DialogueLine

¶

class

DialogueLine

extends

DialogueEntry

The

DialogueLine

class represents a single line of text within the dialogue

Line

.

The

DialogueLine

objects will be delivered to your

DialogueView

with methods

onLineStart()

,

onLineSignal()

,

onLineStop()

, and

onLineFinish()

.

A dialogue line may contain a

character

(the name of the entity who is speaking), and

tags

? a list of hashtag tokens that specify some meta information about the line.

Example of a dialogue line in yarn script:

Hermione

:

Holy cricket! You're Harry Potter.  #surprised

Here the

character

is ?Hermione?, the

text

of the line is ?Holy cricket! You?re Harry Potter.?, and the

tags

list will contain a single entry ?#surprised?.

A dialogue line may also contain inline expressions, which will be re-evaluated every time the line is execut

Jenny

:

My favorite color is {$favoriteColor}, what about you?

After evaluation, the resulting string may be ?My favorite color is vantablack, what about you??.

Lastly, a dialogue line may have markup attributes. These are similar to HTML tags, only they use square b

Jenny

:

My [i]favorite[/i] color is [bb color=$color]{$color}[/bb].

These markup attributes will not be visible in the output (i.e. the resulting text is still ?My favorite color is va

attributes

list, which will specify that there is an attribute

[i]

around the word ?favorite?, and another attribute

[bb]

with parameter

color

around the word ?vantablack?.

Inline expressions cannot contain markup attributes.

Constructors

¶

DialogueLine

(

{

required

LineContent

content

,

Character?

character

,

List<String>?

tags

}

)

Properties

¶

content

?

LineContent?

The content of this Line.

character

?

Character?

The character who is speaking the line. This can be null if the line does not contain a speaker.

text

?

String

The computed text of the line, after substituting all inline expressions, stripping the markup, and processing

This value can only be accessed after the line was

evaluate

d. It may change upon subsequent re-evaluations of the line (which occur each time the line goes through a

DialogueRunner

).

tags

?

List<String>

The list of hashtags associated with the line. If there are no hashtags, the list will be empty.

Each value in the list will start with the

#

symbol.

attributes

?

List<MarkupAttribute>

The list of markup spans associated with the line.

isConst

?

bool

True if the line will never change upon subsequent reruns. That is, when the line does not depend on any d

Methods

¶

evaluate

(

)

Computes the

text

of the line, substituting the current values of all inline expressions.

Normally, you wouldn?t need to call this method manually ? the

DialogueRunner

will take care to do that for you. However, it may be necessary to call this if you need to access

DialogueLine

s outside of a dialogue runner.

Flame fire atlas

¶

Flame fire atlas is a texture atlas lib for Flame. By using

flame_fire_atlas

one can access images and animations stored in a

.fa

texture atlas by referring to them by their named keys.

FireAtlas

¶

FireAtlas is a tool for handling texture atlases. Atlases can be created using the

Fire Atlas Editor

.

Creating Atlas

¶

To create a texture atlas open

Fire Atlas Editor

.

Select new atlas and give the atlas a name, tile width, tile height and an image and press okay. This will tal

To create a new

Sprite

in the atlas, select a portion and click the plus button on top left and give the selection a name and then sel

Sprite

and press

Create

Sprite

. You can now see a preview in the right panel of editor.

To create a new

SpriteAnimation

in the atlas, select a portion and click the plus button on top left and give the selection a name and then sel

Animation

and provide

frame

count

and

steps

times

(in

milliseconds)

and select the checkbox to loop the animation, and then press

Create

Animation

. You can now see a preview of the animation in the right panel of the editor.

Once you are done with editing you can download the fire atlas file from top left with the

download

icon button.

Texture atlas

¶

A

Texture atlas

is an image that contains data from several smaller images that have been packed together to reduce over

Usage

¶

To use the bridge library in your game you just need to add

flame_fire_atlas

to your pubspec.yaml, as can be seen in the

Flame Fire Atlas example

and in the pub.dev

installation instructions

.

Then you have the following methods at your disposal:

import

'package:flame_fire_atlas/flame_fire_atlas.dart'

;

// Load the atlas from your assets

// file at assets/atlas.fa

final

atlas

=

await

FireAtlas

.

loadAsset

(

'atlas.fa'

);

//or when inside a game instance, the loadFireAtlas can be used:

// file at assets/atlas.fa

final

atlas

=

await

loadFireAtlas

(

'atlas.fa'

);

// Get a Sprite with the given key.

FireAtlas

.

getSprite

(

'sprite_name'

)

// Get a SpriteAnimation with the given key.

FireAtlas

.

getAnimation

(

'animation_name'

)

To use FireAtlas in your game, load the fire atlas file in an

onLoad

method, either in your game or a component. Then you can use

getSprite

and

getAnimation

to retrieve the mapped assets.

class

ExampleGame

extends

FlameGame

{

late

FireAtlas

_atlas

;

@override

Future

<

void

>

onLoad

()

async

{

_atlas

=

await

loadFireAtlas

(

'atlas.fa'

```
);
add
(
SpriteComponent
(
size:
Vector2
(
50
,
50
),
position:
Vector2
(
0
,
50
),
sprite:
_atlas
.
getSprite
(
'sprite_name'
),
```

),

);

add

(

SpriteAnimationComponent

(

size:

Vector2

(

150

,

100

),

position:

Vector2

(

150

,

100

),

animation:

_atlas

.

getAnimation

(

'animation_name'

),

),

);

}

}

Full Example

¶

You can check an example

here

.

# Getting Started

¶

## About Flame

¶

Flame is a modular Flutter game engine that provides a complete set of out-of-the-way solutions for games

It provides you with a simple yet effective game loop implementation, and the necessary functionalities that

We also provide stand-alone packages that extend the Flame functionality which can be found in the

Bridge Packages

section.

You can pick and choose whichever parts you want, as they are all independent and modular.

The engine and its ecosystem are constantly being improved by the community, so please feel free to reac

Give us a star if you want to help give the engine exposure and grow the community. :)

## Installation

¶

Add the

flame

package as a dependency in your

pubspec.yaml

by running the following command:

```
flutter pub add flame
```

The latest version can be found on

pub.dev

.

then run

```
flutter
pub
```

get

and you are ready to start using it!

Getting started

¶

There is a set of tutorials that you can follow to get started in the

tutorials folder

.

Simple examples for all features can be found in the

examples folder

.

You can also check out the

awesome flame repository

, it contains quite a lot of good tutorials and articles written by the community to get you started with Flame.

Outside of the scope of the engine

¶

Games sometimes require complex feature sets depending on what the game is all about. Some of these f

Multiplayer (netcode)

¶

Flame doesn?t bundle any network feature, which may be needed to write online multiplayer games.

If you are building a multiplayer game, here are some recommendations of packages/services:

Nakama

: Nakama is an open-source server designed to power modern games and apps.

Firebase

: Provides dozens of services that can be used to write simpler multiplayer experiences.

Supabase

: A cheaper alternative to Firebase, based on Postgres.

2. Start Coding

¶

The Plan

¶

Now that we have the assets loaded and a very rough idea of what classes we will need, we need to start t

Ember should be able to be controlled to move left, right, and jump.

The level will be infinite, so we need a way to randomly load sections of the level.

The objective is to collect stars while avoiding enemies.

Enemies cannot be killed as you need to use the platforms to avoid them.

If Ember is hit by an enemy, it should reduce Ember?s health by 1.

Ember should have 3 lives to lose.

There should be pits that if Ember falls into, it is automatically game over.

There should be a main menu and a game-over screen that lets the player start over.

Now that this is planned out, I know you are probably as excited as I am to begin and I just want to see Em

Note

Why did I choose to make this game an infinite side scrolling platformer?

Well, I wanted to be able to showcase random level loading. No two game plays will be the same. This exa

Loading Assets

¶

For Ember to be displayed, we will need to load the assets. This can be done in

main.dart

, but by so doing, we will quickly clutter the file. To keep our game organized, we should create files that ha

lib

folder called

ember_quest.dart

. In that file, we will add:

```
import
'package:flame/game.dart'
;
class
EmberQuestGame
extends
FlameGame
{
EmberQuestGame
();
@override
Future
<
void
>
onLoad
()
async
{
await
images
.
loadAll
([
'block.png'
,
```

'ember.png'

,

'ground.png'

,

'heart_half.png'

,

'heart.png'

,

'star.png'

,

'water_enemy.png'

,

]);

}

}

As I mentioned in the

assets

section, we are using multiple individual image files and for performance reasons, we should leverage Flam

await

images.loadAll()

takes a list of the file names that are found in

assets/images

and loads them to cache.

Scaffolding

¶

So now that we have our game file, let?s prepare the

main.dart

file to receive our newly created

FlameGame

. Change your entire

main.dart

file to the following:

```
import

'package:flame/game.dart'

;

import

'package:flutter/material.dart'

;

import

'ember_quest.dart'

;

void

main

()

{

runApp

(

const

GameWidget

<

EmberQuestGame

>
```

.

controlled

(

gameFactory:

EmberQuestGame

.

new

,

),

);

}

You can run this file and you should just have a blank screen now. Let?s get Ember loaded!

CameraComponent and World

¶

To move around in the world we are going the use the built-in

CameraComponent

and

World

that exists on the

FlameGame

class. We are going to add all our components to the

world

and follow the player with the

camera

.

import

'package:flame/components.dart'

;

import

'package:flame/game.dart'

;

class

EmberQuestGame

extends

FlameGame

{

@override

Future

<

void

>

onLoad

()

async

{

await

images

.

loadAll

([

'block.png'

,

'ember.png'

,

'ground.png'

,

'heart_half.png'

,

'heart.png'

,

'star.png'

,

'water_enemy.png'

,

]);

// Everything in this tutorial assumes that the position

// of the `CameraComponent`s viewfinder (where the camera is looking)

// is in the top left corner, that's why we set the anchor here.

camera

.

viewfinder

.

anchor

=

Anchor

.

topLeft

;

}

}

Ember Time

¶

Keeping your game files organized can always be a challenge. I like to keep things logically organized by h

lib/actors

and in that folder, create

ember.dart

. In that file, add the following code:

import

'package:flame/components.dart'

;

import

'../ember_quest.dart'

;

class

EmberPlayer

extends

SpriteAnimationComponent

with

HasGameReference

<

EmberQuestGame

>

{

EmberPlayer

```
({
required
super
.
position
,
})
:
super
(
size:
Vector2
.
all
(
64
),
anchor:
Anchor
.
center
);
@override
void
onLoad
()
```

{

animation

=

SpriteAnimation

.

fromFrameData

(

game

.

images

.

fromCache

(

'ember.png'

),

SpriteAnimationData

.

sequenced

(

amount:

4

,

textureSize:

Vector2

.

all

(

16

),

stepTime:

0.12

,

),

);

}

}

This file uses the

HasGameRef

mixin which allows us to reach back to

ember_quest.dart

and leverage any of the variables or methods that are defined in the game class. You can see this in use w

game.images.fromCache('ember.png')

. Earlier, we loaded all the files into cache, so to use that file now, we call

fromCache

so it can be leveraged by the

SpriteAnimation

. The

EmberPlayer

class is extending a

SpriteAnimationComponent

which allows us to define animation as well as position it accordingly in our game world. When we construc

Vector2.all(64)

is defined as the size of Ember in our game world should be 64x64. You may notice that in the animation

SpriteAnimationData

, the

textureSize

is defined as

Vector2.all(16)

or 16x16. This is because the individual frame in our

ember.png

is 16x16 and there are 4 frames in total. To define the speed of the animation,

stepTime

is used and set at

0.12

seconds per frame. You can change the

stepTime

to any length that makes the animation seem correct for your game vision.

