*/\**

*Further progress updates will be written here (much like what I did for my Frogger game)*

*4-23-24 further optimized the collision and velocity upon the ball hitting the player or enemy paddles,*

*now the ball will take into account the upper, center, and lower portions of each paddle. Still have to improve this.*

*4-22-24*

*tried new music tracks for the game and adjusted physics collision between paddes and ball.*

*keep working on collision code...*

*4-19-24*

*Added custom background images, custom paddles sprites, added copyright-free music audio, added*

*custom font for score counters displays.*

*4-18-24*

*added score counters for each player and changed player and AI collision code for when they collide with the ball.*

*4-17-24*

*Worked on enemy AI paddle game logic (how it responds when the ball comes near its goal)*

*and also worked on what happens to the ball once it hits the boundaries of the left or right*

*walls. Built custom functions for random number generation and what should happen when collision between ball and player occurs.*

*4-13-24*

*This is the Phaser Javascript code for the Pong clone created by Carls and Radon. I started*

*by creating the main window for the game (black background at present) and one paddle.  \*/*

*/\**

*The width and height properties set the size of the canvas element that Phaser will create.*

*In this case 1000 x 600 pixels. Your game world can be any size you like,*

*but this is the resolution the game will display in.*

*When you set gravity: { y: 0 }, it means there is no vertical gravity acting on*

*the object. It won’t fall or rise due to gravity. This can be useful for creating*

*scenarios where you want to simulate a zero-gravity environment*

*or where you manually handle the object’s movement without relying on gravity.*

*\*/*

var config = {

    type: Phaser.AUTO,

    width: 1000,

    height: 600,

    physics: {

        default: 'arcade',

        arcade: {

            gravity: { y: 0 },

            debug: false

        }

    },

    scene: {

        preload: preload,

        create: create,

        update: update

    }

};

*// game objects*

var playerPaddle;

var enemyPaddle;

var ball;

*// var platform; unused delete this later*

var playerScore = 0;

var playerScoreText;

var enemyScore = 0;

var enemyScoreText;

*/\* 4-23-24 consider using these variables in the UPDATE method and create separate getRndInteger variables in the collision methods*

*Why? because that way the X velocity for the ball upon hitting the player or enemy paddle will always change slighty*

*instead of staying at a fixed number (Despite using random) \*/*

var xVelocityBallPlayer = getRndInteger(700, 800);

var yVelocityBallPlayerList = [getRndInteger(0, 0), getRndInteger(-300, -400), getRndInteger(300, 400)];

var xVelocityBallEnemy = getRndInteger(-700, -800);

var yVelocityBallEnemyList = [getRndInteger(0, 0), getRndInteger(-300, -400), getRndInteger(300, 400)];

var game = new Phaser.Game(config);

/\*\*  About each function:

 \* preload() {} — a method that defines what we need to load before the scene and from where. We’ll use it to load assets later on.

create(data) {} — a method that gets triggered when a scene is created. In it, we’ll specify positioning for

such scene elements as Character and Enemies.

update(time, delta) {} — a method that gets called with every render frame (on average, 60 times per second).

It’s a game loop in which redrawing, moving objects, etc. occurs.

 \*/

function preload() {

*/\**

*The order in which game objects are displayed matches the order in which you create them. So if you wish to place a star sprite*

*above the background, you would need to ensure that it was added as an image second, after the sky image:*

*\*/*

    this.load.audio('ingameMUSIC','assets/audio/space-120280.mp3');

    this.load.image('spacevoid','assets/backgrounds/Azetude-GameplayBackground2.jpeg');

*//this.load.image('rainbowvoid', 'assets/rainbowvoid.png');*

*// width and height of the frame in pixels*

    this.load.spritesheet('paddle', 'assets/sprites/FirstPaddle.png', { frameWidth: 263, frameHeight: 551 });

    this.load.image('paddleAI','assets/sprites/SecondPaddle.png');

    this.load.image('BlueBall','assets/sprites/BlueBall.png');

}

function create() {

*// activate the mp3 music sound for the main game scene*

    const music = this.sound.add('ingameMUSIC');

    music.play();

*// draw all the game objects onto the screen*

    this.add.image(500,300,'spacevoid');

*//Creates a new Arcade Sprite object with a Static body.*

    /\*\*

     \* In Arcade Physics there are two types of physics bodies:

     \* Dynamic and Static. A dynamic body is one that can move

     \* around via forces such as velocity or acceleration. It can

     \* bounce and collide with other objects and that collision

     \* is influenced by the mass of the body and other elements.

