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, useful methods and example code snippets.

Don't forget you need to add `using DG.Tweening;` when using methods from the DOTween plugin.

Adding some colors

Change the color of the ball, the paddle, the walls (in SpriteRenderer), 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 Controller script.

Random color of the bricks - `Random.ColorHSV()` may come in handy, e.g.:

```
Color randomColor = Random.ColorHSV(0f, 1f, 1f, 1f, 1f, 1f);
```

Color according to the row - All bricks in the same row will have the same color.

Camera shake

Shake the camera whenever the ball hits something.
You may need:

- `Camera.main` - Provides reference to the enabled `Camera`.
- `Camera.DOShakePosition(...)` - Shakes the `Camera`'s position.
- `Component.DOComplete()` - Completes all tweens on the target.

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 setup

When the bricks appear on the screen, they should do something interesting, e.g. fall down to its place, rotate, scale up and/or down, fade in. We can also initialize each brick with a random delay.
You may need:

- `Transform.DOMoveY(...)` - Tweens the `Transform`'s Y position.
- `Transform.DOMove(...)` - Tweens the `Transform`'s position.

- `Transform.DORotate(...)` - Tweens the `Transform`'s rotation.
- `Transform.DOScale(...)` - Tweens the `Transform`'s scale.
- `SpriteRenderer.DOFade(...)` - Tweens the `SpriteRenderer`'s alpha value.

If you want to set the initial value, you may use the `From(...)` method, e.g.:

```
tweenedSpriteRenderer.DOFade(0f, 0.5f).From(1f);
```

Try different ease settings using the `SetEase(Ease ease)` method, e.g.:

```
tweenedTransform.DOMoveY(...).SetEase(Ease.OutBack);
```

Beware that when restarting the game during the setup the bricks get destroyed but the tween tries to keep playing. When destroying the bricks, we should also complete (or kill) all attached tweens so that we will not get any errors, e.g.:

```
tweenedComponent.DOKill(); // kills all tweens on the target
```

We can also chain the tweens using the `OnComplete(TweenCallback action)` method. The callback is fired when the tween is finished and we may use lambda functions, e.g.:

```
tweenedTransform.DOScale(1.3f, 0.3f)
    .OnComplete(() => tweenedTransform.DOScale(1f, 0.3f));
```

To first wait for some random amount of time before the initialization (falling, rotating, …) there are several options:

1. Use a tween to pass some time (set `endValue` equal to the current value), then do the initialization in `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.

2. Use coroutines.
3. Use the `SetDelay(...)` method from the DOTween plugin and combine it with OnStart(...) callback to start several tweens together, e.g.:

```
tweenedTransform.DOMove(...)
    .SetDelay(...)
    .OnStart(() => {
        // …other tweens…
    });
```

4. Use Sequences with overlapping tweens from DOTween plugin, e.g.:

```
Sequence tweenSequence = DOTween.Sequence();
tweenSequence.Append(tweenedTransform.DOMove(...))
    .Insert(0, tweenedSprite.DOFade(..., tweenSequence.Duration()))
    // …other tweens…
    .SetDelay(...);
```

A random number may be generated using e.g. `Random.Range(...,...)`.

Juicy ball hit

Whenever the ball hits something (bricks, wall), it can do something interesting, for example get bigger and change color for a very short time.
You may need:

- `Transform.DOScale(...)` - Tweens the `Transform`'s scale.
- `SpriteRenderer.DOColor(...)` - Tweens the `SpriteRenderer`'s color.
- `Component.DOComplete()` - Completes all tweens on the target.
- `OnComplete(TweenCallback action)` - For chaining tweens as described here.

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

Juicy brick hit

When the brick is hit by the ball, we don't have to destroy it straight away, instead we can do something more interesting (fade out, scale down, fall down, rotate and/or change color).
You may need:

- `SpriteRenderer.DOFade(...)` - Tweens the `SpriteRenderer`'s alpha value.
- `Transform.DOScale(...)` - Tweens the `Transform`'s scale.
- `Transform.DOMove(...)` - Tweens the `Transform`'s position.
- `Transform.DORotate(...)` - Tweens the `Transform`'s rotation.
- `SpriteRenderer.DOColor(...)` - Tweens the `SpriteRenderer`'s color.
- `OnComplete(TweenCallback action)` - For a callback as described here.

Class `Color` has a property called `grayscale` which contains the color's grayscale value and may be used in the following way:

```
// variable spriteRenderer contains the brick's SpriteRenderer component
float grayscaleValue = spriteRenderer.color.grayscale;
brickSr.color = new Color(grayscaleValue,grayscaleValue,grayscaleValue);
```

We should also disable the collider so that the ball cannot collide with the brick while it is disappearing:

```
brick.GetComponent<BoxCollider2D>().enabled = false;
```

Try different ease settings using the `SetEase(Ease ease)` method, e.g.:

```
tweenedTransform.DOMoveY(...).SetEase(Ease.OutBack);
```

Don't forget to destroy the brick object when all the tweens are finished.

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).

The mouse position is already stored in the `mouPos` field of the `Paddle` script.

Paddle face

To make the paddle more lively we can add a face to it, including:

- Eyes winking in random intervals and rotating towards the ball.
- A smile getting bigger when the paddle hits the ball and frowning in horror when the ball gets far away.

Suitable sprites are already prepared in the `Assets/Sprites` folder.