Now before you rush to run the game again, we have to add Ember to the game world. To do this, go back

ember_quest.dart

and add the following:

import

'package:flame/game.dart'

;

import

'actors/ember.dart'

;

class

EmberQuestGame

extends

FlameGame

{

late

EmberPlayer

_ember

;

@override

Future

<

void

>

onLoad

()

async

{

await

images

.

loadAll

([

'block.png'

,

'ember.png'

,

'ground.png'

,

'heart_half.png'

,

'heart.png'

,

'star.png'

,

'water_enemy.png'

,

]);

camera

.

viewfinder

.

anchor

=

Anchor

.

topLeft

;

_ember

=

EmberPlayer

(

position:

Vector2

(

128

,

canvasSize

.

y

-

70

),

);

world

.

add

(

_ember

);

}

}

Run your game now and you should now see Ember flickering in the lower left-hand corner.

Building Blocks

¶

Now that we have Ember showing on screen and we know our basic environment is all working correctly, it

3. Building the World

!

Overlays

¶

Since a Flame game can be wrapped in a widget, it is quite easy to use it alongside other Flutter widgets in

Game.overlays

enables any Flutter widget to be shown on top of a game instance. This makes it very easy to create things

The feature can be used via the

game.overlays.add

and

game.overlays.remove

methods that mark an overlay to be shown or hidden, respectively, via a

String

argument that identifies the overlay. After that, you can map each overlay to their corresponding Widget in

GameWidget

declaration by providing an

overlayBuilderMap

.

// Inside your game:

final

pauseOverlayIdentifier

=

'PauseMenu'

;

// Marks 'PauseMenu' to be rendered.

overlays

.

add

```
(
pauseOverlayIdentifier
);
// Marks 'PauseMenu' to not be rendered.
overlays
.
remove
(
pauseOverlayIdentifier
);
// On the widget declaration
final
game
=
MyGame
();
Widget
build
(
BuildContext
context
)
{
return
GameWidget
(
```

game:

game

,

overlayBuilderMap:

{

'PauseMenu'

:

(

BuildContext

context

,

MyGame

game

)

{

return

Text

(

'A pause menu'

);

},

},

);

}

The order of rendering for an overlay is determined by the order of the keys in the

overlayBuilderMap

.

An example of this feature can be found

here

.

## Debug features

¶

### FlameGame features

¶

Flame provides some debugging features for the

FlameGame

class. These features are enabled when the

debugMode

property is set to

true

(or overridden to be

true

). When

debugMode

is enabled, each

PositionComponent

will be rendered with their bounding size, and have their positions written on the screen. This way, you can

To see a working example of the debugging features of the

FlameGame

, check this

example

.

### FPS

¶

The FPS reported from Flame might be a bit lower than what is reported from for example the Flutter DevT

FpsComponent

¶

The

FpsComponent

can be added to anywhere in the component tree and will keep track of how many FPS that the game is cu

FpsTextComponent

.

FpsTextComponent

¶

The

FpsTextComponent

is simply a

TextComponent

that wraps an

FpsComponent

, since you most commonly want to show the current FPS somewhere when the

FpsComponent

is used.

ChildCounterComponent

¶

ChildCounterComponent

is a component that renders the number of children of type

T

from a component (

target

) every second. So for example:

So for example, the following will render the number of

SpriteAnimationComponent

that are children of the game

world

:

add

(

ChildCounterComponent

<

SpriteAnimationComponent

>

(

target:

world

,

),

);

TimeTrackComponent

¶

This component allows developers to track time spent inside their code. This can be useful for performance

To use it, add it to your game somewhere (since this is a debug feature, we advise to only add the compon

add

(

TimeTrackComponent

());

Then in the code section that you want to track time, do the following:

void

update

(

double

dt

)

{

TimeTrackComponent

.

start

(

'MyComponent.update'

);

// ...

TimeTrackComponent

.

end

(

'MyComponent.update'

);

}

With the calls above, the added

TimeTrackComponent

will render the elapsed time in microseconds.

Palette

¶

Throughout your game you are going to need to use colors in lots of places. There are two classes on

dart:ui

that can be used,

Color

and

Paint

.

The

Color

class represents a ARGB color in a hexadecimal integer format. So to create a

Color

instance, you just need to pass the color as an integer in the ARGB format.

You can use Dart?s hexadecimal notation to make it really easy; for instance:

0xFF00FF00

is fully opaque green (the ?mask? would be

0xAARRGGBB

).

Note

: The first two hexadecimal digits are for the alpha channel (transparency), unlike on regular (non-A) RGB.

In the Material Flutter package there is a

Colors

class that provides common colors as constants:

import

'package:flutter/material.dart'

show

Colors

;

const

black

=

Colors

.

black

;

Some more complex methods might also take a

Paint

object, which is a more complete structure that allows you to configure aspects related to stroke, colors, filt

Paint

object representing just a single simple plain solid color.

Note:

we don?t recommend that you create a new

Paint

object every time you need a specific

Paint

, since it could potentially lead to a lot of unnecessary objects being created. A better way is to either define

Paint

object somewhere and re-use it (however, do note that the

Paint

class is mutable, unlike

Color

), or to use the

Palette

class to define all the colors that you want to use in your game.

You can create such an object like this:

Paint

green

=

Paint

()..

color

=

const

Color

(

0xFF00FF00

);

To help you with this and to also keep your game?s color palette consistent, Flame adds the

Palette

class. You can use it to easily access both

Color

s and

Paint

s where needed and also to define the colors your game use as constants, so you don?t get those mixed u

The

BasicPalette

class is an example of what a palette can look like, and adds black and white as colors. So to use black or

BasicPalette

; for example, using

color

:

TextConfig

regular

=

TextConfig

(

color:

BasicPalette

.

white

.

color

);

Or using

paint

:

canvas

.

drawRect

(

rect

,

BasicPalette

.

black

.

paint

);

However, the idea is that you can create your own palette, following the

BasicPalette

example, and add the color palette/scheme of your game. Then you will be able to statically access any co

Palette

implementation, from the example game

BGUG

:

import

'dart:ui'

;

import

'package:flame/palette.dart'

;

class

Palette

{

static

PaletteEntry

white

=

BasicPalette

.

white

;

static

PaletteEntry

toastBackground

=

PaletteEntry

(

Color

(

0xFFAC3232

));

static

PaletteEntry

toastText

=

PaletteEntry

(

Color

(

0xFFDA9A00

));

static

PaletteEntry

grey

=

PaletteEntry

(

Color

(

0xFF404040

));

static

PaletteEntry

green

=

PaletteEntry

(

Color

(

0xFF54a286

));

}

A

PaletteEntry

is a

const

class that holds information of a color and it has the following members:

color

: returns the

Color

specified

paint

: creates a new

Paint

with the color specified.

Paint

is a non-

const

class, so this method actually creates a brand new instance every time it?s called. It?s safe to cascade mu

Images

¶

To start off you must have an appropriate folder structure and add the files to the

pubspec.yaml

file, like this:

flutter

:

assets

:

-

assets/images/player.png

-

assets/images/enemy.png

Images can be in any format supported by Flutter, which include: JPEG, WebP, PNG, GIF, animated GIF, a

flame_svg

library.

Loading images

¶

Flame bundles an utility class called

Images

that allows you to easily load and cache images from the assets directory into memory.

Flutter has a handful of types related to images, and converting everything properly from a local asset to an

Image

that can be drawn on Canvas is a bit convoluted. This class allows you to obtain an

Image

that can be drawn on the

Canvas

using the

drawImageRect

method.

It automatically caches any image loaded by filename, so you can safely call it many times.

The methods for loading and clearing the cache are:

load

,

loadAll

,

clear

and

clearCache

. They return

Future

s for loading the images. These futures must be awaited for before the images can be used in any way. If y

load()

operations and then await for all of them at once using

Images.ready()

method.

To synchronously retrieve a previously cached image, the

fromCache

method can be used. If an image with that key was not previously loaded, it will throw an exception.

To add an already loaded image to the cache, the

add

method can be used and you can set the key that the image should have in the cache.

You can also use

ImageExtension.fromPixels()

to dynamically create an image during the game.

For

clear

and

clearCache

, do note that

dispose

is called for each removed image from the cache, so make sure that you don?t use the image afterwards.

Standalone usage

¶

It can manually be used by instantiating it:

```
import

'package:flame/cache.dart'

;

final

imagesLoader

=

Images

();

Image

image

=

await

imagesLoader
```

.

load

(

'yourImage.png'

);

But Flame also offers two ways of using this class without instantiating it yourself.

Flame.images

¶

There is a singleton, provided by the

Flame

class, that can be used as a global image cache.

Example:

import

'package:flame/flame.dart'

;

import

'package:flame/sprite.dart'

;

// inside an async context

Image

image

=

await

Flame

.

images

.

load

(

'player.png'

);

final

playerSprite

=

Sprite

(

image

);

Game.images

¶

The

Game

class offers some utility methods for handling images loading too. It bundles an instance of the

Images

class, that can be used to load image assets to be used during the game. The game will automatically free

The

onLoad

method from the

Game

class is a great place for the initial assets to be loaded.

Example:

class

```
MyGame
extends
Game
{
Sprite
player
;
@override
Future
<
void
>
onLoad
()
async
{
// Note that you could also use Sprite.load for this.
final
playerImage
=
await
images
.
load
(
'player.png'
```

);

player

=

Sprite

(

playerImage

);

}

}

Loaded assets can also be retrieved while the game is running by

images.fromCache

, for example:

class

MyGame

extends

Game

{

// attributes omitted

@override

Future

<

void

>

onLoad

()

async

```
{
// other loads omitted
await
images
.
load
(
'bullet.png'
);
}
void
shoot
()
{
// This is just an example, in your game you probably don't want to
// instantiate new [Sprite] objects every time you shoot.
final
bulletSprite
=
Sprite
(
images
.
fromCache
(
'bullet.png'
```

));

_bullets

.

add

(

bulletSprite

);

}

}

Loading images over the network

¶

The Flame core package doesn?t offer a built in method to loading images from the network.

The reason for that is that Flutter/Dart does not have a built in http client, which requires a package to be u

With that said, it is quite simple to load images from the network once a http client package is chosen by th

Image

can be fetched from the web using the

http

package.

import

'package:http/http.dart'

as

http

;

import

'package:flutter/painting.dart'

;

final

response

=

await

http

.

get

(

'https://url.com/image.png'

);

final

image

=

await

decodeImageFromList

(

response

.

bytes

);

Note

Check

flame_network_assets

for a ready to use network assets solution that provides a built in cache.

