- Step 1 - Setting Up the Scene
- Step 2 - Adding Action to the Ball
- Step 3 - Set up the Pool Table Walls
- Step 4 - Making all the NumBalls
- Step 5 - Create Cue Ball

## Step 1 - Setting Up the Scene

1. Create a scene:
   - In the gallery, click on `Scenes` tab.
   - Press `Add Scene` button.
   - Enter a name for the scene, such as “Main”.
   - Press `Create` button.
   - Click on the *Main* scene in the gallery to open it.
   - Optional: On the top right part of the scene, adjust the `Viewport` to a device that you prefer. It’s useful to make it as large as possible where you can still see the whole scene, without getting scrollbars.
2. Add background image:
   - In the gallery, click on `Assets` tab. Choose `Images`.
   - Drag the *background* image onto the scene.

3. Move the *background* image to cover the whole scene. Select and drag the *background* image, or use the `Property Editor` to set the image's `Left` and `Top` properties to "0".
4. Create the NumBall actor
    - From the gallery, choose `Actors` tab.
    - Click on `Add Actor` button. It opens a window to add an actor.
    - Set `Actor Name` to "NumBall".
    - Under images, choose "Ball number 1".
    - Press `Create Actor` button.
5. Drag the *NumBall* actor to the scene. Run the scene. As you see, nothing happens. In the next video we will learn how to add action to our actor.

## Step 2 - Adding Action to the Ball

1. Make the Ball move.
    - Select the *NumBall* from the Scene
    - The `Property Editor` will open on the right; that will give you access to change the properties of this entity.
    - Scroll down in the list under the `physics` category, and set the `linear velocity` to "10", and run the scene. The ball will start moving toward the top of the scene.
2. Change the moving direction: Go back to `Property Editor` and change the `Velocity Direction` to "45" and run the scene. Do the same thing for "90", "180" and "270". This gives you a sense of how to set the moving direction of the ball.
3. Adjust the ball's shape: Drag and drop another *NumBall* to the scene, where the first ball will have a slight collision with it. Set the first *NumBall*'s `Linear Velocity` to "1" and the `Velocity Direction` to "90" and run the scene. As you can see, even though the Balls are circular, they will have collision on corners too. To fix this, in the first *NumBall* properties, change its `shape` to "circle". Run the scene; we still have a problem with the second NumBall. Fix the `shape` for the other NumBall as well.

4. Adjust the shape for all balls: To fix the shape for any *NumBall* entity that will be created from now on, we need to adjust the shape property of *NumBall* actor:
    - Go to *NumBall* actor.
    - Click on `Image and/or Property Editor`.
    - Change the `shape` to “circle”.

## Step 3 - Set up the Pool Table Walls

To keep the ball on the table we need to put some walls around the table.

1. Create vertical walls: Go to `Actors` and create a new actor; name it “VWall” and use the “vertical wall” image for it. Drag and drop the wall on the scene and move it to the right position.
2. Make the wall motionless: For one of the balls, set the `Linear Velocity` to “20” and `Velocity direction` to “90”. Run the scene. As you see, the wall will move when it is hit by the ball. To keep the wall from moving, we have to make the wall motionless. To do so:
    - Go to `Property Editor` under the `Physics` category and set the `Type`to “Static”. Run the scene again. This time, the wall does not move when it is hit by the ball.
    - To make all the vertical walls motionless, go to the VWall `Image and/or Property Editor`, set the `Type` to “Static”. Drag and drop the second VWall to the scene.
3. Create horizontal walls: Create another actor and name it “HWall”. Use the horizontal wall image for it and set its `Type` to “Static”. Drop the *HWall* on the scene and put it in the right position.
4. Rotate walls: As you can see, some of the walls’ lighting are not toward the table. We can fix this by rotating those walls. To do so, choose each wall entity, in `Property Editor`; under the `Image` set `Image Rotation` to “180” degrees.

## Step 4 - Making all the NumBalls

In this step we will make all the numbered balls and will put them on the scene.

