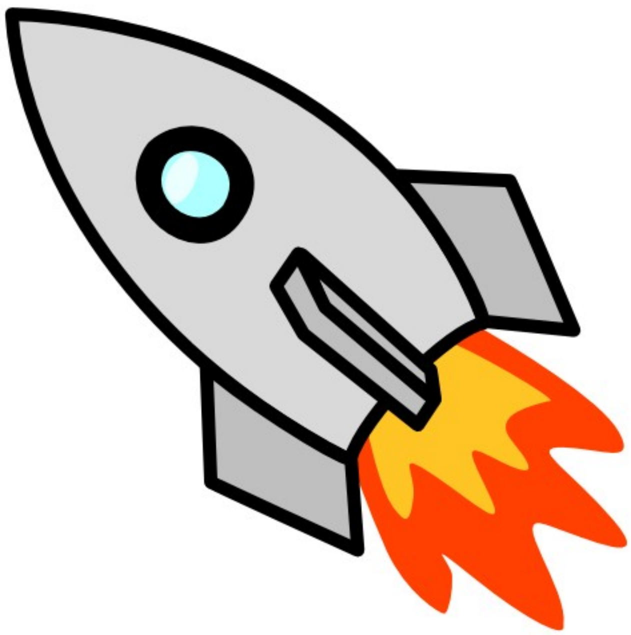
IMPACT

A space shooter

By: Arnav Dhamija, 11 B, National Public School HSR Layout



Acknowledgements

I would like to thank Nalina ma'am and Archana ma'am for teaching us the underlying concepts to make this project possible. Moreover, I would like to thank Aadi Jain for teaching me how to use the kbhit () function, without which, this project would have been very difficult to do. Moreover, online resources such as StackExchange and cprogramming.com were very useful

Contents

OVERVIEW OF C++

EVOLUTION OF C++

The C++ programming language has a history going back to 1979, when Bjorne Stroustrup was doing work for his Ph.D. thesis. One of the languages Stroustrup had the opportunity to work with was a language called Simula, which as the name implies is a language primarily designed for simulations.

Shortly thereafter, he began work on "C with Classes", which as the name implies was meant to be a superset of the C language. His goal was to add object-oriented programming into the C language, which was and still is a language well-respected for its portability without sacrificing speed or low-level functionality. His language included classes, basic inheritance, inlining, default function arguments, and strong type checking in addition to all the features of the C language.

The first C with Classes compiler was called Cfront, which was derived from a C compiler called CPre. It was a program designed to translate C with Classes code to ordinary C. A rather interesting point worth noting is that Cfront was written mostly in C with Classes, making it a self-hosting compiler (a compiler that can compile itself). Cfront would later be abandoned in 1993 after it became difficult to integrate new features into it, namely C++ exceptions.

In 1983, the name of the language was changed from C with Classes to C++. The ++ operator in the C language is an operator for incrementing a variable, which gives some insight into how Stroustrup regarded the language.

FEATURES

C++ is the multi paradigm, compile, free form, general purpose, statistically typed programming language. This is known as middle level language as it comprises of low level and high level language features.
The first commercial implementation of the C++ was released in 1985 and before that the name of language was changed to “C++”. And some new features were added to the language and the main features of the C++ are:

a. Classes
b. Inheritance
c. Data abstraction and encapsulation
d. Polymorphism
e. Dynamic Binding
f. Message passing

ADVANTAGES

1. Portable:the C++ standard is consistent
2. Industrial: evolved to satisfy the needs of software engineers, not computer scientists
3. Efficient: Compiles into highly optimized CPU-specific machine code with little or no runtime overhead.
4. Multi-paradigm: allows the use and penalty-free mixing of procedural, OOP, generic programming, functional programming, etc.
5. Strictly statically typed (unlike Python for example): a large amount of logic and calculations can be performed at compile time (by the type checking/inferring system).
6. Has deterministic memory management (as opposed to Java, C#, and other languages with garbage collectors): the life time of every object is known with absolute precision, which makes destructors useful and RAII possible.

Summary:

This project is an enjoyable shoot 'em up space game. The player must move the spaceship to shoot bullets to destroy incoming enemies. The player starts off with three lives and 15 health points (HP) per life.

Two power ups appear after every 5 and 10 turns. A Life power up provides an extra life and the Double Score power up gives 200 points per enemy kill for the 5 turns after it is obtained.

