Juiciness

With Breakout clone

Proposed improvements

The improvements listed below were taken from the GDC talk Juice it or lose it by Martin Jonasson and Petri Purho.
Each proposed improvement is presented with the basic description of the task, steps of implementation and an example partial solution (a code snippet).

Add some colors

Change the color of the ball, the paddle, the background (in Camera) or the bricks.
The bricks' color may be changed either for all of them in the prefab or during setup of the level in the Game Manager script.
Random color of the bricks:

```
brick.GetComponent<SpriteRenderer>().color = Random.ColorHSV(0f, 1f, 1f, 1f, 1f, 1f);
```

Color according to the row:

```
Color[] colors = new Color[] { Color.yellow, Color.red, Color.magenta, Color.blue, Color.cyan, Color.green };
brick.GetComponent<SpriteRenderer>().color = colors[row];
```

Screen shake

Shake the camera whenever the ball hits something.

```
Camera.main.DOComplete();
Camera.main.DOShakePosition(0.2f, 0.3f);
```

First we need to complete all unfinished tweens, otherwise we would start a new tween before finishing the previous ones, so the camera would end up offset if the tweens come in quick succession.

Bricks falling down

When the bricks appear on the screen, they should fall down to its place.

```
Brick brick = Instantiate<Brick>(brickPrefab, new Vector3(x, 6, 0), Quaternion.identity, bricksParent);
brick.transform.DOMove(new Vector3(x, y, 0), 0.8f).SetEase(Ease.OutBack);
```

Try different ease settings.
Beware that when restarting the game the bricks get destroyed but the tween tries to keep playing. When destroying the bricks, we should also complete all attached tweens so that we will not get any errors.

```
brick.transform.DOKill(); // in GameController.RemoveBrick
```

Bricks rotating

Rotate the bricks when they are appearing.

```
brick.transform.DORotate(new Vector3(0, 0, 180), 0.8f)
    .SetEase(Ease.OutCirc);
```

Bricks scaling up and down

Change the scale of the bricks while they are appearing. Note that the initial scale is 0.5.

```
Vector3 initialScale = brick.transform.localScale;
brick.transform.DOScale(initialScale * 1.3f, 0.2f).OnComplete(() =>
brick.transform.DOScale(initialScale * 0.3f, 0.4f).OnComplete(() =>
brick.transform.DOScale(initialScale, 0.2f)));
```

Bricks initialized with random delay

Each brick will first wait for some time and then do the initialization (falling, rotation, …).
There are several options we could use to delay the initialization.
We could use tween just to pass some time (set endValue equal to the current value).

```
brick.transform.DOMove(brick.transform.position, Random.Range(0.0f,
0.6f)).OnComplete(() => { …the other tweens… });
```

The actual initialization can be done in the `OnComplete` callback.

> Beware that it is very important we have the brick setup in a separate method. Lambda functions in C# capture the variables, so after the delay all the bricks would go to the same location (they would share the `row` and `col` variables) if we placed the tween with lambda function directly in the for loop.

Or we could use coroutines instead.

```
StartCoroutine(SetupBrick(row, col));
//…
private IEnumerator SetupBrick(int row, int col) {
    // …other stuff…
    yield return new WaitForSeconds(Random.Range(0.0f, 0.6f));
    // …all the tweens…
}
```

Or we could use the `.SetDelay` method from the DOTween plugin and combine it with `.OnStart` callback to start several tweens together.

```
brick.transform.DOMove(new Vector3(x, y, 0), 0.8f)
    .SetDelay(Random.Range(0.0f, 0.6f))
    .OnStart(() => {
        // …other tweens…
    });
```

Or we could use `Sequences` with overlapping tweens from DOTween plugin.

```
Sequence tweenSequence = DOTween.Sequence();
tweenSequence.Append(brick.transform.DOMove(new Vector3(x, y, 0), 0.8f))
    .Insert(0, brickSr.DOFade(1f, tweenSequence.Duration()))
    // …other tweens…
    .SetDelay(Random.Range(0.0f, 0.6f));
```

Bricks fading in

The bricks may also be fading in while appearing.

```
brick.GetComponent<SpriteRenderer>().DOFade(1f, 0.8f).From(0f);
```

Stretchy paddle

We can use the offset of the mouse cursor from where the paddle is to figure out the X and Y scale of the paddle. The paddle will get longer and thinner when the cursor is far (and the paddle is therefore moving fast).

```
// in Paddle.Update
float mouseOffset = Mathf.Clamp(mousePos.x - transform.position.x, 0f, 5f)
/ 5f;
transform.localScale = new Vector3(paddleScale.x + mouseOffset * 0.8f,
paddleScale.y - mouseOffset * 0.5f, 1);
```

Ball scaling up on hit

Whenever the ball hits something, it gets bigger for a short while.

```
transform.DOComplete();
transform.DOScale(initialScale * 1.25f, 0.1f).OnComplete(() =>
transform.DOScale(initialScale, 0.1f));
```

We need to complete all previous tweens to ensure correct working even in a situation of quick succession of collisions.

