Non-Player Character Behavior in Computer Games

By Israel Irizarry

But first a little about my career goals...

Outline

- Introduction and History

—---

- Analysis of Finite State Machines
- Pathfinding and Movement
- Finite State Machine and Pathfinding Implementation

—---

- Conclusion
- Future Analysis

Introduction

Overall, this serves two main purposes in relation to NPC behavior: to educate about pathfinding and finite state machines through multiple examples and to inspire further NPC behavior development.

- What is a non-player character (NPC)?
- According to statista.com, revenue from the video game industry will grow by 7.17% annually between 2023 and 2027 worldwide [1].
- NPCs are very important to creating an immersive experience and challenging the player.

Introduction: What is used

- Python and C.
- Unity is a tool that helps game developers and students make video games.
- Supports Javascript and C# scripting using MonoDevelop.

Popular Games Made with Unity

- *Cuphead*
- *Hollow Knight*
- *Beat Saber*
- *Ori and the Blind Forest*
- *Among Us*

A graphic showing Unity’s Logo [2].


ori
AND THE BLIND FOREST

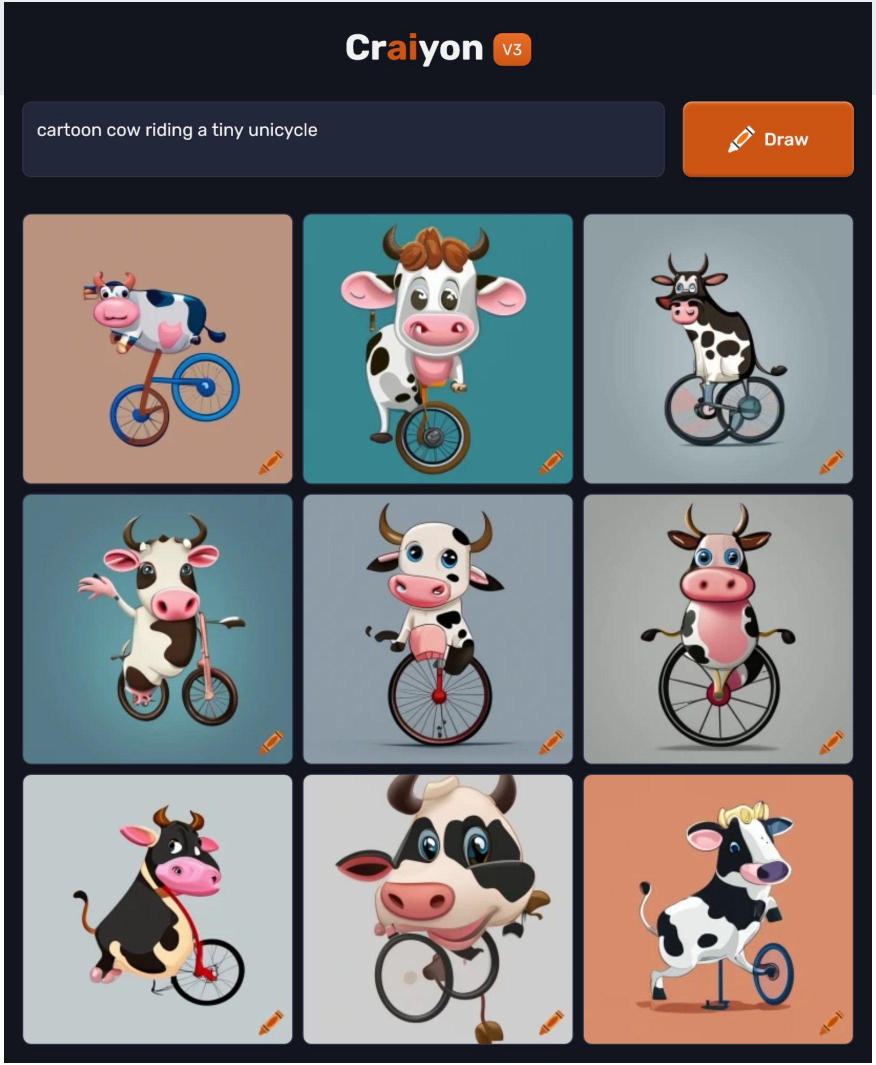
Early History

- The first few NPCs were not in video games, but they were in tabletop games.
- The very first video games had text-based and/or artist renderings to portray NPCs.
- With technological advances throughout the years, more resources meant that developers could make NPCs with more interesting behavior and movements.

Present and Future NPC behavior development

- Today, NPCs can easily deliver a lively feel to almost any game.
- The technological resources of today allow for complex behaviors.
- In the future, AI could help create NPCs that could respond to different characters in unique ways.

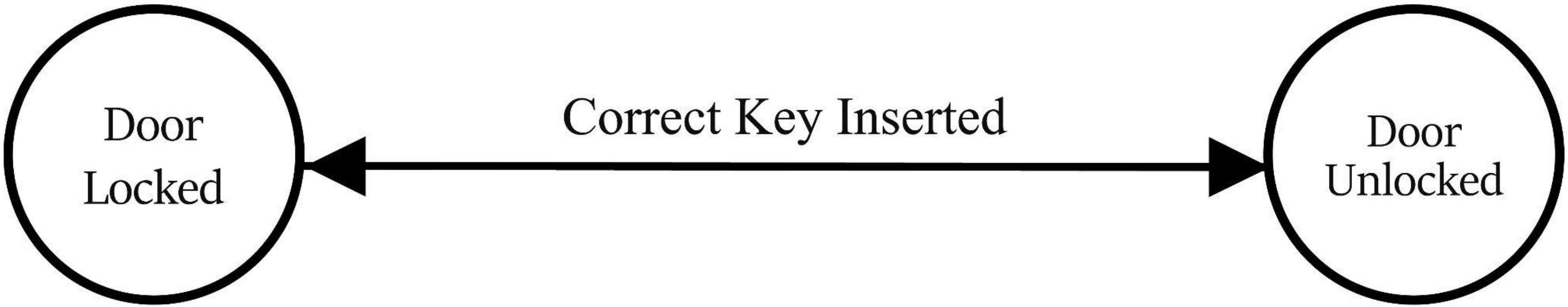
ChatGPT [3]


Craiyon V3
cartoon cow riding a tiny unicycle
Draw

Analysis of Finite State Machines

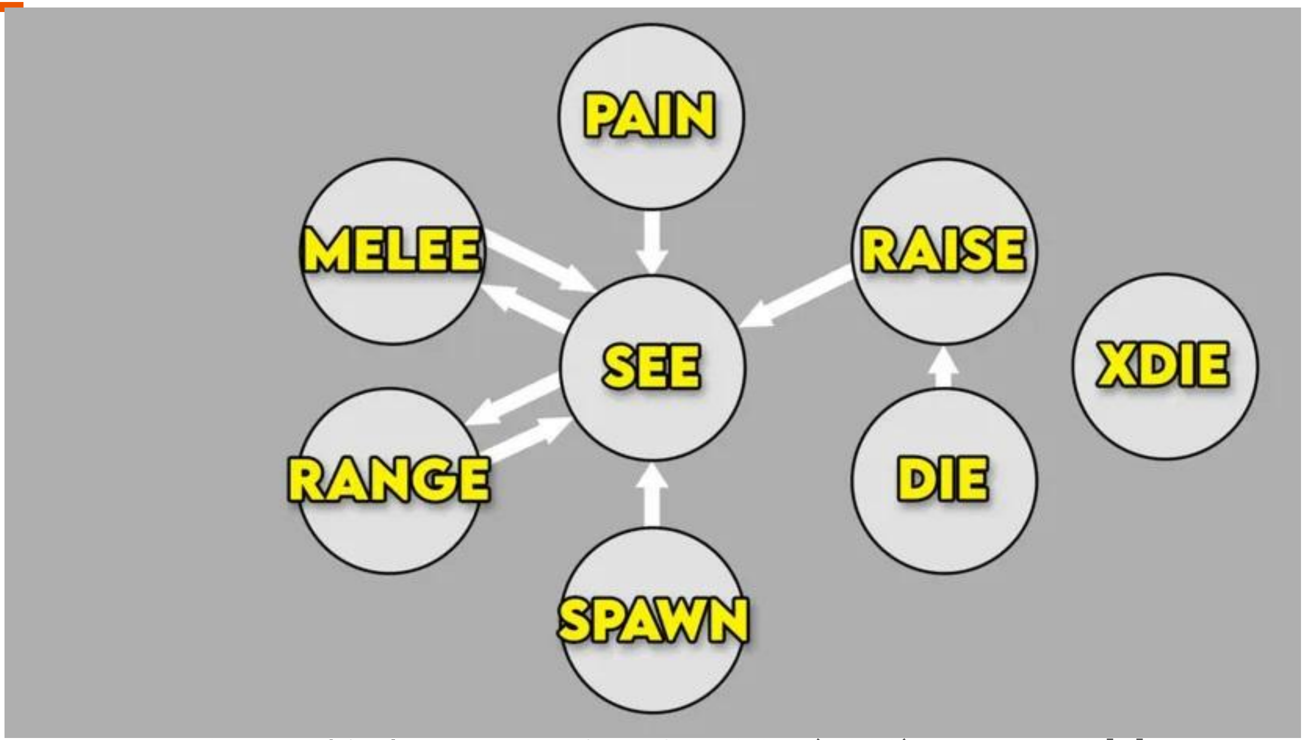
Finite State Machine Modeling

- Finite state machines (FSM) can be defined as the graphical modeling of different states of a program. In game development and more specifically NPC behavior development, finite state machines can dictate what an NPC does or says based on gameplay and player inputs.
- They contain nodes and edges called states and transitions.



DOOM, 1993 [13]

DOOM (1993)



A FSM graphical representation of a *DOOM* (1993) enemy NPC [5].

Player-NPC Interactment

- It is important to understand the player-NPC relationship to make better NPCs.
- When designing FSMs we need to see if it will satisfy the player through immersion and interesting behavior.
- The NPCs in *DOOM* (1993) help immerse the player by transitioning to many different states throughout each fight.
- Different NPC types also have different FSMs

Future NPC FSM Analysis

- As FSMs become more complicated, decisions trees could help them become more manageable.
- Trees, in general, are graphs containing nodes and edges where each node has exactly one input node except for the root node.
- Q-learning algorithms can help NPCs choose an optimal path.
- This could eliminate the need for static, handmade FSMs.

Pathfinding and Movement

This section analyzes a few ways NPCs can be programmed to move around a 2D or 3D gamespace.