The game was inspired and named after the popular 2000 Nokia game 'Space Impact'.

User guide:

Controls are WASD for moving the ship up, left, down, and right respectively. The spacebar is for shooting bullets. On hitting an enemy with a bullet, the player gets 100 points added to their score.

The ship has 15 HP (health points) per life. 1 HP is lost when an enemy passes the ship.

Every 5 turns, either a Life (represented by ❤) or a Double Score (represented by a Ω) start moving on the playing area. Obtaining a Life gives the player an extra life. Obtaining a Double Score gives the player 200 points for every killed enemy for the next 5 turns. A meter displaying how many turns are left before the Double Score is exhausted.

No installation is required, it can either be compiled and run from TurboC++/DOSBOX, or be used by simply opening IMPACT.exe.

Requirements:

Hardware:
Enough to run TurboC++. If the standalone .exe is to be used, the computer must have at least 256 KB RAM and a Pentium II or more recent CPU. However using the .exe by itself is not recommended.

Software:
To run standalone .exe on Windows, only Windows XP or older can be used
To run in DOSBOX, Windows 7/8, and Linux can be used

Header files:

`<iostream.h>`
Used for cin and cout objects.

`<conio.h>`
Used for getch (), gotoxy (,), cprintf () [for coloured text], as well as disabling the cursor

`<dos.h>`
Used for delaying enemy and user movement in the game by 20-50 ms.

`<process.h>`
Used for exit(), to exit the game on pressing the quit key or when GAME OVER

`<time.h>`
Used as a RNG seed along with stdlib.h to generate random numbers for randomising the enemies as well as the power ups x-coordinates

`<stdlib.h>`
Used rand () along with time.h to generate random numbers for randomising the enemies as well as the power ups x-coordinates

User defined functions:

`void boundaries ()`
Used for printing borders within which all gameplay happens. It is called multiple times to avoid boundaries occasionally disappearing.

`void movement (char a, posVector &ship)`
Used for moving the ship within the aforementioned borders. ‘a’ is the parameter for input key and posVector is the current location of the ship on the screen.

`int collisions (posVector obj, posVector enemy, int type)`
Used for detecting several possible types of collisions such as collision of the ship with the enemy, collision of bullet with the enemy, and collision of ship with the power up. All such events trigger different actions and hence, type parameter is for specifying the type of collision. The posVector of the enemy and the ship denote current position of the ship and the enemy on the screen. The function returns an integer value which specifies the event of collision.

`void enemyMovement (posVector &enemy, posVector * &bullets, int scoreDouble)`
Used for animating the movement of enemies across the screen. Furthermore, the collisions function is used to check for collision of the enemy with bullets shot from the ship.

`void lifePrinter (int lives)`
`void healthPrinter (int health)`
`void doublePrinter (int doubleScore)`
Both the lifePrinter and the healthPrinter function have very similar purposes. They are used for printing the amount of health and life left to the player on the side of the screen in the form of a bar. The doublePrinter function similarly prints the number of turns left (out of a maximum of 5) before the Double Score power up is completely used up.

`void splashScreen ()`
Used for printing the game logo and basic instructions at the start of the game.

