Week_22

02/28/2024

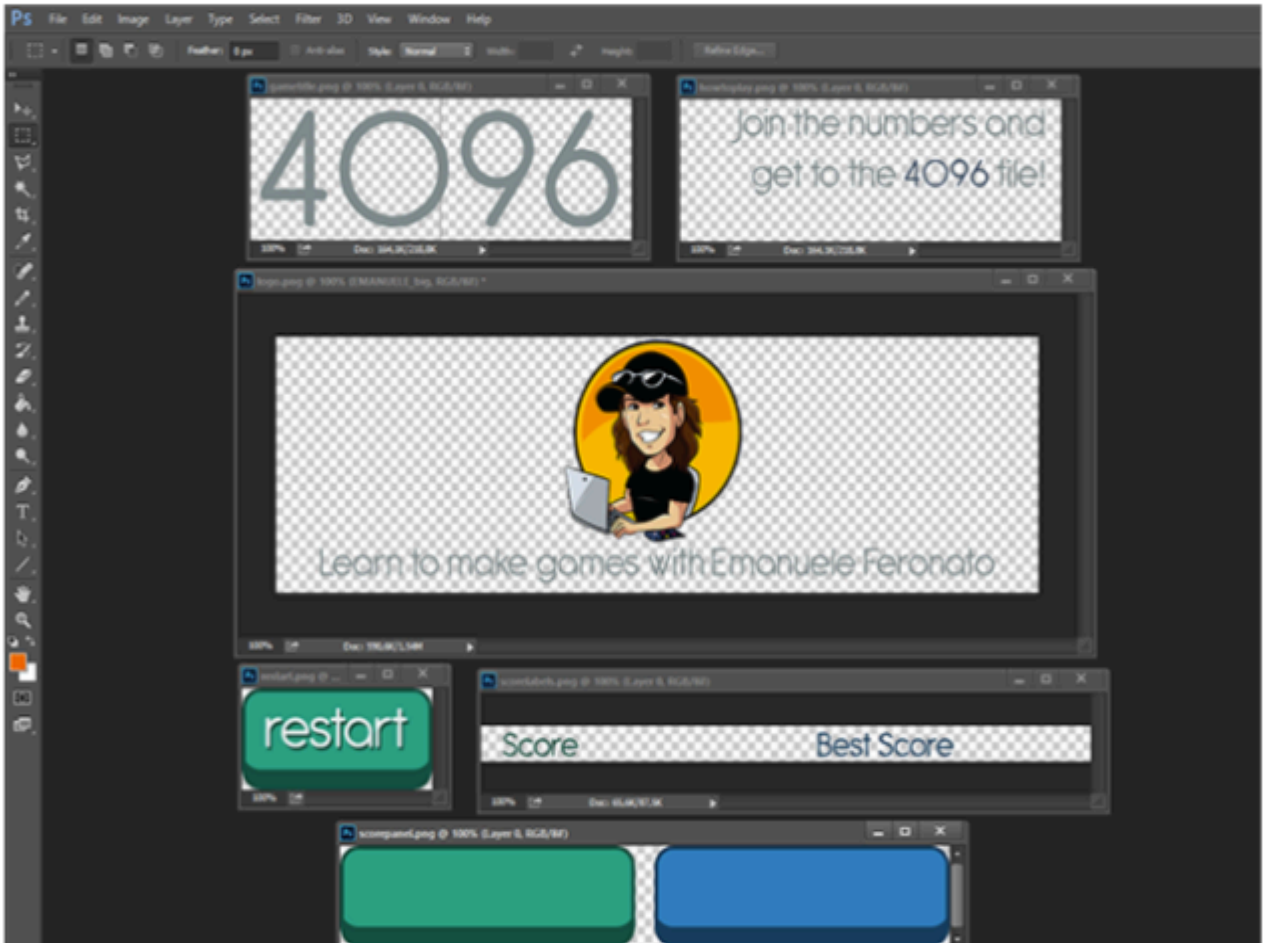
U.I.

U.I. stands for User Interface, and is very important
With every gaming software program. The way your
Users interact with the software, determines the
Experience they have. User Interface goes beyond the
Way your game is played, U.I. includes data (information)
To and from your user. Examples include: status bar,
Configuration and setup screens, features and benefits,
Game or player supplemental information…

You have seen software that has a (below average) U.I. Today we will improve our game's User Interface. Then make the actions interactive.