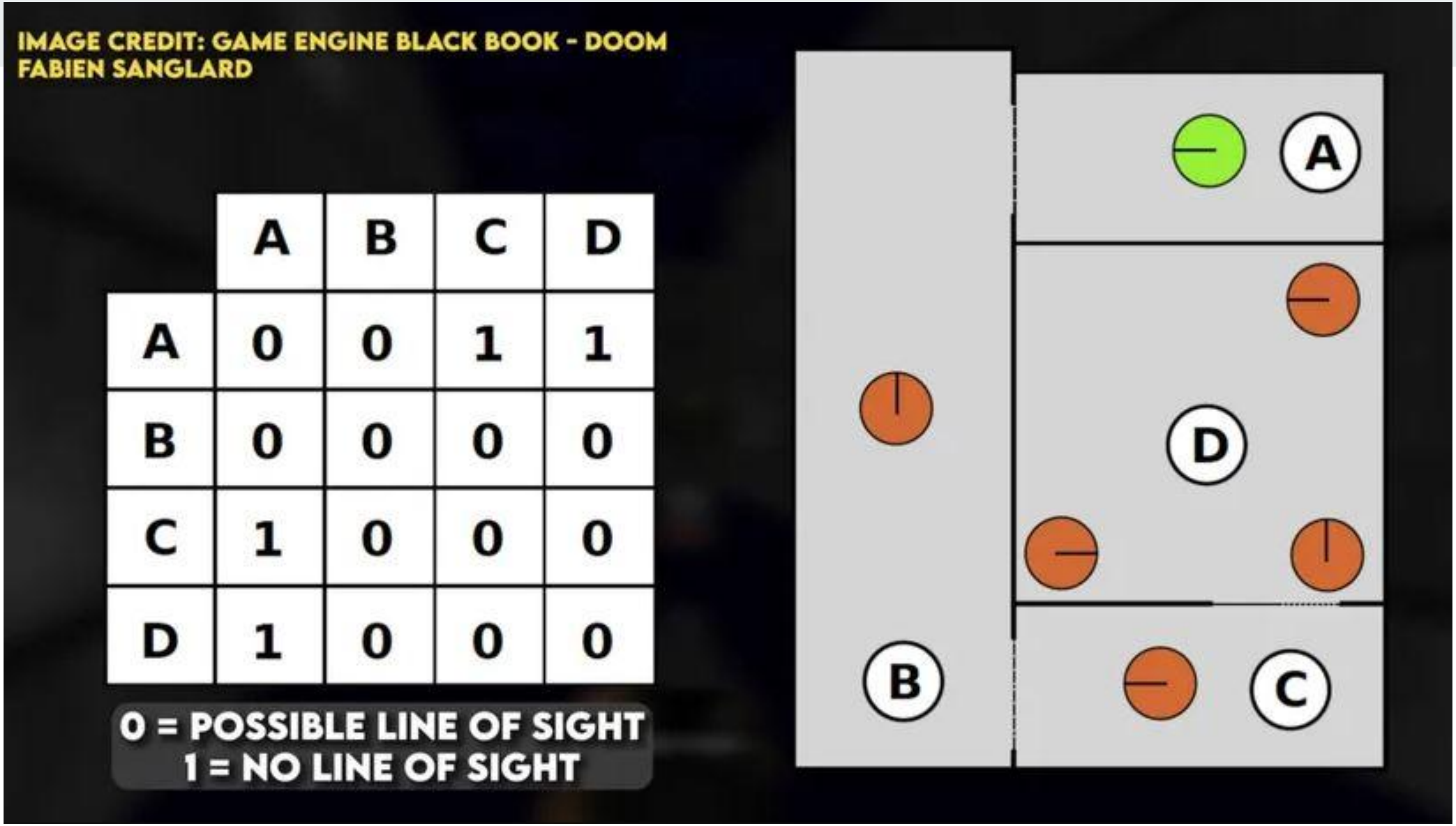
What is Pathfinding?

- Pathfinding in video games refers to how moving characters, NPC or not, find a path to a target location.
- For most games both modern and aged, pathfinding is used to add realistic behavior to NPCs, such as enemies [6].
- Different algorithms can be used to find the shortest path to a target or the path that cost the least (if weighted).

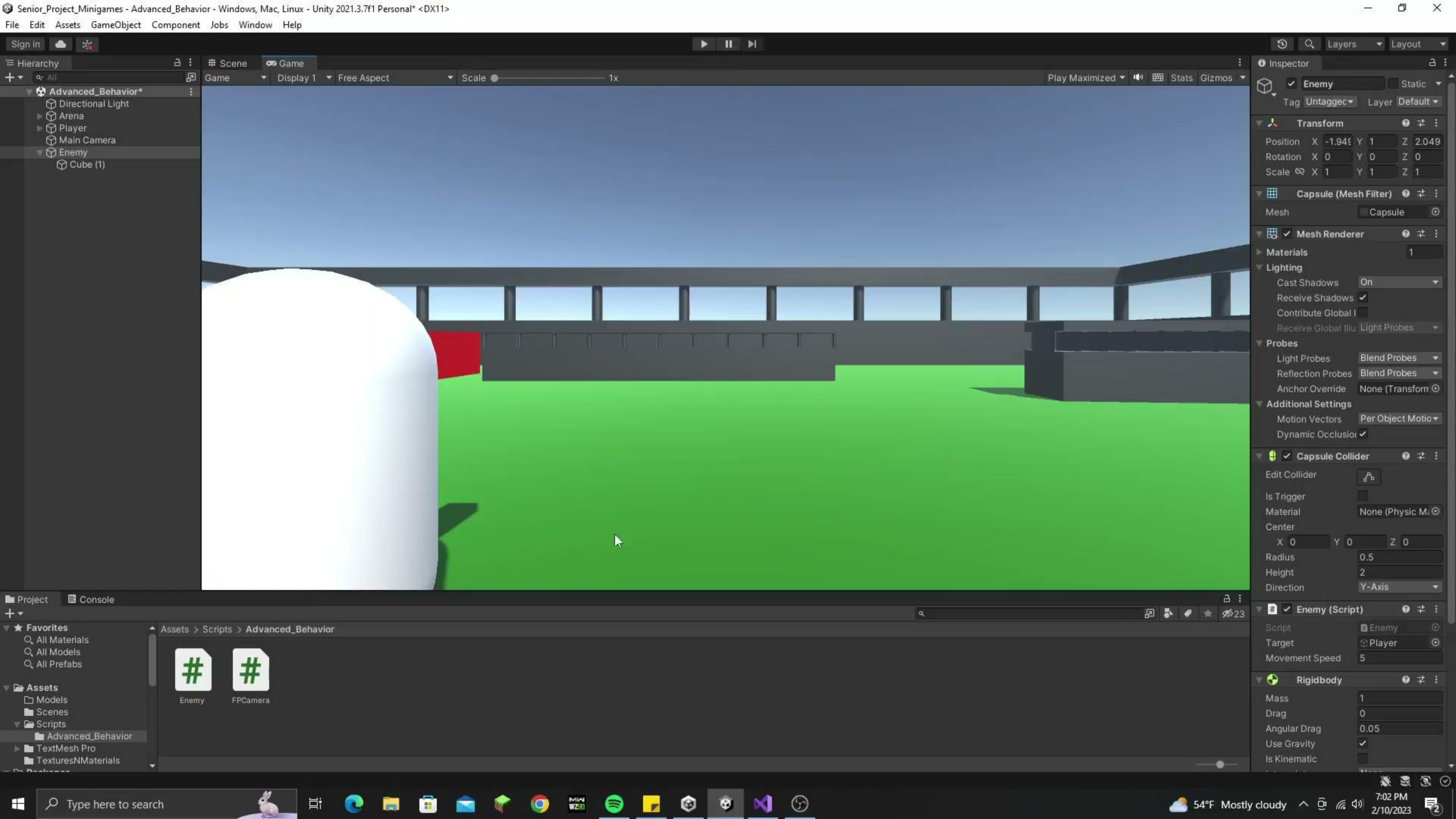
Basic Algorithmic Implementation for Pathfinding

- *DOOM* (1993) has a very basic pathfinding algorithm for their NPC movement.

Screenshot from *DOOM* (1993) [7].



A BLOCKMAP for a level in *DOOM* (1993) [7].

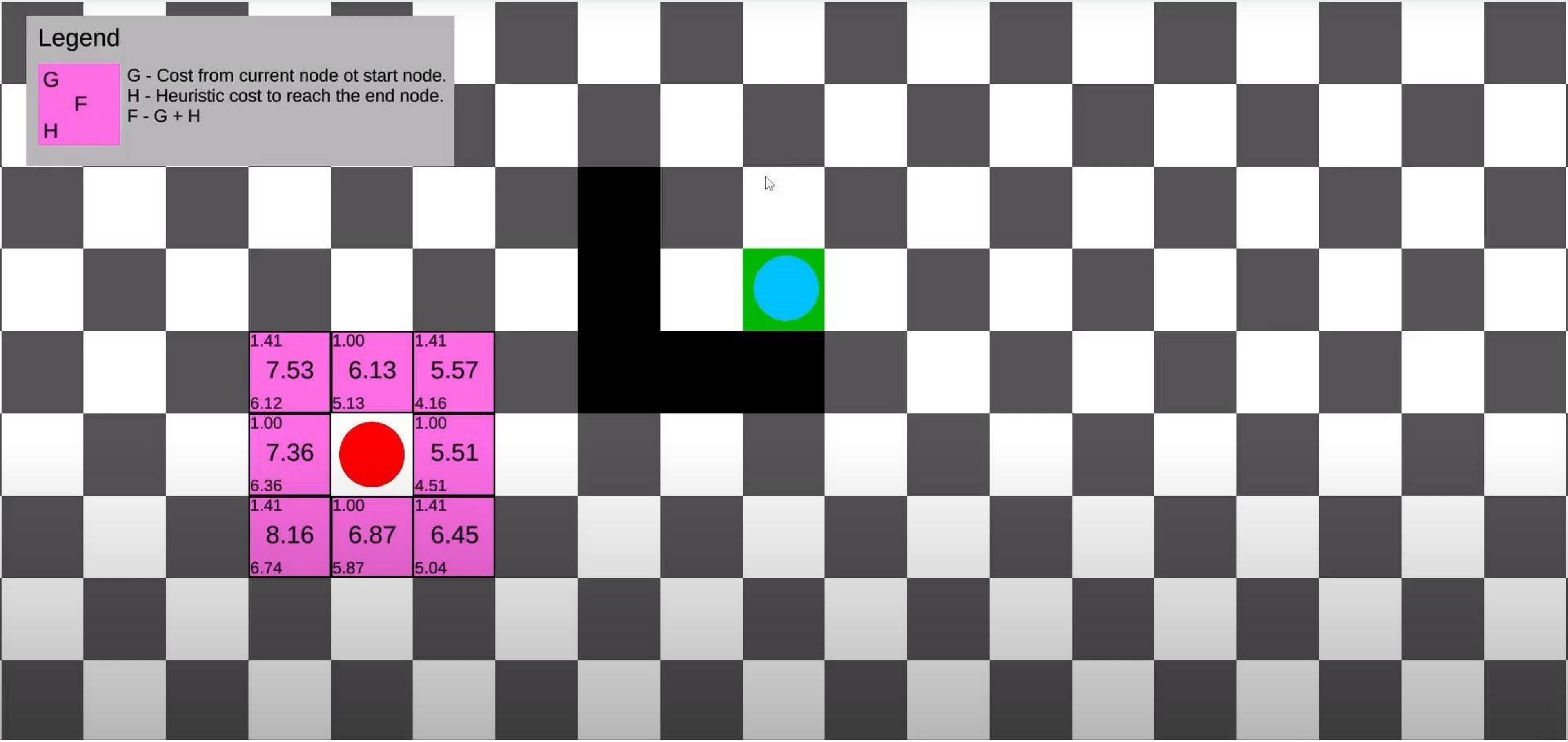

Senior_Project_Minigames - Advanced_Behavior - Windows, Mac, Linux - Unity 2021.3.7f1 Personal* <DX11>
File Edit Assets GameObject Component Jobs Window Help
Sign in
Layers Layout
Hierarchy
Scene Game
Game Display 1 Free Aspect Scale 1x
Play Maximized Stats Gizmos
Advanced_Behavior*
Directional Light
Arena
Player
Main Camera
Enemy
Cube (1)
Inspector
Enemy
Static
Tag Untagged Layer Default
Transform
Position X -1.949 Y 1 Z 2.049
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Capsule (Mesh Filter)
Mesh Capsule
Mesh Renderer
Materials 1
Lighting
Cast Shadows On
Receive Shadows
Contribute Global I
Receive Global Illu Light Probes
Probes
Light Probes Blend Probes
Reflection Probes Blend Probes
Anchor Override None (Transform
Additional Settings
Motion Vectors Per Object Motio
Dynamic Occlusion
Capsule Collider
Edit Collider
Is Trigger
Material None (Physic M
Center
X 0 Y 0 Z 0
Radius 0.5
Height 2
Direction Y-Axis
Enemy (Script)
Script Enemy
Target Player
Movement Speed 5
Rigidbody
Mass 1
Drag 0
Angular Drag 0.05
Use Gravity
Is Kinematic
Project Console
Favorites
All Materials
All Models
All Prefabs
Assets
Models
Scenes
Scripts
Advanced_Behavior
TextMesh Pro
TexturesNMaterials
Assets > Scripts > Advanced_Behavior
Enemy
FPCamera
23
Type here to search
54°F Mostly cloudy
7:02 PM
2/10/2023

