Platformer Game

Base Platformer
Step 00: Setup Game Project
Step 01: Load Overworld Map
Step 02: Add player
Step 03: Move player
Step 04: Add Gravity
Step 05: Add Camera
Step 06: Wall/Floor Collisions
Step 07: Game Over

Bonus Features (Optional):
Bonus Step 0: Load Object Layer

Multiple Levels
Hazards
    Falling Bricks
    Spikes
Enemies
    Walking enemy
    Flying enemy
Collectables
    Coins with score counter
Animation
    left/right walking animations

Step 0: Setup Phaser Game Project

Planning:
Setup the basic organization and codebase to start a new phaser project
Project Organization: designate an organization for your game project.

- assets: directory to hold all of your game art
- scripts: directory to hold all of your game code
- index.html: file that launches your game in the browser

Implementation:
Let's implement the

index.html

```
<html>
    <head>
        <script src="./scripts/phaser.js">    </script>
        <script src="./scripts/Level1.js"> </script>
        <script src="./scripts/game.js">      </script>
    </head>
    <body></body>
</html>
```

Explanation:
Browsers are applications that open and execute html files. The javascript code must be attached to an html file such that the browser can run it. This game will use the Phaser 3 javascript game library. In addition to that, we must create two javascript files:
Must include three javascript files:

- *phaser.js:* html5 game framework
- *Level1.js*: this game's scene class file, contains & manages all rules for this game
- *game.js*: setups the configurations for this game and runs the game through Phaser

game.js

```
var config = new Object();

config.type   = Phaser.CANVAS;                          //HTML Rendering API
config.width  =  32 * 96;                               //32px/tile * 96 tiles/world
config.height = 32 * 16;                                //32px/tile * 16 tiles/world
config.pixelArt = true;                                 //optimized for pixel art
config.scene = [ Level1 ];                              //Scenes in this game
config.physics = { default:'arcade'};                   //Physics: collisions & gravity

var game = new Phaser.Game(config);                     //Start game with these configs
```

Explanation:
Setup the configurations for this game. Must define the renderer type, the screen width, screen height, the game scenes, and the physics properties. Then start the game!

Level1.js

```
class Level1 extends Phaser.Scene
{
    //construct new scene
    constructor(key='Level1')
    {
        super(key);     //set this scene's id within superclass constructor
        this.map_key = 'map1';
        this.map_json = 'level1.json';
    }

    //preload external game assets
    preload()
    {

    }

    //create game data
    create()
    {

    }

    //Update game data
    update()
    {

    }
}
```

Explanation:

Games are made up of a collection of Scenes. Each scene contains all the game objects and the rules for the game. A scene has 4 phases.

- constructor: constructs an empty scene object
- preload: preloads the external art, level, music assets that the scene depends on
- create: creates all the game objects within the scene
- update: manages the game objects during gameplay, updating their state

Step 1: Load Overworld Map

Planning:
After designing your overworld map in Tiled, let's start by loading and displaying the overworld map into the Play Scene. Ensure that your tileset is named: 'tiles.png' and your json file is named: 'worldMap.json'

Implementation:

Level1.js → preload()

```
preload()
{
    this.load.path = 'assets/';                                         //Define file path
    this.load.tilemapTiledJSON( this.map_key, this.map_json);               //Load JSON file
    this.load.image( 'tiles', 'tiles.png' );                            //Load tile images
}
```

Level1.js → create()

```
//Create Game World
create()
{
    this.create_map();          // create level
}
```

Level1.js → create_map()

```
//Load level
create_map()
{
        this.map = this.make.tilemap( {key: this.map_key} ); //setup map object from tilemap
        let groundTiles = this.map.addTilesetImage( 'tiles'); //add tileset image to map obj

        // 1st param: name of tileset in Tiled, 2nd param: tileset image from preload
        this.groundLayer = this.map.createStaticLayer( 'tiles', groundTiles);
}
```

Test:
Try opening the index.html file within the browser and see if the entire game world map loads into the canvas. Note: You may need to 'Zoom Out' your browser view to display the entire map.