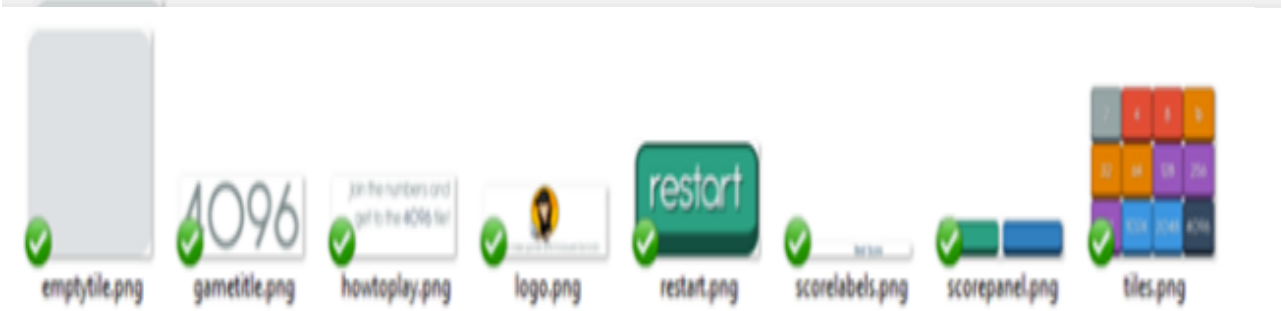
Currently our User Interface is just the game itself:

Lets see what we can do to improve upon this:

