UNIT TITLE: Game Development

Lesson 3:

Coverage Area

- C++ in the gaming industry
- Common math's topics for C++ game development
- Game Theory in Video Game Development
    - Minimax Algorithm Explanation to understand Tic-Tac-Toe Game
- Tic Tac Toe Game in C++ codes.

C++ in the gaming industry

- **C++ is recognized as the industry standard.** *It is also the language you need to learn if you are planning to program in Unreal Engine.* C#, on the other hand, is excellent for beginners and is fully supported by Unity Engine.

- Game developers have been building games with C++ for decades. including Windows, Mac, Linux, Android, and iOS.

- C++ is used in numerous 2D game engines and 3D game engines. **Godot and Unreal Engine.**

- The Unity game engine is written in C#, but its runtime language is C++.

- By now, many gaming APIs are written in C++. Its popularity is not the reason it's great for game development.

C++ has stark advantages for game development when it comes to ensuring high performance. High performance (i.e. minimum latency) is an essential requirement for game design.

C++ vs other common game programming languages...runtime in milliseconds

Programming language	Time (ms)
C++	1,129
Java	3,306
C#	10,797
Python	45,003

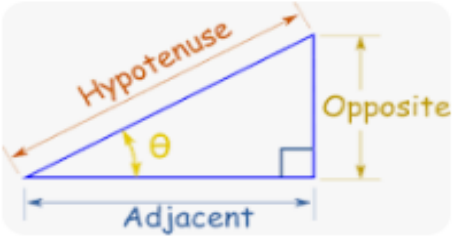
https://haslab.github.io/SAFER/scp21.pdf

some of the common branches of math utilized in game development include:

1. **Algebra** - Linear algebra is the study of vectors. If your game involves the position of an on-screen button, the direction of a camera, or the velocity of a race car, you will have to use vectors. *The better you understand linear algebra; the more control you will have over the behavior of these vectors.*

2. **Trigonometry**- Trigonometry evolved from the exigency to calculate the angles in various fields like astronomy, gaming, and artillery, etc. For this reason, *it is also called the study of relationships between lengths and angles of triangles.*

    - trigonometry has a cardinal role to play in game development. More specifically, sin, cos, and tan get widely used for gaming.



The most obvious use of trigonometry is to : get an object to move in any given direction, without trigonometry this is impossible.

```
//Example code, will move the object speed units in the given direction (degrees)
d2r = pi / 180; //Conversion from degrees to radians
this.x += speed * cos (direction * d2r);
this.y += speed *-sin (direction * d2r);
```

3. **Calculus** – calculus in physics engines

    - Modern video games use physics *to achieve realistic behavior and special effects.*
    - Initially the calculus of physics engines in video games was delegated to detecting collisions between in-game objects (such as player characters, rocks, and dust particles)
    - The integrator of a physics engine would take in information of an object at time *t* and apply that information to formulas in order to determine the new position/vector of said object.

Basic Integrator Formulas and other Determinants

Here are just some of the formulas used by a physics engine integrator:

Position

Derivative: $r(t)$

Integral: $r(t) = r_0 + \int_0^t v dt'$

Velocity

Derivative: $v(t) = \frac{dr}{dt}$

Integral: $v(t) = v_0 + \int_0^t a dt'$

Acceleration

Derivative: $a(t) = \frac{dv}{dt} = \frac{d^2r}{dt^2}$

Integral: $a(t)$

4. Linear Algebra
5. Discrete Mathematics
6. Applied Mathematics

More specific elements of math almost always used in games include:

1. Matrices
2. Delta time
3. Unit and scaling vectors
4. Dot and cross products

Game Theory in Video Game Development

- **Game theory** started *as a mathematical exploration of human behaviors*.
- Zero-sum game
  - If a player is winning, the other player is losing by the same "ämount"
- One of the clearest ways to apply game theory to game development is by *balancing the odds.*
- Game theory can help you design a fair game if the game involves decision-making.
  - Game theory can help players reach optimal decision-making *(one that maximizes the expected utility.)* when confronted by independent and competing actors in a strategic setting.