1. Set the speed to zero, to make sure all of the balls behave the same and none of them are going to have preset speed. Select each ball, and in its `Property Editor` make sure the `Linear Velocity` and `Velocity Direction` is zero.
2. Put all the *NumBalls* on the scene: Start lining up the balls on the scene. Drag and drop more *NumBall* to the scene. You can make a copy of the balls that are already on the scene by pressing the `CTRL` key and then dragging the balls.
3. Change the balls' image: Select each ball; in `Property Editor` under the `Properties`category, change the `Image` for each of them from 1 to 15. These images have been preloaded under the `Assets` for you.
4. Lay the balls: Start putting the balls on the scene where they are supposed to be. Use their `Left` and `Top` properties to adjust their position.

## Step 5 - Create Cue Ball

1. Create the *CueBall* actor: Because the CueBall has different behavior from other balls, we need to create a separate actor for it.
    - Go to `Actors`.
    - Click on `Add Actor`.
    - Set the `Name` to "CueBall".
    - Choose the `Shape` to "Circle".
2. Put the *CueBall* on the scene.
    - Drag and drop *CueBall* on the scene.
    - Move it to the correct its position. Use `Top` and `Left` values as reference.

3. Adjust the moving direction of *CueBall*: As you see, the arrow on the ball is upward, which shows the direction that the ball will be moving towards. To make the *CueBall* move towards other balls, we need to rotate it. To do so, go to `Property Editor`, and under the `Image` set the `Rotation` to “90” degrees.
4. Hit the *CueBall*: Run the scene. As you see, nothing happens. Set the `Linear Velocity` to “50” and `Velocity Direction` to “90”. Run the scene again. As you see, *CueBall* moves and hits the *NumBalls*.

## Step 6 - Adjust Bounce

In the past step, the *NumBalls’* reaction to the hit does not look natural. In this step, we do some adjustments to make the *NumBalls’* movement natural. This is mostly because the real pool balls are more bouncy. Therefore, we need a way to make *Numballs* more bouncy. In Actimator, this is very easy to achieve using the `Bounciness`property of entities.

1. Set the bounciness manually: Select a *NumBall* entity and in `Property Editor`, under `Physics`, change `Bounciness` to “0.8”. Do the same thing for a few balls and run the scene. It looks like it fixed the problem.
2. Set the bounciness programmatically: We can go on with what we were doing in the previous step and change the bounciness of all the *Numballs*. But it’s time-consuming. To save some time, we can program the *NumBalls* to set their bounciness to “0.8” as soon as the game starts.
    - Go to `Actors`.
    - In front of *NumBall* click on `</>`(Program Actors Behavior).
    - Go to `Events` tab, drag and drop `Entity Create` into the `Programming Area`.
    - Go to the `Commands` tab, and find the `Set` command under the `Assignment/Variable`.
    - Drag and drop it inside the `ON_CREATE`event in the `Programming Area`.
    - Find `This Entity` command under the `Actor\Entity`.

- Drag it into the first parameter of the `Set` command that you have already put in `Programming Area`.
- Set the value for `This Entity`command to “Restitution”(Bounciness).
- Set the value for the `Set` command to “0.8”.

The `ON_CREATE` event is triggered whenever an entity of that actor is created. For the existing entities on the scene, as soon as the game starts, which is when you press the play button, the `ON_CREATE` event is triggered for all those entities. This means that when the game starts, the bounciness of all *NumBall* entities will be set to “0.8”.

Run the scene. All the `NumBalls` bounce more naturally now. Do not forget to fix the *CueBall* bounciness: Click on the *CueBall* entity on the scene under the `Property Editor`; change its `Bounciness` to “0.8”.

## Step 7 - Stopping

Run the scene again. As you see, once the balls are hit, they do not come to a stop. In the real world, the balls lose speed gradually due to the friction and finally, they stop. In this step we will find out how to make the balls stop. There is no way in Actimator to define the friction between the balls and the table. The friction is only applied when two entities hit each other, which is either when two balls hit each other or a ball hits the wall. Therefore, we need to simulate the friction by gradually reducing the speed of the balls.

