DEVELOPER'S GUIDE

THE NEWEST 2 PLAYER
VIDEO SKILL GAME

PONG

from ATARI CORPORATION
SYZYGY ENGINEERED

The Team That Pioneered Video Technology

FEATURES

- STRIKING Attract Mode
- Ball Serves Automatically
- Realistic Sounds of Ball Bouncing, Striking Paddle
- Simple to Operate Controls
- ALL SOLID STATE TV and Components for Long, Rugged Life
- ONE YEAR COMPUTER WARRANTY
- Proven HIGH PROFITS in Location After Location
- Low Key Cabinet, Suitable for Sophisticated Locations
- 25¢ per play

THIS GAME IS AVAILABLE FROM YOUR LOCAL DISTRIBUTOR

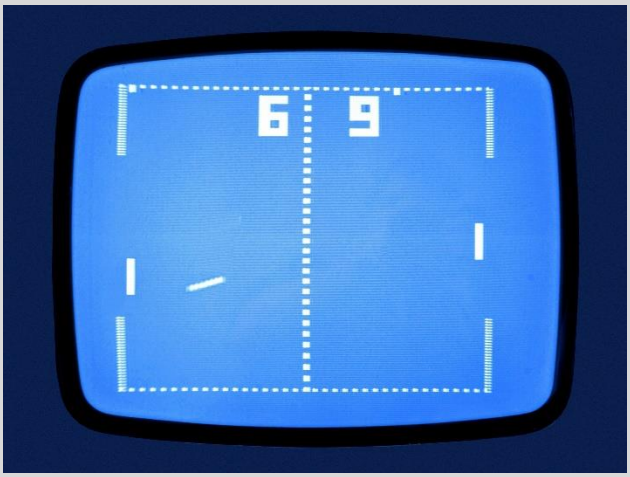
Manufactured by
ATARI INC.

Maximum Dimensions:
WIDTH -26"

DEVELOPER'S GUIDE

OVERVIEW:

Games are just a piece of software- an executable that people can double click and launch.Pong is a table tennis inspired game, and is considered one of the Classic in gaming industry. There has been many iterations of this game from very nostalgic to a very modern look. This is one of its iteration written in C++ programming language using <windows.h> library with the help of the UNICODE. It doesn't use any graphics so all the drawings are drawn with characters.



WIKIPEDIA *: Unicode is an information technology standard for the consistent encoding, representation, and handling of text expressed in most of the world's writing systems.*

The UNICODE is HEX numbers which are to be interpreted by the compiler to create certain CHARACTERS onto the screen. UNICODE functionality is not turned on by default, so it must be enabled first. As this game is entirely CONSOLE based because it does not use any of the GRAPHICS libraries but rather <windows.h> library to create a console window in WINDOWS OPERATING SYSTEM.

NOTE*: This program might not run on other platforms apart from WINDOWS OPERATING SYSTEM (PLATFORM DEPENDENT). This program was made on Visual Studio Community, so it is preferred to be compiled on Visual Studio Community which UNICODE on settings.*

