
void Game_Status::update_h_score(){
    high temp;
    if(current_score>high_score[0].s){
        high_score[0]=temp;
        high_score[0]=high_score[5];
        high_score[5]=temp;

    }
}
void Game_Status::sort_score(){
    Game_Status::high temp;
    for(int i=1;i<5;i++){
        temp=high_score[i];
        int j=i-1;
        while((temp.s>high_score[j].s)&&(j>=0)){
            high_score[j+1]=high_score[j];    //moves element forward
            j=j-1;
        }
        high_score[j+1]=temp;    //insert element in proper place
    }
}
```

WHY WE HAVE USED C++?

Because It is object oriented I could have also used C but C++ being object oriented helped us to deal with complex situation much better than that of C which is not object oriented.

FUTURE PLANS

- IMPLEMENT THIS USING PARALLEL PROGRAMMING BECAUSE RIGHT KNOW WHOLE GAME IS RUNNING IN A SINGLE LOOP WHICH MAKES THE GAME LOOKS FLASHING.
- SPECIAL OBJECT FALLING, BY SHOOTING THAT OBJECT 5 TIMES HEALTH AND LIFE OF THE PLAYER INCREASES.
- BETTER LEVELS.
- BETTER SCORE SAVING METHOD.

BIBLIOGRAPHY

- EN.CPPREFERENCE.COM
- BJARNE STROUSTRUP BOOK