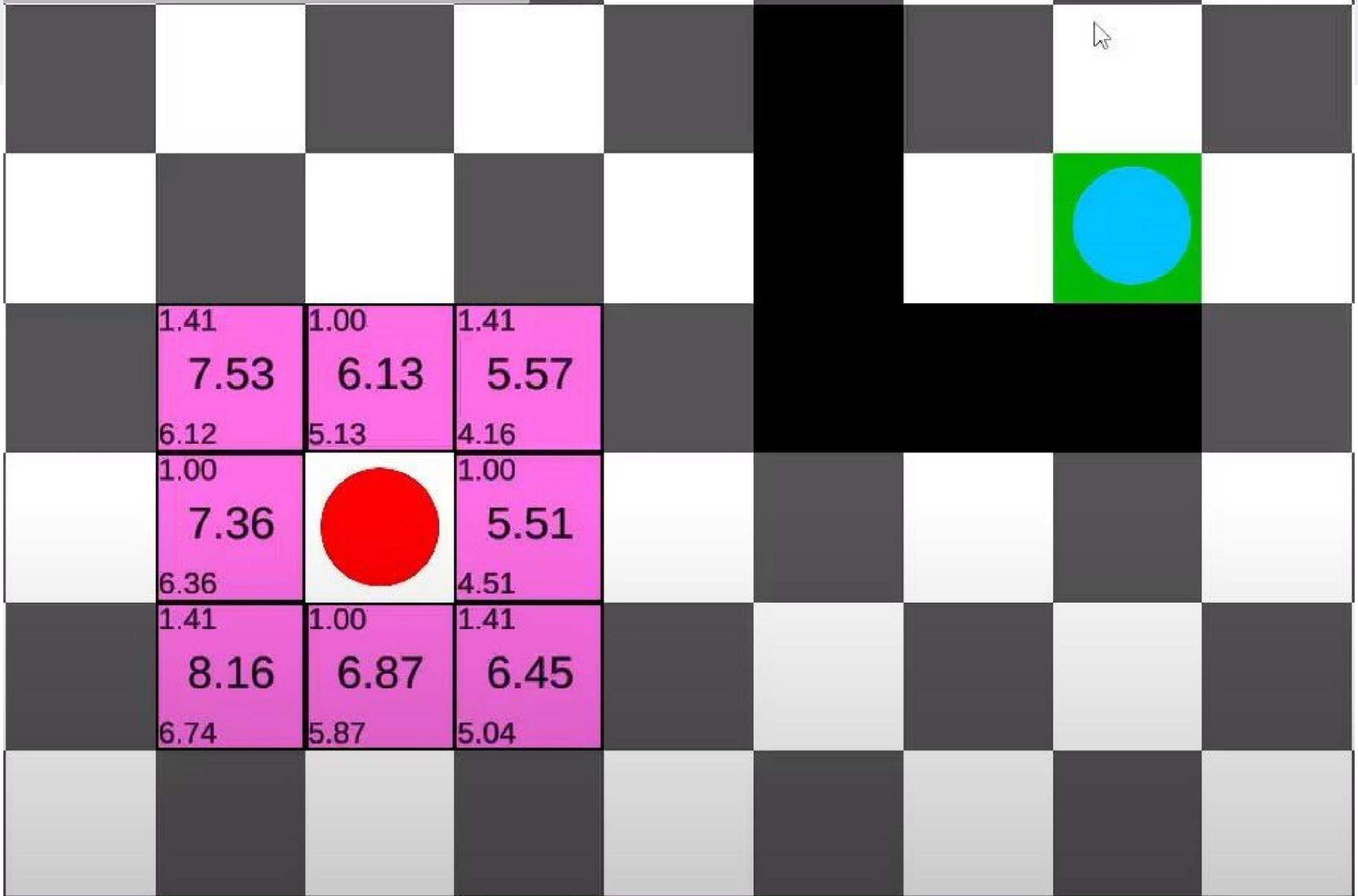
A* Pathfinding Algorithm

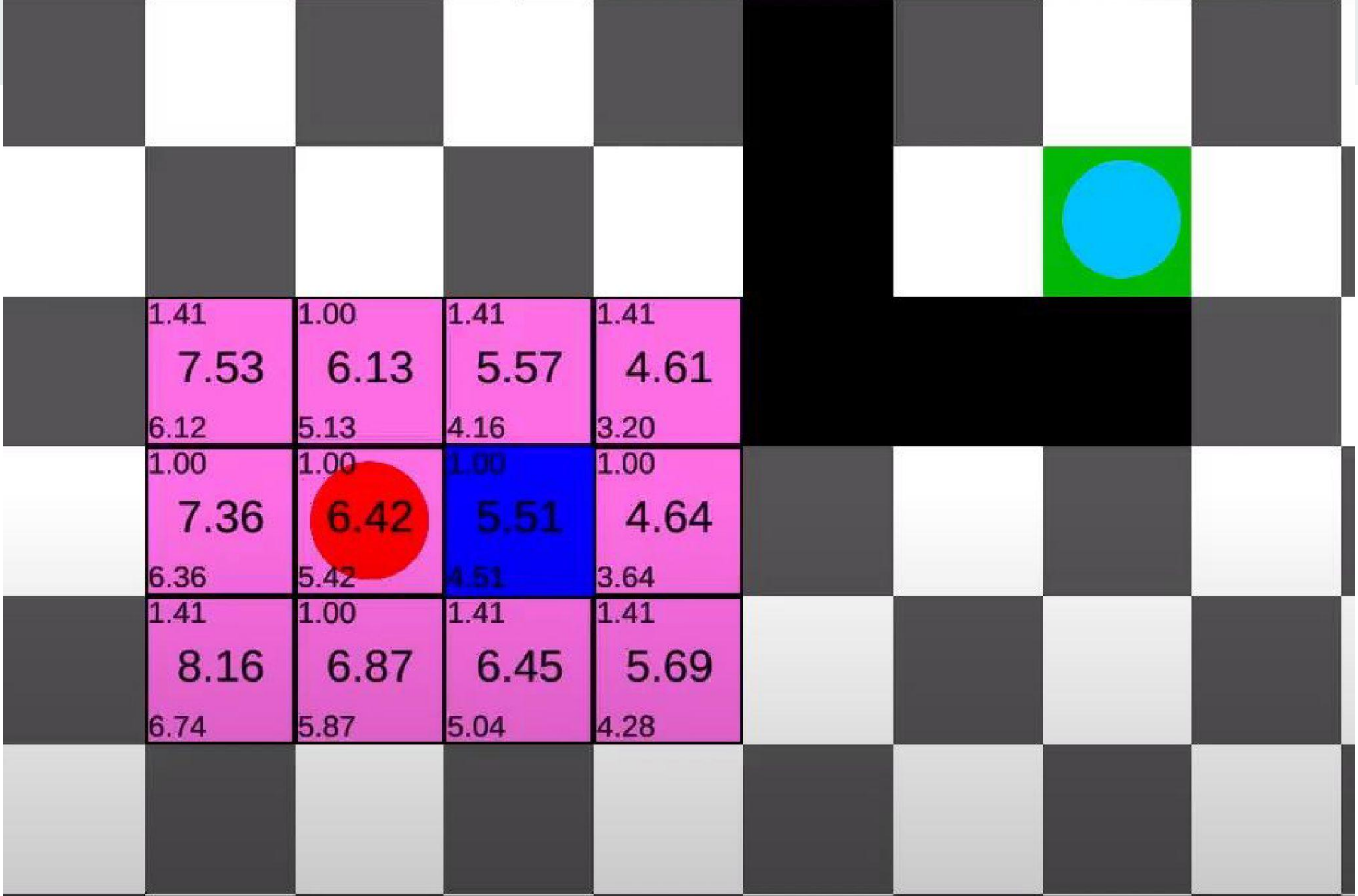
- The A* pathfinding algorithm is a dynamic algorithm that helps find the shortest path, either weighted or unweighted, to a given destination.
- A dynamic pathfinding algorithm allows for real-time pathfinding updating for moving or changing targets.

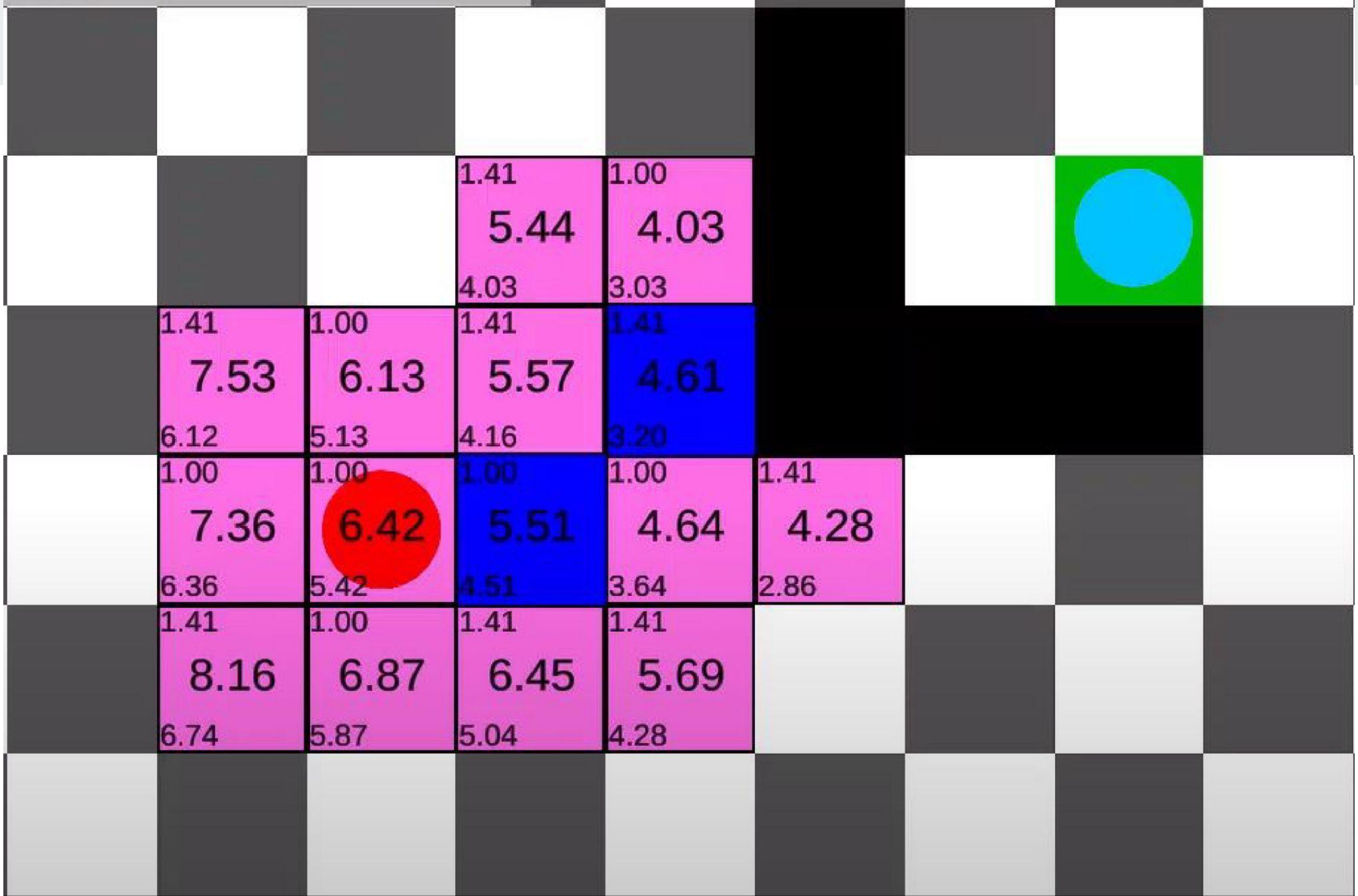
Algorithm for pathfinding through a 2D-grid plane:

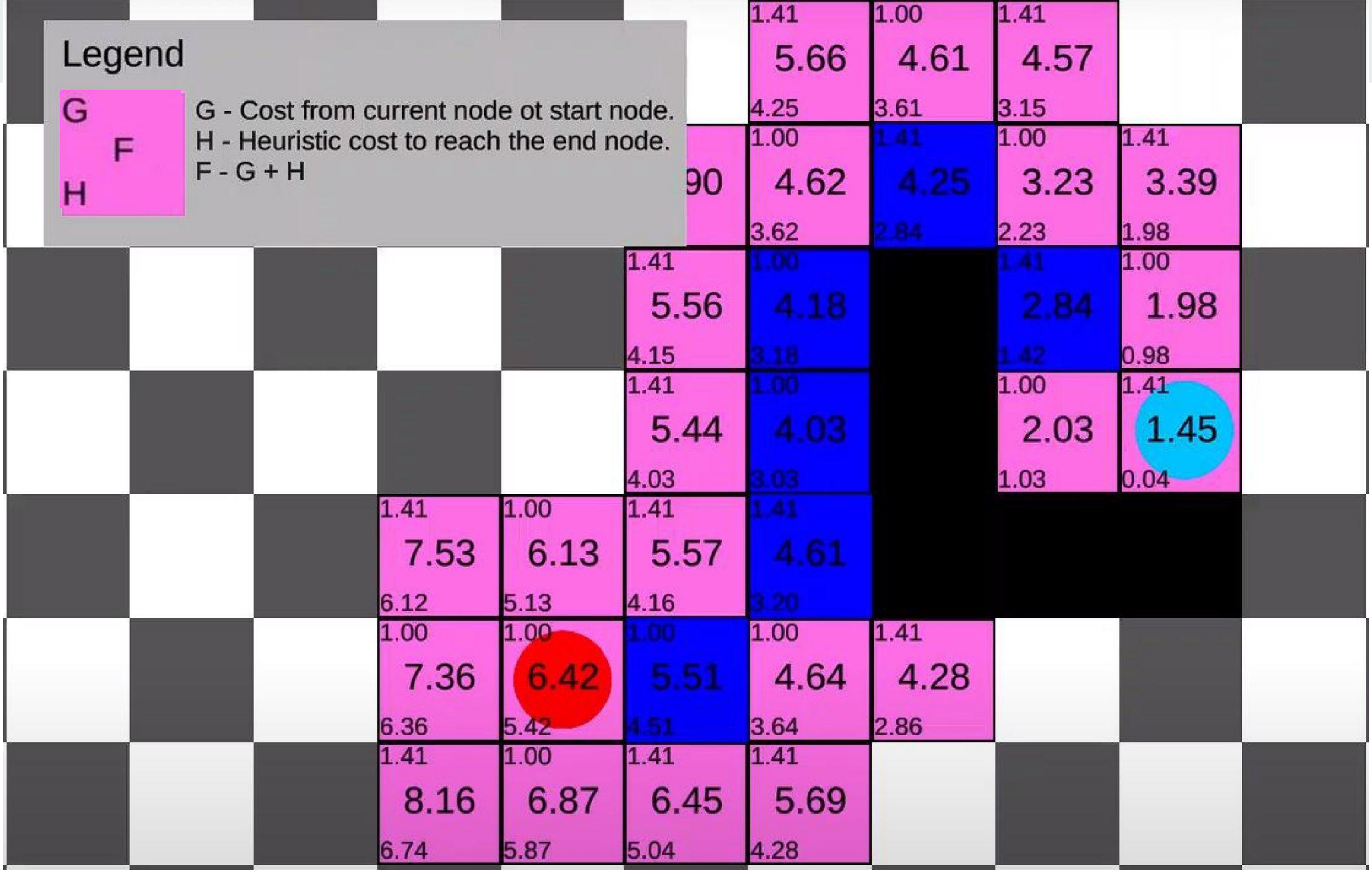
1. For every node neighboring or adjacent to the current node, record its distance from the current node via distance formula on a standard. This is going to be labeled G for any given float value.
2. For every node neighboring or adjacent to the current node, record its distance from the destination node. This is going to be labeled H for any given float value.
3. F is the sum of both G and H.
4. For every node neighboring or adjacent to the current node, choose the next node in the path to be the node with the lowest F value. That node will now be the current node. Repeat until the last node is the destination node.

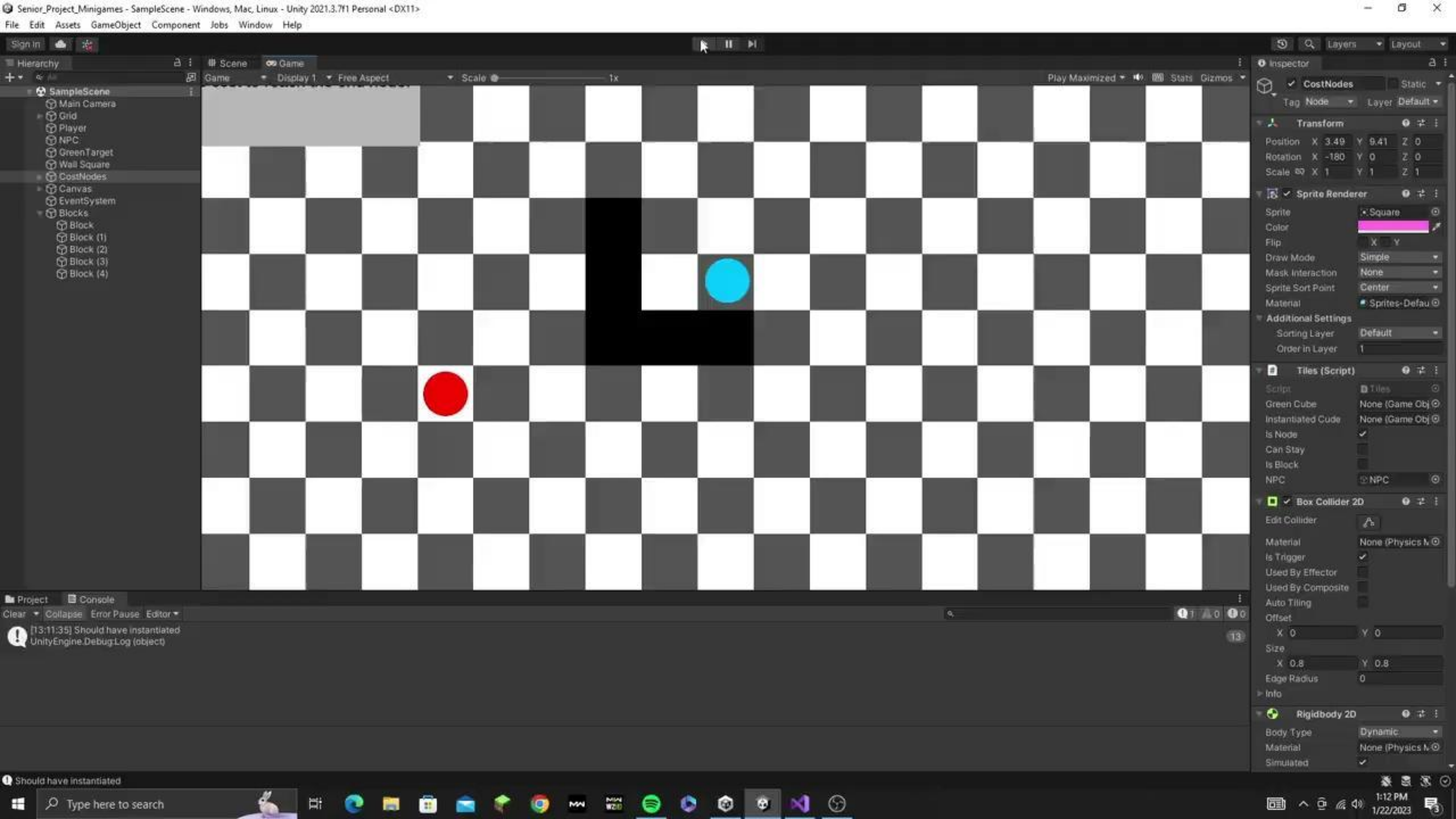

Legend
G
F
H
G - Cost from current node ot start node.
H - Heuristic cost to reach the end node.
F - G + H
1.41
7.53
6.12
1.00
6.13
5.13
1.41
5.57
4.16
1.00
7.36
6.36
1.00
5.51
4.51
1.41
8.16
6.74
1.00
6.87
5.87
1.41
6.45
5.04


1.41
7.53
6.12
1.00
6.13
5.13
1.41
5.57
4.16
1.00
7.36
6.36
1.00
5.51
4.51
1.41
8.16
6.74
1.00
6.87
5.87
1.41
6.45
5.04


1.41
7.53
6.12
1.00
6.13
5.13
1.41
5.57
4.16
1.41
4.61
3.20
1.00
7.36
6.36
1.00
6.42
5.42
1.00
5.51
4.51
1.00
4.64
3.64
1.41
8.16
6.74
1.00
6.87
5.87
1.41
6.45
5.04
1.41
5.69
4.28


1.41
5.44
4.03
1.00
4.03
3.03
1.41
7.53
6.12
1.00
6.13
5.13
1.41
5.57
4.16
1.41
4.61
3.20
1.00
7.36
6.36
1.00
6.42
5.42
1.00
5.51
4.51
1.00
4.64
3.64
1.41
4.28
2.86
1.41
8.16
6.74
1.00
6.87
5.87
1.41
6.45
5.04
1.41
5.69
4.28