Sprite

¶

Flame offers a

Sprite

class that represents an image, or a region of an image.

You can create a

Sprite

by providing it an

Image

and coordinates that defines the piece of the image that that sprite represents.

For example, this will create a sprite representing the whole image of the file passed:

final

image

=

await

images

.

load

(

'player.png'

);

Sprite

player

=

Sprite

(

image

);

You can also specify the coordinates in the original image where the sprite is located. This allows you to us

```
final
image
=
await
images
.
load
(
'player.png'
);
final
playerFrame
=
Sprite
(
image
,
srcPosition:
Vector2
(
32.0
,
0
),
srcSize:
```

Vector2

(

16.0

,

16.0

),

);

The default values are

(0.0,

0.0)

for

srcPosition

and

null

for

srcSize

(meaning it will use the full width/height of the source image).

The

Sprite

class has a render method, that allows you to render the sprite onto a

Canvas

:

final

image

=

await

images

.

load

(

'block.png'

);

Sprite

block

=

Sprite

(

image

);

// in your render method

block

.

render

(

canvas

,

16.0

,

16.0

);

//canvas, width, height

You must pass the size to the render method, and the image will be resized accordingly.

All render methods from the

Sprite

class can receive a

Paint

instance as the optional named parameter

overridePaint

that parameter will override the current

Sprite

paint instance for that render call.

Sprite

s can also be used as widgets, to do so just use

SpriteWidget

class.

A complete example using sprite as widgets can be found

here

.

SpriteBatch

¶

If you have a sprite sheet (also called an image atlas, which is an image with smaller images inside), and w

SpriteBatch

handles that job for you.

Give it the filename of the image, and then add rectangles which describes various part of the image, in ad

You render it with a

Canvas

and an optional

Paint

,

BlendMode

and

CullRect

.

A

SpriteBatchComponent

is also available for your convenience.

See the examples

here

.

ImageComposition

¶

In some cases you may want to merge multiple images into a single image; this is called

Compositing

. This can be useful for example when working with the

SpriteBatch

API to optimize your drawing calls.

For such use cases Flame comes with the

ImageComposition

class. This allows you to add multiple images, each at their own position, onto a new image:

final

composition

=

ImageComposition

()

..

add

(

image1

,

Vector2

(

0

,

0

))

..

add

(

image2

,

Vector2

(

64

,

0

));

..

add

(

image3

,

Vector2

(

128

,

0

),

source

:

Rect

.

fromLTWH

(

32

,

32

,

64

,

64

),

);

Image

image

=

await

composition

.

compose

();

Image

imageSync

=

composition

.

composeSync

();

As you can see, two versions of composing image are available. Use

ImageComposition.compose()

for the async approach. Or use the new

ImageComposition.composeSync()

function to rasterize the image into GPU context using the benefits of the

Picture.toImageSync

function.

Note:

Composing images is expensive, we do not recommend you run this every tick as it affect the performance

Animation

¶

The Animation class helps you create a cyclic animation of sprites.

You can create it by passing a list of equally sized sprites and the stepTime (that is, how many seconds it t

final

a

=

SpriteAnimationTicker

(

SpriteAnimation

.

spriteList

(

sprites

,

stepTime:

0.02

));

After the animation is created, you need to call its

update

method and render the current frame?s sprite on your game instance.

Example:

class

MyGame

extends

Game

{

SpriteAnimationTicker

a

;

MyGame

()

```
{
a
=
SpriteAnimationTicker
(
SpriteAnimation
(...));
}
void
update
(
double
dt
)
{
a
.
update
(
dt
);
}
void
render
(
Canvas
```

c

)

{

a

.

getSprite

().

render

(

c

);

}

}

A better alternative to generate a list of sprites is to use the

fromFrameData

constructor:

const

amountOfFrames

=

8

;

final

a

=

SpriteAnimation

.

fromFrameData

(

imageInstance

,

SpriteAnimationFrame

.

sequenced

(

amount:

amountOfFrames

,

textureSize:

Vector2

(

16.0

,

16.0

),

stepTime:

0.1

,

),

);

This constructor makes creating an

Animation

very easy when using sprite sheets.

In the constructor you pass an image instance and the frame data, which contains some parameters that c

SpriteAnimationFrameData

class to see all the parameters.

If you use Aseprite for your animations, Flame does provide some support for Aseprite animation?s JSON

final

image

=

await

images

.

load

(

'chopper.png'

);

final

jsonData

=

await

assets

.

readJson

(

'chopper.json'

);

final

animation

=

SpriteAnimation

.

fromAsepriteData

(

image

,

jsonData

);

Note:

trimmed sprite sheets are not supported by flame, so if you export your sprite sheet this way, it will have th

Animations, after created, have an update and render method; the latter renders the current frame, and the

Animations are normally used inside

SpriteAnimationComponent

s, but custom components with several Animations can be created as well.

A complete example of using animations as widgets can be found

here

.

SpriteSheet

¶

Sprite sheets are big images with several frames of the same sprite on it and is a very good way to organiz

import

'package:flame/sprite.dart'

;

final

spriteSheet

=

SpriteSheet

(

image:

imageInstance

,

srcSize:

Vector2

.

all

(

16.0

),

);

final

animation

=

spriteSheet

.

createAnimation

(

0

,

stepTime:

0.1

);

Now you can use the animation directly or use it in an animation component.

You can also create a custom animation by retrieving individual

SpriteAnimationFrameData

using either

SpriteSheet.createFrameData

or

SpriteSheet.createFrameDataFromId

:

final

animation

=

SpriteAnimation

.

fromFrameData

(

imageInstance

,

SpriteAnimationData

([

spriteSheet

.

createFrameDataFromId

(

1

,

stepTime:

0.1

),

// by id

spriteSheet

.

createFrameData

(

2

,

3

,

stepTime:

0.3

),

// row, column

spriteSheet

.

createFrameDataFromId

(

4

,

stepTime:

0.1

),

// by id

]),

);

If you don?t need any kind of animation and instead only want an instance of a

Sprite

on the

SpriteSheet

you can use the

getSprite

or

getSpriteById

methods:

spriteSheet

.

getSpriteById

(

2

);

// by id

spriteSheet

.

getSprite

(

0

,

0

);

// row, column

You can see a full example of the

SpriteSheet

class

here

.

Expressions

¶

The

expressions

in YarnSpinner provide a way to dynamically change the flow or the content of the dialogue, based on

variables

, combined with

operators

or

function

calls. They are used in several places:

to insert a dynamic text into a

line

;

to create or update a

variable

;

as part of a

command

such as

<<if>>

or

<<set>>

;

to compute the values of

markup

attributes.

An expression always evaluates synchronously, meaning that it cannot wait for user?s input, nor perform a

user-defined command

that waits for the calculation to succeed and then stores the result into some global

variable

, which can then be accessed from an expression.

flame_lottie

¶

This package allows you to load and add Lottie animations to your Flame game.

The native Lottie libraries (such as

lottie-android

) are maintained by

Airbnb

.

The Flutter package

lottie

, on which this wrapper is based on, is by developed

xaha.dev

and can be found on

pub.dev

.

Usage

¶

To use it in your game you just need to add

flame_lottie

to your pubspec.yaml.

Simply load the Lottie animation using the

loadLottie

method and the

LottieBuilder

. It allows all the various ways of loading a Lottie file:

Lottie.asset

, for obtaining a Lottie file from an AssetBundle using a key.

Lottie.network

, for obtaining a lottie file from a URL.

Lottie.file

, for obtaining a lottie file from a File.

Lottie.memory

, for obtaining a lottie file from a Uint8List.

? and add it as

LottieComponent

to your Flame ? game.

Example:

class

MyGame

extends

FlameGame

{

...

@override

Future

<

void

>

onLoad

()

async

{

final

asset

=

Lottie

.

asset

(

'assets/LottieLogo1.json'

);

final

animation

=

await

loadLottie

(

asset

);

add

(

LottieComponent

(

animation

,

repeating:

true

,

```
// Continuously loop the animation.
size:
Vector2
.
all
(
400
),
),
);
}
...
}
```

Space Shooter Game Tutorial

¶

In this tutorial, we will follow a step-by-step process for coding a game using the Flame engine.

This tutorial assumes that you have at least some familiarity with common programming concepts, and with

Dart

programming language.

## Effects

¶

An effect is a special component that can attach to another component in order to modify its properties or a

For example, suppose you are making a game with collectible power-up items. You want these power-ups

Let?s add a

`ScaleEffect`

to grow the item from 0 to 100% when the power-up first appears. Add another infinitely repeating alternati

`MoveEffect`

in order to make the item move slightly up and down. Then add an

`OpacityEffect`

that will ?blink? the item 3 times, this effect will have a built-in delay of 30 seconds, or however long you wa

`RemoveEffect`

that will automatically remove the item from the game tree after the specified time (you probably want to tim

`OpacityEffect`

).

As you can see, with a few simple effects we have turned a simple lifeless sprite into a much more interest

## Overview

¶

The function of an

`Effect`

is to effect a change over time in some component?s property. In order to achieve that, the

`Effect`

must know the initial value of the property, the final value, and how it should progress over time. The initial

`EffectController`

s.

There are multiple effects provided by Flame, and you can also

create your own

. The following effects are included:

MoveByEffect

MoveToEffect

MoveAlongPathEffect

RotateEffect.by

RotateEffect.to

ScaleEffect.by

ScaleEffect.to

SizeEffect.by

SizeEffect.to

AnchorByEffect

AnchorToEffect

OpacityToEffect

OpacityByEffect

ColorEffect

SequenceEffect

RemoveEffect

An

EffectController

is an object that describes how the effect should evolve over time. If you think of the initial value of the effe

There are multiple effect controllers provided by the Flame framework as well:

EffectController

LinearEffectController

ReverseLinearEffectController

CurvedEffectController

ReverseCurvedEffectController

PauseEffectController

RepeatedEffectController

InfiniteEffectController

SequenceEffectController

SpeedEffectController

DelayedEffectController

NoiseEffectController

RandomEffectController

SineEffectController

ZigzagEffectController

Built-in effects

¶

Effect

¶

The base

Effect

class is not usable on its own (it is abstract), but it provides some common functionality inherited by all othe

The ability to pause/resume the effect using

effect.pause()

and

effect.resume()

. You can check whether the effect is currently paused using

effect.isPaused

.

Property

removeOnFinish

(which is true by default) will cause the effect component to be removed from the game tree and garbage-c

Optional user-provided

onComplete

, which will be invoked when the effect has just completed its execution but before it is removed from the g

The

reset()

method reverts the effect to its original state, allowing it to run once again.