Ball changing color on hit

Whenever the ball hits something, it changes color for a short while.

```
sr.DOComplete();
sr.DOColor(Color.white, 0.1f).OnComplete(() => sr.DOColor(initialColor,
0.1f));
```

Brick fading out after being hit

When the brick is hit by the ball, it fades out. We should also disable the collider so that the ball cannot collide with the brick while it is disappearing.

```
// in GameController.RemoveBrick
brick.GetComponent<BoxCollider2D>().enabled = false;
brick.GetComponent<SpriteRenderer>().DOFade(0f,
0.3f).SetEase(Ease.InOutQuad).OnComplete(() => Destroy(brick.gameObject));
```

Brick shrinking after being hit

When the brick is hit by the ball, it starts scaling down.

```
brick.transform.DOScale(0f, 0.3f);
```

Brick falling down after being hit

When the brick is hit by the ball, it starts falling down.

```
brick.transform.DOMove(new Vector3(brick.transform.position.x, -6f, 1f),
0.3f);
```

Brick rotating after being hit

When the brick is hit by the ball, it starts rotating.

```
brick.transform.DORotate(Vector3.forward * 270f, 0.3f);
```

Brick changing color after being hit

When the brick is hit by the ball, it changes its color, e.g. to grayscale.

```
SpriteRenderer brickSr = brick.GetComponent<SpriteRenderer>();
float grayscaleValue = brickSr.color.grayscale;
brickSr.color = new Color(grayscaleValue,grayscaleValue,grayscaleValue);
```

Game fade in/out

The object with black overlay is already prepared and referenced from the Game Manager script. Use it to fade out at the end of the game and then fade in again when everything from the previous round is removed and a new round is going to be set up.

```
blackOverlay.DOFade(1f, 0.5f).OnComplete(() => {
      blackOverlay.DOFade(0f, 1f).OnComplete(() => SetupLevel());
});
// destroy previous bricks
```

Winking eyes on the paddle

Add eyes to the paddle and make them wink in random intervals.
Create two new GameObjects under the paddle, assign them sprite Assets/Sprites/eye-outter.png, set Order in Layer (in SpriteRendeder component) to 1. Then add another new GameObject under each of these with sprite Assets/Sprites/eye-inner.png.
Example of good settings:

- outer eyes: position (+/- 0.35, 0, 0), scale (0.35, 0.4, 1)
- inner eyes: position (0, 0.12, 0), scale (1.2, 1.2, 1)

Pass the outer eyes into the Paddle script. Once in a while, scale the eyes in the Y axis to 0 and back.

```
private float eyeWinkCooldown = 2f;
private Vector3 eyeScale; // set in Start to the initial scale

// in Paddle.Update
eyeWinkCooldown -= Time.deltaTime;
if (eyeWinkCooldown < 0) { // check whether the cooldown has passed
    eyeWinkCooldown = Random.Range(0.8f, 2f); // choose a random cooldown
time
    leftEye.transform.DOScaleY(0f, 0.07f).OnComplete(() =>
leftEye.transform.DOScaleY(eyeScale.y, 0.07f));
    rightEye.transform.DOScaleY(0f, 0.07f).OnComplete(() =>
rightEye.transform.DOScaleY(eyeScale.y, 0.07f));
}
```

Rotating the eyes towards the ball

Each eye has separate outer and inner parts so that it is easy to change the position of the inner ones as if the eyes were looking in some direction. Use this to look towards the ball.
Pass the inner eyes into the paddle script and change their position in the Update method to offset them from the center in the direction towards the ball.

```
private float eyeOffset = 0.12f;

// direction from the center of the paddle to the ball, normalized
```

```
Vector3 lookDir = (ball.transform.position -
transform.position).normalized;
// set local position, relative to the parent
leftEyeInner.transform.localPosition = lookDir * eyeOffset;
rightEyeInner.transform.localPosition = lookDir * eyeOffset;
```

Smile on the paddle

Add a smile to the paddle, make it smile when the paddle hits the ball, or frown in horror if the ball gets far away.
Add a new GameObject under the Paddle, assign it a sprite Assets/Sprites/smile.png, set Order in Layer (in SpriteRendeder component) to 1.
Example of good settings: position (0, 0, 0), scale (0.3, 0.02, 1).
Pass the smile object into the Paddle script. Whenever the paddle hits the ball, scale the Y value of the smile for a while.