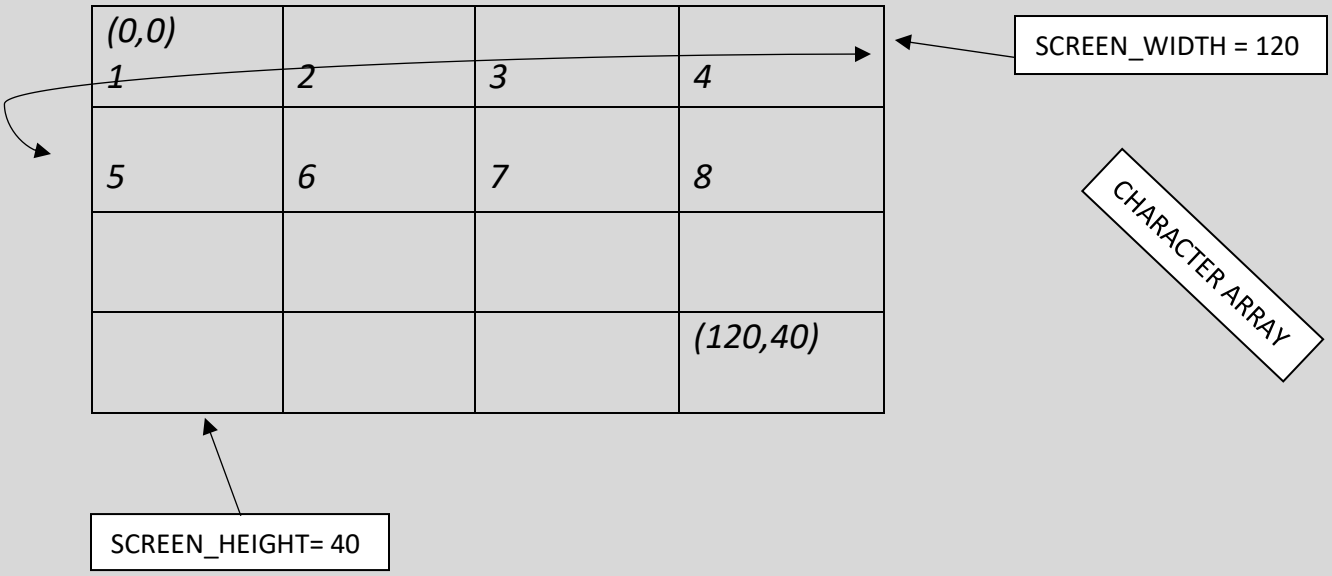
This game has very same aesthetic and playing feel as its predecessors and uses one puck and 2 paddles controlled by each of the PLAYERS. Keeping the keys pressed slide the paddles either UP or DOWN to defend from being scored points. Puck moves in all direction and gets deflected when collided with a Paddle surface

DEVELOPER'S GUIDE

or Lower and Upper borders. The Player who scores 5 points first WINS the game. This is just very basic description of the Program, but more things will be found out when the Game is actually played.

SOLUTION DESIGN:

As we know that this is a Command Line Program, so there are no graphics involved; it is purely generated using text. As we know all the characters of the text occupies the same amount of area, so we can treat them as pixels.