Debugging:

Step 2: Add Player

Planning:

Now that the world map is loaded into the Scene, let's next add the player into the starting location.

Implementation:

index.html

```
<html>
    <head>
        <script src="./scripts/phaser.js">    </script>
        <script src="./scripts/Player.js">    </script>
        <script src="./scripts/Level1.js"> </script>
        <script src="./scripts/game.js">      </script>
    </head>
    <body></body>
</html>
```

Player.js

```
class Player extends Phaser.Physics.Arcade.Sprite
{
    constructor(scene)
    {
        super(scene, 0, 0, 'player');

        let start = scene.map.findObject('items', obj => obj.name === 'player' );
        this.x = start.x;
        this.y = start.y ;
        this.setOrigin(0.5,1);
        this.depth = 1;
        this.speed = 200;

        scene.add.existing(this);
    }
}
```

Level1.js → preload()

```
preload()
{
    this.load.path = 'assets/';                                          //Define file path
    this.load.tilemapTiledJSON( this.map_key, 'lvl1-map.json');          //Load JSON file
    this.load.image( 'tiles', 'tiles.png' );                            //Load tile images
    this.load.image( 'player', 'player.png');
}
```

Level1.js → create()

```
//Create Game World
create()
{
    this.create_map();                          //create map
    this.player = new Player(this);             //create player
}
```

Step 3: Move Player

Planning:
Now that the player is in the game, let's add movement.

Implementation:

Player.js

```
class Player extends Phaser.Physics.Arcade.Sprite
{
    constructor(scene)
    {
        super(scene, 0, 0, 'player');

        let start = scene.map.findObject('items', obj=> obj.name === 'start');
        this.x = start.x;
        this.y = start.y;
        this.setOrigin(1,1);
        this.depth = 1;
        this.speed = 200;

        scene.add.existing(this);
        scene.physics.add.existing(this);
        this.arrow_keys = scene.input.keyboard.addKeys('up,left,right');
    }

    //move player
    move() {
        //verify that player has a physics body to move (for multilevel)
        if (this.body === undefined)
            return;

        // reset velocity
        this.body.velocity.x = 0;
        // take care of character movement
        if ( this.arrow_keys.up.isDown )
        {
            this.body.velocity.y = -250;
        }
        if ( this.arrow_keys.left.isDown )
        {
            this.body.velocity.x = -this.speed;
        }
        if ( this.arrow_keys.right.isDown )
        {
            this.body.velocity.x = this.speed;
        }
    }
}
```

Level1.js → update()

```
//update game state
update()
{
    this.player.move();
}
```

Testing:
The player should now slowly move around the world when the arrow keys are pressed. Gravity should pull them down.

Step 4: Add Gravity

Planning:
Now that the player is in the game, let's add movement.

Implementation:

Level1.js → setup_physics()

```
//setups physics and collisions
setup_physics()
{
    this.physics.world.gravity.y = 600;
}
```

Level1.js → create()

```
//Create Game World
create()
{
    this.create_map();                          //create map
    this.player = new Player(this);             //create player
    this.setup_physics();                       //setup physics
}
```

Step 5: Add Camera

Planning:
Now that the player is in the game, let's add a camera to focus on the player.

Implementation:

game.js

```
var config = new Object();

config.type  = Phaser.CANVAS;                                   //HTML Rendering API
config.width  =  32 * 24;                                       //32px/tile * 8 tiles/screen
config.height = 32 * 16;                                        //32px/tile * 8 tiles/screen
config.pixelArt = true;                                         //optimized for pixel art
config.scene = [ Level1 ];                                      //Scenes for this game
config.physics = { default:'arcade' }                 //Physics: collisions & gravity

var game = new Phaser.Game(config);                   //Start game with these configurations
```

Level1.js → setup_camera()

```
//Setup camera to follow player
setup_camera()
{
    this.cameras.main.startFollow(this.player);
    this.cameras.main.setBounds(0,0,this.map.widthInPixels, this.map.heightInPixels);
    this.cameras.main.setBackgroundColor('rgb(204, 207, 255)');
}
```

*Implement these three statements one-by-one and test the game each time in between

Level1.js → create()

```
//Create Game World
create()
{
    this.create_map();                          //create map
    this.player = new Player(this);             //create player
    this.setup_physics();                       //setup physics
    this.setup_camera();                        //setup camera
}
```

Step 6: Ground/Wall Collisions

Planning:
Now that the player moves and the camera follows. Let's add wall collisions using physics.
In Tiled, give custom property to all tiles in tiles layer with 'terrain' property of 'block'.