1. Program the *NumBall* decrease of speed:
    - Go to `Actors`.
    - Go to *NumBall* programs.
    - Go to `Events` tab, and drag `Tick`event to `Programming Area`. This event will occur 20 times per second. Therefore, it is a suitable event to program the gradual decrease of speed.

  - We need to check the *NumBall* speed each time and if it’s not zero yet, we need to decrease it a small amount. To check the speed, we will drag the `IF` from the `Control Flow Toolbar`inside the `Tick` event.
  - Go to `Commands` tab, and find `Relational` command under the `Logic` category. Drag and drop it inside the `IF` command’s `Condition`parameter.
  - Find `This Entity` under `Actor/Entity`, and drop it as a first parameter of `Relational`.
  - Set `Relational` operator to “>=”.
  - Set the second parameter of `Relational` to “0.03”.
  - Find `Increase` command under the `Assignment/Variable` category, and drop it into the `IF` body.
  - Change `Increase` command to “Decrease”.
  - Drag a `This Entity` command into the first parameter of `Decrease`command.
  - Set `This Entity` parameter to “speed”.
  - Set decreasing amount to “0.03”.
2. Stop *NumBalls* from rotation: Run the scene; as you can see, the balls will stop moving but they will still rotate due to their *angular velocity*. We need to set their angular speed to zero.
    - Drop under `IF` under the `Tick`event.
    - Set `Relational` as `This Entity`“speed” < “0.03”
    - Find `Set` command under `Assignment/Variable`, and drop it under the `IF` body.
    - Drag a `This Entity` command into the first parameter of `Set`command.
    - Set the parameter of `This Entity`to “Angular Speed”.
    - Leave the value to “0”.

    - Run the scene and check that all the *NumBalls* come to a complete stop now.
3. Repeat all the above steps for “CueBall”.

Run the scene and make sure it’s working as expected.

## Program the Cue

### Step 8 - Program Cue Ball Turning & Shooting

The *CueBall* will just shoot once at the beginning of the game because of its preset speed. But we want to be able to rotate it with right and left arrow keys, and shoot it whenever we click on the `CueBall`.

1. Program the *CueBall* to turn right, when the player presses the right arrow key. To do so, we have to check constantly for whether the right key has been pressed or not, and if so, we rotate the ball in a clockwise direction.
    - Go to *CueBall’s* program.
    - Go to `Events`, and find `Draw (Before Render)` under `Graphics`events. This event will happen 60 times per second, right before refreshing the game screen at every frame. Drop it in the `Programming Area`.
    - Drop an `IF` command from the `Control Flow` toolbar.
    - Find the `Key` command under the `Logic` category and drop it inside the `IF` command’s `Condition`parameter.
    - Set `Key` value to “Right Arrow” by pressing the right arrow key.
    - Find `Increase` command under the `Assignment/Variable` category. Drop it inside the `IF` body.
    - Find `This Entity` command under `Actor/Entity` category and put it inside the first parameter of `Increase`.
    - Set `This Entity` parameter to “Rotation”.
    - Set `Increase` amount to “2”. Increasing the rotation degree causes the entity to rotate clockwise.

2. Program the *CueBall* to turn left, , when the player presses the left arrow key. To do so, we have to check constantly for whether the left key has been pressed or not, and if so, we rotate the ball in a counterclockwise direction.
    - Put another `IF` under the `Draw (Before Render)` event.
    - Drop the `Key`“Left Arrow” command inside the `Condition` parameter of `IF`.
    - Use `Increase` & `This Entity`commands and make this command: `Decrease This Entity` “Rotation” by “2”. Decreasing the rotation degree causes the entity to rotate counterclockwise.
3. Program shooting the *CueBall*. To do so, whenever the player clicks or taps the *CueBall*, we give it speed towards its current direction.
    - Find the `Mouse Down` event under the `Entity Mouse/Touch` events. Drop it inside the `Programming Area`. This event will be triggered when you click or tap on this entity.
    - Drop a `Speed` command inside the event.
    - Set the `Speed` parameter to “50”.
    - Now we have to set the direction of the *CueBall* by setting the `Angle`parameter of the `Speed` command. We need the *CueBall* to move towards its current direction shown by the arrow on the *CueBall*. As the *CueBall* moves during the game, it rotates and the arrow’s direction rotates with it too. Because in the first place, we set the arrow direction to have the same value as the *CueBall’s* `Rotation` parameter, we can use the `Rotation` parameter to get the *CueBall’s* current direction. To do so, drop `This Entity` to the `Angle`parameter of `Speed` command. Set `This Entity` parameter to “Rotation”.