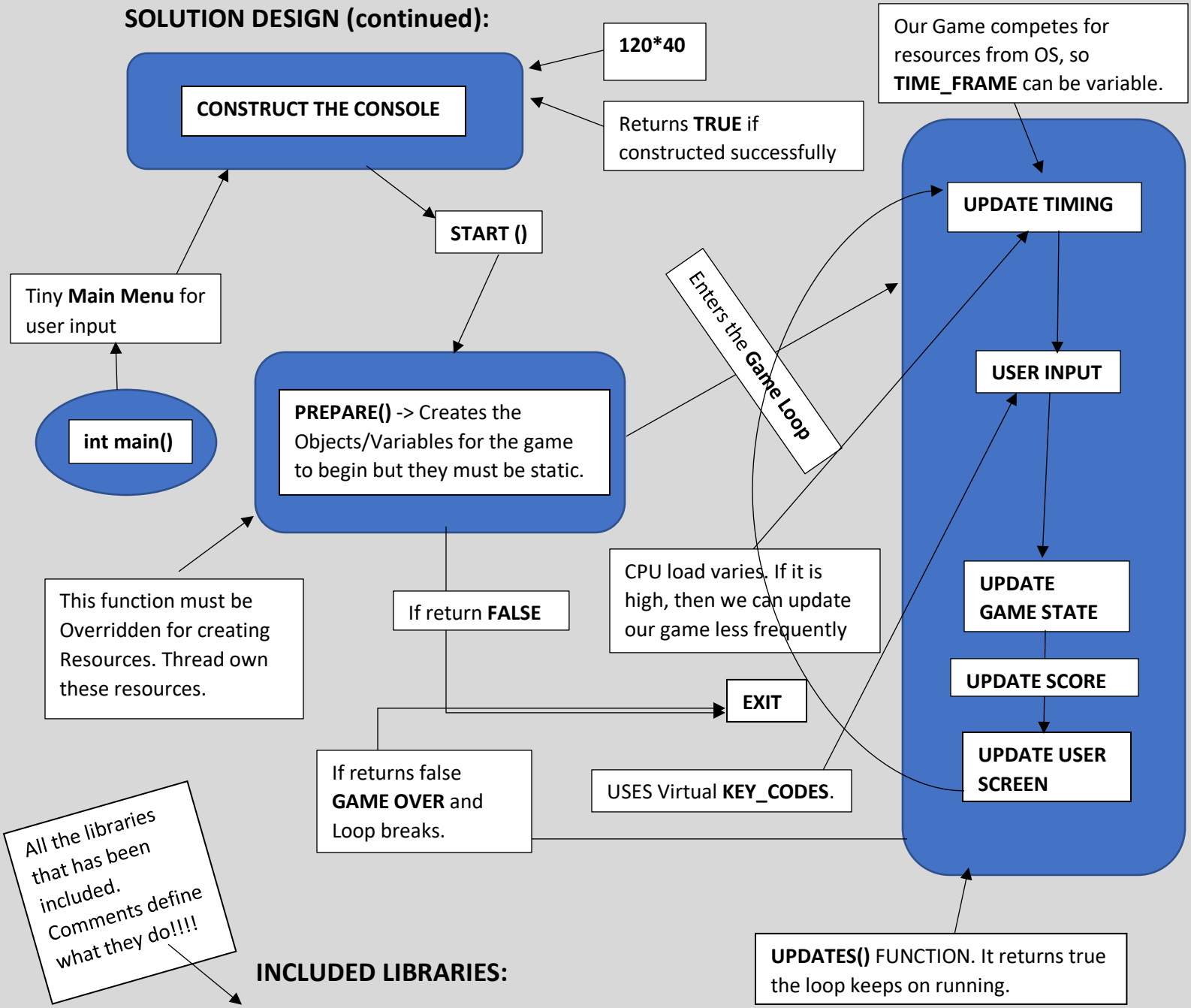


*It is actually 1D array, but we treat it as 2D as it wraps right back around on the new line. The screen is made up of the **CHAR_INFO** struct which made up of 2 things:*

1. *Characters - >represented through UNICODE (16 bit encoding) which we can use with the help of UNSIGNED SHORT or WCHAR_T.*
2. *Palette -> FOREGROUND AND BACKGROUND COLOR- > helps to give color to our console.*

DEVELOPER'S GUIDE

SOLUTION DESIGN (continued):



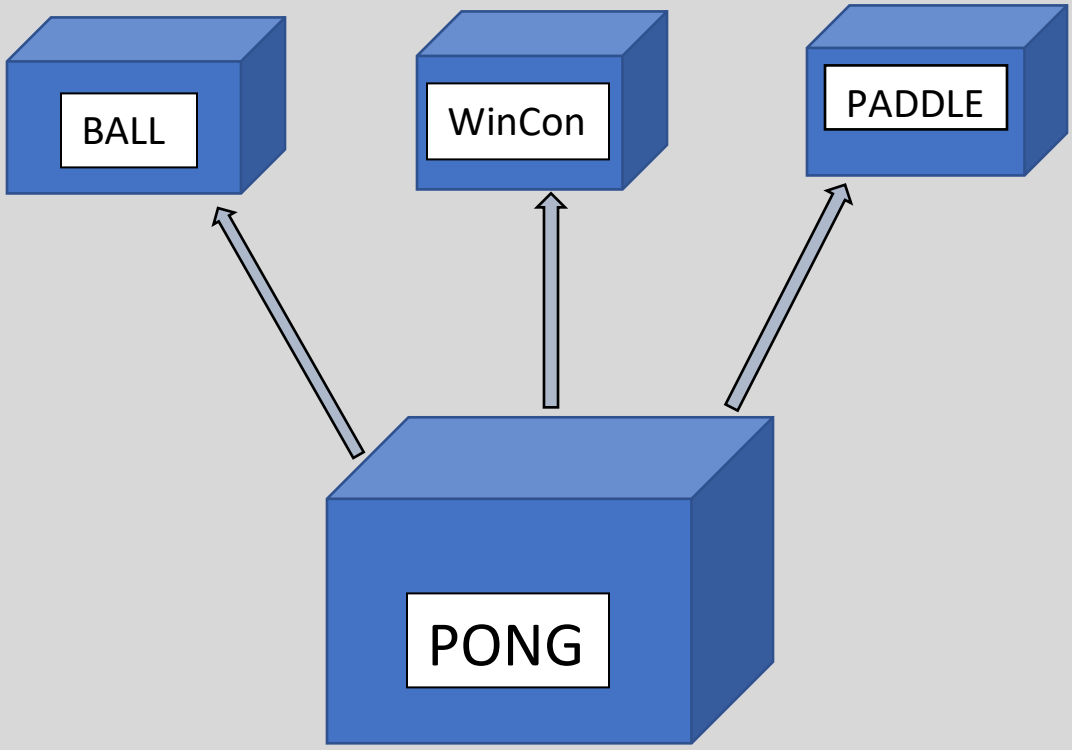
INCLUDED LIBRARIES:

```
#include <iostream>// Input-Output Standard Library
#include <chrono>//Library for taking out the time points to get the Frame Rate.
#include <thread> //Library for Thread Function
#include <sstream> // WSTRING and many more on BUFFER.
#include <fstream>// Library for writing into and reading from a file.

using namespace std;

#include <windows.h> // Windows Library for Windows commands (therefore it's platform dependent).
```