Implementation:

Level1.js → create_map()

```
//Load level
create_map()
{
        this.map = this.make.tilemap( {key: 'map'} );          //set up map object from tilemap
        let groundTiles = this.map.addTilesetImage( 'tiles');  //add tileset images to the map obj

        // 1st param: name of tileset in Tiled, 2nd param: tileset image from preload.
        this.groundLayer = this.map.createStaticLayer( 'tiles', groundTiles);

        //collisions based on tile type in custom properties
        var ground_block = { terrain: 'block'};
        this.groundLayer.setCollisionByProperty( ground_block );
}
```

Level1.js → setup_physics()

```
//setups physics and collisions
setup_physics()
{
    this.physics.world.gravity.y = 600;
    this.physics.add.collider( this.player, this.groundLayer );
}
```

Player.js → move()

```
//move player
move()
{
    // reset velocity
    this.body.velocity.x = 0;

     // take care of character movement
     if ( this.arrowKeys.up.isDown && this.body.onFloor() )
     {
         this.body.velocity.y = -250;
     }
     if ( this.arrowKeys.left.isDown )
     {
         this.body.velocity.x = -this.speed;
     }
     if ( this.arrowKeys.right.isDown )
     {
         this.body.velocity.x = this.speed;
     }
}
```

Testing:
Player should now collide with the floor and wall blocks

Step 7: Game Over

Planning:
Now add a lose condition. If the player falls off the stage, the level should restart

Implementation:

Level1.js → game_over()

```
//check player lose conditions
game_over()
{
    if (this.player.y > this.map.heightInPixels)          //check if player is lower than level
    {
        this.scene.restart();
    }
}
```

Level1.js → update()

```
//update game state
update()
{
    this.player.move();
    this.game_over();
}
```

Additional Bonus Features

Basic Game complete! Let's now add bonus features and objects into the level
-More Levels
-Hazards like falling bricks or spikes
-Collectables
-Animations
-Enemies

Bonus, Step 0: Load Object Layer

Planning:
Now let's add the object layer into the game world from the Tiled map. Item images should be in its own sprite sheet.

Implementation:

Level1.js → preload()

```
//preload external game assets
preload()
{
    this.load.path = 'assets/';                                          //Define file path
    this.load.tilemapTiledJSON( this.map_key, 'lvl1-map.json');          //Load JSON file
    this.load.image( 'tiles', 'tiles.png' );                             //Load tile images
    var player_size = {frameWidth: 64, frameHeight: 64}
    this.load.spritesheet( 'player', 'player.png', player_size );
    var tile_size = {frameWidth: 32, frameHeight: 32}
    this.load.spritesheet( 'itemMap', 'items.png', tile_size );
    this.load.spritesheet( 'tileMap', 'tiles.png', tile_size );
}
```

Level1.js → create()

```
//Create Game World
create()
{
    this.create_map();                      //create map
    this.player = new Player(this);         //create player
    this.setup_camera();                    //setup camera
    this.create_objects();                  //create objects/items
    this.setup_physics();                   //setup physics
}
```

Level1.js → create_objects()

```
//Create items from object layer
create_objects()
{

}
```

Level1.js → setup_objects()

```
setup_objects(objGroup){
    for(var obj of objGroup)
    {
        this.physics.add.existing(obj);
        obj.body.immovable = true;
        obj.body.allowGravity = false;
    }
}
```

Bonus #1: More Levels

Planning:

Let's add new levels to the game.