MoveByEffect

¶

This effect applies to a

PositionComponent

and shifts it by a prescribed

offset

amount. This offset is relative to the current position of the target:

move_by_effect.dart

1

import

'package:doc_flame_examples/flower.dart'

;

2

import

'package:flame/effects.dart'

;

3

import

'package:flame/game.dart'

;

4

5

class

MoveByEffectGame

extends

FlameGame

{

6

bool

reset

=

false

;

7

@override

8

Future

<

void

>

onLoad

()

async

{

9

final

flower

=

Flower

(

10

size:

60

,

11

position:

canvasSize

/

2

,

12

onTap:

(

flower

)

{

13

if

(

reset

=

!

reset

)

{

14

flower

.

add

(

15

MoveEffect

.

by

(

16

Vector2

(

30

,

30

),

17

EffectController

(

duration:

1.0

),

18

),

19

);

20

}

else

{

21

flower

.

add

(

22

MoveEffect

.

by

(

23

Vector2

(

-

30

,

-

30

),

24

EffectController

(

duration:

1.0

),

25

),

26

);

27

}

28

},

29

);

30

add

(

flower

);

31

}

32

}

Code

final

effect

=

MoveByEffect

(

Vector2

(

0

,

-

10

),

EffectController

(

duration:

0.5

),

);

If the component is currently at

Vector2(250,

200)

, then at the end of the effect its position will be

Vector2(250,

190)

.

Multiple move effects can be applied to a component at the same time. The result will be the superposition

MoveToEffect

¶

This effect moves a

PositionComponent

from its current position to the specified destination point in a straight line.

move_to_effect.dart

1

import

'package:doc_flame_examples/flower.dart'

;

2

import

'package:flame/effects.dart'

;

3

import

'package:flame/game.dart'

;

4

5

class

MoveToEffectGame

extends

FlameGame

{

6

bool

reset

=

false

;

7

@override

8

Future

<

void

>

onLoad

()

async

{

9

final

flower

=

Flower

(

10

size:

60

,

11

position:

canvasSize

/

2

,

12

onTap:

(

flower

)

{

13

if

(

reset

=

!

reset

)

{

14

flower

.

add

(

15

MoveEffect

.

to

(

16

Vector2

(

30

,

30

),

17

EffectController

(

duration:

1.0

),

18

),

19

);

20

}

else

{

21

flower

.

add

(

22

MoveEffect

.

to

(

23

size

/

2

,

24

EffectController

(

duration:

1.0

),

25

),

```
26
);
27
}
28
},
29
);
30
add
(
flower
);
31
}
32
}
Code
final
effect
=
MoveToEffect
(
Vector2
(
100
```

,

500

),

EffectController

(

duration:

3

),

);

It is possible, but not recommended to attach multiple such effects to the same component.

MoveAlongPathEffect

¶

This effect moves a

PositionComponent

along the specified path relative to the component?s current position. The path can have non-linear segme

Vector2.zero()

in order to avoid sudden jumps in the component?s position.

move_along_path_effect.dart

1

import

'dart:ui'

;

2

3

import

'package:doc_flame_examples/flower.dart'

;

4

import

'package:flame/effects.dart'

;

5

import

'package:flame/game.dart'

;

6

7

class

MoveAlongPathEffectGame

extends

FlameGame

{

8

bool

reset

=

false

;

9

@override

10

Future

<
void
>
onLoad
()
async
{
11
final
flower
=
Flower
(
12
size:
60
,
13
position:
canvasSize
/
2
,
14
onTap:
(

flower

)

{

15

if

(

reset

=

!

reset

)

{

16

flower

.

add

(

17

MoveAlongPathEffect

(

18

Path

()..

quadraticBezierTo

(

100

,

0

,

50

,

-

50

),

19

EffectController

(

duration:

1.5

),

20

),

21

);

22

}

else

{

23

flower

.

add

(

24

MoveAlongPathEffect

(

25

Path

()..

quadraticBezierTo

(

-

100

,

0

,

-

50

,

50

),

26

EffectController

(

duration:

1.5

),

27

),

28

);

29

}

30

},

31

);

32

add

(

flower

);

33

}

34

}

Code

final

effect

=

MoveAlongPathEffect

(

Path

()..

quadraticBezierTo

(

100

,

0

,

50

,

-

50

),

EffectController

(

duration:

1.5

),

);

An optional flag

absolute:

true

will declare the path within the effect as absolute. That is, the target will ?jump? to the beginning of the patl

Another flag

oriented:

true

instructs the target not only move along the curve, but also rotate itself in the direction the curve is facing a

RotateEffect.by

¶

Rotates the target clockwise by the specified angle relative to its current orientation. The angle is in radians

tau

/4 in radians) clockwise:

rotate_by_effect.dart

```
1
import
'package:doc_flame_examples/flower.dart'
;
2
import
'package:flame/effects.dart'
;
3
import
'package:flame/game.dart'
;
4
import
'package:flame/geometry.dart'
;
5
6
class
RotateByEffectGame
extends
```

FlameGame

{

7

bool

reset

=

false

;

8

@override

9

Future

<

void

>

onLoad

()

async

{

10

final

flower

=

Flower

(

11

size:

60

,

12

position:

canvasSize

/

2

,

13

onTap:

(

flower

)

{

14

if

(

reset

=

!

reset

)

{

15

flower

.

add

(

16

RotateEffect

.

by

(

17

tau

/

4

,

18

EffectController

(

duration:

2

),

19

),

20

);

21

}

else

{

22

flower

.

add

(

23

RotateEffect

.

by

(

24

-

tau

/

4

,

25

EffectController

(

duration:

2

),

26

),

27

);

28

}

29

},

30

);

31

add

(

flower

);

32

}

33

}

Code

final

effect

=

RotateEffect

.

by

(

tau

/

4

,

EffectController

(

duration:

2

),

);

RotateEffect.to

¶

Rotates the target clockwise to the specified angle. For example, the following will rotate the target to look

tau

/4 east, 180º=tau/2 south, and 270º=tau*3/4 west):

rotate_to_effect.dart

1

import

'package:doc_flame_examples/flower.dart'

;

2

import

'package:flame/effects.dart'

;

3

import

'package:flame/game.dart'

;

4

import

'package:flame/geometry.dart'

;

5

6

class

RotateToEffectGame

extends

FlameGame

{

7

bool

reset

=

false

;

8

@override

9

Future

<

void

>

onLoad

()

async

{

10

final

flower

=

Flower

(

11

size:

60

,

12

position:

canvasSize

/

2

,

13

onTap:

(

flower

)

{

14

if

(

reset

=

!

reset

)

{

15

flower

.

add

(

16

RotateEffect

.

to

(

17

tau

/

4

,

18

EffectController

(

duration:

2

),

19

),

20

);

21

}

else

{

22

flower

.

add

(

23

RotateEffect

.

to

(

24

-

tau

/

4

,

25

EffectController

(

duration:

2

),

26

),

27

);

28

}

29

},

30

);

31

add

(

flower

);

32

}

33

}

Code

final

effect

=

RotateEffect

.

to

(

tau

/

4

,

EffectController

(

duration:

2

),

);

ScaleEffect.by

¶

This effect will change the target?s scale by the specified amount. For example, this will cause the compon

scale_by_effect.dart

1

import

'package:doc_flame_examples/flower.dart'

;

2

import

'package:flame/effects.dart'

;

3

import

'package:flame/game.dart'

;

4

5

class

ScaleByEffectGame

extends

FlameGame

{

6

bool

reverse

=

false

;

7

bool

hold

=

false

;

8

@override

9

Future

<

void

>

onLoad

()

async

{

10

final

flower

=

Flower

(

11

size:

60

,

12

position:

canvasSize

/

2

,

13

onTap:

(

flower

)

{

14

if

(

hold

)

{

15

return

;

16

}

17

hold

=

true

;

18

if

(

reverse

=

!

reverse

)

{

19

flower

.

add

(

20

ScaleEffect

.

by

(

21

Vector2

.

all

(

1.5

),

22

EffectController

(

duration:

0.3

),

23

onComplete:

()

=>

hold

=

false

,

24

),

25

);

26

}

else

{

27

flower

.

add

(

28

ScaleEffect

.

by

(

29

Vector2

.

all

(

1

/

1.5

),

30

EffectController

(

duration:

0.3

),

31

onComplete:

()

=>

hold

=

false

,

32

),

33

);

34

}

35

},

36

);

37

add

(

flower

);

38

}

39

}

Code

final

effect

=

ScaleEffect

.

by

(

Vector2

.

all

(

1.5

),

EffectController

(

duration:

0.3

),

);

ScaleEffect.to

¶

This effect works similar to

ScaleEffect.by

, but sets the absolute value of the target?s scale.

scale_to_effect.dart

1

import

'package:doc_flame_examples/flower.dart'

;

2

import

'package:flame/effects.dart'

;

3

import

'package:flame/game.dart'

;

4

5

class

ScaleToEffectGame

extends

FlameGame

{

6

bool

reverse

=

false

;

7

bool

hold

=

false

;

8

@override

9

Future

<

void

>

onLoad

()

async

{

10

final

flower

=

Flower

(

11

size:

60

,

12

position:

canvasSize

/

2

,

13

```
onTap:
(
flower
)
{
14
if
(
hold
)
{
15
return
;
16
}
17
hold
=
true
;
18
if
(
reverse
=
```

!

reverse

)

{

19

flower

.

add

(

20

ScaleEffect

.

to

(

21

Vector2

.

all

(

0.5

),

22

EffectController

(

duration:

0.5

),

23

onComplete:

()

=>

hold

=

false

,

24

),

25

);

26

}

else

{

27

flower

.

add

(

28

ScaleEffect

.

to

(

29

Vector2

.

all

(

1

),

30

EffectController

(

duration:

0.5

),

31

onComplete:

()

=>

hold

=

false

,

32

),

33

);

34

}

35

},

36

);

37

add

(

flower

);

38

}

39

}

Code

final

effect

=

ScaleEffect

.

to

(

Vector2

.

all

(

0.5

),

EffectController

(

duration:

0.5

),

);

SizeEffect.by

¶

This effect will change the size of the target component, relative to its current size. For example, if the targe

Vector2(100,

100)

, then after the following effect is applied and runs its course, the new size will be

Vector2(120,

50)

:

size_by_effect.dart

1

import

'package:doc_flame_examples/ember.dart'

;

2

import

'package:flame/components.dart'

;

3

import

'package:flame/effects.dart'

;

4

import

'package:flame/game.dart'

;

5

6

class

SizeByEffectGame

extends

FlameGame

{

7

bool

reset

=

false

;

8

@override

9

Future

<
void
>
onLoad
()
async
{
10
final
ember
=
EmberPlayer
(
11
position:
size
/
2
,
12
size:
Vector2
(
45
,
40

),

13

onTap:

(

ember

)

{

14

if

(

reset

=

!

reset

)

{

15

ember

.

add

(

16

SizeEffect

.

by

(

17

Vector2

(

-

15

,

30

),

18

EffectController

(

duration:

0.75

),

19

),

20

);

21

}

else

{

22

ember

.

add

(

23

SizeEffect

.

by

(

24

Vector2

(

15

,

-

30

),

25

EffectController

(

duration:

0.75

),

26

),

27

);

28

}

29

},

30

)..

anchor

=

Anchor

.

center

;

31

32

add

(

ember

);

33

}

34

}

Code

final

effect

=

SizeEffect

.

by

(

Vector2

(

-

15

,

30

),

EffectController

(

duration:

1

),

);

The size of a

PositionComponent

cannot be negative. If an effect attempts to set the size to a negative value, the size will be clamped at zero

Note that for this effect to work, the target component must implement the

SizeProvider

interface and take its

size

into account when rendering. Only few of the built-in components implement this API, but you can always m

implements

SizeEffect

to the class declaration.