DEVELOPER'S GUIDE

SOLUTION APPROACH & FILES:

In total there are 4 HEADER FILES and 3 SOURCE files with 3 classes Ball, Paddle and WinCon inherited by the Pong class. (MULTIPLE INHERITANCE)



MUTIPLE INHERITANCE!!!

There are three parent classes and one of them is abstract (using PURE VITUAL FUNCTIONS)-> WinCon. Those functions have been overridden in Pong Class, so Polymorphism has been used as well in the project.

```
//////////////////////////////PURE VIRTUAL FUNCTIONS ////////////////////////////////////

   //These functions has to be Overided (Virtual)
   virtual bool PREPARE() = 0;// Some Basic Creation
   virtual bool UPDATES(float TIME_DIFFERENCE) = 0; //FREQUENTLY UPDATED
```

PREPARE() *function makes sure that Ball Direction variables has been initialized for Ball object to move in that particular direction when game starts. This function must return true for us to move into the Game Loop and inside it into the* ***UPDATES()*** *function. Updates() function runs in a loop and observe User input, Updated Game States, Logics of Movement ,Updated Score, and Updated drawing onto the screen.*

DEVELOPER'S GUIDE

1.Ball.h → (HEADER FILE).

```
//NO CPP FILE becasue it's just one constructor and Attributes/////

//////////////////////////////////////////Ball Class////////////////
//(Parent)
class Ball
{
public: //public becasue of the data Mondification
    float BALL_X, BALL_Y; //Ball Position
    float INIT_X, INIT_Y; //Default Position
    int BALL_DX, BALL_DY; //Ball Directions
    float SPEED; //Speed of the Ball
public:
    //Multiple Parameter Constructor with Default Arguments
    Ball(float X_POS = 0.0, float Y_POS = 0.0, float SPEED= 60.0f)
    {
        INIT_X = X_POS;
        INIT_Y = Y_POS;
        BALL_X = X_POS;
        BALL_Y = Y_POS;
        BALL_DX = -1, BALL_DY = -1;
        this->SPEED = SPEED;
    }
};
#endif // !BALL_H
```

This is one of the Parent class and it is only a dot H file because there is not any function associated to this classes apart from its constructor which is a multiparameter constructor with default arguments. This class's attributes have been made public so that their values can be changed easily as BALL object is a very fast-moving object in the game. We cannot have function calls to grab the values as there will be overheads. So all the values changes and mechanics has been laid out in CPP file of Pong Class overridden ***Updates()*** *function.*