     \*/

*// (used for testing physics) platform = this.physics.add.staticSprite(500,620,'rainbowvoid');*

*//the creation of a Physics Sprite and the creation of some animations that it can use.*

*/\* The sprite was created via the Physics Game Object Factory (this.physics.add)*

*which means it has a Dynamic Physics body by default. \*/*

    playerPaddle = this.physics.add.sprite(50,300,'paddle');

    console.log(`The height of the player paddle (internal height) ${playerPaddle.displayHeight}`);

    console.log(`The width of the player paddle (internal width) ${playerPaddle.displayWidth}`);

*// setSize - Sets the internal size of this Game Object, as used for frame or physics body creation.*

    playerPaddle.setSize(263, 420);

*/\* setBounce - Bounce is the amount of restitution, or elasticity, the body has when it collides with another object.*

*A value of 1 means that it will retain its full velocity after the rebound. A value of 0 means it will not rebound at all. \*/*

*//playerPaddle.setBounce(0.8);*

*//setScale - Sets the scale of this Game Object (Vertical and horizontal) this is like changing the actual sprite size!*

    playerPaddle.setScale(0.3, 0.3);

*/\* setPushable Sets if this Body can be pushed by another Body.*

*A body that cannot be pushed will reflect back all of the velocity it is given to the colliding body.*

*If that body is also not pushable, then the separation will be split between them evenly.*

*If you want your body to never move or seperate at all, see the setImmovable method. \*/*

    playerPaddle.setPushable(false);

    playerPaddle.setCollideWorldBounds(true);

*/\* The sprite is then set to collide with the world bounds. The bounds, by default, are on the outside of the game dimensions.*

*As we set the game to be 1000 x 600 then the player won't be able to run outside of this area.*

*It will stop the player from being able to run off the edges of the screen or jump through the top.\*/*

*//frameRate - The frame rate of playback, of the current animation, in frames per second. 0 by default*

*//repeat - The number of times to repeat playback of the current animation. -1 val means animation will repeat forever*

    this.anims.create({

        key: 'up',

        frames: this.anims.generateFrameNumbers('paddle', { start: 0}),

        frameRate: 10,

        repeat: -1

    });

    this.anims.create({

        key: 'down',

        frames: this.anims.generateFrameNumbers('paddle', { start: 0 }),

        frameRate: 10,

        repeat: -1

    });

    /\*\*

     \* This populates the cursors object with four properties: up, down, left, right,

     \* that are all instances of Key objects. Then all we need to do is poll these in

     \* our update loop.

     \*/

    cursors = this.input.keyboard.createCursorKeys();

*// ENEMY AI paddle*

    enemyPaddle = this.physics.add.sprite(config.width - 50, 300, 'paddleAI');

    enemyPaddle.setScale(0.3, 0.3);

    enemyPaddle.setCollideWorldBounds(true);

    enemyPaddle.setPushable(false);

*/\* The ball sprite was created via the Physics Game Object Factory (this.physics.add)*

*which means it has a Dynamic Physics body by default.\*/*

    ball = this.physics.add.sprite(500,300,'BlueBall');

*//setScale - Sets the scale of this Game Object (Vertical and horizontal) this is like changing the actual sprite size!*

    ball.setScale(0.2,0.2);

*// setSize - Sets the internal size of this Game Object, as used for frame or physics body creation.*