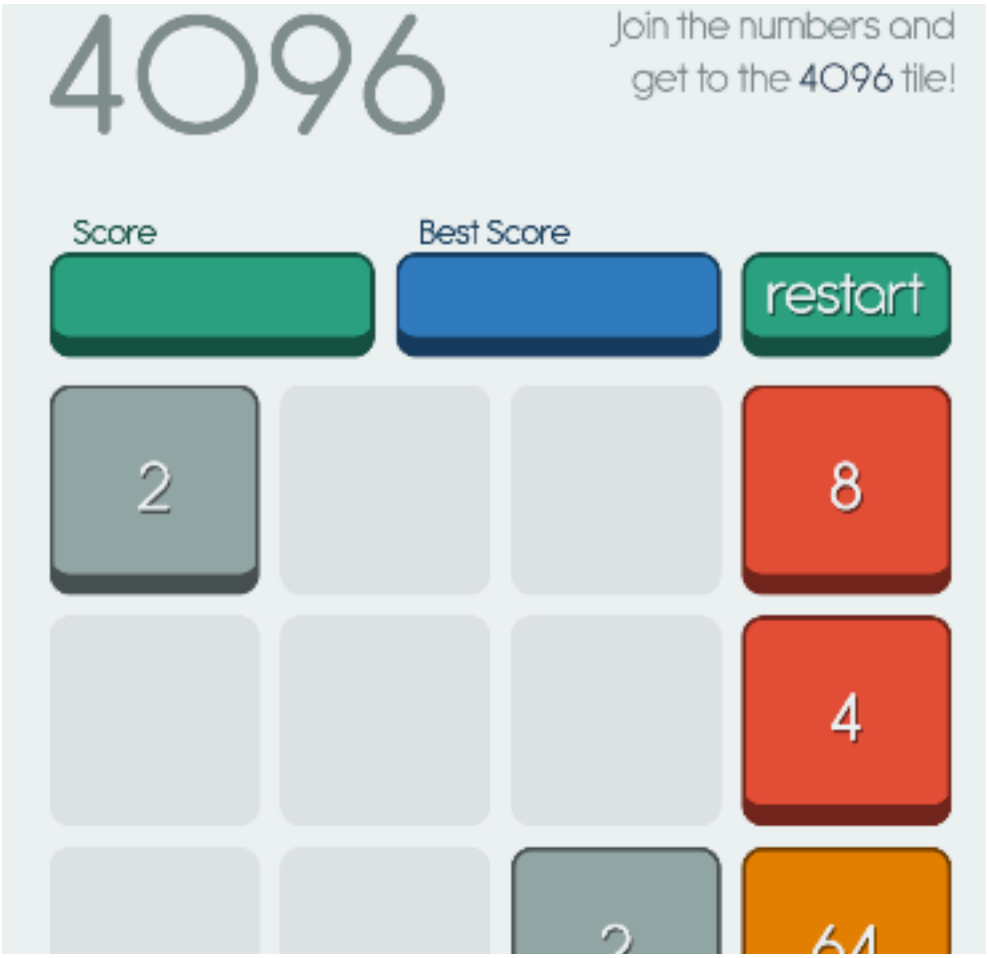


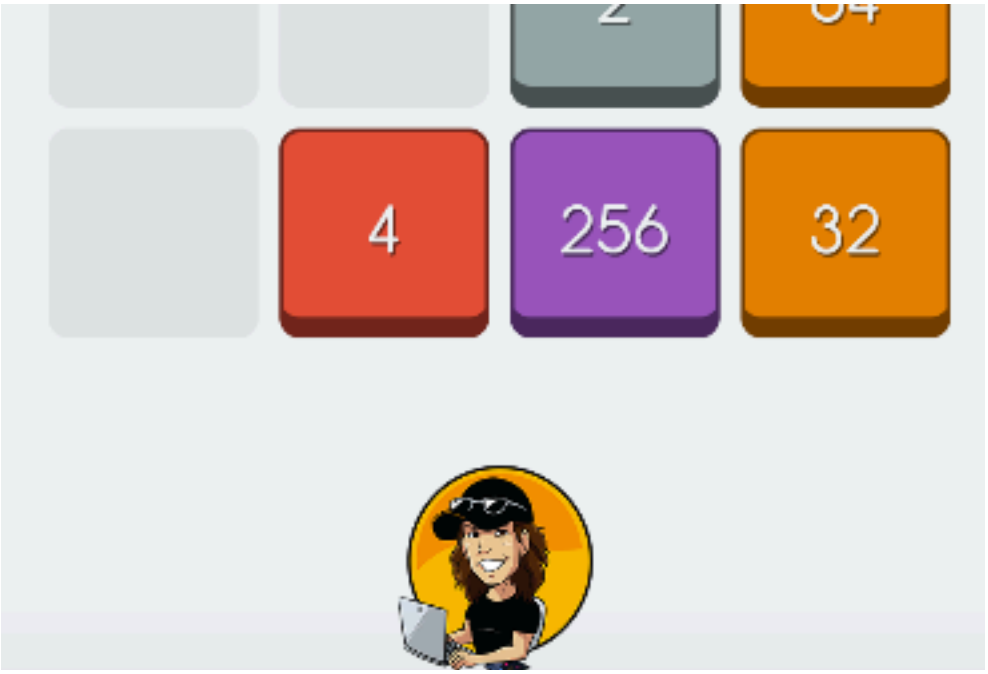
We have the game title, the game instructions, my branded logo, a restart button, a single image two rectangles where to display current score and best score, and a single image with two labels.

All these images will be saved in the good old sprites folder inside assets folder, which now should look this way:



Here is a sample from Coach Arthur





This is the right time for you to unleash your creativity and draw your own assets.

Keep in mind all your images should have the same theme, or your game will look like early 1990 websites with a lot of random images popping here and there.

Also, you should draw buttons and panels which fit with each row/column board configuration, or at least the most common ones.

Then we add the required lines to `preload` method to preload the new images.

```
preload(){
    this.load.image("restart", "assets/sprites/restart.png");
    this.load.image("scorepanel", "assets/sprites/scorepanel.png");
    this.load.image("scorelabels", "assets/sprites/scorelabels.png");
    this.load.image("logo", "assets/sprites/logo.png");
    this.load.image("howtoplay", "assets/sprites/howtoplay.png");
    this.load.image("gametitle", "assets/sprites/gametitle.png");
    this.load.image("emptytile", "assets/sprites/emptytile.png");
    this.load.spritesheet("tiles", "assets/sprites/tiles.png", {
        frameWidth: gameOptions.tileSize,
        frameHeight: gameOptions.tileSize
    });
    this.load.audio("move", ["assets/sounds/move.ogg", "assets/sounds/move.mp3"]);
    this.load.audio("grow", ["assets/sounds/grow.ogg", "assets/sounds/grow.mp3"]);
}
```

In `create` method we'll add to the stage the images we just preloaded

In `create` method we'll add to the stage the images we just preloaded.

Although it's only a matter of adding images, something you should already know, there is a couple of new concepts.