DEVELOPER'S GUIDE

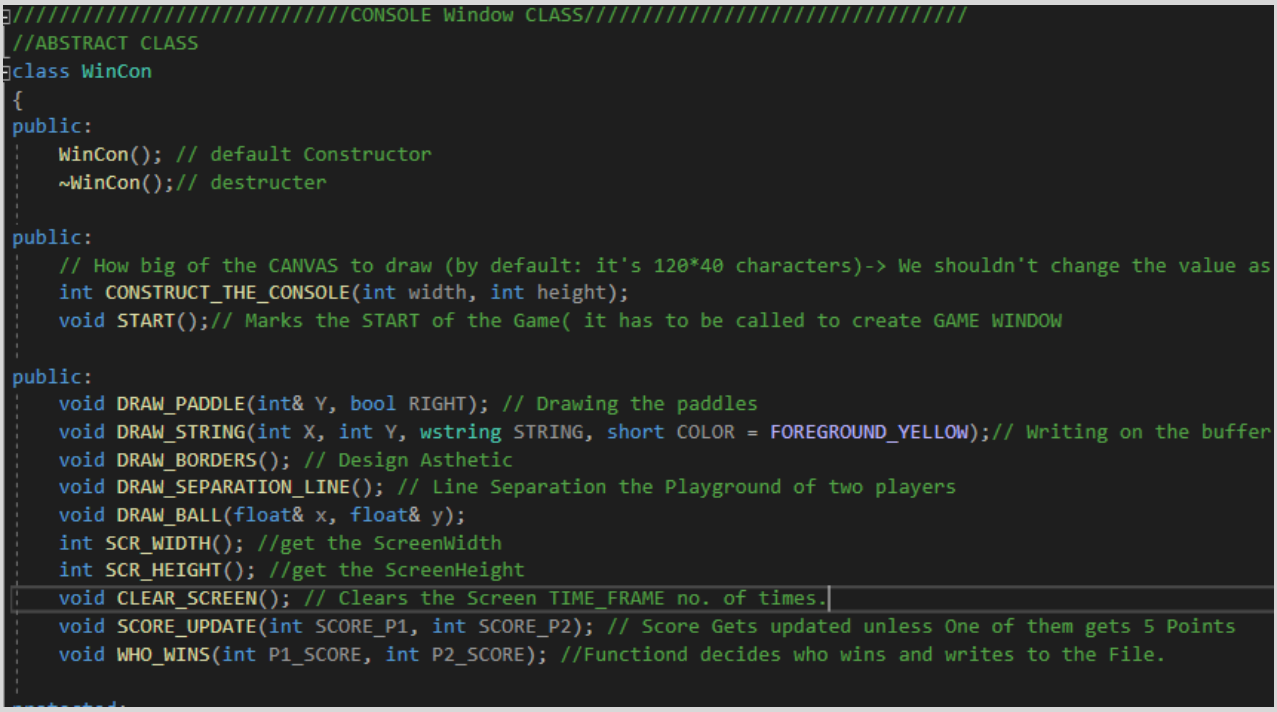
2. Paddle -> (HEADER FILE)

```
//Paddle Class
//(Parennt)
class Paddle
{
public://public becasue of the data Mondification
    float PLAYER_X, PLAYER_Y; // Position of Paddles with their X and Y
    float DEFAULT_X, DEFAULT_Y; // Default Position
    int SCORE; //SCORE OF THE Player
public:
    //Multiple Parameter Constructor with Default Arguments
    Paddle(float XPOS = 0, float YPOS = 0)
    {
        DEFAULT_X = XPOS;
        DEFAULT_Y = YPOS;
        PLAYER_X = XPOS;
        PLAYER_Y = YPOS;
        SCORE = 0;
    }
};

#endif // !PADDLE_H
```

This is one of the Parent class as well and same as before it is only a dot H file because it has only its Multiparameter constructor with default arguments. Same as ball class all values has been made public so the values of Paddle obj can be manipulated easily. User uses virtual keys to slide the paddle up and down with a certain speed that implements that TIME_FRAME variable which has been made possible with help of chrono library. So, the change in position is really quick that's not movement functions rather all the codes is written inside ***UPDATES()*** *overridden function inside Pong Class.*

DEVELOPER'S GUIDE

3.WinCon -> (HEADER FILE)



```
//////////////////////////////CONSOLE Window CLASS////////////////////////////////
//ABSTRACT CLASS
class WinCon
{
public:
    WinCon(); // default Constructor
    ~WinCon();// destructer

public:
    // How big of the CANVAS to draw (by default: it's 120*40 characters)-> We shouldn't change the value as
    int CONSTRUCT_THE_CONSOLE(int width, int height);
    void START();// Marks the START of the Game( it has to be called to create GAME WINDOW

public:
    void DRAW_PADDLE(int& Y, bool RIGHT); // Drawing the paddles
    void DRAW_STRING(int X, int Y, wstring STRING, short COLOR = FOREGROUND_YELLOW);// Writing on the buffer
    void DRAW_BORDERS(); // Design Asthetic
    void DRAW_SEPARATION_LINE(); // Line Separation the Playground of two players
    void DRAW_BALL(float& x, float& y);
    int SCR_WIDTH(); //get the ScreenWidth
    int SCR_HEIGHT(); //get the ScreenHeight
    void CLEAR_SCREEN(); // Clears the Screen TIME_FRAME no. of times.
    void SCORE_UPDATE(int SCORE_P1, int SCORE_P2); // Score Gets updated unless One of them gets 5 Points
    void WHO_WINS(int P1_SCORE, int P2_SCORE); //Functiond decides who wins and writes to the File.
```

