class BaseScene1 extends Phaser.Scene {

    cursors;

    enemies;

    player;

    door;

    s

    blocks = [];

    keys;

    haskey = false;

    isDown = false;

    doorlocked = true;

    constructor(config, tilesetImageURL, tilemapURL, tilesetItemsURL) {

        super(config);

        this.tilesetImageURL = tilesetImageURL;

        this.tilemapURL = tilemapURL;

        this.tilesetItemsURL = tilesetImageURL;

    }

    preload() {

        this.load.image('tileset-image', this.tilesetImageURL);

        this.load.image('mountains', '/assets/backgroundCastles.png');

        this.load.image("items", this.tilesetItemsURL);

        this.load.image('tilesets', this.tilesetItemsURL);

        this.load.image('doors', '/assets/door.png');

        this.load.image('bblocker', 'assets/block\_exported.png');

        this.load.image('yblocker', 'assets/ylock\_exported.png');

        this.load.image('gblocker', 'assets/glock\_exported.png');

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

        this.load.image('left', 'assets/leftarrow.png');

        this.load.image('right', 'assets/rightarrow.png');

        this.load.image('up', 'assets/uparrow.png');

        this.load.spritesheet('player', 'assets/wizard.png', {

            frameWidth: 32,

            frameHeight: 32

        });

        this.load.spritesheet('enemy', 'assets/enemyblue.png', {

            frameHeight: 32,

            frameWidth: 32

        });

        this.load.tilemapTiledJSON('level', this.tilemapURL);

    }

    create() {

        this.map = this.make.tilemap({

            key: 'level'

        });

        this.keys = this.physics.add.staticGroup();

        this.enemies = this.physics.add.group();

        this.doors = this.physics.add.staticGroup();

        this.blockers = this.physics.add.staticGroup();

        this.map.tileset = this.map.addTilesetImage('floor', 'tileset-image');

        this.physics.world.setBounds(0, 0, this.map.widthInPixels, this.map.heightInPixels);

        this.map.createStaticLayer('background', this.map.tileset, 0, 0);

        this.map.createStaticLayer('floor', this.map.tileset, 0, 0);

        //this.createplayer();

        let objectLayer = this.map.getObjectLayer("objects");

        if (objectLayer) {

            objectLayer.objects.forEach(function (object) {

                object = this.retrieveCustomProperties(object); //Check if the object has any custom properties in Tiled and assign them to the object

                // console.log(object)

                if (object.type === "spawnpoint") {

                    //Create player

                    this.createPlayer(object);

                } else if (object.type === "enemyspawner") {

                    this.createEnemy(object);

                    console.log('this is working');

                } else if (object.type === "blocker") {

                    this.blocks.push(object);

                } else if (object.type === "door") {

                    this.createdoors(object);

                } else if (object.type === "keys") {

                    this.createkey(object);

                }

            }, this); //Set context for object layer

        }

        for (var i = 0; i < this.blocks.length; i++) {

            this.createblockers(this.blocks[i]);

        }

        this.mountains = this.add.image(this.map.widthInPixels, this.map.heightInPixels, 'mountains');

        //console.log(this.mountains);

        this.mountains.setScale(10, 10);

        this.mountains.setDepth(-1);

        this.camera = this.cameras.getCamera("");

        this.camera.startFollow(this.player);

        this.camera.setBounds(0, 0, this.map.widthInPixels, this.map.height \* this.map.tileHeight);

        this.camera.zoom = 1;

        this.createCollision();

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

        this.movementButtons = {};

        this.createButtons();

        this.anims.create({

            key: 'Left',

            frames: this.anims.generateFrameNumbers('player', {

                start: 7,

                end: 9

            }),

            frameRate: 10,

            repeat: -1,

        });

        this.anims.create({

            key: 'Right',

            frames: this.anims.generateFrameNumbers('player', {

                start: 7,

                end: 8

            }),

            frameRate: 10,

            repeat: -1

        });

        this.anims.create({

            key: 'turn',

            frames: [{

                key: 'player',

                frame: 0

            }],

            frameRate: 10,

            repeat: -1

        })

        //this.createbutton();

    }

    update() {

        //console.log(this.player);

        if (this.cursors.right.isDown || this.movementButtons.right.isDown) {

            this.player.setVelocityX(250);

            this.player.flipX = false;

            this.player.anims.play('Right', true)

            this.isDown = true;

        } else if (this.cursors.left.isDown || this.movementButtons.left.isDown) {

            this.player.setVelocityX(-250);

            this.player.flipX = true;

            this.player.anims.play('Left', true);

        } else {

            this.player.setVelocityX(0);

            this.player.anims.play('turn')