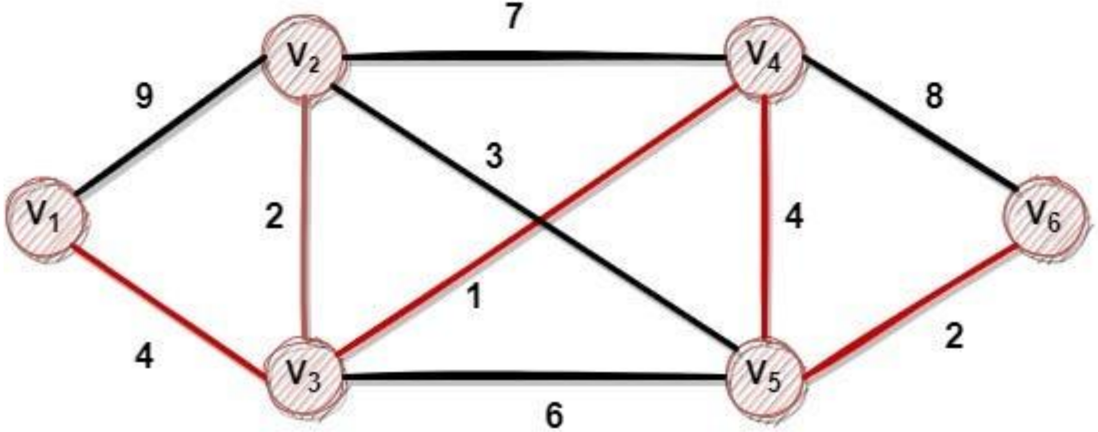

Legend
G
F
H
G - Cost from current node ot start node.
H - Heuristic cost to reach the end node.
F - G + H
1.41 5.66 4.25
1.00 4.61 3.61
1.41 4.57 3.15
90
1.00 4.62 3.62
1.41 4.25 2.84
1.00 3.23 2.23
1.41 3.39 1.98
1.41 5.56 4.15
1.00 4.18 3.18
1.41 2.84 1.42
1.00 1.98 0.98
1.41 5.44 4.03
1.00 4.03 3.03
1.00 2.03 1.03
1.41 1.45 0.04
1.41 7.53 6.12
1.00 6.13 5.13
1.41 5.57 4.16
1.41 4.61 3.20
1.00 7.36 6.36
1.00 6.42 5.42
1.00 5.51 4.51
1.00 4.64 3.64
1.41 4.28 2.86
1.41 8.16 6.74
1.00 6.87 5.87
1.41 6.45 5.04
1.41 5.69 4.28


Senior_Project_Minigames - SampleScene - Windows, Mac, Linux - Unity 2021.3.7f1 Personal <DX11>
File Edit Assets GameObject Component Jobs Window Help
Sign in
Layers Layout
Hierarchy
SampleScene
Main Camera
Grid
Player
NPC
GreenTarget
Wall Square
CostNodes
Canvas
EventSystem
Blocks
Block
Block (1)
Block (2)
Block (3)
Block (4)
Scene Game
Game Display 1 Free Aspect Scale 1x
Play Maximized Stats Gizmos
Inspector
CostNodes Static
Tag Node Layer Default
Transform
Position X 3.49 Y 9.41 Z 0
Rotation X -180 Y 0 Z 0
Scale X 1 Y 1 Z 1
Sprite Renderer
Sprite Square
Color
Flip X Y
Draw Mode Simple
Mask Interaction None
Sprite Sort Point Center
Material Sprites-Defau
Additional Settings
Sorting Layer Default
Order in Layer 1
Tiles (Script)
Script Tiles
Green Cube None (Game Obj
Instantiated Cude None (Game Obj
Is Node
Can Stay
Is Block
NPC NPC
Box Collider 2D
Edit Collider
Material None (Physics M
Is Trigger
Used By Effector
Used By Composite
Auto Tiling
Offset
X 0 Y 0
Size
X 0.8 Y 0.8
Edge Radius 0
Info
Rigidbody 2D
Body Type Dynamic
Material None (Physics M
Simulated
Project Console
Clear Collapse Error Pause Editor
[13:11:35] Should have instantiated
UnityEngine.Debug:Log (object)
13
Should have instantiated
Type here to search
1:12 PM
1/22/2023

Dijkstra’s Algorithm

- Dijkstra’s algorithm is an algorithm that finds the shortest path to any node in a weighted graph.
- Dijkstra’s algorithm is not a dynamic algorithm, so when implementing unrestricted NPC movement, it is not very useful in most applications.
- Some games, however, may benefit from using Dijkstra’s algorithm.

Dijkstra's Algorithm



	Distance	Previous Node	Visited
V_1	0	None	True
V_2	6	V3	True
V_3	4	V_1	True
V_4	5	V3	True
V_5	9	V4	True
V6	11	V5	True

Path: V_1 - V_3 - V_4 - V_5 - V_6

Total Cost: 11

Source #14

Finite State Machine and Pathfinding Implementation

This section serves to show different methods to implement FSMs into simulations and video games. This section will also mention the use of previously discussed pathfinding.

Chess

- In creating the following chess project, no common chess algorithm was used.
- The overall pathfinding used in this project was restricted to the traditional chess piece movement set except for a few special capture moves.

Lets take a look at the program...

Chess Project Algorithm (C) - Playerside

1. The player chooses a piece to move.
2. The player then chooses a destination for that piece.
3. If the destination is possible via the selected piece's movement rules, the piece is replaced with a space, and the piece's destination spot is filled with the piece's ASCII character.

```
        //If there still two kings on the board it proceeds with the game.
        if (kingCount == 2 && (pState == 1 || (pState == 2 && playerTurn == 1))) {

            //Gets input on what piece to move.
            printf("\nEnter piece to move: ");
            scanf("%s", input);
            x = input[1];
            y = input[0];
            x -= '0';

            //Gets input on where to move the piece.
            printf("Enter the destination location: ");
            scanf("%s", input);
            x1 = input[1];
            y1 = input[0];
            x1 -= '0';

            //Formatting.
            printf("___________________________________\n");
            for (int q = 0; q < 8; q++) {
                //Checks if the piece is part of the current player's team.
                for (int i = 0; i < 2; i++) {
                    for (int j = 0; j < 8;j++) {
                        if (xAxis[q] == y && playerTurn == 1 && chessBoard[8-x][q] == player1[i][j]) {
                            isYours += 1;
                        }
                        if (xAxis[q] == y && playerTurn == 2 && chessBoard[8-x][q] == player2[i][j]) {
                            isYours += 1;
                        }
                    }
                }
            }

```

```
//Checks if valid input was entered and places pieces in proper positions.
for (int q = 0; q < 8; q++) {

    //If the piece is part of the team, then the piece's position is replaced by a space.
    if ((xAxis[q] == y && x>0 && x<9) && isYours >= 1) {
        isValid += 1;
        qPos = q;
    }

    //Places the piece in its new position if the new position is within the rules.
    if ((xAxis[q] == y1 && x1>0 && x1<9) && isYours >= 1) {
        isValid += 1;
        qPos1 = q;
    }
}

//Prints an error message if invalid input was entered.
if (isValid != 2 && isYours != 0) {
    printf("***Incorrect format. You must enter a valid letter and number: ex. 'e7'.");
}

//Prints an error method if you try to move a piece that is not yours.
if (isYours == 0) {
    printf("***You have to choose a piece on your team and on the board.");
    if (playerTurn == 1) {
        printf("\nStill Player 1's turn.");
    }
    else {
        printf("\nStill Player 2's turn.");
    }
}
```

```
    //Changes the turn to the next player and prints it to screen.
    if (isYours >= 1 && isValid == 2) {
        if (chessBoard[8-x][qPos] == 'p' || chessBoard[8-x][qPos] == 'P') {
            canMove = _Pawn_(x, y, x1, y1, chessBoard, xAxis, player1, player2, playerTurn);
        }
        if (chessBoard[8-x][qPos] == 'r' || chessBoard[8-x][qPos] == 'R') {
            canMove = _Rook_(x, y, x1, y1, chessBoard, qPos, qPos1, player1, player2, playerTurn);
        }
        if (chessBoard[8-x][qPos] == 'b' || chessBoard[8-x][qPos] == 'B') {
            canMove = _Bishop_(x, y, x1, y1, chessBoard, qPos, qPos1, player1, player2, playerTurn);
        }
        if (chessBoard[8-x][qPos] == 'n' || chessBoard[8-x][qPos] == 'N') {
            canMove = _Knight_(x, y, x1, y1, chessBoard, qPos, qPos1, player1, player2, playerTurn);
        }
        if (chessBoard[8-x][qPos] == 'k' || chessBoard[8-x][qPos] == 'K') {
            canMove = _King_(x, y, x1, y1, chessBoard, qPos, qPos1, player1, player2, playerTurn);
        }
        if (chessBoard[8-x][qPos] == 'q' || chessBoard[8-x][qPos] == 'Q') {
            canMove = _Queen_(x, y, x1, y1, chessBoard, qPos, qPos1, player1, player2, playerTurn);
        }
        if (canMove == 1) {
            current = chessBoard[8-x][qPos];
            chessBoard[8-x][qPos] = ' ';
            chessBoard[8-x1][qPos1] = current;
            _ChessBoard_(chessBoard, xAxis);

            //If player is playing against the computer, than it changes to player 2.
            if (pState == 2) {
                playerTurn = 2;
            }
        }
        else {
            printf("***You cannot move this piece here.");
        }
    }
}
```

```
Would you rather play against another person (take turns on the same computer)
or play against the computer?
You can also have the program play against itself.

Enter 1 for PvP, 2 for PvC:, or 3 for CvC: 2

    ---------------------------------
8   | R | N | B | Q | K | B | N | R |
    ---------------------------------
7   | P | P | P | P | P | P | P | P |
    ---------------------------------
6   |   |   |   |   |   |   |   |   |
    ---------------------------------
5   |   |   |   |   |   |   |   |   |
    ---------------------------------
4   |   |   |   |   |   |   |   |   |
    ---------------------------------
3   |   |   |   |   |   |   |   |   |
    ---------------------------------
2   | p | p | p | p | p | p | p | p |
    ---------------------------------
1   | r | n | b | q | k | b | n | r |
    ---------------------------------
      a   b   c   d   e   f   g   h
Enter piece to move: a2
Enter the destination location: a4
```

```
   ---------------------------------
8  | R | N | B | Q | K | B | N | R |
   ---------------------------------
7  | P | P | P | P | P | P | P | P |
   ---------------------------------
6  |   |   |   |   |   |   |   |   |
   ---------------------------------
5  |   |   |   |   |   |   |   |   |
   ---------------------------------
4  | p |   |   |   |   |   |   |   |
   ---------------------------------
3  |   |   |   |   |   |   |   |   |
   ---------------------------------
2  |   | p | p | p | p | p | p | p |
   ---------------------------------
1  | r | n | b | q | k | b | n | r |
   ---------------------------------
     a   b   c   d   e   f   g   h
```