*//ball.setSize(10,10);*

*//Sets the Body's velocity (horizontal and vertical)*

    ball.setVelocity(xVelocityBallEnemy,yVelocityBallEnemyList[0]);

*/\*Set the X and Y values of the gravitational pull to act upon this Arcade Physics Game Object.*

*Values can be positive or negative. Larger values result in a stronger effect.*

*If only one value is provided, this value will be used for both the X and Y axis. \*/*

*//ball.setGravityY(200);*

*/\* setBounce - Bounce is the amount of restitution, or elasticity, the body has when it collides with another object.*

*A value of 1 means that it will retain its full velocity after the rebound. A value of 0 means it will not rebound at all. \*/*

    ball.setBounce(1,1);

*/\* setCollideWorldBounds - Sets whether this Body collides with the world boundary.*

*Optionally also sets the World Bounce values. If the Body.worldBounce is null, it's set to a new Phaser.Math.Vector2 first.*

*\*/*

    ball.setCollideWorldBounds(true);

*// collider method takes two objects and tests for collision and performs separation against them.*

    this.physics.add.collider(playerPaddle, ball,playerHitsBall);

    this.physics.add.collider(enemyPaddle, ball, enemyHitsBall);

    console.log(`The width of ball is ${ball.width}`);

    console.log(`The x coordinate of enemy paddle is ${enemyPaddle.x}`);

    console.log(`The y coordinate of enemy paddle is ${enemyPaddle.y}`);

    console.log(`The width of enemy paddle is ${enemyPaddle.width}`);

    console.log(`The height of enemy paddle is ${enemyPaddle.height}`);

    playerScoreText = this.add.text(340, 0, `player score: ${playerScore}`, { fontFamily: 'Dream MMA', fontSize: '25px', fill: "#FFF", fixedWidth:330});

    enemyScoreText = this.add.text(505, 0, `enemy score: ${enemyScore}`, { fontFamily: 'Dream MMA', fontSize: '25px', fill: "#FFF", fixedWidth:330});

}

function update() {

    /\*\*The first thing it does is check to see if the up key is being held down.

     \* If it is we apply a negative vertical velocity and start the 'up' running animation.

     \* If they are holding down 'down' instead we literally do the opposite.

     \* By clearing the velocity and setting it in this manner, every frame,

     \* it creates a 'stop-start' style of movement.

    The player sprite will move only when a key is held down and stop immediately they are not.

    Phaser also allows you to create more complex motions, with momentum and acceleration,

    but this gives us the effect we need for this game.

     \*/

    if (cursors.up.isDown) {

        playerPaddle.setVelocityY(-500);

        playerPaddle.anims.play('up', true);

    }

    else if (cursors.down.isDown) {

        playerPaddle.setVelocityY(500);

        playerPaddle.anims.play('down', true);

    }

    else{

*// without this final else block, your pong paddle will float like its underwater between the top wall and bottom wall.*

        playerPaddle.setVelocityY(0);

    }

*/\* main algorithm ideas:*

*when the ball gets close to the enemy paddle,*

*move the enemy paddle so it can block the ball from passing it via updating (or setting) its velocity.*

*Stop moving the paddle when the ball is not within its vicinity*

*by stopping its velocity. \*/*

    console.log(`The x velocity of ball is ${ball.body.velocity.x.toString()} and the y velocity of ball is ${ball.body.velocity.y.toString()}`);

*// ball is greater than or equal a width before the enemy paddle and a height below the enemys paddle.*

    if (ball.x >= (enemyPaddle.x - 350) && ball.y >= enemyPaddle.y + 60) {

*// make the paddle move to collide with the ball.*

        enemyPaddle.setVelocityY(400);

    }

*// ball is greater than or equal a width before the enemy paddle and less than a height above the enemys paddle.*

    else if (ball.x >= (enemyPaddle.x - 350) && ball.y < enemyPaddle.y - 60) {

*// make the paddle move to collide with the ball.*

        enemyPaddle.setVelocityY(-400);

    }

    else {

        enemyPaddle.setVelocityY(0);