An alternative to

SizeEffect

is the

ScaleEffect

, which works more generally and scales both the target component and its children.

SizeEffect.to

¶

Changes the size of the target component to the specified size. Target size cannot be negative:

size_to_effect.dart

```
1
import
'package:doc_flame_examples/ember.dart'
;
2
import
'package:flame/components.dart'
;
3
import
'package:flame/effects.dart'
;
4
import
'package:flame/game.dart'
;
5
```

6

class

SizeToEffectGame

extends

FlameGame

{

7

bool

reset

=

false

;

8

@override

9

Future

<

void

>

onLoad

()

async

{

10

final

ember

=

EmberPlayer

(

11

position:

size

/

2

,

12

size:

Vector2

(

45

,

40

),

13

onTap:

(

ember

)

{

14

if

(

reset

=

!

reset

)

{

15

ember

.

add

(

16

SizeEffect

.

to

(

17

Vector2

(

90

,

80

),

18

EffectController

(

duration:

0.75

),

19

),

20

);

21

}

else

{

22

ember

.

add

(

23

SizeEffect

.

to

(

24

Vector2

(

45

,

```
40
),
25
EffectController
(
duration:
0.75
),
26
),
27
);
28
}
29
},
30
)..
anchor
=
Anchor
.
center
;
31
32
```

add

(

ember

);

33

}

34

}

Code

final

effect

=

SizeEffect

.

to

(

Vector2

(

90

,

80

),

EffectController

(

duration:

1

),

);

AnchorByEffect

¶

Changes the location of the target?s anchor by the specified offset. This effect can also be created using

AnchorEffect.by()

.

anchor_by_effect.dart

1

import

'package:doc_flame_examples/flower.dart'

;

2

import

'package:flame/effects.dart'

;

3

import

'package:flame/game.dart'

;

4

5

class

AnchorByEffectGame

extends

FlameGame

{

6

bool

reset

=

false

;

7

@override

8

Future

<

void

>

onLoad

()

async

{

9

final

flower

=

Flower

(

10

size:

60

,

11

position:

canvasSize

/

2

,

12

onTap:

(

flower

)

{

13

if

(

reset

=

!

reset

)

{

14

flower

.

```
add
(
15
AnchorByEffect
(
16
Vector2
(
0.5
,
0.5
),
17
EffectController
(
speed:
1
),
18
),
19
);
20
}
else
{
```

21

flower

.

add

(

22

AnchorByEffect

(

23

Vector2

(

-

0.5

,

-

0.5

),

24

EffectController

(

speed:

1

),

25

),

26

);

27

}

28

},

29

);

30

add

(

flower

);

31

}

32

}

Code

final

effect

=

AnchorByEffect

(

Vector2

(

0.1

,

0.1

),

EffectController

(

speed:

1

),

);

AnchorToEffect

¶

Changes the location of the target?s anchor. This effect can also be created using

AnchorEffect.to()

.

anchor_to_effect.dart

1

import

'package:doc_flame_examples/flower.dart'

;

2

import

'package:flame/components.dart'

;

3

import

'package:flame/effects.dart'

;

4

import

'package:flame/game.dart'

;

5

6

class

AnchorToEffectGame

extends

FlameGame

{

7

@override

8

Future

<

void

>

onLoad

()

async

{

9

final

flower

=

Flower

(

10

size:

60

,

11

position:

canvasSize

/

2

,

12

onTap:

(

flower

)

{

13

if

(

flower

.

anchor

==

Anchor

.

center

)

{

14

flower

.

add

(

15

AnchorToEffect

(

16

Anchor

.

bottomLeft

,

17

EffectController

(

speed:

1

),

18

),

19

);

20

}

else

{

21

flower

.

add

(

22

AnchorToEffect

(

23

Anchor

.

center

,

24

EffectController

(

speed:

1

),

25

),

26

);

27

}

28

},

29

);

30

add

(

flower

);

31

}

32

}

Code

final

effect

=

AnchorToEffect

(

Anchor

.

center

,

EffectController

(

speed:

1

),

);

OpacityToEffect

¶

This effect will change the opacity of the target over time to the specified alpha-value. It can only be applie

OpacityProvider

.

opacity_to_effect.dart

1

import

'package:doc_flame_examples/flower.dart'

;

2

import

'package:flame/components.dart'

;

3

import

'package:flame/effects.dart'

;

4

```
import
'package:flame/game.dart'
;
5
6
class
OpacityToEffectGame
extends
FlameGame
{
7
bool
reset
=
false
;
8
9
@override
10
Future
<
void
>
onLoad
()
```

```
async
{
11
final
flower
=
Flower
(
12
position:
size
/
2
,
13
size:
60
,
14
onTap:
_onTap
,
15
)..
anchor
=
```

Anchor

.

center

;

16

17

add

(

flower

);

18

}

19

20

void

_onTap

(

Flower

flower

)

{

21

if

(

reset

=

!

reset

)

{

22

flower

.

add

(

23

OpacityEffect

.

to

(

24

0.2

,

25

EffectController

(

duration:

0.75

),

26

),

27

);

28

}

else

{

29

flower

.

add

(

30

OpacityEffect

.

fadeIn

(

31

EffectController

(

duration:

0.75

),

32

),

33

);

34

}

35

}

36

}

Code

final

effect

=

OpacityEffect

.

to

(

0.2

,

EffectController

(

duration:

0.75

),

);

If the component uses multiple paints, the effect can target one more more of those paints using the

target

parameter. The

HasPaint

mixin implements

OpacityProvider

and exposes APIs to easily create providers for desired paintIds. For single paintId

opacityProviderOf

can be used and for multiple paintIds and

opacityProviderOfList

can be used.

opacity_effect_with_target.dart

```
1
import
'package:doc_flame_examples/flower.dart'
;
2
import
'package:flame/components.dart'
;
3
import
'package:flame/effects.dart'
;
4
import
'package:flame/game.dart'
;
5
6
class
```

OpacityEffectWithTargetGame

extends

FlameGame

{

7

bool

reset

=

false

;

8

9

// This reference needs to be stored because every new instance of

10

// OpacityProvider caches the opacity ratios at the time on creation.

11

late

OpacityProvider

_borderOpacityProvider

;

12

13

@override

14

Future

<

void

>

onLoad

()

async

{

15

final

flower

=

Flower

(

16

position:

size

/

2

,

17

size:

60

,

18

onTap:

_onTap

,

19

)..

anchor

=

Anchor

.

center

;

20

21

_borderOpacityProvider

=

flower

.

opacityProviderOfList

(

22

paintIds:

const

[

FlowerPaint

.

paintId1

,

FlowerPaint

.

paintId2

],

23

);

24

25

add

(

flower

);

26

}

27

28

void

_onTap

(

Flower

flower

)

{

29

if

(

reset

=

!

reset

)

{

30

flower

.

add

(

31

OpacityEffect

.

to

(

32

0.2

,

33

EffectController

(

duration:

0.75

),

34

target:

_borderOpacityProvider

,

35

),

36

);

37

}

else

{

38

flower

.

add

(

39

OpacityEffect

.

fadeIn

(

40

EffectController

(

duration:

0.75

),

41

target:

_borderOpacityProvider

,

42

),

43

);

44

}

45

}

46

}

Code

final

effect

=

OpacityEffect

.

to

(

0.2

,

EffectController

(

duration:

0.75

),

target:

component

.

opacityProviderOfList

(

paintIds:

const

[

paintId1

,

paintId2

],

),

);

The opacity value of 0 corresponds to a fully transparent component, and the opacity value of 1 is fully opa

OpacityEffect.fadeOut()

and

OpacityEffect.fadeIn()

will animate the target into full transparency / full visibility respectively.

OpacityByEffect

¶

This effect will change the opacity of the target relative to the specified alpha-value. For example, the follow

90%

:

opacity_by_effect.dart

1

import

'package:doc_flame_examples/ember.dart'

;

2

import

'package:flame/components.dart'

;

3

import

'package:flame/effects.dart'

;

4

import

'package:flame/game.dart'

;

5

6

class

OpacityByEffectGame

extends

FlameGame

{

7

bool

reset

=

false

;

8

@override

9

Future

<

void

>

onLoad

()

async

{

10

final

ember

=

EmberPlayer

(

11

position:

size

/

2

,

12

size:

size

/

4

,

13

onTap:

(

ember

)

{

14

if

(

reset

=

!

reset

)

{

15

ember

.

add

(

16

OpacityEffect

.

by

(

17

0.9

,

18

EffectController

(

duration:

0.75

),

19

),

20

);

21

}

else

{

22

ember

.

add

(

23

OpacityEffect

.

by

(

24

-

0.9

,

25

EffectController

(

duration:

0.75

),

26

),

27

);

28

}

29

},

30

)..

anchor

=

Anchor

.

center

;

31

32

add

(

ember

);

33

}

34

}

Code

final

effect

=

OpacityEffect

.

by

(

0.9

,

EffectController

(

duration:

0.75

),

);

Currently this effect can only be applied to components that have a

HasPaint

mixin. If the target component uses multiple paints, the effect can target any individual color using the

paintId

parameter.

GlowEffect

¶

Note

This effect is currently experimental, and its API may change in the future.

This effect will apply the glowing shade around target relative to the specified

glow-strength

. The color of shade will be targets paint color. For example, the following effect will apply the glowing shad

10

:

glow_effect.dart

1

import

'package:flame/components.dart'

;

2

import

'package:flame/effects.dart'

;

3

import

'package:flame/game.dart'

;

4

5

import

'package:flutter/material.dart'

;

6

7

class

GlowEffectExample

extends

FlameGame

{

8

@override

9

Future

<

void

>

onLoad

()

async

{

10

final

paint

=

Paint

()..

color

=

const

Color

(

0xff39FF14

);

11

12

add

(

13

CircleComponent

(

14

radius:

size

.

y

/

4

,

15

position:

size

/

2

,

16

anchor:

Anchor

.

center

,

17

paint:

paint

,

18

)..

add

(

19

GlowEffect

(

20

10.0

,

21

EffectController

(

22

duration:

2

,

23

infinite:

true

,

24

),

25

),

26

),

27

);

28

}

29

}

Code

final

effect

=

GlowEffect

(

10.0

,

EffectController

(

duration:

3

),

);

Currently this effect can only be applied to components that have a

HasPaint

mixin.