Chess Project Algorithm (C) - Computerside

1. Out of all of the computer's pieces, check if it can capture a player's piece.
2. If so, the piece is added to an array that contains all the pieces that can capture.
3. Choose a random piece out of that array and have it capture the first enemy piece it can capture.
4. If there are no pieces that can capture, choose a random piece that can move to a new position and have it move to the first position it can move to.

```
//Start of Computer playing portion--------------------------------|

//CvC computer rules for computer 1.
if (playerTurn == 1 && pState == 3) {

    //Delays the computer's response.
    printf("\nComputer 1 is thinking...");
    int milliSec = 1000 * 1;
    time_t startTime = clock();
    while(clock() < startTime + milliSec);
    printf("\n.\n");

    //Declares and initializes computer variables.
    int counter, stopSearch;
    counter = 0;
    stopSearch = 0;

    //Records all piece positions for the computer's side.
    for (int i = 0; i < 8; i++) {
        for (int j = 0; j < 8; j++) {
            for (int k = 0; k < 6; k++) {
                if (player1Pieces[k] == chessBoard[i][j]) {
                    player2CurPosX[counter] = 8-i;
                    player2CurPosY[counter] = xAxis[j];
                    counter += 1;
                }
            }
        }
    }
```

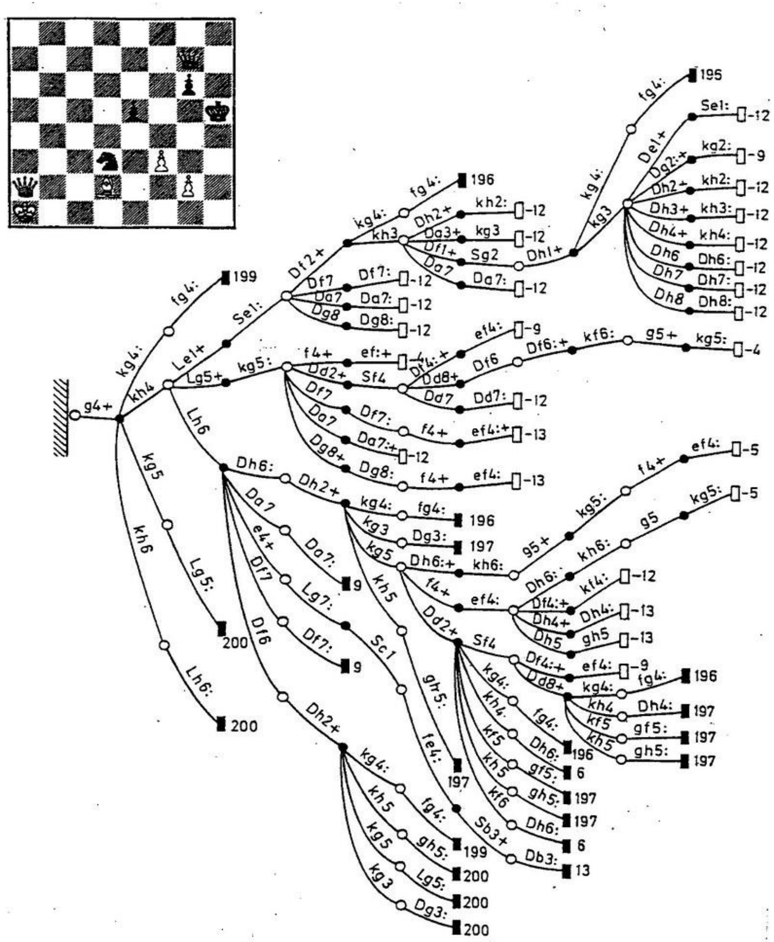
```
//For all the computer's pieces, check for every spot on the board if it can move to that spot.
for (int i = 0; i < 16; i++) {
    // x and y printf("%c%d, ", player2CurPosY[i], player2CurPosX[i]);

    for (int k = 0; k < 8; k++) {

        for (int j = 0; j < 8; j++) {
            // x1 and y1 printf("%c at %c%d", chessBoard[i][j], xAxis[j], 8-i);

            //Gets the numerical position for each letter variable.
            for (int q = 0; q < 8; q++) {
                if (xAxis[q] == player2CurPosY[i]) {
                    qPos = q;
                }
                if (xAxis[q] == xAxis[j]) {
                    qPos1 = q;
                }
            }

            //Checks if the computer can move the piece or not.
            if (chessBoard[8-player2CurPosX[i]][qPos] == 'p' && ((k-2 == player2CurPosX[i] && xAxis[qPos1] == xAxis[qPos])|| k-1 == player2CurPosX[i])) {
                canMove = _Pawn_(player2CurPosX[i], player2CurPosY[i], k, xAxis[j], chessBoard, xAxis, player1, player2, playerTurn);
            }
            if (chessBoard[8-player2CurPosX[i]][qPos] == 'r') {
                canMove = _Rook_(player2CurPosX[i], player2CurPosY[i], k, xAxis[j], chessBoard, qPos, qPos1, player1, player2, playerTurn);
            }
            if (chessBoard[8-player2CurPosX[i]][qPos] == 'b') {
                canMove = _Bishop_(player2CurPosX[i], player2CurPosY[i], k, xAxis[j], chessBoard, qPos, qPos1, player1, player2, playerTurn);
            }
            if (chessBoard[8-player2CurPosX[i]][qPos] == 'n') {
                canMove = _Knight_(player2CurPosX[i], player2CurPosY[i], k, xAxis[j], chessBoard, qPos, qPos1, player1, player2, playerTurn);
            }
            if (chessBoard[8-player2CurPosX[i]][qPos] == 'k') {
                canMove = _King_(player2CurPosX[i], player2CurPosY[i], k, xAxis[j], chessBoard, qPos, qPos1, player1, player2, playerTurn);
            }
            if (chessBoard[8-player2CurPosX[i]][qPos] == 'q') {
                canMove = _Queen_(player2CurPosX[i], player2CurPosY[i], k, xAxis[j], chessBoard, qPos, qPos1, player1, player2, playerTurn);
            }

```

```c
//Checks if a piece can capture.
for (int i = 0; i < 64; i++) {
    capturePos[i] = -1;
    if (player2Capture[i] != -1) {
        captureCount = 1;
    }
}

//If a piece can capture, Choose a random piece that can capture.
if (captureCount == 1) {
    for (int i = 0; i < 64; i++) {
        if (player2Capture[i] != -1) {
            capturePos[probCount] = i;
            probCount += 1;
        }
    }
    //Chooses one piece that can move and moves it to the new spot.
    srand((unsigned) time(&t));
    randCount = rand() % probCount;
    printf("___________________________________\n");
    printf("Computer 1 moved %c at %c%d to %c%d", chessBoard[8-player2PosX[capturePos[randCount]]][player2PosY[capturePos[randCount]]], xAxis[player2PosY[capturePos[randC
    chessBoard[8-player2DestX[capturePos[randCount]]][player2DestY[capturePos[randCount]]] = chessBoard[8-player2PosX[capturePos[randCount]]][player2PosY[capturePos[randC
    chessBoard[8-player2PosX[capturePos[randCount]]][player2PosY[capturePos[randCount]]] = ' ';
}

//If no pieces can capture, move a random piece.
if (captureCount == 0) {
    for (int i = 0; i < 64; i++) {
        if (player2PosY[i] != -1) {
            probCount += 1;
        }
    }
    //Chooses one piece that can move and moves it to the new spot.
    srand((unsigned) time(&t));
    randCount = rand() % probCount;
    printf("___________________________________\n");
    printf("Computer 1 moved %c at %c%d to %c%d", chessBoard[8-player2PosX[randCount]][player2PosY[randCount]], xAxis[player2PosY[randCount]], player2PosX[randCount], xA
    chessBoard[8-player2DestX[randCount]][player2DestY[randCount]] = chessBoard[8-player2PosX[randCount]][player2PosY[randCount]];
    chessBoard[8-player2PosX[randCount]][player2PosY[randCount]] = ' ';

}
```

```
The computer is thinking...
.

.

.
________________________________________________
The computer moved P at f7 to f5
    -----------------------------------------
8   | R | N | B | Q | K | B | N | R |
    -----------------------------------------
7   | P | P | P | P | P |   | P | P |
    -----------------------------------------
6   |   |   |   |   |   |   |   |   |
    -----------------------------------------
5   |   |   |   |   |   | P |   |   |
    -----------------------------------------
4   | p |   |   |   |   |   |   |   |
    -----------------------------------------
3   |   |   |   |   |   |   |   |   |
    -----------------------------------------
2   |   | p | p | p | p | p | p | p |
    -----------------------------------------
1   | r | n | b | q | k | b | n | r |
    -----------------------------------------
      a   b   c   d   e   f   g   h
```

Decision Tree Representation

- Decision trees can be used to represent large and complex FSMs. These FSMs can be used to make pathfinding decisions.
- Chess decision trees can be extremely large, so it is common to use decision trees [8].

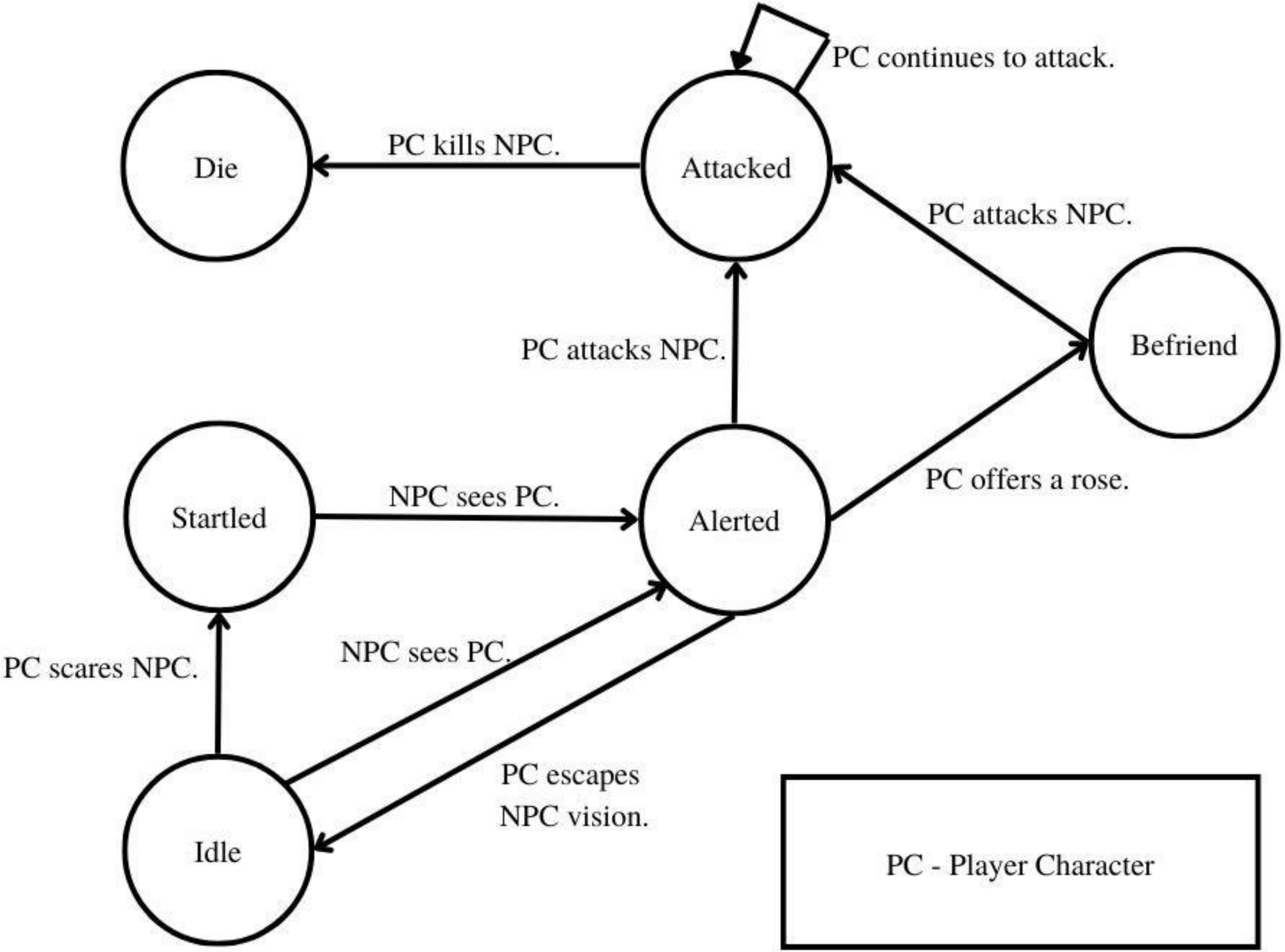


FSM and Pathfinding Combination

- A more organic movement that is controlled by random number generators (RNG) or seeds can produce realistic NPC movement.
- An NPC's pathfinding can tell an NPC where to go, and at the same time, an FSM can be run to conduct state transitions while the NPC is moving to the next node in the pathfinding algorithm.