WinCon *is the biggest class in the Pong Game as it contains all the drawing functions as well the constructor and destructors which is later ultimately inherited into the Pong Game.*

Default Constructor *of WinCon initializes the* ***ScreenWidth*** *and* ***ScreenHeight*** *variables and gets the Handle to the Output Console. Meanwhile destructor deallocates the* ***Screen_Buffer*** *memory.* ***CONSTRUCT_THE_COSNSOLE()*** *method creates the console with the Screen_Buffer which sits behind it for the drawings to be made. This is done specifically using commands from Windows library!!!*

DEVELOPER'S GUIDE

```
//Set up the RECTANGLE( 2D coordinates) Canvas and passing it as CONSOLE WINDOW INFO.
RECT_WIN = { 0, 0, 1, 1 };
SetConsoleWindowInfo(HANDLE_CONSOLE_OUT, TRUE, &RECT_WIN);

// SIZE OF THE SCREEN BUFFER
//USING COORD (coordinates) ASSUMING THE CANVAS IS A 2D.
COORD coord = { (short)SCREEN_WIDTH, (short)SCREEN_HEIGHT };

// PHYSICAL WINDOW SIZE (THE APPEARANCE)
RECT_WIN = { 0, 0, (short)SCREEN_WIDTH - 1, (short)SCREEN_HEIGHT - 1 };

// ALLOCATING MEMORY TO THE SCREEN BUFFER of type ( CHAR_INFO ). IT'S A 16 BIT UNICODE.
SCREEN_BUFFER = new CHAR_INFO[SCREEN_WIDTH * SCREEN_HEIGHT];

//Initializing the CHAR_INTO STRUCT with memset to 0
memset(SCREEN_BUFFER, 0, sizeof(CHAR_INFO) * SCREEN_WIDTH * SCREEN_HEIGHT);
```

RECT_WIN *is a* ***SMALL_RECT*** *type with coordinates being the corners of the rectangle which is the window size(initialization)*

COORD *helps to form a 2D canvas with ScreenHeight and ScreenWidth*

SCREEN_BUFFER *has been allocated of size (ScreenHeight *ScreenWidth) of type* ***CHAR_INFO*** *struct.*

memset *just initializes the entire screen buffer with 0.*

SetConsoleWindowInfo() *passes the CONSOLE OUTPUT HANDLE to it and makes* ***RECT_WIN*** *its window.*

```
////////////////////////////////////////GAME THREAD FUNCTION////////////////////////////////////
void WinCon::GAME_THREAD()
{
```

Meanwhile in the ***GAME_THREAD*** *function runs a Game Loop but before getting inside the loop we take two TIME_POINTS and we compare the one with the one inside the loop to get the time difference ->****TIME_PASSED*** *variable and then it is counted to get the* ***TIME_FRAME****. Our Game competes for resources from OS, so* ***TIME_FRAME*** *can be variable on different machines. It's actually time in seconds but very small, and we divide 1 by it -> 1/TIME_FRAME = frames/sec. It is part of CHRONO library and it gives better user experience even when CPU is under high load.*

DEVELOPER'S GUIDE

```
//////////////////////////////////////GAME THREAD FUNCTION//////////////////////////////////
void WinCon::GAME_THREAD()
{
    // Create user resources as part of this thread
    if (!PREPARE())
        ACTIVE = false;
    auto TIME_POINT1 = chrono::system_clock::now();
    auto TIME_POINT2 = chrono::system_clock::now();

    while (ACTIVE)
    {
        ////////////////////// Run as fast as possible/////////////
        while (ACTIVE){


            ////////////////////////////////////TIMING (USING CHRONO LIBRARY)//////////////////
            //To get the ----> 1/Frame-Rate
            TIME_POINT2 = chrono::system_clock::now();
            chrono::duration<float> TIME_PASSED = TIME_POINT2 - TIME_POINT1;
            TIME_POINT1 = TIME_POINT2;
            float TIME_FRAME = TIME_PASSED.count();
```