Run the scene and make sure it’s working as expected.

## Step 9 - Cue Ball Rotation Touch Controls

We want our game to be playable on smartphones and tablets too. Using the keyboard during the game is not comfortable on those devices. In this step we are going to add arrow buttons to control the rotation of *CueBall*.

1. Create the arrow button actors:
    - Go to `Actors` tab.
    - Click On `Add Actor` button.
    - Set the `Actor Name` to “Right”.
    - Choose the “Right” for the `Image`.
    - Set the `Has Physics` to “False”. This causes the `Right` button to not collide with any other entity so we can place them anywhere we want on the scene.
    - Press `Create Actor` button.
    - Repeat these steps to make the “Left” actor .
    - Drop and position and entity of each of the *Right* and *Left* actors on the scene.
2. Program the *Right* actor to turn the *CueBall* clockwise when the player clicks/taps on it.
    - Go to the *Right* actor’s program.
    - Go to `Events`.
    - Find `Mouse Down` event under `Entity Mouse/Touch` events. Drop it in the `Programming Area`. This event will be triggered when you click on this entity, which is the *Right* button.
    - Find `Increase` command under `Assignment/Variable` category. Drop it under the event.
    - Find `Entity` command under `Actor/Entity` category and put it as the first parameter of `Increase`.
    - Go back to the scene, select the *CueBall*, and in `Property Editor` find the `Name` value

    - Set `Entity` parameter to the *CueBall*`Name` value.
    - Set `Increase` amount to “2”.
3. Program the *Left* actor to turn the *CueBall* counterclockwise when the player clicks/taps on it. To do so, repeat the same steps as for the *Right* actor, only this time for the *Left* actor. Change the `Increase`command’s first parameter to “Decrease”, so that the ball turns in the reverse direction.

Run the scene and make sure it’s working as expected.

## Step 10 - Create the Pocket Actor and Program Collision

Play the scene. As you can see, the balls do not have any reaction when they hit a pocket. They have to be removed from the scene when they hit a pocket. In this step, we are going to add this feature to our game.

1. Adding *Pockets*: Even though we have the pockets in the pool table image on the scene, they can not have any reaction with the balls. To be able to define behaviors for pockets, we have to create a *Pocket* actor.
    - Go to `Actors`.
    - Click on `Add Actor`.
    - Set the `Name` to “Pocket”.
    - Choose “Pocket” for the `Image`.
    - Set the `Type` to “Static”. So it won’t move when the balls hit it.
    - Set the `Shape` to “Circle”, to make its collision boundary circular.
    - Drop six *Pocket* entities to their positions on the table. You can use the `CTRL` key to copy a *Pocket* entity.
2. Programming *NumBall* reaction to colliding with pockets:
    - Go to `Actors`
    - Go to *NumBall’s* program.
    - Go to `Events`.

        - Find the `Collision Start` event under the `Physics` events, and drop it into the `Programming Area`.
        - Now we need to check if the collided actor is a *Pocket*, we need to remove the ball.
        - Drop `IF` under the event.
        - Under `Commands`, find the `Relational` command under the `Logic` category and drop it into `IF` command's `Condition`parameter.
        - Find the `Event` command under the `Event` category in `Commands`tab. Drop it into the first parameter of `Relational`command.
    - **Leave the `Relational` command's operator as "==".**
            - Set the `Event` command parameter to "Collided Actor".
            - Find `Actor` command under the `Actor/Entity` category. Drop it as the second parameter of `Relational`.
            - Set the `Actor` command parameter to "Pocket".
            - Find `Destroy` command under the `Actor/Entity`category. Drop it inside the `IF` body.
            - Find `This Entity` command under the `Actor/Entity`category. Drop it into the `Destroy` command's first parameter. Change the `This Entity` command's parameter to `Name`.
            - Run the scene and make sure that the *Numballs* are removed as expected.