Minimax Algorithm Explanation to understand Tic-Tac-Toe Game

- In game theory, minimax is a decision rule *used to minimize the worst-case potential loss*
- Minimax is a recursive algorithm which is used to choose an optimal move for a player assuming that the other player is also playing optimally.
- It is used in games such *as tic-tac-toe, go, chess, Isola, checkers*, and many other two-player games.
- **Minimax Algorithm** (trying to find the best move to make)
  - Back tracking algorithm
  - Best move strategy used
  - Max will try to maximize its utility (Best Move)
  - Min will try to minimize its utility (worst move)

What are the rules of tic-tac-toe?

The game is played on a grid that's 3 squares by 3 squares.

1. You are X, your friend (or the computer in this case) is O. Players take turns putting their marks in empty squares.
2. The first player to get 3 of her marks in a row (up, down, across, or diagonally) is the winner.
3. When all 9 squares are full, the game is over.

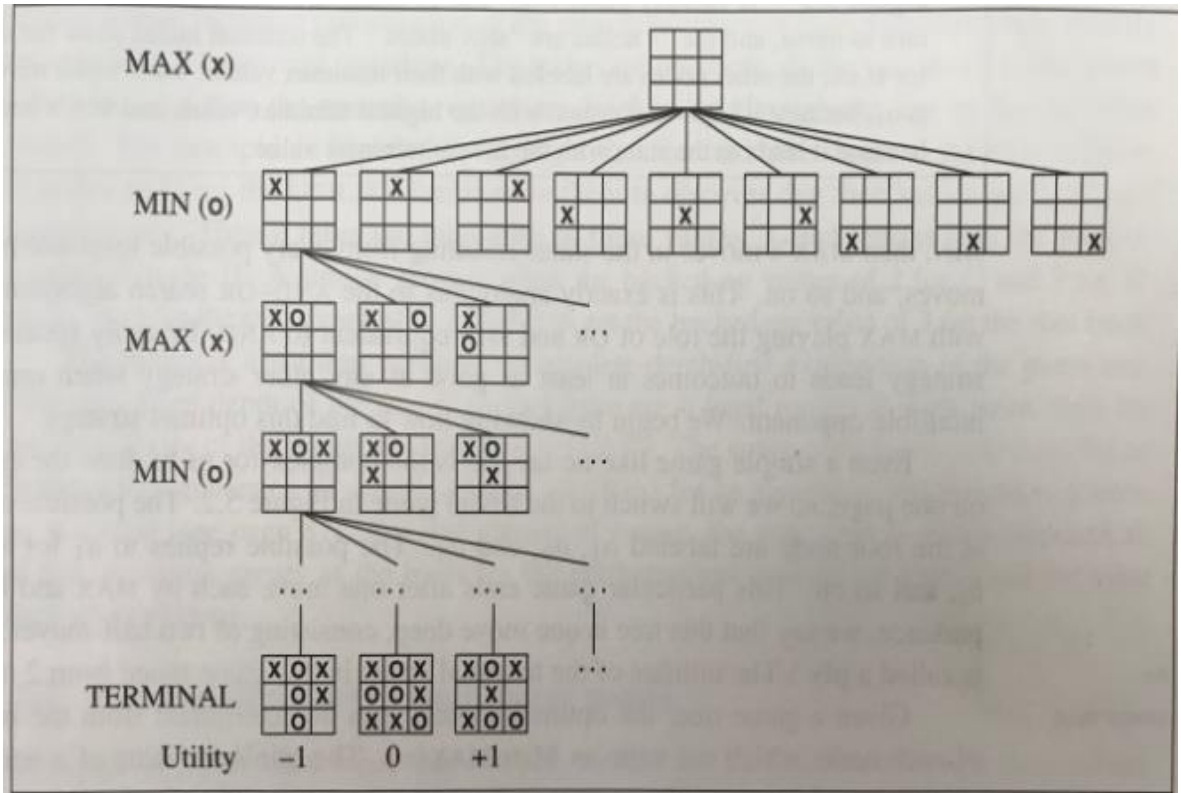
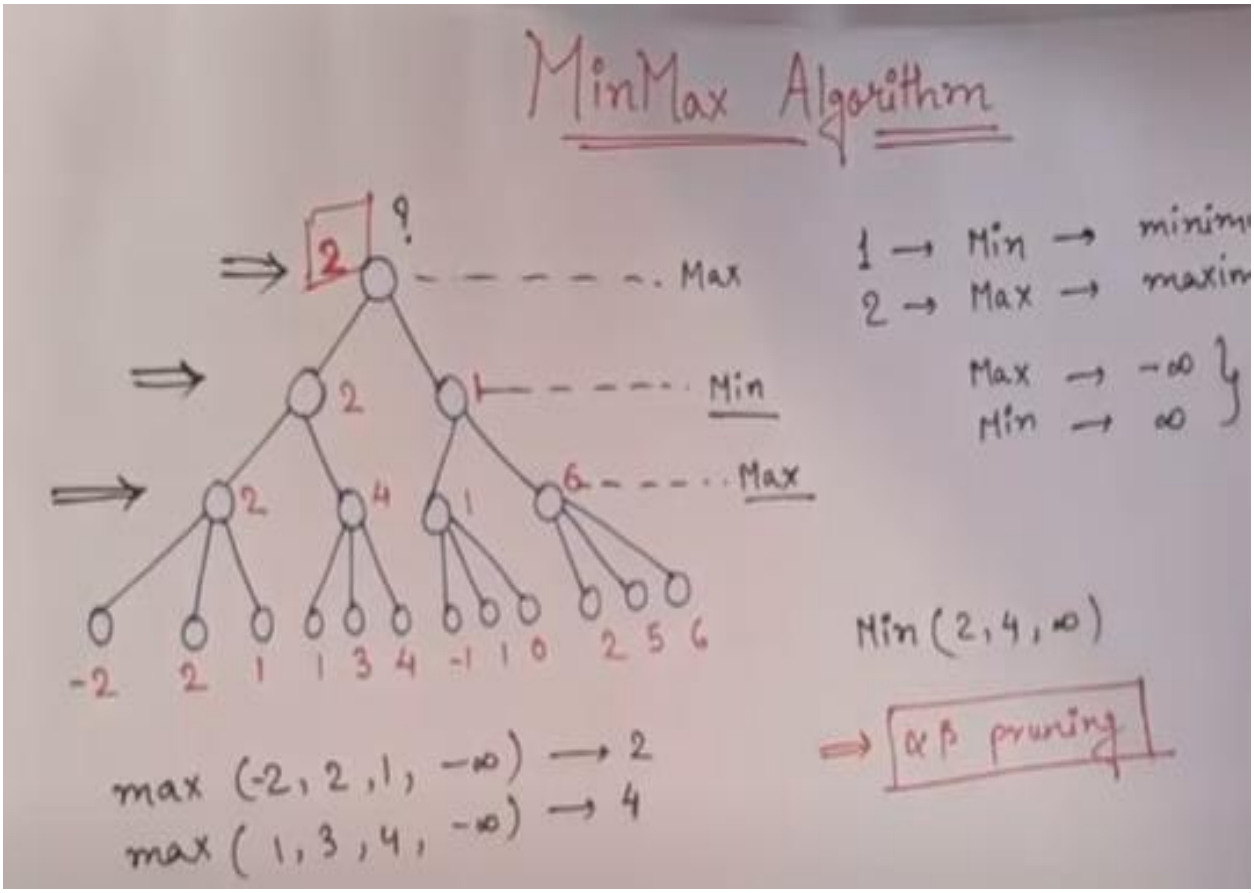


Figure 5.1 A (partial) game tree for the game of tic-tac-toe. The top node is the initial state, and MAX moves first, placing an X in an empty square. We show part of the tree, giving alternating moves by MIN (O) and MAX (X), until we eventually reach terminal states, which can be assigned utilities according to the rules of the game.

- For a win-or-lose game like chess or tic-tac-toe, **there are only three possible values**-win, lose, or draw (often assigned numeric values of 1, -1, and 0 respectively).
- A computer can compute all possible outcomes for a relatively simple game like tic-tac-toe (disregarding symmetric game states, there are 9! = 362,880 possible outcomes;
- the X player has a choice of 9 spaces, then the O has a choice of 8, etc.), but it won't help.



