Dodger Game

Base Platformer
Step 00: Setup Game Project
Step 01: Load Background
Step 02: Add player
Step 03: Move player
Step 04: Add Enemy
Step 05: Move Enemy
Step 06: Game Over/ Collisions

Bonus Features:

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/PlayScene.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
- *PlayScene.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 =
{
    type: Phaser.CANVAS,                                    //HTML Rendering API
    width:  640,                                           //64px/tile * 48 tiles/world
    height: 480,                                           //64px/tile * 40 tiles/world
    pixelArt: true,                                        //optimized for pixel art
    scene: [ PlayScene ],                                 //Scenes for this game
    physics: { default:'arcade', arcade:{ gravity:{y:0} }  }    //Physics for collisions & gravity
};

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

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!

PlayScene.js

```
class PlayScene extends Phaser.Scene
{
    //construct new scene
    constructor()
    {
        super('play');      //set this scene's id within superclass constructor
    }

    //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.'

Implementation:

PlayScene.js → preload()

```
preload()
{
    this.load.path = 'assets/';                                    //Define file path
    this.load.image( 'background', 'background.png' );             //Load tile images
}
```

PlayScene.js → create()

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

PlayScene.js → create_map()

```
//Load level
create_map()
{
        this.add.image(640/2, 480/2, 'background');
}
```

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.

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/PlayScene.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, 300, 200, 'player');
        this.depth = 1;
        this.speed = 200;

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

PlayScene.js → preload()

```
preload()
{
    this.load.path = 'assets/';                                    //Define file path
    this.load.image( 'background', 'background.png' );             //Load background image
    this.load.image( 'player', 'player.png' );                     //Load player image
}
```

PlayScene.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, 300, 200, 'player');
        this.depth = 1;
        this.speed = 200;

        scene.add.existing(this);
        scene.physics.add.existing(this);
        this.setCollideWorldBounds(true);       //don't go out of the map

        this.arrowKeys = new Set()
        scene.input.keyboard.on('keydown', this.updateArrowKeys,this );
        scene.input.keyboard.on('keyup',   this.updateArrowKeys,this );
    }

    updateArrowKeys(event)
    {
        if (event.type === 'keydown')
            this.arrowKeys.add(event.key);
        else if (event.type === 'keyup')
            this.arrowKeys.delete(event.key);
    }

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

        // take care of character movement
        if ( this.arrowKeys.has("ArrowUp") )
        {
            this.body.velocity.y = -this.speed;
        }
        if ( this.arrowKeys.has("ArrowDown") )
        {
            this.body.velocity.y = this.speed;
        }
        if ( this.arrowKeys.has("ArrowLeft" ) )
        {
            this.body.velocity.x = -this.speed;
        }
        if ( this.arrowKeys.has("ArrowRight" ) )
        {
            this.body.velocity.x = this.speed;
        }
    }
}
```

PlayScene.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 Enemy

Planning:
Now let's spawn enemies into the game

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

Enemy.js

```
class Enemy extends Phaser.Physics.Arcade.Sprite
{
    constructor(scene)
    {
        super(scene, -32, -32, 'enemy');
        this.x = Phaser.Math.Between(0, 640);
        this.depth = 1;
        this.speed = Phaser.Math.Between(2, 10);

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

PlayScene.js → preload()

```
preload()
{
    this.load.path = 'assets/';                                    //Define file path
    this.load.image( 'background', 'background.png' );             //Load background image
    this.load.image( 'player', 'player.png' );                     //Load player image
    this.load.image( 'enemy', 'dragon.png' );                     //Load player image
}
```

PlayScene.js → create()

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

PlayScene.js → create_objects()

```
create_objects()
{
    this.enemies = this.physics.add.group();

    this.time.addEvent({
        delay: 200,
        callback: function() { this.enemies.add( new Enemy(this)) },
        callbackScope: this,
        loop: true
    }, this);
}
```

Step 5: Move Enemies

Planning:
Now that the enemies spawn, start moving them around

Implementation:

Enemy.js → move()

```
//move Enemy
move()
{
        this.y += this.speed;
}
```

PlayScene.js → update()

```
//update game state
update()
{
    this.player.move();
    this.enemies.children.iterate( (enemy) => enemy.move() );
}
```

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

Step 6: Game Over

Planning:
Now add a lose condition. If the player touches monster restart the scene.
Implementation:

PlayScene.js → create()

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

PlayScene.js → setup_physics()

```
//setups physics and collisions
setup_physics()
{
    this.physics.add.overlap(this.player,this.enemies,this.game_over,null,this);
}
```

PlayScene.js → game_over()

```
game_over(player, enemy)
{
    this.scene.restart();
}
```

Additional Bonus Features

Basic Game complete! Let's now add bonus features and objects into the level
-More Levels
-Different Enemies
-Collectables
-Animations
-Power ups
-Timer
-bullets