    }

*// check if ball passes the right wall then update players score and reset balls position to the center again.*

    if (ball.x >= config.width - 20)

    {

        ball.x = config.width / 2

        ball.y = config.height / 2

        ball.setVelocity(xVelocityBallEnemy,yVelocityBallEnemyList[2]);

        playerScore++;

        playerScoreText.setText(`player score: ${playerScore}`);

    }

*// check if ball passes the left wall then update enemys score and reset its position to the center again.*

    if (ball.x <= 15) {

        ball.x = config.width / 2

        ball.y = config.height / 2

        ball.setVelocity(xVelocityBallPlayer, yVelocityBallPlayerList[2]);

        enemyScore++;

        enemyScoreText.setText(`enemy score: ${enemyScore}`);

    }

}

*/\**

*This custom function returns a random number in a custom range that is inclusive of the min and max.*

*\*/*

function getRndInteger(minNum, maxNum) {

    return Math.floor(Math.random() \* (maxNum - minNum + 1)) + minNum;

}

*/\**

*This custom function will cause the velocity of the ball to change upon collision with the player paddle.*

*The balls velocity will vary depending if the ball collides with the upper, center, or lower portion of the*

*players paddle.*

*\*/*

function playerHitsBall(){

    console.log(`Ball y pos = ${ball.y}`);

    console.log(`playerPaddle y pos = ${playerPaddle.y}`);

    var xVelocityBall = getRndInteger(700, 800);

*// when the ball hits the middle portion of the paddle then it shoud bounce back at a 0-1 degree angle.*

    if ((ball.y == playerPaddle.y) || (ball.y >= (playerPaddle.y - 15) && ball.y <= (playerPaddle.y + 15))){

        console.log("ball hit middle portion of player paddle");

        ball.setVelocity(xVelocityBall, yVelocityBallPlayerList[0]);

    }

    else if (ball.y < playerPaddle.y){

*// when the ball hits the upper portion of the paddle, then it should bounce at a 45 - 50 degree angle.*

        console.log("ball hit upper portion of player paddle");

        ball.setVelocity(xVelocityBallPlayer, yVelocityBallPlayerList[1]);

    }

    else{

*// when the ball hits the lower portion of the paddle, then it should bounce at a negative 45 - 50 degree angle.*

        console.log("ball hit lower portion of player paddle");

        ball.setVelocity(xVelocityBallPlayer, yVelocityBallPlayerList[2]);

    }

*/\* setBounce - Bounce is the amount of restitution, or elasticity, the body has when it collides with another object.*

*A value of 1 means that it will retain its full velocity after the rebound. A value of 0 means it will not rebound at all. \*/*

*//ball.setBounce(1, 1);*

}

*/\**

*This custom function will cause the velocity of the ball to change upon collision with the enemy paddle.*

*The balls velocity will vary depending if the ball collides with the upper, center, or lower portion of the*

*enemy paddle.*

*\*/*

function enemyHitsBall(){

    console.log(`Ball y pos = ${ball.y}`);

    console.log(`enemyPaddle y pos = ${enemyPaddle.y}`);

*// when the ball hits the middle portion of the paddle then it shoud bounce back at a 0-1 degree angle.*

    if (ball.y == enemyPaddle.y) {

        console.log("ball hit middle portion of enemy paddle");

        ball.setVelocity(xVelocityBallEnemy,yVelocityBallEnemyList[0]);

    }

    else if (ball.y < enemyPaddle.y) {

*// when the ball hits the upper portion of the paddle, then it should bounce at a 45 - 50 degree angle.*

        console.log("ball hit upper portion of enemy paddle");

        ball.setVelocity(xVelocityBallEnemy, yVelocityBallEnemyList[1]);

    }

    else {

*// when the ball hits the lower portion of the paddle, then it should bounce at a negative 45 - 50 degree angle.*

        console.log("ball hit lower portion of enemy paddle");

        ball.setVelocity(xVelocityBallEnemy, yVelocityBallEnemyList[2]);

    }

}