Tic Tac Toe Game in C++

```
//Importing the inbuild libraries in CPP
#include <iostream>
#include <stdlib.h>
using namespace std;
//Array for the board
char board[3][3] = {{'1','2','3'},{'4','5','6'},{'7','8','9'}};
//Variable Declaration
int choice;
int row,column;
char turn = 'X';
bool draw = false;

//Function to show the current status of the gaming board

void display_board(){
```

```
    //Rander Game Board LAYOUT

    cout<<"PLAYER - 1 [X]t PLAYER - 2 [O]nn";
    cout<<"tt     |     |     n";
    cout<<"tt  "<<board[0][0]<<"  | "<<board[0][1]<<"  |  "<<board[0][2]<<" n";
    cout<<"tt_____|_____|_____n";
    cout<<"tt     |     |     n";
    cout<<"tt  "<<board[1][0]<<"  | "<<board[1][1]<<"  |  "<<board[1][2]<<" n";
    cout<<"tt_____|_____|_____n";
    cout<<"tt     |     |     n";
    cout<<"tt  "<<board[2][0]<<"  | "<<board[2][1]<<"  |  "<<board[2][2]<<" n";
    cout<<"tt     |     |     n";
}

//Function to get the player input and update the board

void player_turn(){
    if(turn == 'X'){
        cout<<"ntPlayer - 1 [X] turn : ";
    }
    else if(turn == 'O'){
        cout<<"ntPlayer - 2 [O] turn : ";
    }
    //Taking input from user
    //updating the board according to choice and reassigning the turn Start

    cin>> choice;

    //switch case to get which row and column will be update

    switch(choice){
        case 1: row=0; column=0; break;
        case 2: row=0; column=1; break;
        case 3: row=0; column=2; break;
        case 4: row=1; column=0; break;
        case 5: row=1; column=1; break;
        case 6: row=1; column=2; break;
        case 7: row=2; column=0; break;
        case 8: row=2; column=1; break;
        case 9: row=2; column=2; break;
```

```
        default:
            cout<<"Invalid Move";
    }

    if(turn == 'X' && board[row][column] != 'X' && board[row][column] != 'O'){
        //updating the position for 'X' symbol if
        //it is not already occupied
        board[row][column] = 'X';
        turn = 'O';
    }else if(turn == 'O' && board[row][column] != 'X' && board[row][column] !=
'O'){
        //updating the position for 'O' symbol if
        //it is not already occupied
        board[row][column] = 'O';
        turn = 'X';
    }else {
        //if input position already filled
        cout<<"Box already filled!n Please choose another!!nn";
        player_turn();
    }
    /* Ends */
    display_board();
}

//Function to get the game status e.g. GAME WON, GAME DRAW GAME IN
CONTINUE MODE

bool gameover(){
    //checking the win for Simple Rows and Simple Column
    for(int i=0; i<3; i++)
    if(board[i][0] == board[i][1] && board[i][0] == board[i][2] || board[0][i] ==
board[1][i] && board[0][i] == board[2][i])
    return false;

    //checking the win for both diagonal

    if(board[0][0] == board[1][1] && board[0][0] == board[2][2] || board[0][2] ==
board[1][1] && board[0][2] == board[2][0])
    return false;
```