SequenceEffect

¶

This effect can be used to run multiple other effects one after another. The constituent effects may have dif

The sequence effect can also be alternating (the sequence will first run forward, and then backward); and a

sequence_effect.dart

```
1
import
'package:doc_flame_examples/flower.dart'
;
2
import
'package:flame/effects.dart'
;
3
import
'package:flame/game.dart'
;
4
5
class
SequenceEffectGame
extends
FlameGame
{
6
@override
7
Future
<
void
>
```

onLoad

()

async

{

8

final

flower

=

Flower

(

9

size:

40

,

10

position:

canvasSize

/

2

,

11

onTap:

(

flower

)

{

12

flower

.

add

(

13

SequenceEffect

([

14

ScaleEffect

.

by

(

15

Vector2

.

all

(

1.5

),

16

EffectController

(

duration:

0.2

,

alternate:

true

),

17

),

18

MoveEffect

.

by

(

19

Vector2

(

30

,

-

50

),

20

EffectController

(

duration:

0.5

),

21

),

22

MoveEffect

.

by

(

23

Vector2

(

-

30

,

50

),

24

EffectController

(

duration:

0.5

),

25

),

26

ScaleEffect

.

by

(

27

Vector2

.

all

(.

75

),

28

EffectController

(

duration:

0.2

,

alternate:

true

),

29

),

30

]),

31

);

32

},

33

);

34

add

(

flower

);

35

}

36

}

Code

final

effect

=

SequenceEffect

([

ScaleEffect

.

by

(

Vector2

.

all

(

1.5

),

EffectController

(

duration:

0.2

,

alternate:

true

,

),

),

MoveEffect

.

by

(

Vector2

(

30

,

-

50

),

EffectController

(

duration:

0.5

,

),

),

OpacityEffect

.

to

(

0

,

EffectController

(

duration:

0.3

,

),

),

RemoveEffect

(),

]);

RemoveEffect

¶

This is a simple effect that can be attached to a component causing it to be removed from the game tree af

remove_effect.dart

1

import

'package:doc_flame_examples/ember.dart'

;

2

```
import
'package:flame/components.dart'
;
3
import
'package:flame/effects.dart'
;
4
import
'package:flame/events.dart'
;
5
import
'package:flame/game.dart'
;
6
7
class
RemoveEffectGame
extends
FlameGame
with
TapDetector
{
8
static
```

const

double

delayTime

=

3

;

9

late

EmberPlayer

ember

;

10

late

TextComponent

textComponent

;

11

final

RemoveEffect

effect

=

RemoveEffect

(

delay:

delayTime

);

12

13

@override

14

Future

<

void

>

onLoad

()

async

{

15

add

(

16

ember

=

EmberPlayer

(

17

position:

size

/

2

,

18

size:

Vector2

(

45

,

40

),

19

)..

anchor

=

Anchor

.

center

,

20

);

21

add

(

textComponent

=

TextComponent

()..

position

=

Vector2

.

all

(

5

));

22

}

23

24

@override

25

void

onTap

()

{

26

if

(

children

.

contains

(

ember

))

```
{
27
ember
.
add
(
effect
);
28
}
else
{
29
effect
.
reset
();
30
add
(
ember
);
31
}
32
}
```

```
33
34
@override
35
void
update
(
double
dt
)
{
36
textComponent
.
text
=
37
(
effect
.
controller
.
progress
*
delayTime
).
```

toStringAsFixed

(

2

);

38

super

.

update

(

dt

);

39

}

40

}

Code

final

effect

=

RemoveEffect

(

delay:

3.0

);

ColorEffect

¶

This effect will change the base color of the paint, causing the rendered component to be tinted by the prov

Usage example:

color_effect.dart

```
1
import
'package:doc_flame_examples/ember.dart'
;
2
import
'package:flame/components.dart'
;
3
import
'package:flame/effects.dart'
;
4
import
'package:flame/game.dart'
;
5
import
'package:flutter/animation.dart'
;
6
import
'package:flutter/cupertino.dart'
```

;

7

8

class

ColorEffectExample

extends

FlameGame

{

9

bool

reset

=

false

;

10

11

@override

12

Future

<

void

>

onLoad

()

async

{

13

final

ember

=

EmberPlayer

(

14

position:

size

/

2

,

15

size:

size

/

4

,

16

onTap:

(

ember

)

{

17

if

(

reset

=

!

reset

)

{

18

ember

.

add

(

19

ColorEffect

(

20

const

Color

(

0xFF00FF00

),

21

EffectController

(

duration:

1.5

),

22

opacityTo:

0.6

,

23

),

24

);

25

}

else

{

26

ember

.

add

(

27

ColorEffect

(

28

const

Color

(

0xFF1039DB

),

29

EffectController

(

duration:

1.5

),

30

opacityTo:

0.6

,

31

),

32

);

33

}

34

},

35

)..

anchor

=

Anchor

.

center

;

36

37

add

(

ember

);

38

}

39

}

Code

final

effect

=

ColorEffect

(

const

Color

(

0xFF00FF00

),

EffectController

(

duration:

1.5

),

opacityFrom

=

0.2

,

opacityTo:

0.8

,

);

The

opacityFrom

and

opacityTo

arguments will determine ?how much? of the color that will be applied to the component. In this example th

Note:

Due to how this effect is implemented, and how Flutter?s

ColorFilter

class works, this effect can?t be mixed with other

ColorEffect

s, when more than one is added to the component, only the last one will have effect.

Creating new effects

¶

Although Flame provides a wide array of built-in effects, eventually you may find them to be insufficient. Lu

Each effect extends the base

Effect

class, possibly via one of the more specialized abstract subclasses such as

ComponentEffect<T>

or

Transform2DEffect

.

The

Effect

class? constructor requires an

EffectController

instance as an argument. In most cases you may want to pass that controller from your own constructor. L

Lastly, you will need to implement a single method

apply(double

progress)

that will be called at each update tick while the effect is active. In this method you are supposed to make c

In addition, you may want to implement callbacks

onStart()

and

onFinish()

if there are any actions that must be taken when the effect starts or ends.

When implementing the

apply()

method we recommend to use relative updates only. That is, change the target property by incrementing/d

Effect controllers

¶

EffectController

¶

The base

EffectController

class provides a factory constructor capable of creating a variety of common controllers. The syntax of the

EffectController

({

required

double

duration

,

Curve

curve

=

Curves

.

linear

,

double

?

reverseDuration

,

Curve

?

reverseCurve

,

bool

alternate

=

false

,

double

atMaxDuration

=

0.0

,

double

atMinDuration

=

0.0

,

int

?

repeatCount

,

bool

infinite

=

false

,

double

startDelay

=

0.0

,

VoidCallback

?

onMax

,

VoidCallback

?

onMin

,

});

duration

? the length of the main part of the effect, i.e. how long it should take to go from 0 to 100%. This parameter

duration

seconds.

curve

? if given, creates a non-linear effect that grows from 0 to 100% according to the provided

curve

.

reverseDuration

? if provided, adds an additional step to the controller: after the effect has grown from 0 to 100% over the

duration

seconds, it will then go backwards from 100% to 0 over the

reverseDuration

seconds. In addition, the effect will complete at progress level of 0 (normally the effect completes at progre

reverseCurve

? the curve to be used during the ?reverse? step of the effect. If not given, this will default to

curve.flipped

.

alternate

? setting this to true is equivalent to specifying the

reverseDuration

equal to the

duration

. If the

reverseDuration

is already set, this flag has no effect.

atMaxDuration

? if non-zero, this inserts a pause after the effect reaches its max progress and before the reverse stage. D

atMinDuration

? if non-zero, this inserts a pause after the reaches its lowest progress (0) at the end of the reverse stage.

repeatCount

? if greater than one, it will cause the effect to repeat itself the prescribed number of times. Each iteration w

infinite

? if true, the effect will repeat infinitely and never reach completion. This is equivalent to as if

repeatCount

was set to infinity.

startDelay

? an additional wait time inserted before the beginning of the effect. This wait time is executed only once, e

.started

property returns false. The effect?s

onStart()

callback will be executed at the end of this waiting period.

Using this parameter is the simplest way to create a chain of effects that execute one after another (or with

onMax

? callback function which will be invoked right after reaching its max progress and before the optional paus

onMin

? callback function which will be invoked right after reaching its lowest progress at the end of the reverse st

The effect controller returned by this factory constructor will be composited of multiple simpler effect contro

In addition to the factory constructor, the

EffectController

class defines a number of properties common for all effect controllers. These properties are:

.started

? true if the effect has already started. For most effect controllers this property is always true. The only exc

DelayedEffectController

which returns false while the effect is in the waiting stage.

.completed

? becomes true when the effect controller finishes execution.

.progress

? current value of the effect controller, a floating-point value from 0 to 1. This variable is the main ?output?

.duration

? total duration of the effect, or

null

if the duration cannot be determined (for example if the duration is random or infinite).

LinearEffectController

¶

This is the simplest effect controller that grows linearly from 0 to 1 over the specified

duration

:

final

ec

=

LinearEffectController

(

3

);

ReverseLinearEffectController

¶

Similar to the

LinearEffectController

, but it goes in the opposite direction and grows linearly from 1 to 0 over the specified duration:

final

ec

=

ReverseLinearEffectController

(

1

);

CurvedEffectController

¶

This effect controller grows non-linearly from 0 to 1 over the specified

duration

and following the provided

curve

:

final

ec

=

CurvedEffectController

(

0.5

,

Curves

.

easeOut

);

ReverseCurvedEffectController

¶

Similar to the

CurvedEffectController

, but the controller grows down from 1 to 0 following the provided

curve

:

final

ec

=

ReverseCurvedEffectController

(

0.5

,

Curves

.

bounceInOut

);

PauseEffectController

¶

This effect controller keeps the progress at a constant value for the specified time duration. Typically, the

progress

would be either 0 or 1:

final

ec

=

PauseEffectController

(

1.5

,

progress:

0

);

RepeatedEffectController

¶

This is a composite effect controller. It takes another effect controller as a child, and repeats it multiple time

final

ec

=

RepeatedEffectController

(

LinearEffectController

(

1

),

10

);

The child effect controller cannot be infinite. If the child is random, then it will be re-initialized with new ranc

InfiniteEffectController

¶

Similar to the

RepeatedEffectController

, but repeats its child controller indefinitely.

final

ec

=

InfiniteEffectController

(

LinearEffectController

(

1

));

SequenceEffectController

¶

Executes a sequence of effect controllers, one after another. The list of controllers cannot be empty.

final

ec

=

SequenceEffectController

([

LinearEffectController

(

1

),

PauseEffectController

(

0.2

),

ReverseLinearEffectController

(

1

),

]);

SpeedEffectController

¶

Alters the duration of its child effect controller so that the effect proceeds at the predefined speed. The initia

DurationEffectController

.

The

SpeedEffectController

can only be applied to effects for which the notion of speed is well-defined. Such effects must implement th

MeasurableEffect

interface. For example, the following effects qualify:

MoveByEffect

,

MoveToEffect

,

MoveAlongPathEffect

,

RotateEffect.by

,

RotateEffect.to

.