These are the new lines, we'll add them at the very beginning of `create` method.

```
create(){
    var restartXY = this.getTilePosition(-0.8, gameOptions.boardSize.cols - 1);
    var restartButton = this.add.sprite(restartXY.x, restartXY.y, "restart");
    var scoreXY = this.getTilePosition(-0.8, 1);
    this.add.image(scoreXY.x, scoreXY.y, "scorepanel");
    this.add.image(scoreXY.x, scoreXY.y - 70, "scorelabels");
    var gameTitle = this.add.image(10, 5, "gametitle");
    gameTitle.setOrigin(0, 0);
    var howTo = this.add.image(game.config.width, 5, "howtoplay");
    howTo.setOrigin(1, 0);
    var logo = this.add.sprite(game.config.width / 2, game.config.height, "logo");
    logo.setOrigin(0.5, 1);
    // same as before
}
```

Let's explain these new lines one by one.

We want to place the restart button above the rightmost column, so here is how we calculate restart button position:

```
var restartXY = this.getTilePosition(-0.8, gameOptions.boardSize.cols - 1);
```

`getTilePosition` method also accepts non-integers, non-negative numbers as argument. The method simply calculates a `Point` with `x` and `y` coordinates, it's up to you to adjust the values of its arguments to make your restart button look nice.

With these settings, restart button – which has been drawn with the same length as a tile – will be placed some pixels above the rightmost column, in the position stored in `restartXY` variable.

```
var restartButton = this.add.sprite(restartXY.x, restartXY.y, "restart");
```

And now we just added the button to the canvas.

```
var scoreXY = this.getTilePosition(-0.8, 1);
```

Same thing for `scoreXY` variable, will store the position of the score panel, which has been with the same length as three tiles and two tile spacings.

```
this.add.image(scoreXY.x, scoreXY.y, "scorepanel");
```

And we added the score panel.

```
this.add.image(scoreXY.x, scoreXY.y - 70, "scorelabels");
```

On top of the score panel we added the score labels.

Why having two images, one for the score panel and one for the score labels, rather than one single image with both score panels and score labels in it?

It's a matter of making the game customizable. I prefer to keep hardcoded text away from other graphic assets, should I need to change it or remove it I don't have to worry about the rest of the assets.

```
var gameTitle = this.add.image(10, 5, "gametitle");
```

The game title is added at the very top of the canvas.

```
gameTitle.setOrigin(0, 0);
```

And we set its origin, or anchor point, on the top left corner.

> The **anchor** or **origin** of an image or sprite sets the origin point of the image.
>
> When you add an image at `x, y` you actually add an image in a position so that its origin is `x, y`.
>
> Setting the origin to `0.5, 0.5` – which is also the default value – means the image origin is the center of the image.
>
> Setting the origin to `1, 1` means the image origin is on the bottom right corner.
>
> Setting the origin to `0, 0` means the image origin is on the top left corner.
>
> Any value from 0 to 1 is accepted.
>
> `setOrigin(x, y)` method sets image origin to `x, y`.

Game instructions will be placed on the opposite side of the title, at the rightmost side of the canvas:

```
var howTo = this.add.image(game.config.width, 5, "howtoplay");
```

With the origin set on the top right corner.

```
howTo.setOrigin(1, 0);
```

And finally the logo, placed in the bottom center of the canvas, with its origin also in its bottom center:

```
var logo = this.add.sprite(game.config.width / 2, game.config.height, "logo");
logo.setOrigin(0.5, 1);
```

We aren't just building a HTML5 cross platform game, we are also allowing room for some customization without adding exceptions to the code.

Carefully designing your interface will allow you to save a lot of time.

Making sprites interactive

Having a restart button is quite useless if you do not restart the game when clicking on it.

Adding interactivity on sprites turning them into buttons is very easy, a matter of a couple of lines of code, to be added to create method.

Next:

Handling a score system

The original 2048 game has a simple yet challenging score system: each time you

upgrade a tile, your score increases by the value of the upgraded tile.

If you merge two "2" tiles into a "4", you will get 4 points. If you merge two "16" tiles into a "32", you will get 32 points, and so on.

First, a new property needs to be added in `create` method of `playGame` class:

```
create(){
    this.score = 0;
    // same as before
}
```

`score` will keep count of the score made during the game, and it starts at zero.