3. Programming *CueBall* reaction to colliding with pockets: As you see, we still need to fix the *CueBall* so that whenever it hits any pocket it should go back to the start point.
        - Go to `Actors`.
        - Go to *CueBall* program.
        - Go to `Events`.
        - Find the `Collision Start` event under the `Physics` events and drop it into the `Programming Area`.

        - Now we need to check that if the collided actor is a *Pocket*, we have to remove the *CueBall*.
        - Drop the `IF` under the event.
        - Under `Commands`, find the `Relational` command under the `Logic` and drop it into the `IF`parameter.
        - Find `Event` command under the `Event` category in `Commands` tab. Drop it into the first parameter of `Relational`.
    - **Leave the `Relational` command's operator as "==".**
            - Set the `Event` command parameter to "Collided Actor".
            - Find `Actor` command under the `Actor/Entity` category. Drop it into the second parameter of `Relational`command.
            - Set the `Actor` command parameter to "Pocket".
            - Find `Set` command under `Assignment/Variable` category. Drop it into the `IF` body.
            - Find `Entity` command under `Actor/Entity` category and put it as the first parameter of `Set`.
            - Set `Entity` parameter to "Top". Set the amount to "336", which is the starting top position of the *CueBall* in pixels.
            - Do the last 4 steps and set `Left` to "342", which is the starting left position of the *CueBall* in pixels.

# Stop the *CueBall* after hitting the pocket: Run the scene. As you can see, the `CueBall` will be moved to the starting point after the collision. But it still has its linear and angular speed values that causes the ball to move and rotate even after being moved to its starting position. To fix this:

- Do the same steps as for the `Top` and `Left` of the *CueBall* to set its `Speed` to “0”.
- Set the `Angular Speed` to “0”.
- Set the `Rotation` to “90”.

Run the scene and make sure everything works as expected.

## Step 11 - Create Restart Button

In this step, we add a restart button on the scene that takes the game to its starting point, whenever it is clicked or tapped by the player.

1. Adding Restart actor:
    - Go to `Actors`.
    - Click on `Add Actor`.
    - Set the `Name` to “Restart”.
    - Choose “Restart” for the `Image`.
    - Set the `Type` to “Static”, so it won’t move when the balls hit it. We can also set its `Has Physics` to “False” to disable its physics completely.
    - Set the `Shape` to “Circle”, to match its image.
    - Drop it onto the scene and move it to a good position.
2. Programming the `Restart` button: it takes us back to the main scene, whenever the player clicks on it.
    - Go to `Actors`
    - Go to *Restart* program.
    - Go to `Events`.
    - Find `Mouse Down` event under `Entity Mouse/Touch` events, and drop it into the `Programming Area`.
    - Find `Switch Scene` command under the `Scene` category.
    - Drop it into the the `ON_MOUSEDOWN`event.

- Set the `Switch Scene` parameter to “Main”.

Run the scene and make sure it works as it should.

## Step 12 - Add Sound to the Game

In this step, we add some sound to the game. We have 4 preloaded audio files under the `Assets => Audio`:

- Ball & Ball
- Ball & Wall
- NumBall & Pocket
- CueBall & Pocket

In this step we will program the balls to play the sound when they collide.

1. Adding the *NumBall hitting pockets* Audio:
    - Go to `Actors`
    - Go to *NumBall* program. We have already programmed its collision with the *Pocket*.
    - Find `Play Audio` command under the Assets` category, and drop it before the `Destroy` command in the `ON_STARTCONTACT` event.
    - Run the scene and make sure it works as it should.
2. Adding the *NumBall hitting the walls* Audio:
    - Set the `Play Audio` parameter to “NumBall-Pocket”.
    - Go to `Events`.
    - Find the `Collision End` under the `Physics` events and drop it into the `Programming Area`.
    - Drop an `IF` command from the `Control Flow Toolbar` under the event.