The parameter

speed

is in units per second, where the notion of a ?unit? depends on the target effect. For example, for move eff

final

ec1

=

SpeedEffectController

(

LinearEffectController

(

0

),

speed:

1

);

final

ec2

=

EffectController

(

speed:

1

);

// same as ec1

DelayedEffectController

¶

Effect controller that executes its child controller after the prescribed

delay

. While the controller is executing the ?delay? stage, the effect will be considered ?not started?, i.e. its

.started

property will be returning

false

.

final

ec

=

DelayedEffectController

(

LinearEffectController

(

1

),

delay:

5

);

NoiseEffectController

¶

This effect controller exhibits noisy behavior, i.e. it oscillates randomly around zero. Such effect controller c

final

ec

=

NoiseEffectController

(

duration:

0.6

,

frequency:

10

);

RandomEffectController

¶

This controller wraps another controller and makes its duration random. The actual value for the duration is

RepeatedEffectController

or

InfiniteEffectController

.

final

ec

=

RandomEffectController

.

uniform

(

LinearEffectController

(

0

),

// duration here is irrelevant

min:

0.5

,

max:

1.5

,

);

The user has the ability to control which

Random

source to use, as well as the exact distribution of the produced random durations. Two distributions ?

.uniform

and

.exponential

are included, any other can be implemented by the user.

SineEffectController

¶

An effect controller that represents a single period of the sine function. Use this to create natural-looking ha

SineEffectControllers

with different periods, will create a

Lissajous curve

.

final

ec

=

SineEffectController

(

period:

1

);

ZigzagEffectController

¶

Simple alternating effect controller. Over the course of one

period

, this controller will proceed linearly from 0 to 1, then to -1, and then back to 0. Use this for oscillating effect

EffectController

).

final

ec

=

ZigzagEffectController

(

period:

2

);

See also

¶

Examples of various effects

.

Controlling the player and adding some graphics

¶

Now that we have the base for our game and a component for our player, let?s add some interactivity to it.

There are a couple of ways of doing that in Flame. For this tutorial, we will do that by using one of Flame?s

PanDetector

.

This detector will make our game class receive pan (or drag) events. To do so, we just need to add the

PanDetector

mixin to our game class and override its listener methods; in our case, we will use the

onPanUpdate

method. The updated code will look like the following:

```
import

'package:flame/input.dart'

;

class

SpaceShooterGame

extends

FlameGame

with

PanDetector

{

late

Player

player

;

@override
```

void

onLoad

()

{

// omitted

}

@override

void

onPanUpdate

(

DragUpdateInfo

info

)

{

}

}

At this point, our game should be receiving all the pan update inputs, but we are not doing anything with th

We now need a way to move our player. That can be achieved by simply saving our

Player

component to a variable inside our game class, adding a method

move

to our

Player

, and just connect them:

class

Player

extends

PositionComponent

{

static

final

_paint

=

Paint

()..

color

=

Colors

.

white

;

@override

void

render

(

Canvas

canvas

)

{

canvas

.

drawRect

(

size

.

toRect

(),

_paint

);

}

void

move

(

Vector2

delta

)

{

position

.

add

(

delta

);

}

}

class

SpaceShooterGame

extends

FlameGame

with

PanDetector

{

late

Player

player

;

@override

Future

<

void

>

onLoad

()

async

{

await

super

.

onLoad

();

player

=

Player

()

..

position

=

size

/

2

..

width

=

50

..

height

=

100

..

anchor

=

Anchor

.

center

;

add

(

player

);

}

```
@override
void
onPanUpdate
(
DragUpdateInfo
info
)
{
player
.
move
(
info
.
delta
.
game
);
}
}
```

That is it! If you drag the screen, the player should follow your movement and we have just implemented ou

Before we move to our next step, let?s replace that boring white rectangle with some cool graphics.

Flame provides many classes to help us with graphical rendering. For this step, we are going to use the

`Sprite`

class.

`Sprite`

s are used in Flame to render static images or portions of them in the game. To render a

Sprite

inside a

FlameGame

, we should use the

SpriteComponent

class, which wraps the

Sprite

features into a component.

So let?s refactor our current implementation, first, we can replace our inheritance from

PositionComponent

to

SpriteComponent

(which is a component that extends from

PositionComponent

) and load the sprite:

```
class
Player
extends
SpriteComponent
{
void
move
(
Vector2
delta
```

)

{

position

.

add

(

delta

);

}

}

class

SpaceShooterGame

extends

FlameGame

with

PanDetector

{

late

Player

player

;

@override

Future

<

void

>?

onLoad

()

async

{

await

super

.

onLoad

();

final

playerSprite

=

await

loadSprite

(

'player-sprite.png'

);

player

=

Player

()

..

sprite

=

playerSprite

..

x

=

size

.

x

/

2

..

y

=

size

.

y

/

2

..

width

=

50

..

height

=

100

..

anchor

=

Anchor

.

center

;

add

(

player

);

}

@override

void

onPanUpdate

(

DragUpdateInfo

info

)

{

player

.

move

(

info

.

delta

.

game

);

}

}

And now, you should see a small blue spaceship on the screen!

A couple of notes worth mentioning:

Unlike

PositionComponent

,

SpriteComponent

has an implementation for the

render

method, so we can delete the previous override.

FlameGame

has a couple of methods for loading assets, like

loadSprite

. Those methods are quite useful, because when used,

FlameGame

will take care of cleaning any cache when the game is removed from the Flutter widget tree.

Before we close this step, there is one small improvement that we can do. Right now, we are loading the sp

Just like

FlameGame

, components also have an

onLoad

method that can be overridden to do initializations. But before we implement our player?s load method, not

loadSprite

method from the

FlameGame

class.

That is not a problem! Every time our component needs to access things from its game class, we can mix

HasGameRef

mixin; that will add a new variable to our component called

gameRef

which will point to the game instance where the component is running. Now, let?s refactor our game a little

class

Player

extends

SpriteComponent

with

HasGameRef

<

SpaceShooterGame

>

{

Player

()

:

super

(

size:

Vector2

(

100

,

150

),

anchor:

Anchor

.

center

,

);

@override

Future

<

void

>

onLoad

()

async

{

await

super

.

onLoad

();

sprite

=

await

gameRef

.

loadSprite

(

'player-sprite.png'

);

position

=

gameRef

.

size

/

2

;

}

void

move

(

Vector2

delta

)

{

position

.

add

(

```
delta
);
}
}
class
SpaceShooterGame
extends
FlameGame
with
PanDetector
{
late
Player
player
;
@override
Future
<
void
>
onLoad
()
async
{
await
super
```

.

onLoad

();

player

=

Player

();

add

(

player

);

}

@override

void

onPanUpdate

(

DragUpdateInfo

info

)

{

player

.

move

(

info

.

delta

.

game

);

}

}

If you run the game now, you will not notice any visual differences, but now we have a more scalable struct

Run

main.dart

1

import

'package:flame/components.dart'

;

2

import

'package:flame/events.dart'

;

3

import

'package:flame/game.dart'

;

4

import

'package:flame/input.dart'

;

5

```
import
'package:flutter/material.dart'
;
6
7
void
main
()
{
8
runApp
(
GameWidget
(
game:
SpaceShooterGame
()));
9
}
10
11
class
SpaceShooterGame
extends
FlameGame
with
```

PanDetector

{

12

late

Player

player

;

13

14

@override

15

Future

<

void

>

onLoad

()

async

{

16

player

=

Player

();

17

add

(

player

);

18

}

19

20

@override

21

void

onPanUpdate

(

DragUpdateInfo

info

)

{

22

player

.

move

(

info

.

delta

.

global

);

23

}

24

}

25

26

class

Player

extends

SpriteComponent

with

HasGameReference

<

SpaceShooterGame

>

{

27

Player

()

28

:

super

(

29

size:

Vector2

(

100

,

150

),

30

anchor:

Anchor

.

center

,

31

);

32

33

@override

34

Future

<

void

>

onLoad

()

async

{

35

await

super

.

onLoad

();

36

37

sprite

=

await

game

.

loadSprite

(

'player-sprite.png'

);

38

39

position

=

game

.

size

/

2

;

40

anchor

=

Anchor

.

center

;

41

}

42

43

void

move

(

Vector2

delta

)

{

44

position

.

add

(

delta

);

45

}

46

}

Code

Gesture Input

¶

This is documentation for gesture inputs attached directly on the game class, most of the time you want to

TapCallbacks

and

DragCallbacks

for that.

For other input documents, see also:

Keyboard Input

: for keystrokes

Other Inputs

: For joysticks, game pads, etc.

Intro

¶

Inside

package:flame/gestures.dart

you can find a whole set of

mixin

s which can be included on your game class instance to be able to receive touch input events. Below you c

mixin

s and its methods:

Touch and mouse detectors

¶

- TapDetector - onTap - onTapCancel - onTapDown - onLongTapDown - onTapUp - SecondaryTap

Mouse only events

- MouseMovementDetector - onMouseMove - ScrollDetector - onScroll

It is not possible to mix advanced detectors (

MultiTouch*

) with basic detectors of the same kind, since the advanced detectors will

always win the gesture arena

and the basic detectors will never be triggered. So for example, you can?t use both

MultiTouchTapDetector

and

PanDetector

together, since no events will be triggered for the latter (there is also an assertion for this).

Flame?s GestureApi is provided by Flutter?s Gesture Widgets, including

GestureDetector widget

,

RawGestureDetector widget

and

MouseRegion widget

, you can also read more about Flutter?s gestures

here

.

PanDetector and ScaleDetector

¶

If you add a

PanDetector

together with a

ScaleDetector

you will be prompted with a quite cryptic assertion from Flutter that says:

Note

Having both a pan gesture recognizer and a scale gesture recognizer is redundant; scale is a superset of p

Just use the scale gesture recognizer.

This might seem strange, but

onScaleUpdate

is not only triggered when the scale should be changed, but for all pan/drag events too. So if you need to u

onScaleUpdate

(+

onScaleStart

and

onScaleEnd

).

For example you could do something like this if you want to move the camera on pan events and zoom on

late

double

startZoom

;

@override

void

onScaleStart

(

_

)

{

startZoom

=

camera

.

zoom

;

}

@override

void

onScaleUpdate

(

ScaleUpdateInfo

info

)

{

final

currentScale

=

info

.

scale

.

global

;

if

(

!

currentScale

.

isIdentity

())

{

camera

.

zoom

=

startZoom

*

currentScale

.

y

;

}

else

{

camera

.

translateBy

(

-

info

.

delta

.

global

);

camera

.

snap

();

}

}

In the example above the pan events are handled with

info.delta

and the scale events with

info.scale

, although they are theoretically both from underlying scale events.

This can also be seen in the

zoom example

.