Implementation:

Level2.js

```
//level2 scene
class Level2 extends Level1
{
   constructor()
   {
        super('Level2');
        this.map_key ='map2';
        this.map_json ='level2.json';
   }
}
```

index.html

```
<html>
    <head>
        <script src="./scripts/phaser.js">    </script>
        <script src="./scripts/Player.js">    </script>
        <script src="./scripts/Level1.js">    </script>
        <script src="./scripts/Level2.js">    </script>
        <script src="./scripts/game.js">      </script>
    </head>
    <body></body>
</html>
```

game.js

```
var config = new Object();

config.type = Phaser.CANVAS;                                    //HTML Rendering API
config.width =  32 * 96;                                      //32px/tile * 96 tiles/screen
config.height = 32 * 16;                                      //32px/tile * 16 tiles/screen
config.pixelArt = true;                                        //optimized for pixel art
config.scene = [ Level1, Level2 ];                                 //Scenes for this game
config.physics = { default:'arcade'};                        //Physics: collisions & gravity

var game = new Phaser.Game(config);                         //Start game with these configs
```

Level1.js → create_objects()

```
//Create items from object layer
create_objects()
{
    var goal_image = {key:'itemMap', frame: 3 };
    this.goal = this.map.createFromObjects( 'items','goal', goal_image);
    this.setup_objects(this.goal);
}
```

**Note: frame value should match the tile's countable position within the itemMap sprite sheet*

Level1.js → next_scene()

```
//start the next level
next_scene(player,goal)
{
    this.scene.start('Level2');
}
```

Level1.js → setup_physics()

```
//setups physics and collisions
setup_physics()
{
    this.physics.world.gravity.y = 600;
    this.physics.add.collider( this.player, this.groundLayer );
    this.physics.add.overlap(this.player,this.goal,this.next_scene,null,this);
}
```

Testing:

When the player touches the goal, Level2 should start

Bonus #2: Hazards: Deadly Tiles

Step 1: Add First Hazard:

Planning:
Add stationary tiles that can kill player such as lava, spikes, acid, water.

Implementation:

Level1.js → create_objects()

```
//Create items from object layer
create_objects()
{
    var hazard1_image =  {key:'itemMap', frame: 2 }
    this.group_hazard1 = this.map.createFromObjects( 'items','hazard1',hazard1_image);
    this.setup_objects(this.group_hazard1);
}
```

**Note: frame value should match the tile's countable position within the itemMap sprite sheet*

Level1.js → setup_physics()

```
//setups physics and collisions
setup_physics()
{
    this.physics.world.gravity.y = 600;
    this.physics.add.collider( this.player, this.groundLayer );
    this.physics.add.overlap(this.player,this.group_hazard1,this.game_over,null,this);
}
```

Level1.js → game_over()

```
//check player lose conditions
game_over(player=null, hazard=null)
{
    if (this.player.y > this.map.heightInPixels)     //check if player is lower than level
    {
        this.scene.restart();
    }
    if (hazard !== null)                            //check if player is touching hazard
    {
        this.scene.restart();
    }
}
```

Testing:
When the player touches the hazard, the game over occurs.

Step 2: Add Second Hazard:

Planning:
Add stationary tiles that can kill player such as lava, spikes, acid, water.

Implementation:

Level1.js → create_objects()

```
//Create items from object layer
create_objects()
{
    var hazard2_image =  {key:'itemMap', frame: 1 }
    this.group_hazard2 = this.map.createFromObjects( 'items','hazard2',hazard2_image);
    this.setup_objects(this.groupHazard2);
}
```

**Note: frame value should match the tile's countable position within the itemMap sprite sheet*

Level1.js → setup_physics()

```
//setups physics and collisions
setup_physics()
{
    this.physics.world.gravity.y = 600;
    this.physics.add.collider( this.player, this.groundLayer );
    this.physics.add.overlap(this.player,this.group_hazard1,this.game_over,null,this);
    this.physics.add.overlap(this.player,this.group_hazard2,this.game_over,null,this);
}
```

Testing:
When the player touches the hazard, the game over occurs.

Bonus #3: Hazards: Falling Tiles

Planning:
Add stationary tiles that fall when player touches them.