Inside `makeMove` method in the routine which checks if the tile value needs to be updated, we'll add the line which handles the score:

```
if(newRow != curRow || newCol != curCol){
    var newPos = this.getTilePosition(newRow, newCol);
    var willUpdate = this.boardArray[newRow][newCol].tileValue == tileValue;
    this.moveTile(this.boardArray[curRow][curCol].tileSprite, newPos, willUpdate);
    this.boardArray[curRow][curCol].tileValue = 0;
    if(willUpdate){
        this.boardArray[newRow][newCol].tileValue ++;
        this.score += Math.pow(2, this.boardArray[newRow][newCol].tileValue);
        this.boardArray[newRow][newCol].upgraded = true;
    }
    else{
        this.boardArray[newRow][newCol].tileValue = tileValue;
    }
}
```

When a tile updates its value, the score is increased by the power of two of the new tile value.

> `Math.pow(base, exponent)` JavaScript method returns `base` to the `exponent`

power.

Finally, each time we refresh the board, we also refresh the score, adding one more line at the beginning of `refreshBoard` method:

```
refreshBoard(){
    this.scoreText.text = this.score.toString();
    // same as before
}
```

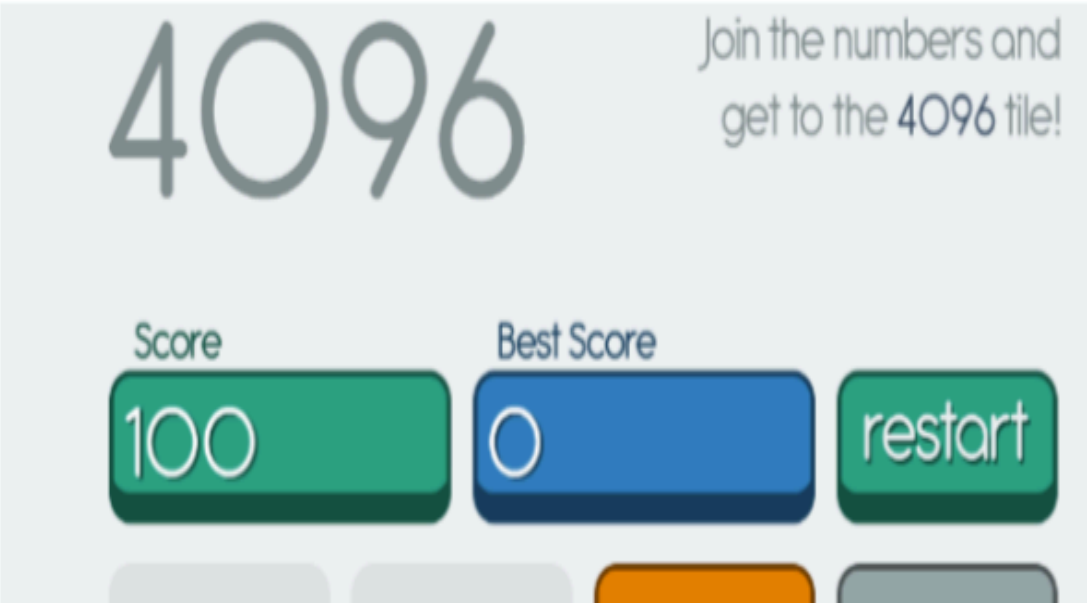
And now `scoreText` bitmap text is updated.

`text` property of a bitmap text sets the text to show.

`toString` JavaScript method converts a number to a string.

Now play the game and see your score increase as you merge tiles:

Now play the game and see your score increase as you merge tiles:



If you restart the game, score also restarts from zero.

Saving the best score

When playing a game, you will quickly realize there is no point in making a great score if you can't save it and try to beat it later.

We are going to cover how to save your best score, and keep it saved even if you close the browser window or turn off your computer or device.

All modern browsers support local storage, a way used by web pages to locally store data in a key/value notation.

The information you save will continue to be stored even when you shut down your device and can be read every time you launch your game.

This is exactly what we need.

Let's create a new item in `gameOptions` global object:

```
var gameOptions = {
    tileSize: 200,
    tileSpacing: 20,
    boardSize: {
        rows: 4,
        cols: 4
    },
    tweenSpeed: 50,
    swipeMaxTime: 1000,
    swipeMinDistance: 20,
    swipeMinNormal: 0.85,
    aspectRatio: 16/9,
    localStorageName: "topscore4096"
}
```

`localStorageName` item stores the name of the local storage variable, so each

time you will change `topscore4096` with something else, you will reset your best score.