FSM Example 1

- In creating an FSM, the states and transitions need to be clearly defined.
- It is also worth noting that for all states except for the starting and ending states, there must be at least one transition entering a node and/or one transition exiting a node for the state to be considered “reachable”.
- An unreachable node is not advised as it adds unnecessary complexity while not benefiting the FSM.


PC continues to attack.
Die
PC kills NPC.
Attacked
PC attacks NPC.
Befriend
PC attacks NPC.
Startled
NPC sees PC.
Alerted
PC offers a rose.
NPC sees PC.
PC scares NPC.
PC escapes
NPC vision.
Idle
PC - Player Character

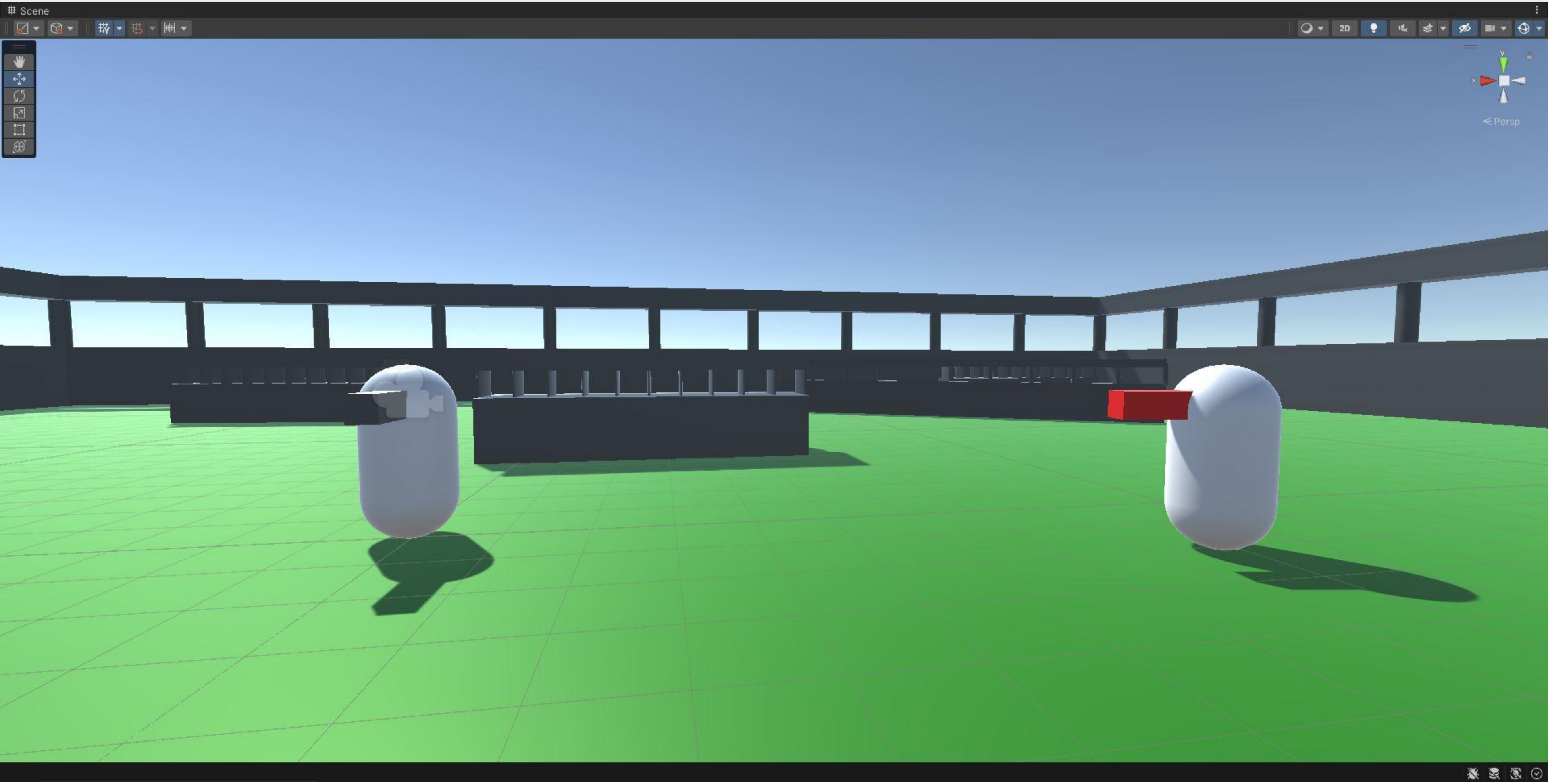
```
#Until the NPC dies, the FSM will continue to run.
    while not isDead:

            #Initial state
            if isIdle:
                print("Choose an option")
                print("1. Sneak behind the NPC and say boo!")
                print("2. Walk in front of the person.\n")
                answer = input("Enter a path number: ")

                #Transitions available to current state
                if (answer == '1'):
                    isStartled = True
                    isIdle = False

                if (answer == '2'):
                    isAlerted = True
                    isIdle = False

```

```
        #State available after IDLE state
        if isStartled:

            isStartled = False
            isAlerted = True

        #State available after IDLE or STARTLED state
        if isAlerted:
            answer = input("Enter a path number: ")

            #Transitions available to current state
            if answer == '1':
                isIdle = True
                isAlerted = False
            if answer == '2':
                print("ee")
                isMad = True
                isAlerted = False
            if answer == '3':
                isAlerted = False
```

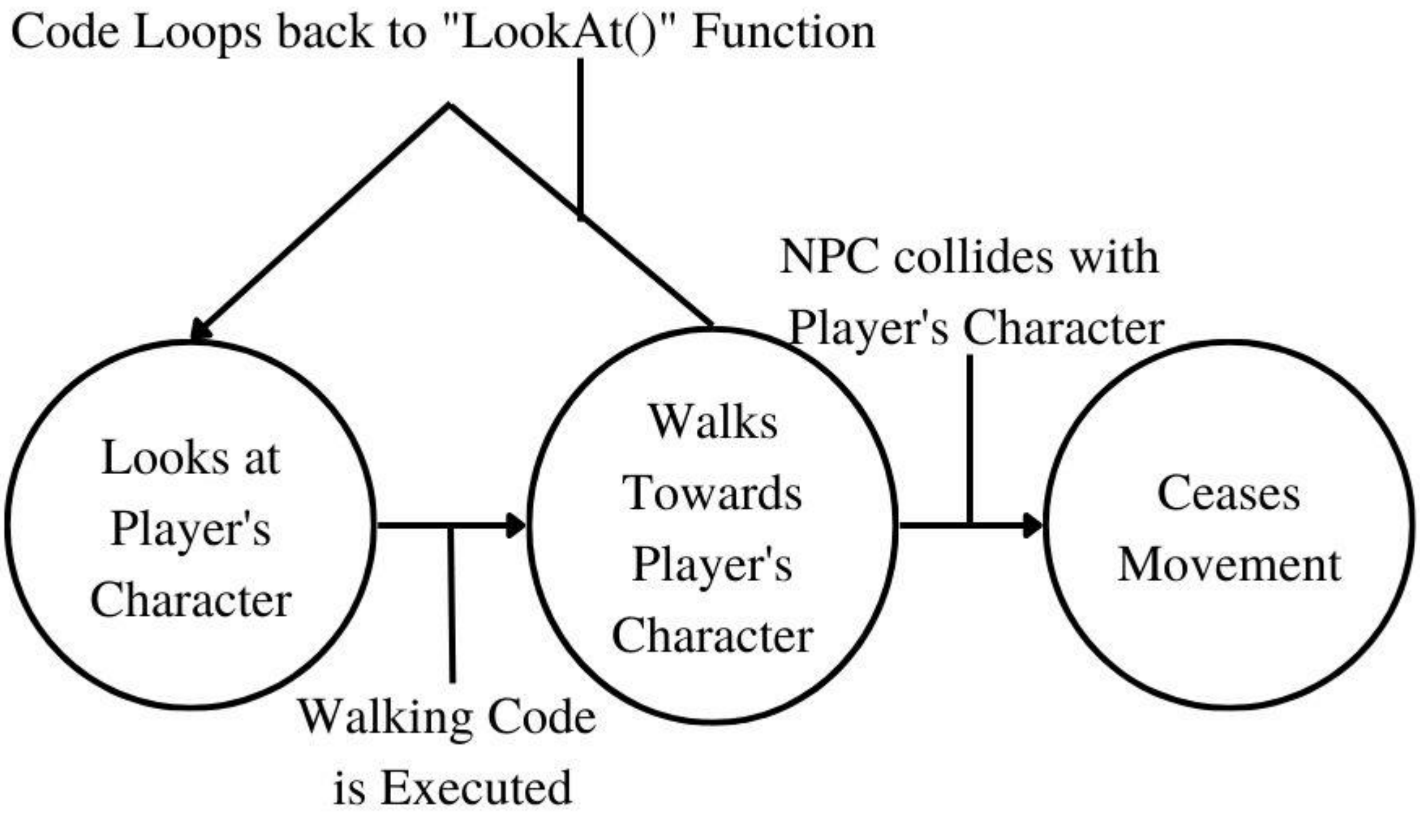
```
You see an NPC in the distance and walk up to it.

It has 3 health points
Choose an option
1. Sneak behind the NPC and say boo!
2. Walk in front of the person.

Enter a path number: 1
__________________________________________________________
        --------
     ---__    __---
    - _/        \_ -
   -   [ ]    [ ]   -
   -                -
   -       ____     -
    -    /    \    -
     ---  \__/  ---
        --------
NPC - OH! You scared me!
***The NPC has transitioned to the STARTLED state.***

__________________________________________________________
        --------
     ---__    __---
    -  /        \  -
   -   [ ]    [ ]   -
   -                -
   -       _____    -
    -     |_____|   -
     ---          ---
        --------
NPC - Hello there.
***The NPC has transitioned to the ALERTED state.***

Choose an option
1. Say nothing.
2. Attack.
3. Give rose.

Enter a path number:
```