        } //Check for space bar press

        if (Phaser.Input.Keyboard.JustDown(this.cursors.space)|| this.movementButtons.up.isDown) {

            this.player.setVelocityY(-500);

            this.isDown = true;

           this.movementButtons.up = this.isDown =false;

        }

    }

    createdoors(object) {

        this.doors.create(object.x, object.y, 'doors');

        //console.log(this.doors);

    }

    createblockers(object) {

        var blocker = this.blockers.create(object.x, object.y, object.name);

        blocker.colour = object.colour;

    }

    createkey(object) {

        var key = this.keys.create(object.x, object.y, object.colour + 'key');

        key.colour = object.colour;

    }

    retrieveCustomProperties(object) {

        if (object.properties) { //Check if the object has custom properties

            if (Array.isArray(object.properties)) { //Check if from Tiled v1.3 and above

                object.properties.forEach(function (element) { //Loop through each property

                    this[element.name] = element.value; //Create the property in the object

                }, object); //Assign the word "this" to refer to the object

            } else { //Check if from Tiled v1.2.5 and below

                for (var propName in object.properties) { //Loop through each property

                    object[propName] = object.properties[propName]; //Create the property in the object

                }

            }

            delete object.properties; //Delete the custom properties array from the object

        }

        return object; //Return the new object w/ custom properties

    }

    createPlayer(object) {

        this.player = this.physics.add.sprite(object.x, object.y, 'player', 0);

        this.player.setCollideWorldBounds(true);

        this.player.setScale(1.5)

        this.player.setDepth(2);

        this.player.ownedKeys = {

            green: false,

        }

    }

    createCollision() {

        // this is making a name and making = layer from tiled named 'floor'

        //and tilesetImageURL = my ground.png image that is used to make floor

        let collisionLayer = this.map.getLayer('floor').tilemapLayer;

        collisionLayer.setCollisionBetween(0, 1000)

        this.physics.add.collider(this.player, collisionLayer);

        this.physics.add.collider(this.enemies, collisionLayer);

        this.physics.add.collider(this.enemies, this.player, this.enemyattack, null, this);

        this.physics.add.collider(this.player, this.keys, this.collidekey, null, this);

        this.physics.add.collider(this.player, this.blockers, this.colliderblock, null, this);

        this.physics.add.collider(this.player,this.doors,this.opendoor, null,this);

    }

    colliderblock(player, blocker) {

        if (this.player.ownedKeys[blocker.colour] == false) {

            this.doorlocked = true;

        } else if (this.player.ownedKeys[blocker.colour] == true) {

            console.log("player has key ")

            this.doorlocked = false;

        }

        console.log(blocker);

    }

    opendoor(player,door){

        if(this.doorlocked = true){

            console.log("door is open")

         }

    }

    collidedoor(player,door) {

        if(this.doorlocked = false){

            console.log("door is locked find key ")

        }

    }

    collidekey(player, key) {

        this.player.ownedKeys[key.colour] = true;

        this.destroyBlock(key);

        if(this.keys.colour = this.player.ownedKeys[key.colour]){

            key.destroy();

            console.log(this.ownedKeys);

        }

    }

    destroyBlock(key) {

        this.blockers.getChildren().forEach(function (blocker) {

            if (blocker.colour === key.colour) {

                blocker.destroy();

            }

        });

    }

    enemyattack() {

        console.log("enemy hit you ");

    }

    createEnemy(object) {

        let origin = {

            x: object.x,

            y: object.y + object.height

        };

        let dest = {

            x: object.x + object.width,

            y: object.y + object.height

        };

        let line = new Phaser.Curves.Line(origin, dest);

        let enemy = this.add.follower(line, origin.x, origin.y, 'enemy');

        this.physics.add.existing(enemy);

        this.enemies.add(enemy);

        enemy.startFollow({

            duration: 5000,

            repeat: -1,

            yoyo: true,

            ease: 'Sine.easeInOut'

        })

    }

    movement() {

    }

    createButtons() {

        /\*this.movementButtons.left = \*/

        this.createButton(10, 2800, "left");

        this.createButton(80, 2800, 'right');

        this.createButton(500, 2800, "up");

    }

    createButton(x, y, texture, callback) {

        var button = this.add.sprite(x, y, texture);

        button.setInteractive()

        button.on('pointerdown', function (x, y, texture) {

            console.log('this is working')

            this.isDown = true;

            console.log(this.isDown);

        })

        button.on('pointerup', function (x, y, texture) {

            this.isDown = false;

        })

        button.on('pointermove', function(x,y,texture){

            this.isDown = false;

        })

        //What does the event listener want to do?

        this.movementButtons[texture] = button;

    }

}