You may need:

- `Random.Range(...,...)` - Generates a random number in the given range.
- `Transform.DOScale(...)` - Tweens the `Transform`'s scale.
- `Transform.DOScaleY(...)` - Tweens the `Transform`'s Y scale.
- `Vector3.normalized` - To get the `Vector3` normalized.
- `Transform.localPosition` - Get/set the position relative to the parent transform.
- `Transform.position` - Get/set the world space position of the `Transform`.
- `Component.DOComplete()` - Completes all tweens on the target.
- `OnComplete(TweenCallback action)` - For chaining tweens as described here.

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. Quickly scaling the eyes in the Y axis looks like winking.

By scaling the smile in the Y axis to negative values you can make it look frowning.

We need to complete all previous tweens before starting a new one to ensure correct working even in a situation of quick succession of tweens.

Game fade in/out

The object with black overlay is already prepared and referenced from the `GameController` script (field `blackOverlay`). 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.
You may need:

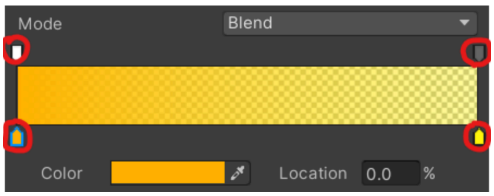
- `Image.DOFade(...)` - Tweens the `Image`'s alpha value.
- `OnComplete(TweenCallback action)` - For a callback as described here.
- `SetDelay(float delay)` - To start the tween with the given delay.

Ball trail

Add a trail behind the ball for some distance. We can use a built-in Trail effect.

Right-click the object in the Hierarchy → Effects → Trail. Then choose reasonable settings in the `TrailRenderer` component in the Inspector, e.g.:

- `Width` getting smaller in time.
- `Color` gradient goes from one color to another (you can modify the values by clicking on the marks).



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;      // when resetting
trail.enabled = true;       // when starting
```

Puff of smoke

Use the particle system to add a puff of smoke whenever the ball hits something (paddle, brick, wall).
Right-click the object in the Hierarchy → 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`, `Looping`, `Start Lifetime`, `Start Speed`, `Start Size`, `Start Color`, `Simulation Space`, `Play On Awake`.
- *Emission* module - `Rate over Time`, `Rate over Distance`, `Bursts` (`Count`).
- *Shape* module - `Shape`, `Radius`.
- *Renderer* module - `Material`.

To play the effect use its `Play()` method from a script, e.g.:

```
public ParticleSystem smoke; // public field
// ...
smoke.Play();
```

Confetti

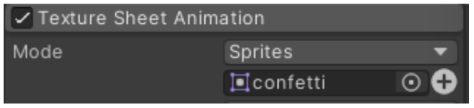
Whenever the ball successfully hits the paddle, launch some confetti to reward the player.
Right-click the object in the Hierarchy → Effects → Particle System.

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

- `Transform` - `Rotation`.

- *Particle System* module - `Duration`, `Looping`, `Start Speed`, `Start Size`, `Start Rotation`, `Start Color`, `Gravity Modifier`, `Simulation Space`, `Play On Awake`.
- *Emission* module - `Rate over Time`, `Rate over Distance`, `Bursts` (`Time`, `Count`).
- *Shape* module - `Shape`, `Radius`, `Arc`, `Mode`.
- *Color over Lifetime* module - `Color`.
- *Texture Sheet Animation* module - `Mode`.
- *Trails* module - `Mode`, `Inherit Particle Color`.
- *Renderer* module - `Material`, `TrailMaterial`.

To add our own sprite for the particles, enable the *Texture Sheet Animation* module, set `Mode` to `Sprites` and add the desired sprite into the field. You can use an already prepared sprite `confetti` in `Assets/Sprites`.



We can also add trails to the particles by enabling the *Trails* module and setting the `TrailMaterial` in the *Renderer* module to `Default-ParticleSystem`.

To play the effect use its `Play()` method from a script, e.g.:

```
public ParticleSystem confetti; // public field
// ...
confetti.Play();
```

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.

You may 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 may choose your own sounds as well, e.g. from freesound.org. Just remember to keep an eye on the license terms and follow them.)

For playing sounds we need an `AudioSource` component on the object which should produce the sound. Then we should assign it the `AudioClip` and eventually set other parameters (`Volume`, `Play On Awake`, `Loop`).
You may need:
- `AudioSource.Play()` - Plays the `AudioClip` assigned to the `AudioSource`. Another call while the previous one is still playing will result in canceling the already playing clip.

- `AudioSource.PlayOneShot(AudioClip clip)` - Plays the `AudioClip` assigned to the `AudioSource` but allows overlapping sounds as opposed to `Play()`.

```
audioSource.PlayOneShot(audioSource.clip); // to play the assigned clip
```

Slow motion

To enjoy all the effects even more, we can add slow motion - whenever a particular key (e.g. Space) is pressed, everything slows down. The slowing down and speeding up should be smooth using tweening.
You may need:

- `Time.timeScale` - Get/set the scale at which time passes, e.g.:

```
if (Input.GetKeyDown(KeyCode.Space))
    Time.timeScale = 0.1f;  // slow the game ten times
if (Input.GetKeyUp(KeyCode.Space))
    Time.timeScale = 1f;  // return speed back to normal
```

Apart from the useful extension methods for Unity components, there is also a very generic way of tweening which allows us to tween almost any value:

- `static DOTween.To(getter, setter, to, float duration)`

It may be used with lambda functions for tweening a numerical value in the following way:

```
private float tweenedNumber;  // some data field

DOTween.To(
    () => tweenedNumber,     // getter, returns the field's value
    x => tweenedNumber = x,  // setter, sets the value x to the field
    0.5f,                    // the end value
    1f                       // the duration
);
```

Try different ease settings using the `SetEase(Ease ease)` method, e.g.:

```
DOTween.TO(...).SetEase(Ease.OutQuart);
```