DEVELOPER'S GUIDE

4.Pong -> (HEADER FILE)

```
//Pong Class
//Child Class
class Pong :public WinCon, public Ball, public Paddle //Multiple Inheritance
{
public:
    //BOOL VARIABLES just to decide when to Finish the Game and when to Start a New One.
    bool GAME_OVER = false;
    bool GAME_RESET = true;

    //Pointer Objecte Variable
    Ball* ball_obj;
    Paddle* PLAYER1;
    Paddle* PLAYER2;
public:
    //Default Constructor
    Pong();

    //Destructor
    ~Pong();

public:
    bool PREPARE()//To initialize some basic variables before the Game Starts.
    {

        BDX = ball_obj->BALL_DX;
        BDY = ball_obj->BALL_DY;
        return true; //Must return true to Carry on to the Game
    }

    bool UPDATES(float TIME_FRAME);//Objects' variables get updated and DRAWN to the Screen
```

Pong is a Child Class and it inherits from all the earlier three classes. Pong's default constructor allocates memory to ***ball_obj, PLAYER1, PLAYER2*** *objects and destructor deallocates when Pong object inside* ***int main()*** *function finishes -> that only happens when the Game Loop breaks /GAME_THREAD ends****.(MEMORY MANAGEMENT is used here)***

*ALL the controls, mechanics and logics are in the UPDATES() function that has been overridden****. EXCEPTION HANDLING*** *is used to check if there occurs any unforeseen condition. This function is also used there Most of them has not been enclosed in functions to save time that is wasted in calling function (overheads). The input and values of ball_obj, PLAYER1 and PLAYER 2 objects change frequently for better response.*

DEVELOPER'S GUIDE

MAIN CPP FILE

There are 2 local functions:

1. *Main Menu -> which takes the user input in the form of string which is their names, and displays the information on the Command Line with the instructions about the GAME.*
2. *Those names are written into the file called **GAME_DATA.txt** and also who won the game**. (using FILE MANAGEMENT)***

```
void WRITE(string& P1, string& P2)
{
    fstream WRITE;
    WRITE.open("Game_Data.txt", ios::app);////GAME_DATA.txt file created if not already exist and later it is appeded
    {
        WRITE << "\nP1 = " << P1 << " vs " << " P2 = " << P2;
    }
    WRITE.close();//File Close
}


////////////////////////////////Game Menu//////////////////////////////////
    //VERY SIMPLE MENU TO nagivate to the GAME//

void MAIN_MENU()
{

    cout << "------------------------------------------------P O N G------------------------------------------------\n" << endl;

    cout << endl << "\tEnter your Name to PLAY->  ";
    string PLAYER_1; //Taking input for Name of the PLayer 1.
    getline(cin, PLAYER_1);
    cout << "\n\t" << PLAYER_1 << " is Player 1!";

    cout << endl << "\tThe Controls for " << PLAYER_1 << " are 'W' to MOVE UP and 'S' to MOVE DOWN" << endl;//Outputing some basic instruction

    cout << endl << "\tEnter the Name to PLAY AGAINST with->  ";
    string PLAYER_2; //Taking input for Name of the PLayer 2.
    getline(cin, PLAYER_2);
    cout << "\n\t" << PLAYER_2 << " is Player 2!";
    cout << endl << "\tThe Controls for " << PLAYER_2 << " are 'UP ARROW' to MOVE UP and 'DOWN ARROW' to MOVE DOWN" << endl << endl;//Outputing so

    cout << "\tWhosoever get 5 points first-> WINS " << endl << "\tPLayer 1 STARTS" << endl << endl; //GET 5 POINTS AND WIN THE GAME

    WRITE(PLAYER_1, PLAYER_2);
}
```

In main() function the Pong object is instantiated. CONSTRUCT_THE_CONSOLE is called to create the console and then START() function is called to start the GAME_THREAD.