We are only saving the best score at the moment, but you can save anything you want: the number of games played, the total time spent playing the game, and so on.

As soon as we start the game, we have to check if there is a high score saved in local storage, so we are going to add a few lines to `create` method:

```
create(){
    this.score = 0;
    var restartXY = this.getTilePosition(-0.8, gameOptions.boardSize.cols - 1);
    var restartButton = this.add.sprite(restartXY.x, restartXY.y, "restart");
    restartButton.setInteractive();
    restartButton.on("pointerdown", function(){
        this.scene.start("PlayGame");
    }, this);
    var scoreXY = this.getTilePosition(-0.8, 1);
    this.add.image(scoreXY.x, scoreXY.y, "scorepanel");
    this.add.image(scoreXY.x, scoreXY.y - 70, "scorelabels");
    var textXY = this.getTilePosition(-0.92, -0.4);
    this.scoreText = this.add.bitmapText(textXY.x, textXY.y, "font", "0");
    textXY = this.getTilePosition(-0.92, 1.1);
    this.bestScore = localStorage.getItem(gameOptions.localStorageName);
    if(this.bestScore == null){
        this.bestScore = 0;
    }
    this.bestScoreText = this.add.bitmapText(textXY.x, textXY.y, "font",
        this.bestScore.toString());
    // same as before
}
```

Let's see what we did:

```
this.bestScore = localStorage.getItem(gameOptions.localStorageName);
```

A new property called `bestScore` is added and gets the value currently saved in local storage.

> `localStorage.getItem(keyName)` method of local storage returns `keyName`'s value or `null` if `keyName` does not exist.
>
> `null` represents JavaScript's intentional absence of any value.

At this time `bestScore` can have a value o can be `null`.

```
if(this.bestScore == null){
    this.bestScore = 0;
}
```

Actually, `null` is not a high score to be proud of, so we are setting `bestScore` to zero if we have a `null` value.

Now we just have to show the best score in `bestScoreText` bitmap text.

```
this.bestScoreText = this.add.bitmapText(textXY.x, textXY.y, "font",
        this.bestScore.toString());
```

At the beginning of the game, the best score – or zero – will be displayed in the proper place.

When to update it? After each successful move, when it's time to refresh the board, in `refreshBoard` method:

```
refreshBoard(){
    this.scoreText.text = this.score.toString();
    if(this.score > this.bestScore){
        this.bestScore = this.score;
        localStorage.setItem(gameOptions.localStorageName, this.bestScore);
        this.bestScoreText.text = this.bestScore.toString();
```

```
    }
    for(var i = 0; i < gameOptions.boardSize.rows; i++){
        for(var j = 0; j < gameOptions.boardSize.cols; j++){
            var spritePosition = this.getTilePosition(i, j);
            this.boardArray[i][j].tileSprite.x = spritePosition.x;
            this.boardArray[i][j].tileSprite.y = spritePosition.y;
            var tileValue = this.boardArray[i][j].tileValue;
            if(tileValue > 0){
                this.boardArray[i][j].tileSprite.visible = true;
                this.boardArray[i][j].tileSprite.setFrame(tileValue - 1);
                this.boardArray[i][j].upgraded = false;
            }
            else{
                this.boardArray[i][j].tileSprite.visible = false;
            }
        }
    }
    this.addTile();
}
```

When will you beat a best score? When your current score is higher than the best score. This is exactly what we are checking in this `if` statement.

```
if(this.score > this.bestScore){
    // rest of the script
}
```

Once `score` is greater than `bestScore`, first we update `bestScore`:

```
this.bestScore = this.score;
```

Then we save the best score in local storage:

```
localStorage.setItem(gameOptions.localStorageName, this.bestScore);
```

We are saving best score as soon as it updates, so if for some reason the player quits the game before it ends, the best score has already been saved.

`localStorage.setItem(keyName, keyValue)` method adds keyName to the

`localStorage.setItem(keyName, keyValue)` method adds `keyName` to the storage, or updates it to `keyValue` if it already exists.

Finally, we update the best score bitmap text:

```
this.bestScoreText.text = this.bestScore.toString();
```

Test the game, quit it anytime, restart your device, your high score will remain.