Source code:

```
/*
 mmmmm  m    m mmmmm    mm      mmm mmmmmmm
   #    ##  ## #   "#   ##   m"   "   #
   #    # ## # #mmm#"  #  #  #        #
   #    # "" # #       #mm#  #        #
 mm#mm  #    # #      #    #  "mmm"   #
*/
//20140127RC
//Developer: Arnav Dhamija
#include <iostream.h>
#include <conio.h>
#include <dos.h>
#include <process.h>
#include <time.h>
#include <stdlib.h>

//boundary limits
const int topBound = 1;
const int lowBound = 24; //use 44 on Windows XP
const int rBound = 60;
const int lBound = 20;
const int maxHealth = 15;
const int spawn = 10;

unsigned long int score = 0;

//enemy/ship symbols
const char eSym = 157;
const char shipSym = 143;

struct posVector
{
    int x;
    int y;
    int status;
};

void boundaries ()
{
    textcolor (YELLOW);
    char sym = '.';
    int i;
    /*
    //top border; use only if needed
    for (i = lBound; i < rBound; i++)
    {
        gotoxy (i, topBound);
        cprintf ("%c", sym);
    } */
    //bottom border
    for (i = lBound; i <= rBound; i++)
    {
        gotoxy (i, lowBound);
        cprintf ("%c", sym);
    }
    //right border
    for (i = topBound; i < lowBound; i++)
    {
        gotoxy (lBound, i);
        cprintf ("%c", sym);
    }
    //left border
    for (i = topBound; i < lowBound; i++)
    {
        gotoxy (rBound, i);
        cprintf ("%c", sym);
    }
}

void movement (char a, posVector &ship)
{
        textcolor (LIGHTGREEN);
        if (a == 'w' && ship.y > topBound + 1)
        {
            gotoxy (ship.x, ship.y);
            cprintf (" ");
            gotoxy (ship.x, --ship.y);
            cprintf ("%c", shipSym);
        }
        else if (a == 'a' && ship.x > lBound + 1)
        {
            gotoxy (ship.x, ship.y);
            cprintf (" ");
            gotoxy (--ship.x, ship.y);
            cprintf ("%c", shipSym);
        }
        else if (a == 's' && ship.y < lowBound - 1)
        {
            gotoxy (ship.x, ship.y);
            cprintf (" ");
            gotoxy (ship.x, ++ship.y);
            cprintf ("%c", shipSym);
        }
        else if (a == 'd' && ship.x < rBound - 1)
        {
            gotoxy (ship.x, ship.y);
            cprintf (" ");
            gotoxy (++ship.x, ship.y);
            cprintf ("%c", shipSym);
        }
        else if (a == 'p')
        {
            textcolor (RED);
            gotoxy (1, 13);
            cprintf ("PAUSED");
            gotoxy (1, 14);
            cprintf ("STRIKE ANY KEY TO");
            gotoxy (1, 15);
            cprintf ("CONTINUE");
            getch ();
            gotoxy (1, 13);
            cprintf ("                  ");
            gotoxy (1, 14);
            cprintf ("                  ");
            gotoxy (1, 15);
            cprintf ("                  ");
        }
        else if (a == 'q')
            exit (0);
}
```

```
int collisions (posVector obj, posVector enemy, int type)
{
    if (obj.x == enemy.x && obj.y == enemy.y && enemy.status == 1 && type == 0)
        return 1;
    else if (obj.x == enemy.x && obj.y == enemy.y && obj.status == 1 && enemy.status == 1
&& type == 1)
        return 2;
    return 0;
}

void enemyMovement (posVector &enemy, posVector * &bullets, int scoreDouble)
{
    textcolor (LIGHTBLUE);
    for (int i = 0; i < 30; i++)
        if (collisions (bullets[i], enemy, 1) == 2)
        {
            bullets[i].status = 0;
            enemy.status = 0;
            gotoxy (bullets[i].x, bullets[i].y);
            cprintf (" ");
            gotoxy (bullets[i].x, bullets[i].y - 1);
            cprintf (" ");
            if (scoreDouble > 0)
                score += 200;
            else
                score += 100;
        }
    if (enemy.y <= lowBound && enemy.y > 1 && enemy.status != 0)
    {
        gotoxy (enemy.x, enemy.y);
        cprintf ("%c", eSym);
        gotoxy (enemy.x, enemy.y - 1);
        cprintf (" ");
    }

    if (enemy.status == 0)
    {
        gotoxy (enemy.x, enemy.y);
        cprintf (" ");
        gotoxy (enemy.x, enemy.y - 1);
        cprintf (" ");
    }
}

void lifePrinter (int lives)
{
    gotoxy (1, 4);
    textcolor (LIGHTGREEN);
    cprintf ("LIVES:");
    gotoxy (1, 5);
    cout << "               ";

    for (int i = 1; i <= lives; i++)
    {
        textcolor (LIGHTRED);
        gotoxy (i, 5);
        cprintf ("%c", char (3));
    }
}

void healthPrinter (int health)
```