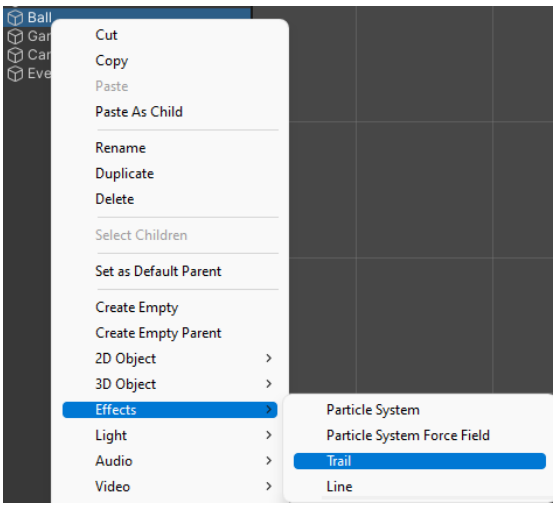
```
private Vector3 smileScale;  // set in Start to the initial scale

smile.transform.DOComplete();
smile.transform.DOScale(new Vector3(smileScale.x, smileScale.y * 9, 1f),
0.3f).OnComplete(() => smile.transform.DOScale(smileScale, 0.8f));
```

If the ball gets far away (check the ball's Y coordinate), scale the Y value of the smile to the negative values. Have bool variables to indicate whether the tween was initiated.

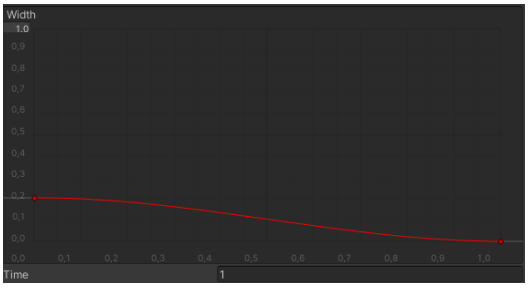
```
// check where is the ball and react accordingly
if (ball.transform.position.y > 1 && !frowningStarted) {
    frowningStarted = true;
    smilingStarted = false;
    smile.transform.DOComplete();
    smile.transform.DOScale(new Vector3(smileScale.x, smileScale.y * -9,
1f), 0.4f);
}
if (ball.transform.position.y < 1 && !smilingStarted) {
    frowningStarted = false;
    smilingStarted = true;
    smile.transform.DOComplete();
    smile.transform.DOScale(smileScale, 0.3f);
}
```

Ball trail

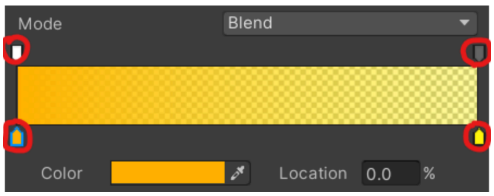
Add a trail behind the ball for some distance. We can use a built-in effect.
Add another GameObject under the Ball, in the menu choose Effects → Trail.



Choose reasonable settings in the Inspector.
Width can be 0.2 at the beginning and 0 at the end, Time 1.



Color gradient going from orange to yellow and alpha from 255 to 100 (you can modify the values by clicking on the marks).

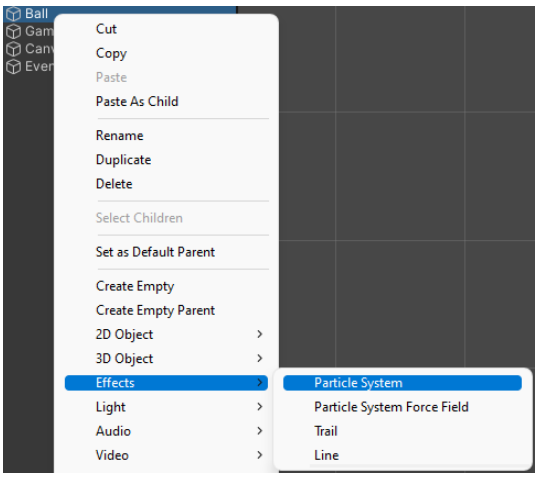


Set the Order in Layer in the ball's SpriteRenderer component to 1, so that the trail is behind.
You may disable and enable the trail from script when the ball is resetting between games.

```
public TrailRenderer trail; // pass in via public field
trail.enabled = false; // in Ball.Reset
trail.enabled = true; // in Ball.StartMoving
```