Implementation:

Level1.js → create_objects()

```
//Create items from object layer
create_objects()
{
    var fall_image = {key:'tileMap', frame: 3 };
    this.group_fall = this.map.createFromObjects( 'items','fall', fall_image);
    this.setup_objects(this.group_fall);
}
```

**Note: frame value should match the tile's countable position within the itemMap sprite sheet*

Level1.js → setup_physics()

```
//setups physics and collisions
setup_physics()
{
    this.physics.world.gravity.y = 600;
    this.physics.add.collider( this.player, this.groundLayer );
    this.physics.add.collider(this.player,this.group_fall,this.add_gravity,null,this);
}
```

Level1.js → add_gravity()

```
//check player lose conditions
add_gravity(player, hazard)
{
    hazard.body.gravity.y = -1;
    hazard.body.allowGravity = true;
}
```

Player.js → move()

```
//move player
move() {
    this.body.velocity.x = 0;                                    // reset velocity

     // take care of character movement
     if ( this.arrowKeys.up.isDown && ( this.body.onFloor() || this.body.touching.down) ) {
         this.body.velocity.y = -250;
     }
     if ( arrowKeys.left.isDown ) {
         this.body.velocity.x = -this.speed;
     }
     if ( arrowKeys.right.isDown ) {
         this.body.velocity.x = this.speed;
     }
}
```

Testing: When the player touches the hazard, it drops down by gravity.

Bonus #4: Collectables

Planning:
Add collectable items such as coins into the level (example code uses coins)

Implementation:

Level1.js → create_objects()

```
//Create items from object layer
create_objects()
{
    var collect_image = {key: 'itemMap', frame: 0 }
    this.group_collect =  this.map.createFromObjects( 'items','collect', collect_image);
    this.setup_objects( this.group_collect );
}
```

**Note: frame value should match the tile's countable position within the itemMap sprite sheet*

Level1.js → coin_take()

```
//pick up a key
collect_take( player, collect )
{
    collect.destroy();
}
```

Level1.js → setup_physics()

```
//setups physics and collisions
setup_physics()
{
    this.physics.world.gravity.y = 600;
    this.physics.add.collider( this.player, this.groundLayer );
    this.physics.add.overlap( this.player, this.group_collect, this.coin_take, null, this );
}
```

Testing:
The player should now be able to interact with objects. Collectables disappear when player touches them.

Bonus #5: Animations

Planning:
Animate player when moving left or right

Implementation:

Player.js → constructor

```
constructor(scene)
{
    super(scene, 0, 0, 'player');

    let start = scene.map.findObject('objects', obj=> obj.type === 'start');
    this.x = start.x;
    this.y = start.y;
    this.setOrigin(0.5,1);
    this.depth = 1;
    this.speed = 200;

    scene.add.existing(this);
    scene.physics.add.existing(this);

    this.arrowKeys = scene.input.keyboard.setKeys("up,left,right");

    this.setup_animations(scene);
}
```

Player.js → setup_animations()

```
//setup the player animations
setup_animations(scene){
    var left_animation = new Object();
    left_animation.key = 'left';
    left_animation.frames = scene.anims.generateFrameNumbers('player', {start:2, end:3});
    left_animation.frameRate = 6;
    scene.anims.create(left_animation);

    var right_animation = new Object();
    right_animation.key = 'right';
    right_animation.frames = scene.anims.generateFrameNumbers('player', {start:0, end:1});
    right_animation.frameRate = 6;
    scene.anims.create(right_animation);

   this.anims.play('right',false);
}
```

**Note: start/end values should match the tile's countable start/end position within the player sprite sheet*

Player.js → move()

```
move(){
    this.body.velocity.x=0;                              // reset x velocity to 0

    if(this.arrowkeys.up.isDown && (this.body.onFloor()){
        this.body.velocity.y= -this.speed *2;
    }
    if(this.arrowkeys.left.isDown ){
        this.body.velocity.x= -this.speed;
```

```
        this.anims.play('left',true);                   //play animation
    }
    if(this.arrowkeys.right.isDown){
        this.body.velocity.x= this.speed;
        this.anims.play('right',true);                   //play animation
    }

}
```