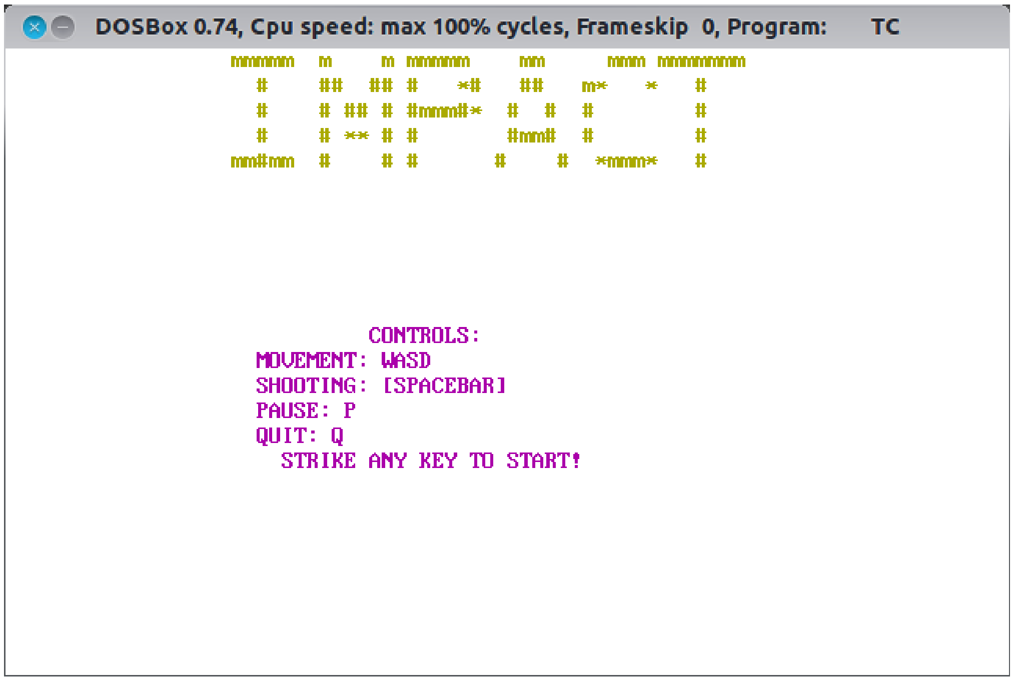
```
{
    _setcursortype (_NOCURSOR);
    int i;
    for (i = 1; i <= maxHealth; i++)
    {
        gotoxy (i, 8);
        cout << ' ';
    }
    for (i = 1; i <= health; i ++)
    {
        textcolor (LIGHTGREEN);
        gotoxy (1, 7);
        cprintf ("HEALTH:");
        if (health > 8 )
            textcolor (LIGHTGREEN);
        else if (health > 4)
            textcolor (YELLOW);
        else
            textcolor (LIGHTRED);
        gotoxy (i, 8);
        cprintf ("%c", char (219)); //or 179
    }
}

void doublePrinter (int doubleScore)
{
    _setcursortype (_NOCURSOR);
    int i;
    for (i = 1; i <= 5; i++)
    {
        gotoxy (i, 11);
        cout << ' ';
    }
    for (i = 1; i <= doubleScore; i++)
    {
        textcolor (LIGHTGREEN);
        gotoxy (1, 10);
        cprintf ("DOUBLE SCORE!");
        if (doubleScore > 3 )
            textcolor (LIGHTGREEN);
        else if (doubleScore > 2)
            textcolor (YELLOW);
        else
            textcolor (LIGHTRED);
        gotoxy (i, 11);
        cprintf ("%c", char (219)); //or 179
    }
}

void splashScreen ()
{
    _setcursortype(_NOCURSOR);
    textcolor (LIGHTBLUE);
    gotoxy (18, 1);
    cprintf (" mmmmm  m    m mmmmm    mm     mmm mmmmmmm\n");
    gotoxy (18, 2);
    cprintf ("   #    ##  ## #   *#   ##   m*   *   #  \n");
    gotoxy (18, 3);
    cprintf ("   #    # ## # #mmm#*  #  #  #        #  \n");
    gotoxy (18, 4);
    cprintf ("   #    # ** # #       #mm#  #        #   \n");
    gotoxy (18, 5);
    cprintf (" mm#mm  #    # #      #    #  *mmm*   #   \n");

    textcolor (LIGHTGREEN);
    gotoxy (lBound + 10, lowBound / 2);
    cprintf ("CONTROLS:");
    gotoxy (lBound + 1, lowBound / 2 + 1);
    cprintf ("MOVEMENT: WASD");
    gotoxy (lBound + 1, lowBound / 2 + 2);
    cprintf ("SHOOTING: [SPACEBAR]");
    gotoxy (lBound + 1, lowBound / 2 + 3);
    cprintf ("PAUSE: P");
    gotoxy (lBound + 1, lowBound / 2 + 4);
    cprintf ("QUIT: Q");
    gotoxy (lBound + 3, lowBound / 2 + 5);
    cprintf ("STRIKE ANY KEY TO START!");
    getch ();
    clrscr ();
}

void main ()
{
    clrscr ();
    splashScreen ();
    posVector ship, enemy[3], bullets[30], lifeUp, scoreUp;
    textcolor (LIGHTGREEN);
    int sep, i, lives = 3, count = 0, del = 45, j, k = 0, o, sep1, heat = 0, cooling = 0,
enemyCount = 0, health, scoreDouble = 0;
    char sym = '.';
    unsigned int seed;
    char a;
    _setcursortype (_NOCURSOR);

    //sanitising bullets
    for (i = 0; i < 30; i++)
        bullets[i].status = 0;

    //initialising ship position
    ship.x = (rBound + lBound) / 2;
    ship.y = (topBound + lowBound) * 3 / 4;

    time_t tmp;
    seed = (unsigned)time (&tmp);
    srand (seed);

    //some initialisation
    boundaries ();
    gotoxy (ship.x, ship.y);
    cout << shipSym;

    health = maxHealth;Ubuntu
    healthPrinter (health);

    while (1) //safe infinite loop :)
    {
        count++;
        enemy[0].status = 1;
        enemy[1].status = 1;
        enemy[2].status = 1;
        scoreUp.status = 1;
        //powerups spawn once in <spawn value>
        if ((count % spawn) == 0)
            lifeUp.status = 1;
        else
            lifeUp.status = 0;

        if (((count + 5) % spawn) == 0)
            scoreUp.status = 1;
        else
            scoreUp.status = 0;

        if ((count % 6) == 0 && (del / 1.2) > 20)
            del /= 1.2;

        //randomizing the x-coordinates of all enemies
        enemy[0].x = (rand () % (rBound - lBound - 2)) + lBound + 1;
        enemy[1].x = (rand () % (rBound - lBound - 2)) + lBound + 1;