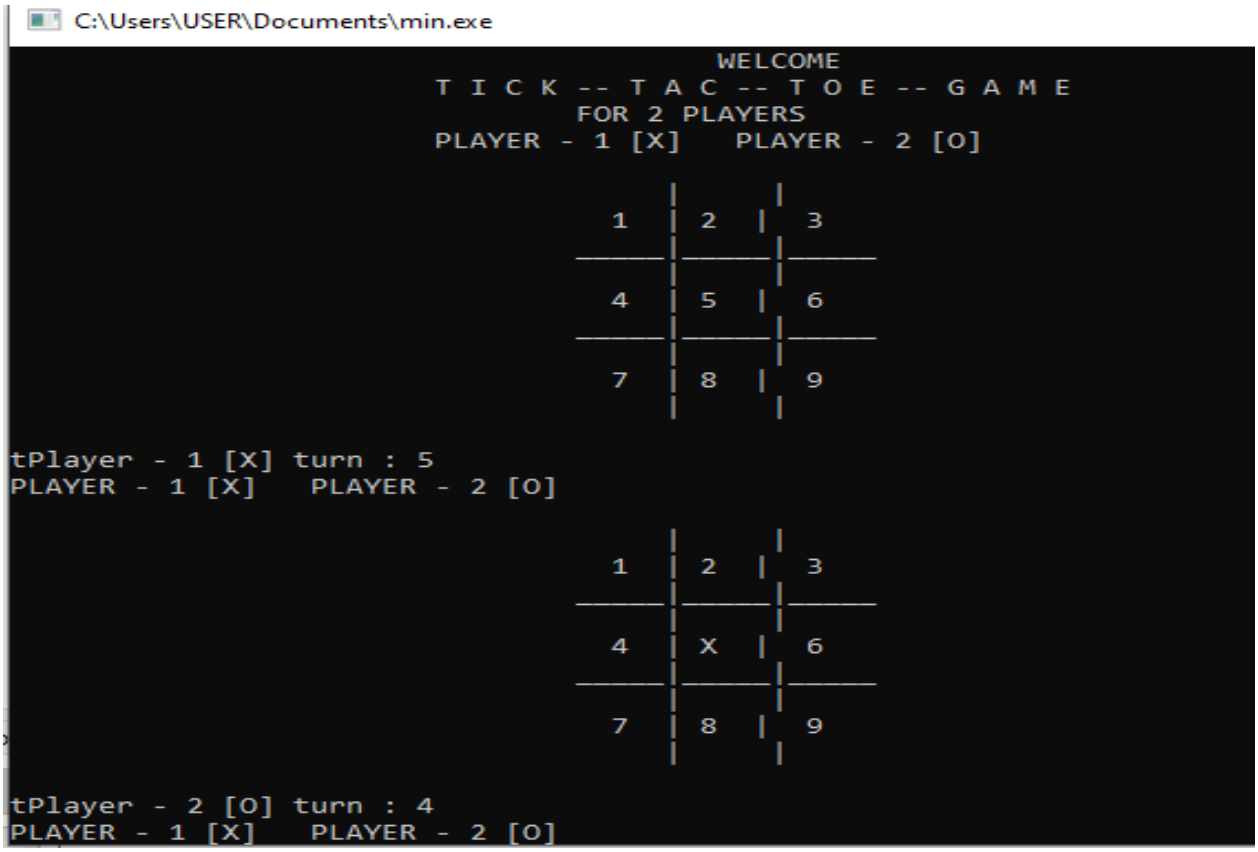
```
    //Checking the game is in continue mode or not
    for(int i=0; i<3; i++)
    for(int j=0; j<3; j++)
    if(board[i][j] != 'X' && board[i][j] != 'O')
    return true;

    //Checking the if game already draw
    draw = true;
    return false;
}

//Program Main Method

int main()
{
    cout<<"tttT I C K -- T A C -- T O E -- G A M Ettt";
    cout<<"nttttFOR 2 PLAYERSnttt";
    while(gameover()){
        display_board();
        player_turn();
        gameover();
    }
    if(turn == 'X' && draw == false){
        cout<<"nnCongratulations!Player with 'X' has won the game";
    }
    else if(turn == 'O' && draw == false){
        cout<<"nnCongratulations!Player with 'O' has won the game";
    }
    else
    cout<<"nnGAME DRAW!!!nn";
}
```

Output


C:\Users\USER\Documents\min.exe
WELCOME
T I C K -- T A C -- T O E -- G A M E
FOR 2 PLAYERS
PLAYER - 1 [X] PLAYER - 2 [O]
1 | 2 | 3
4 | 5 | 6
7 | 8 | 9
tPlayer - 1 [X] turn : 5
PLAYER - 1 [X] PLAYER - 2 [O]
1 | 2 | 3
4 | X | 6
7 | 8 | 9
tPlayer - 2 [O] turn : 4
PLAYER - 1 [X] PLAYER - 2 [O]