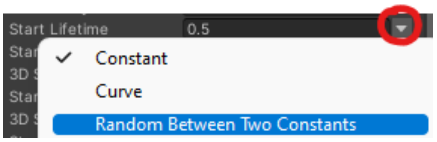
Puff of smoke on ball hit

Use the particle system to add a puff of smoke whenever the ball hits something (paddle, brick, wall).
Add another GameObject under the Ball, in the menu choose Effects → Particle System.



Play with different settings in the Inspector (for more details go to the document containing description of the Particle System). E.g.:

- Particle System module - *Duration* 0.5, *Looping* false, *Start Lifetime* as Random Between Two Constants 0.3 and 0.5, *Start Speed* 0.2, *Start Size* 0.1, *Start Color* as Random Between Two Colors (white and orange), *Simulation Space* World (so that the particles do not move with the ball), *Play On Awake* false.



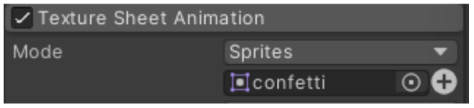
- Emission module - *Rate over Time* 0, *Rate over Distance* 0, in *Bursts* add one burst at time 0.000 with *Count* as Random Between Two Constants 5 and 9.
- Shape module - *Shape* Circle, *Radius* 0.3.
- Renderer module - *Material* Sprites-Default

Pass the Particle System into the Paddle script and call its Play method on collision.

```
public ParticleSystem smoke; // public field

// … in OnCollisionEnter2D
smoke.Play();
```

Confetti when the ball hits the paddle

Similarly to the puff of smoke, create a Particle System object under the Paddle object, set its properties, pass it to the Paddle script and play it on collision.
To add our own sprite for the particles, enable the Texture Sheet Animation module, set Mode to Sprites and add Assets/Sprites/confetti.png into the field.



We can also add trails to the particles by enabling the Trails module and setting the TrailMaterial in the Renderer module to Default-ParticleSystem.
Example of settings:

- Transform - *Rotation* (0,0,65).
- Particle System module - *Duration* 1, *Looping* false, *Start Speed* 10, *Start Size* 0.05, *Start Rotation* as Random Between Two Constants 0 and 180, *Start Color* as

Random Between Two Colors, *Gravity Modifier* 2, *Simulation Space* Local, *Play On Awake* false.
- Emission module - *Rate over Time* 0, *Rate over Distance* 0, one burst in *Bursts* with *Count* 30, and starting at *Time* 0
- Shape module - *Shape* Sphere, *Radius* 0.5, *Arc* 50, *Mode* Random
- Color over Lifetime - alfa at the end set to 0
- Texture Sheet Animation - *Mode* Sprites, Assets/Sprites/confetti.png
- Trails module - *Mode* Particles, *Inherit Particle Color* true
- Renderer module - *Material* Default-ParticleSystem, *TrailMaterial* Default-ParticleSystem

Sounds

Add some sounds to the game, for example when the ball hits the wall, the brick or the paddle. There can also be some background music.
We can use the sounds already prepared in Assets/Sounds:

- music.wav - for the background music
- yay.wav - when the ball hits the paddle
- beep.wav - when the ball hits the brick
- short_beep.wav - when the ball hits the wall

(However you can also choose your own sounds, e.g. from freesound.org. Just remember to keep an eye on the license terms and follow them.)
Add an AudioSource component to the object which should produce the sound (create an empty object for the music), assign it the AudioClip and set Volume. Leave the Play On Awake and Loop options checked only for the background music, the other sound will be controlled from a script (a new script will be created for the walls).

```
private AudioSource audioSource; // field to store the AudioSource

// in Start
audioSource = GetComponent<AudioSource>();

// wherever necessary (e.g. in OnCollisionEnter2D)
audioSource.PlayOneShot(audioSource.clip);
```

We could also use the AudioSource.Play() method but that would not allow overlapping sounds (which is necessary e.g. for hitting the paddle in a quick succession) because it cancels the already playing clip.