        enemy[2].x = (rand () % (rBound - lBound - 2)) + lBound + 1;
        lifeUp.x = (rand () % (rBound - lBound - 2)) + lBound + 1;
        scoreUp.x = (rand () % (rBound - lBound - 2)) + lBound + 1;

        //randomizing separation between enemies
        sep = (rand () % 10);
        sep1 = (rand () % 10);

        enemy[0].y = 2;
        enemy[1].y = enemy[0].y - sep;
        enemy[2].y = enemy[1].y - sep1;
        lifeUp.y = 0;
        scoreUp.y = 0;

        while (enemy[2].y <= lowBound)
        {
            //boundaries tend to get damaged, this repairs them
            boundaries ();

            //bullet free space detection
            for (int j = 0; j < 30; j++)
            {
                if (bullets[j].status == 0)
                    break;

            }

            //WASD navigation
            if (kbhit ())
            {
                a = getch ();
                if (a != ' ')
                    movement (a, ship);
                else
                {
                    if (cooling == 0)
                    {
                        bullets[j].x = ship.x;
                        bullets[j].y = ship.y - 1;
                        bullets[j].status = 1;
                        heat++;
                    }
                    if (heat >= 15)  //change this value to allow more/less bullets on
screen
```

```
        {
            gotoxy (1, 13);
            cout << "             ";
            gotoxy (1, 13);
            textcolor (LIGHTRED);
            cprintf ("OVERHEATING!");
            cooling = 1;
        }
        else
        {
            gotoxy (1, 13);
            cout << "              ";
        }
    }
}

//bullets with full support for collisions :)
for (i = 0; i < 30; i++)
{
    textcolor (LIGHTGREEN);
    if (bullets[i].status == 1 && bullets[i].y > 1)
    {
        //bullet - ship collisions
        for (o = 0; o < 3; o++)
            if (collisions (bullets[i], enemy[o], 1) == 2)
            {
                gotoxy (enemy[o].x, enemy[o].y);
                cprintf (" ");
                gotoxy (enemy[o].x, enemy[o].y - 1);
                cprintf (" ");
                bullets[i].status = 0;
                enemy[o].status = 0;
                if (scoreDouble > 0)
                    score += 200;
                else
                    score += 100;
            }
        if (bullets[i].status == 1)
        {
            gotoxy (bullets[i].x, --bullets[i].y);
            cprintf ("|");
            gotoxy (bullets[i].x, bullets[i].y + 1);
            cprintf (" ");
        }
    }
    else if (bullets[i].y <= 1 && bullets[i].status == 1)
    {
        gotoxy (bullets[i].x, bullets[i].y);
        cout << ' ';
        bullets[i].status = 0;
    }
    if (bullets[i].status == 0)
        k++;
}

//detecting whether ship hits the enemy
for (i = 0; i < 3; i++)
    if (collisions (ship, enemy[i], 0) == 1)
    {
        if (lives == 1)
        {
            clrscr ();
            textcolor (LIGHTGREEN);
            cprintf ("GAME OVER : (");
            cout << '\n';
            cprintf ("SCORE: ");
            cprintf ("%d", score);
            getch ();
            exit (0);
        }
        else
        {
            enemy[i].status = 0;
            gotoxy (1, 10);
            cout << "             ";
            gotoxy (1, 10);
            health = maxHealth;
            textcolor (LIGHTRED);
            cprintf ("-1 LIFE");
            --lives;
        }
    }

//detecting whether ship hits lifeUp
if (collisions (ship, lifeUp, 0))
{
    textcolor (GREEN);
    lives++;
    lifeUp.status = 0;
    gotoxy (ship.x, ship.y - 1);