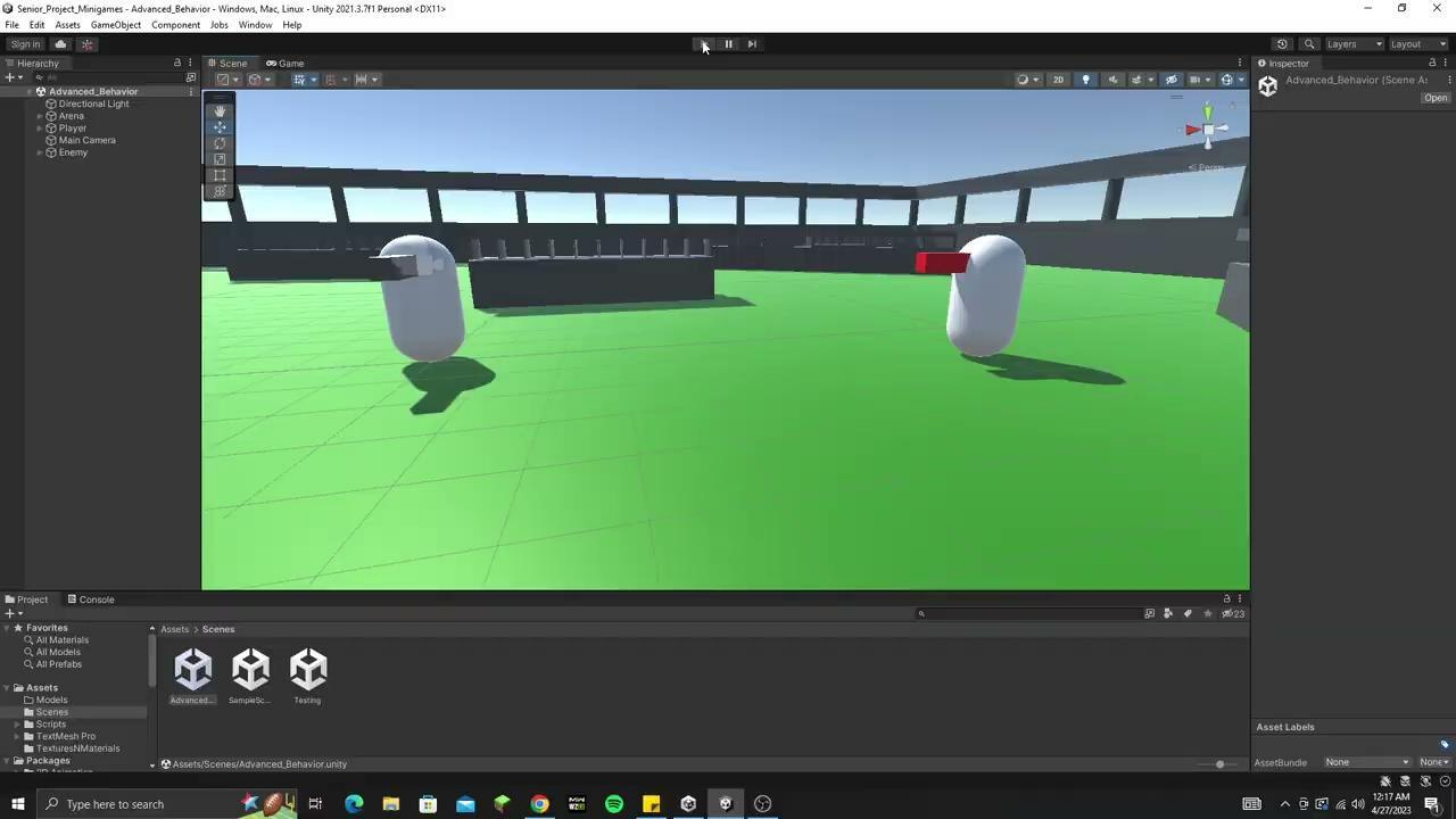
FSM Example 2

- In a three-dimensional implementation of an FSM using Unity, an NPC was tasked to follow the player's character throughout an arena with walls scattered about it.
- To create a more immersive environment, this example had the player controlling the player character from a first-person perspective. Similar to the NPC's design, the player's character was designed to look like the NPC if viewed from an outside angle.



Scene
2D
y
x
Persp


Code Loops back to "LookAt()" Function
NPC collides with Player's Character
Looks at Player's Character
Walks Towards Player's Character
Ceases Movement
Walking Code is Executed


Senior_Project_Minigames - Advanced_Behavior - Windows, Mac, Linux - Unity 2021.3.7f1 Personal <DX11>
File Edit Assets GameObject Component Jobs Window Help
Sign in
Layers
Layout
Hierarchy
Scene
Game
Inspector
Advanced_Behavior
Directional Light
Arena
Player
Main Camera
Enemy
Advanced_Behavior (Scene A:
Open
Project
Console
Favorites
All Materials
All Models
All Prefabs
Assets
Models
Scenes
Scripts
TextMesh Pro
TexturesNMaterials
Packages
Assets > Scenes
Advanced...
SampleSc...
Testing
Assets/Scenes/Advanced_Behavior.unity
Asset Labels
AssetBundle
None
None
Type here to search
12:17 AM
4/27/2023

Conclusion

- Pathfinding algorithms like A* and Dijkstra enable NPCs to move through the game world intelligently, improving the player's experience.
- FSMs allow game developers to model the behavior of NPCs in a structured way, allowing for intelligent reactions to different situations in the game world.
- Pathfinding and FSMs are crucial components for creating immersive and realistic video game NPCs.

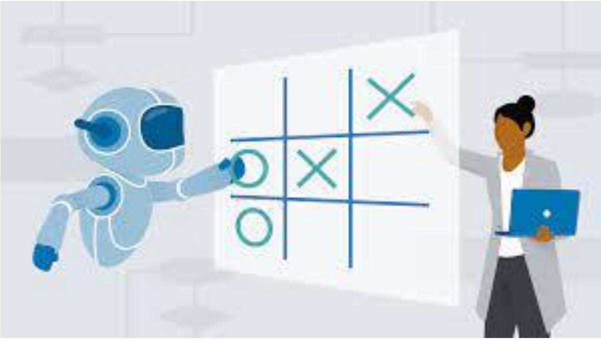
Future Analysis

- Uninteresting and robotic NPCs “have gained a reputation for lacking self-awareness and being awkward, dimwitted, or even annoying” [9]
- With further research on how NPCs can move around virtual worlds and how developers can use FSMs, future games could be made more efficient and introduce new NPC behavior mechanics.

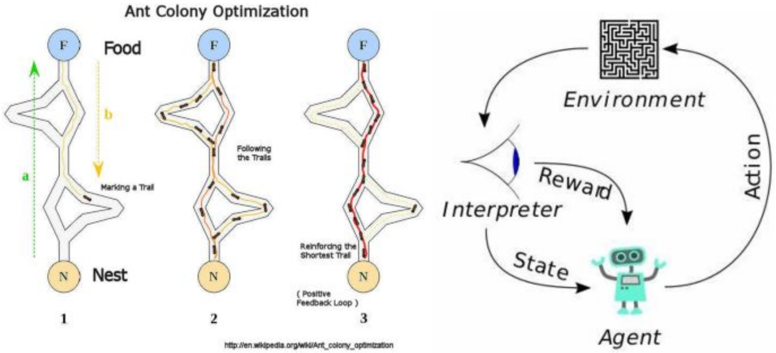
Unique NPC FSM design in *The Last of Us Part 2* [12].

Sons of the Forest - Source 15


236



AI could change the way NPCs are developed [10].



Ant Colony Optimization algorithm [11].

References

1. Video games - worldwide: Statista market forecast. Statista. (n.d.). Retrieved February 20, 2023, from https://www.statista.com/outlook/dmo/digital-media/video-games/worldwide
2. Wikimedia Foundation. (2023, April 11). *Unity (Game Engine)*. Wikipedia. Retrieved April 17, 2023, from https://en.wikipedia.org/wiki/Unity_%28game_engine%29
3. Lindzon, J. (2022, November 10). *OpenAI Dall-e 2: The 200 best inventions of 2022*. Time. Retrieved April 17, 2023, from https://time.com/collection/best-inventions-2022/6225486/dall-e-2/
4. Sanchez, V. (2023, March 12). *The origin and evolution of chat GPT: The natural language model that is changing the game*. RoutineHub Blog. Retrieved April 17, 2023, from https://blog.routinehub.co/origin-and-evolution-of-chat-gpt/
5. Tommy ThompsonBloggerMay 02, 2022. (2022, May 2). The Ai of Doom (1993). Game Developer. Retrieved March 14, 2023, from https://www.gamedeveloper.com/blogs/the-ai-of-doom-1993
6. Rafiq, A., Asmawaty Abdul Kadir, T., \& Normaziah Ihsan, S. (2020). Pathfinding algorithms in game development. IOP Conference Series: Materials Science and Engineering, 769(1), 012021. https://doi.org/10.1088/1757-899x/769/1/012021
7. Edwards, D. (2019, August 19). *Doom (1993) review - welcome to hell*. TheXboxHub. Retrieved April 18, 2023, from https://www.thexboxhub.com/doom-1993-review-welcome-to-hell/
8. Kevinbinz. (2017, June 27). Decision trees in chess. Fewer Lacunae. Retrieved March 25, 2023, from https://kevinbinz.com/2015/02/26/decision-trees-in-chess/
9. Inworld Team. (2022, October 21). What is an NPC? tracing the history (and future) of NPC meaning in games, memes, and Tik Tok. The developer platform for AI characters. Retrieved 30 April 14, 2023, from https://www.inworld.ai/blog/what-is-an-npc-tracing-the-history-and-future-of- npc-meaning-in-games-memes-and-tik-tok
10. Sidhwani, P. (2021, April 9). *The future of AI in gaming*. TechStory. Retrieved April 19, 2023, from https://techstory.in/the-future-of-ai-in-gaming/

11. Liu, Y., Cao, B., & Li, H. (2020, March 31). *Improving ant colony optimization algorithm with Epsilon greedy and Levy flight - complex & intelligent systems*. SpringerLink. Retrieved April 19, 2023, from https://link.springer.com/article/10.1007/s40747-020-00138-3

12. Manaloto, N. (2020, June 2). *How naughty dog made enemies smarter in the last of US 2*. UnGeek. Retrieved April 19, 2023, from https://www.ungeek.ph/2020/06/how-naughty-dog-made-enemies-smarter-in-the-last-of-us-2/

13.Doom 1993 wallpapers - Wallpaper Cave. WallpaperCave. (n.d.). Retrieved April 23, 2023, from https://wallpapercave.com/doom-1993-wallpapers

14. Soularidis, A. (2022, May 25). An introduction to dijkstra's algorithm: Theory and python implementation. Medium. Retrieved April 23, 2023, from https://python.plainenglish.io/dijkstras-algorithm-theory-and-python-implementation-c1135402c321

15. *Sons of the Forest release date, trailer, gameplay, and more*. (n.d.). The Loadout. Retrieved April 24, 2023, from https://www.theloadout.com/sons-of-the-forest/release-date

16. *Ori and the Blind Forest Trailer*. (n.d.). Www.youtube.com. https://www.youtube.com/watch?v=cklw-Yu3moE

Thank you all!

This wouldn't have been possible without the guidance and teachings of Dr. Nichols, Dr. Fuller, and all my professors I have had throughout my time here at Cumberland University.

Questions?