- Go to the `Commands`.
- Find `Relational` command under the `Logic` and drop it into `IF`parameter.
- Find `Event` command under the `Event` category in `Commands` tab. Drop it as the first parameter of `Relational`.
- Set the `Event` command parameter to “Collided Actor”.
- Find `Actor` command under the `Actor/Entity` category. Drop it as the second parameter of `Relational`.
- Set the `Actor` command parameter to “VWall”.
- Drop `Play Audio` under the `IF` body.
- Set the `Play Audio` parameter to “Ball-Wall”.
- Do the same for *HWall*.
- Do the Same for *CueBall*, just change the `Play Audio` parameter to “Ball-Ball”.
- Run the scene and make sure it works as it should.

3. Programming CueBall Audio. Similar to previous steps:
    - Under the collision event with pockets, add `Play Audio` and set its parameter to “CueBall-Pocket”.
    - Add two `IF` commands for collision with *VWall* and *HWall*.
    - Add `Play Audio` with “Ball-Wall” under the `IF` bodies.
    - You do not need to program the Ball-Ball collision sound because you have already programmed it under the *NumBall*.

Run the scene and make sure it works as expected.

## Step 13 - Publish Your Game

Congratulations on making it this far! This is the last step of this course. We are going to publish our game on the Web and share it with our friends.

### Prepare the project’s Web page

If you have not done so already, now is the time to set up the project's Web page. The project's Web page is located at *actimator.com/app/[project-domain]*, where *project-domain* is the unique domain for your project. The page contains the game's name, icon, tagline, description, screenshots, etc. It also contains a play button that allows others to play your game online.

We edit the project's Web page from its `Settings` page. To go to the `Settings` page, click on the `Publish` button on the top right side of the Lab. It opens the publish dialog, which contains a link to the project's `Settings` page. Click on the link. From the `Settings` page, we do the following:

1. Choose a proper `Domain`. The domain is unique for each project, and it appears on the game's link. Make sure you choose a nice and easy domain for your project.
2. Write the `Tagline` (short description) and `Description` (long description) for the game.
3. Upload game icon. Create an icon for your game and upload it from here.
4. Upload splash screen. Splash screen is a large image that appears while the game is loading.
5. Upload Screenshots. Take a few screenshots of the game and upload them. Screenshots appear on the project's Web page in a slider.

When you are done, go to the project's Web page and see if it looks great. Do any adjustments that you can to make the page look awesome.

## Save a version

In order to publish our game, we have to save a version of our project. When we save a version, Actimator copies the current project and its assets to a separate location so later on when you change the project, the saved version remains unaffected. This allows us to keep working on our project while people are playing the saved version of our game. Whenever we have a new update to the game, we save it as a new version and publish it.

To save a version:

- Click on the `arrow` icon on the `Save`button. It opens the `Save Version` dialog.
- Enter a number for this version.
- Enter a description for this version.
- Press the `Save a Version` button.

## Publish the game on the Web

Now that we have saved a version of our game, we are ready to make our game available online so that others can play it. To publish our game, we go to the publish page using the link provided in the Publish dialog. On the publish page:

1. Choose the `Web` tab.
2. Choose the version that you just saved.
3. Press the `Publish to Web` button.

Voila! Your game is published. You can see a *Play* button in the project's Web page, under the game icon.

## Share your game

Anyone who visits your game's Web page can play it by clicking the *Play* button. You can share these links with your social network:

- Game Project Page: *https://wwww.actimator.com/app/[your-project-domain]*
- Game Play Page: *https://wwww.actimator.com/app/play/[your-project-domain]*

## Congratulations!

Congratulations for creating and sharing your video game with Actimator. By now, you have gained a solid grasp of how Actimator works. Explore the Actimator Website and make more games. Happy Game Making!