    cprintf ("%c", ' ');
    gotoxy (ship.x, ship.y);
    cprintf ("%c", shipSym);
}

//similar for scoreUp
if (collisions (ship, scoreUp, 0))
{
    textcolor (GREEN);
    scoreDouble = 5;
    scoreUp.status = 0;
    gotoxy (ship.x, ship.y - 1);
    cprintf ("%c", ' ');
    gotoxy (ship.x, ship.y);
    cprintf ("%c", shipSym);
}

//GAME OVER
if (lives == 0)
{
    clrscr ();
    textcolor (LIGHTGREEN);
    cprintf ("GAME OVER : (");
    cout << '\n';
    cprintf ("SCORE: ");
    cprintf ("%d", score);
    getch ();
    exit (0);
}

//invalidates all enemies crossing the border
for (i = 0; i < 3; i++)
```

```
    if (enemy[i].y == lowBound)
        enemy[i].status = 0;

//invalidates lifeUp
if (lifeUp.y == lowBound)
{
    lifeUp.status = 0;
    gotoxy (lifeUp.x, lifeUp.y - 1);
    cprintf ("%c", ' ');
}

//invalidates scoreUp
if (scoreUp.y == lowBound)
{
    scoreUp.status = 0;
    gotoxy (scoreUp.x, scoreUp.y - 1);
    cprintf ("%c", ' ');
}

if (enemyCount % 2 == 0) //delays the loop a bit
{
    //power up movement bit
    if (lifeUp.status == 1)
    {
        if (lifeUp.y > 0)
        {
            textcolor (LIGHTRED);
            gotoxy (lifeUp.x, lifeUp.y);
            cprintf("%c", 3);
            gotoxy (lifeUp.x, lifeUp.y - 1);
            cprintf("%c", ' ');
        }
        lifeUp.y++;
    }

    if (scoreUp.status == 1)
    {
        if (scoreUp.y > 0)
        {
            textcolor (YELLOW);
            gotoxy (scoreUp.x, scoreUp.y);
            cprintf("%c", 234);
            gotoxy (scoreUp.x, scoreUp.y - 1);
            cprintf("%c", ' ');
        }
        scoreUp.y++;
    }

    //moving the enemies
    for (i = 0; i < 3; i++)
        enemyMovement (enemy[i], bullets, scoreDouble);
    //little block; essential for incrementing the loop
    enemy[1].y = enemy[0].y - sep;
    enemy[2].y = enemy[1].y - sep1;
    enemy[0].y++;
}

//makes the game playable :)
delay (del);

//health reduction
```

```
            for (i = 0; i < 3; i++)
                if (enemy[i].y == lowBound && enemy[i].status == 1)
                {
                    --health;
                    enemy[i].status = 0;
                }

            //if health reaches 0
            if (health <= 0)
            {
                gotoxy (1, 10);
                textcolor (LIGHTRED);
                cprintf ("-1 LIFE");
                --lives;
                health = maxHealth;
            }

            //just to ensure that the ship doesn't go away after it is struck by an enemy
            gotoxy (ship.x, ship.y);
            textcolor (LIGHTGREEN);
            cprintf ("%c", shipSym);

            //little touches: score and formatting
            gotoxy (1, 1);
            textcolor (LIGHTGREEN);
            cprintf ("SCORE:");
            gotoxy (1, 2);
            cout << "                 ";
            gotoxy (1, 2);
            textcolor (LIGHTGREEN);
            cprintf ("%d",score);
            healthPrinter (health);
            lifePrinter (lives);

            //calls to print the doublePrinter
            if (scoreDouble > 0)
                doublePrinter (scoreDouble);
            else
            {
                gotoxy (1, 10);
                cout << "             ";
                gotoxy (1, 11);
                cout << "             ";
            }

            enemyCount++;
        }

        //some cleaning for good UI
        gotoxy (1, 10);
        cout << "             ";
        gotoxy (1, 11);
        cout << "             ";
        heat = 0;
        cooling = 0;
        if (scoreDouble > 0)
            scoreDouble--;
    }
}
```