Testing:
Player should now animate when moving

Bonus #6: Enemies

Step 1: Add the Enemy

Planning:
Add multiple enemies into the level

Implementation:

index.html

```
<html>
    <head>
        <script src="./scripts/phaser.js">   </script>
        <script src="./scripts/Player.js">   </script>
        <script src="./scripts/Enemy.js">    </script>
        <script src="./scripts/Level1.js">   </script>
        <script src="./scripts/game.js">     </script>
    </head>
    <body></body>
</html>
```

Enemy.js

```
class Enemy extends Phaser.Physics.Arcade.Sprite
{
    constructor(scene, start)
    {
        super(scene, start.x, start.y, 'enemy');

        this.start = start;
        this.depth = 1;
        this.speed = 120;

        scene.add.existing(this);
        scene.physics.add.existing(this);
        this.body.velocity.x = -this.speed;
        this.body.bounce.x = 1;
    }
}
```

Level1.js → preload()

```
preload()
{
    this.load.path = 'assets/';                                         //Define file path
    this.load.tilemapTiledJSON( this.map_key, 'lvl1-map.json');             //Load JSON file
    this.load.image( 'tiles', 'tiles.png' );                            //Load tile images
    this.load.spritesheet( 'player', 'player.png', {frameWidth: 64, frameHeight: 64} );
    this.load.image( 'enemy', 'enemy.png' );
}
```

Level1.js → create()

```
//Create Game World
create()
{
    this.create_map();                          //create map
    this.player = new Player(this);             //create player
    this.setup_camera();                        //setup camera
    this.create_monster();                      //create enemies
    this.create_objects();                      //create objects/items
    this.setup_physics();                       //setup physics
}
```

Level1.js → create_monsters()

```
create_monsters()
{
    this.group_monsters = [];
    let enemy_tiles = this.map.filterObjects('items', (obj) => obj.name==='enemy');
    for (let tile of enemy_tiles)
    {
        let enemy_config = {x: tile.x, y: tile.y};
        let monster = new Enemy(this, enemy_config );
        this.group_monsters.push( monster );
    }
}
```

Level1.js → setup_physics()

```
//setups physics and collisions
setup_physics()
{
    this.physics.world.gravity.y = 600;
    this.physics.add.collider( this.player, this.groundLayer );
    this.physics.add.collider( this.group_monsters, this.groundLayer );
    this.physics.add.overlap( this.player,this.group_monsters, this.game_over, null, this );
}
```

**Note: Add colliders with groupMonsters to groupBricks and groupSpikes if they also exists in your game.*

Testing:

Enemies should render into scene, Enemies should collide with floor, and restart level if the player touches them.

Step 2: Respawn Enemy

Planning:
Respawn Enemy to starting position if it fell off stage.

Implementation:

Enemy.js → respawn()

```
respawn()
{
    this.x = this.start.x;
    this.y = this.start.y;
    this.body.velocity.y = 0;
}
```

Level1.js → respawn_enemy()

```
respawn_enemy()
{
    for (let enemy of this.group_monsters)
    {
        if(enemy.y > this.map.heightInPixels)
        {
            enemy.respawn();
        }
    }
}
```

Level1.js → update()

```
//update game state
update()
{
    this.player.move();
    this.game_over();
    this.respawn_enemy();
}
```

Testing:
Enemies should respawn into starting location if they move off the stage

Step 3: Kill Enemy

Planning:
Player can kill an enemy by jumping on top of it

Implementation:

Level1.js → attack()

```
attack(player, enemy)
{
    if (player.body.touching.down)
        enemy.destroy()
    else
        this.game_over(player, enemy);
}
```

Level1.js → setup_physics()

```
//setups physics and collisions
setup_physics()
{
    this.physics.world.bounds.width = this.groundLayer.width;
    this.physics.world.bounds.height = this.groundLayer.height;
    this.physics.add.collider( this.player, this.groundLayer );
    this.physics.add.collider( this.groupMonsters, this.groundLayer );
    this.physics.add.overlap( this.player, this.groupMonsters, this.attack, null, this );
}
```

**Note: Add colliders with groupMonsters to groupBricks and groupSpikes if they also exists in your game.*

Testing:
Enemies should disappear if the player jumps on top of them