DEVELOPER'S GUIDE

External solutions

```
////////////DRAW STRING FUNCTION taken from the INTERNET. As it is a Unicode we have a posibility of COLOR as well./
void WinCon::DRAW_STRING(int X, int Y, wstring STRING, short COLOR)
{
    for (size_t i = 0; i < STRING.size(); i++)
    {
        SCREEN_BUFFER[Y * SCREEN_WIDTH + X + i].Char.UnicodeChar = STRING[i];
        SCREEN_BUFFER[Y * SCREEN_WIDTH + X + i].Attributes = COLOR;
    }
}
```

This DRAW_STRING function is taken front the internet. It helps to print the string onto the SCREEN_BUFFER. Although this function is very simple but the printing strings onto the buffer was a tough task compared to drawing the shapes.

```
////////////////////////KeyBoard Input (Borrowed Code from internet)//////////
for (int i = 0; i < 256; i++)
{
    KEY_NEW_STATE[i] = GetAsyncKeyState(i);

    KEYS[i].Pressed = false;

    if (KEY_NEW_STATE[i] != KEY_OLD_STATE[i])
    {
        if (KEY_NEW_STATE[i] & 0x8000)
        {
            KEYS[i].Pressed = !KEYS[i].Held;
            KEYS[i].Held = true;
        }
        else
        {
            KEYS[i].Held = false;
        }
    }

    KEY_OLD_STATE[i] = KEY_NEW_STATE[i];
```

As we're not using the CONSOLE INPUT HANDLE so it's different than CONSOLE INPUT BUFFER.

```
HANDLE_CONSOLE_IN = GetStdHandle(STD_INPUT_HANDLE);
```

I used the above code which compares key states NEWER and OLDER states of keys when a key is pressed and when it is held.

DEVELOPER'S GUIDE

MASSIVE HELP FROM MICROSOFT DEVEOPER'S NETWORK (MSDN)

Some references underneath.

https://docs.microsoft.com/en-us/windows/console/console-input-buffer#keyboard-events

https://docs.microsoft.com/en-us/windows/console/creation-of-a-console

https://docs.microsoft.com/en-us/windows/console/getconsolescreenbufferinfo

https://docs.microsoft.com/en-us/windows/console/console-handles

https://docs.microsoft.com/en-us/windows/console/console-screen-buffers

GAME MAKING STYLE IS INSPIRED FROM:

https://community.onelonecoder.com/

THE CODE UNDERNEATH IS LEARNT FROM MSDN not really taken from the internet

```
////////////////////// Update Title & Present Screen Buffer////////////////////////////////////////
SetConsoleTitle(L"PONG");//Game Name

HWND console = GetConsoleWindow(); //GET THE HANDLE type HWND

//Set the toggle to no                                          E WINDOW
SetWindowLong(console,                                        e, GWL_STYLE) & ~WS_MAXIMIZEBOX & ~WS_SIZEBOX);

//SET THE WINDOW in LEFT-HAND corner of the MONITOR
MoveWindow(console, 100, 0, 990,690, TRUE);

//WRITE TO THE CONSOLE (EVERYTHING DONE SO FAR)
WriteConsoleOutput(HANDLE_CONSOLE_OUT, SCREEN_BUFFER, { (short)SCREEN_WIDTH, (short)SCREEN_HEIGHT }, { 0,0 }, &
```

HWND __stdcall GetConsoleWindow()
Search Online

A few very basic commands like:

***SetConsoleTitle** -> takes in a string.*

***HWND**-> is a window Handle*

***SetWindowLong**-> turns off the resize toggle as I want the GAME to be one proper shape.*

***MoveWindow**-> takes the handle and puts the windows 100 pixels from left and from top with 990*690 window size.*

***WriteConsoleOutput**t -> takes the CONSOLE OUTPUT HANDLE and prints everything that is on the SCREE_BUFFER.*

DEVELOPER'S GUIDE

Changes in project application form:

Most of the techniques has been used:

- *Objected Oriented Approach*
- *Multiple Inheritance*
- *Polymorphism*
- *Memory Management*
- *File Management*
- *Exception Handling*

Testing

I have multiple runs of the game but not even one time it went out of the bound or Buffer Overrun occurred. It shows no errors and not even one single Warning. It compiles correctly both in DEBUG and RELEASE mode. I have implemented Exception Handling as well to catch any error if there is any.

I have made destructor to deallocate all the memory that has been dynamically allocated to not have any memory leaks.

When the ball goes out of the bound then it is coordinates are reset and goes back to its initial position so again no buffer overrun.

The Collision sometimes is perfect and sometimes a bit off. The ball seldom times stops on the paddle for half a second or less due to the casting of floats to int.

The Game runs perfectly fine and compiles without any error. Although this game is Platform dependent only on Windows. UNICODE must be enabled with Visual Studio Community as the preferred Compiler.