Screenshots (colours inverted):

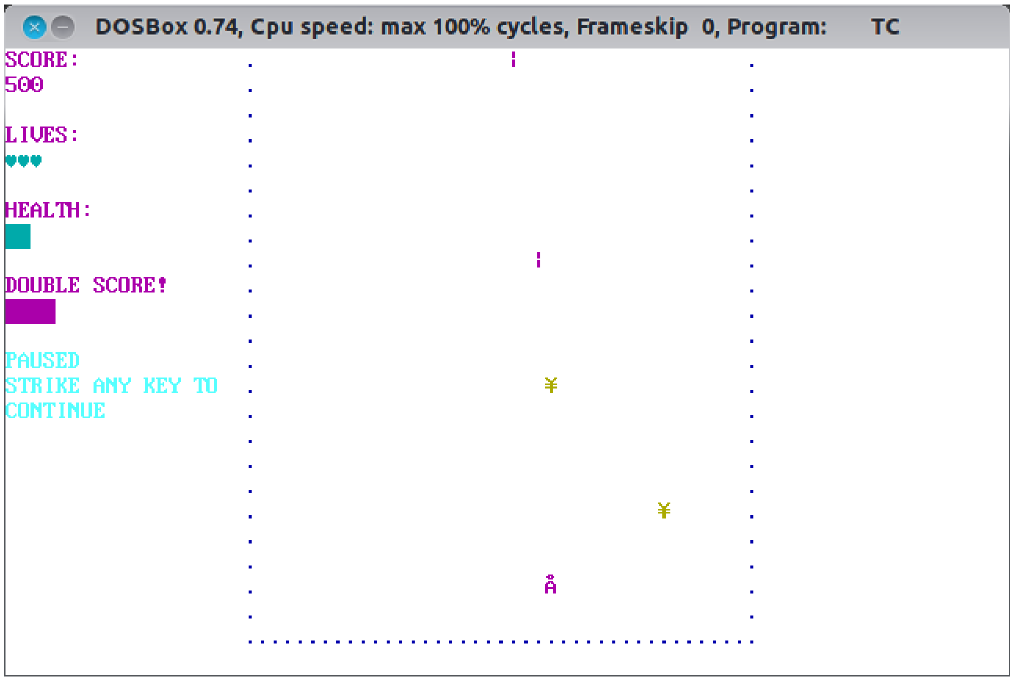


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
mmmmmm  m     m  mmmmmm     mm      mmmm mmmmmmmm
  #     ##   ##  #    *#    ##    m*    *   #
  #     #  ## #  #mmmm#*  #   #   #         #
  #     #  ** #  #        #mmm#   #         #
mm#mm   #     #  #       #     #   *mmmm*   #
```

```
         CONTROLS:
MOVEMENT: WASD
SHOOTING: [SPACEBAR]
PAUSE: P
QUIT: Q
  STRIKE ANY KEY TO START!
```

Introduction screen



DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
SCORE:
500

LIVES:
♥♥♥

HEALTH:
█

DOUBLE SCORE!
████

PAUSED
STRIKE ANY KEY TO
CONTINUE
```

Gameplay, including the Double Score Power Up in effect

Shortcomings:

The game uses no graphics libraries or files. This makes the graphics seem rather primitive. Moreover, only three enemies are displayed on the screen at a time and due to the lack of levels or boss battles, the game can feel rather repetitive after extended use.

Bibliography:

cplusplus.com
stackoverflow.com
cprogramming.com
dosbox.com