Mouse cursor

¶

It is also possible to change the current mouse cursor displayed on the

GameWidget

region. To do so the following code can be used inside the

Game

class

mouseCursor

.

value

=

SystemMouseCursors

.

move

;

To already initialize the

GameWidget

with a custom cursor, the

mouseCursor

property can be used

GameWidget

(

game:

MouseCursorGame

(),

mouseCursor:

SystemMouseCursors

.

move

,

);

Event coordinate system

¶

On events that have positions, like for example

Tap*

or

Drag

, you will notice that the

eventPosition

attribute includes 2 fields:

global

and

widget

. Below you will find a brief explanation about each of them.

global

¶

The position where the event occurred considering the entire screen, same as

globalPosition

in Flutter?s native events.

widget

¶

The position where the event occurred relative to the

GameWidget

position and size, same as

localPosition

in Flutter?s native events.

Example

¶

class

MyGame

extends

FlameGame

with

```
TapDetector
{
// Other methods omitted
@override
bool
onTapDown
(
TapDownInfo
info
)
{
print
(
"Player tap down on
${
info
.
eventPosition
.
widget
}
"
);
return
true
;
```

```
}
@override
bool
onTapUp
(
TapUpInfo
info
)
{
print
(
"Player tap up on
${
info
.
eventPosition
.
widget
}
"
);
return
true
;
}
}
```

You can also check more complete examples

here

.

GestureHitboxes

¶

The

GestureHitboxes

mixin is used to more accurately recognize gestures on top of your

Component

s. Say that you have a fairly round rock as a

SpriteComponent

for example, then you don?t want to register input that is in the corner of the image where the rock is not di

PositionComponent

is rectangular by default. Then you can use the

GestureHitboxes

mixin to define a more accurate circle or polygon (or another shape) for which the input should be within fo

You can add new hitboxes to the component that has the

GestureHitboxes

mixin just like they are added in the below

Collidable

example.

More information about how to define hitboxes can be found in the hitbox section of the

collision detection

docs.

An example of how to use it can be seen

here

.

<<local>>

¶

The

<<local>>

command creates a new variable within the current node, and initializes it to some starting value. Thus, it is

<<declare>>

, except that the variable it creates is visible within a single node only.

The syntax of the

<<local>>

command can be one of the following:

<<

local

$VARIABLE

=

EXPRESSION

>>

<<

local

$VARIABLE

=

EXPRESSION

as

TYPE

>>

This would create a variable with the name

$VARIABLE

(all variables in YarnSpinner start with a

$

sign), and assign it the value of

EXPRESSION

. In the second form, it will ensure that the type of the expression is equal to

TYPE

, otherwise a compile-time error will be thrown. Thus, the second form serves as the explicit annotation for

The following restrictions apply:

each local variable can be declared only once within a node;

the name of a local variable cannot coincide with the name of any global variable.

Examples

¶

In this example the variable

$roll

will only be needed temporarily within this one node, so it wouldn?t make sense to declare it as global.

title

:

a_dice_roll

---

<<

local

$roll

=

dice

(

6

)>>

<<

if

$roll

==

1

>>

You've rolled 1, rotten luck...

<<

elseif

$roll

==

2

>>

You've rolled 2, which is still below the average. Try harder!

<<

elseif

$roll

==

3

>>

You've rolled 3.14159265 (well, almost).

<<

elseif

$roll

==

4

>>

Your roll is an unlucky number. Please roll again

<<

else

>>

You've rolled 10 (when rounded to the nearest ten). Good job!

<<

endif

>>

===

Node

¶

The

Node

class represents a single

node

within the

.yarn

script. The objects of this class will be delivered to your

DialogueView

s with the methods

onNodeStart()

,

onNodeFinish()

.

Properties

¶

title

String

The title (name) of the node.

tags

Map<String,

String>

Additional tags specified in the header of the node. The map will be empty if there were no tags besides th

title

tag.

iterator

Iterator<DialogueEntry>

The content of the node, which is a sequence of

DialogueLine

s,

DialogueChoice

s, or

Command

s.

3. Cards

¶

In this chapter we will begin implementing the most visible component in the game ? the

Card

component, which corresponds to a single real-life card. There will be 52

Card

objects in the game.

Each card has a

rank

(from 1 to 13, where 1 is an Ace, and 13 is a King) and a

suit

(from 0 to 3: hearts ?, diamonds ?, clubs ?, and spades ?). Also, each card will have a boolean flag

faceUp

, which controls whether the card is currently facing up or down. This property is important both for renderin

The rank and the suit are simple properties of a card, they aren?t components, so we need to make a decis

int

, or as an

enum

, or as objects. The choice will depend on what operations we need to perform with them. For the rank, we

Rank

and

Suit

as classes.

Suit

¶

Create file

suit.dart

and declare an

@immutable

class

Suit

there, with no parent. The

@immutable

annotation here is just a hint for us that the objects of this class should not be modified after creation.

Next, we define the factory constructor for the class:

Suit.fromInt(i)

. We use a factory constructor here in order to enforce the singleton pattern for the class: instead of creatin

_singletons

list:

factory

Suit

.

fromInt

(

int

index

)

{

assert

(

index

>=

0

&&

index

<=

3

);

return

_singletons

[

index

];

}

After that, there is a private constructor

Suit._()

. This constructor initializes the main properties of each

Suit

object: the numeric value, the string label, and the sprite object which we will later use to draw the suit sym

klondikeSprite()

function that we created in the previous chapter:

Suit

.

_

(

this

.

value

,

this

.

label

,

double

x

,

double

y

,

double

w

,

double

h

)

:

sprite

=

klondikeSprite

(

x

,

y

,

w

,

h

);

final

int

value

;

final

String

label

;

final

Sprite

sprite

;

Then comes the static list of all

Suit

objects in the game. Note that we define it as static variable so it is evaluated lazily (as if it was marked wit

late

keyword) meaning that it will be only initialized the first time it is needed. This is important: as we can see a

static

final

List

<

Suit

>

_singletons

=

[

Suit

.

_

(

0

,

'?'

,

1176

,

17

,

172

,

183

),

Suit

.

_

(

1

,

'?'

,

973

,

14

,

177

,

182

),

Suit

.

–

(

2

,

'?'

,

974

,

226

,

184

,

172

),

Suit

.

_

(

3

,

'?'

,

1178

,

220

,

176

,

182

),

];

The last four numbers in the constructor are the coordinates of the sprite image within the sprite sheet

klondike-sprites.png

. If you?re wondering how I obtained these numbers, the answer is that I used a free online service

spritecow.com

? it?s a handy tool for locating sprites within a sprite sheet.

Lastly, I have simple getters to determine the ?color? of a suit. This will be needed later when we need to e

/// Hearts and Diamonds are red, while Clubs and Spades are black.

bool

get

isRed

=>

value

<=

1

;

bool

get

isBlack

=>

value

>=

2

;

Rank

¶

The

Rank

class is very similar to

Suit

. The main difference is that

Rank

contains two sprites instead of one, separately for ranks of ?red? and ?black? colors. The full code for the

Rank

class is as follows:

import

'package:flame/components.dart'

;

import

'package:flame/flame.dart'

;

import

'package:flutter/foundation.dart'

;

@immutable

class

Rank

{

factory

Rank

.

fromInt

(

int

value

)

{

assert

(

value

>=

1

&&

value

<=

13

);

return

_singletons

[

value

-

1

];

}

Rank

.

_

(

this

.

value

,

this

.

label

,

double

x1

,

double

y1

,

double

x2

,

double

y2

,

double

w

,

double

h

,

)

:

redSprite

=

klondikeSprite

(

x1

,

y1

,

w

,

h

),

blackSprite

=

klondikeSprite

(

x2

,

y2

,

w

,

h

);

final

int

value

;

final

String

label

;

final

Sprite

redSprite

;

final

Sprite

blackSprite

;

static

final

List

<

Rank

>

_singletons

=

[

Rank

.

_

(

1

,

'A'

,

335

,

164

,

789

,

161

,

120

,

129

),

Rank

.

–

(

2

,

'2'

,

20

,

19

,

15

,

322

,

83

,

125

),

Rank

.

_

(

3

,

'3'

,

122

,

19

,

117

,

322

,

80

,

127

),

Rank

.

_

(

4

,

'4'

,

213

,

12

,

208

,

315

,

93

,

132

),

Rank

.

_

(

5

,

'5'

,

314

,

21

,

309

,

324

,

85

,

125

),

Rank

.

–

(

6

,

'6'

,

419

,

17

,

414

,

320

,

84

,

129

),

Rank

.

–

(

7

,

'7'

,

509

,

21

,

505

,

324

,

92

,

128

),

Rank

.

_

(

8

,

'8'

,

612

,

19

,

607

,

322

,

78

,

127

),

Rank

.

_

(

9

,

'9'

,

709

,

19

,

704

,

322

,

84

,

130

),

Rank

.

–

(

10

,

'10'

,

810

,

20

,

805

,

322

,

137

,

127

),

Rank

.

–

(

11

,

'J'

,

15

,

170

,

469

,

167

,

56

,

126

),

Rank

.

_

(

12

,

'Q'

,

92

,

168

,

547

,

165

,

132

,

128

),

Rank

.

_

(

13

,

'K'

,

243

,

170

,

696

,

167

,

92

,

123

),

];

}

Card component

¶

Now that we have the

Rank

and the

Suit

classes, we can finally start implementing the

Card

component. Create file

components/card.dart

and declare the

Card

class extending from the

PositionComponent

:

class

Card

extends

PositionComponent

{}

The constructor of the class will take integer rank and suit, and make the card initially facing down. Also, w

cardSize

constant defined in the

KlondikeGame

class:

Card

(

int

intRank

,

int

intSuit

)

:

rank

=

Rank

.

fromInt

(

intRank

),

suit

=

Suit

.

fromInt

(

intSuit

),

_faceUp

=

false

,

super

(

size:

KlondikeGame

.

cardSize

);

final

Rank

rank

;

final

Suit

suit

;

bool

_faceUp

;

The

_faceUp

property is private (indicated by the underscore) and non-final, meaning that it can change during the lifetim

bool

get

isFaceUp

=>

_faceUp

;

bool

get

isFaceDown

=>

!

_faceUp

;

void

flip

()

=>

_faceUp

=

!

_faceUp

;

Lastly, let?s add a simple

toString()

implementation, which may turn out to be useful when we need to debug the game:

@override

String

toString

()

=>

rank

.

label

+

suit

.

label

;

// e.g. "Q?" or "10?"

Before we proceed with implementing the rendering, we need to add some cards into the game. Head over

KlondikeGame

class and add the following at the bottom of the

onLoad

method:

final

random

=

Random

();

for

(

var

i

=

0

;

i

<

7

;

i

++

)

{

for

(

var

j

=

0

;

j

<

4

;

j

++

)

{

final

card

=

Card

(

random

.

nextInt

(

13

)

+

1

,

random

.

nextInt

(

4

))

..

position

=

Vector2

(

100

+

i

*

1150

,

100

+

j

*

1500

)

..

addToParent

(

world

);

if

(

random

.

nextDouble

()

<

0.9

)

{

// flip face up with 90% probability

card

.

flip

();

}

}

}

This snippet is a temporary code ? we will remove it in the next chapter ? but for now it lays down 28 rando

Rendering

¶

In order to be able to see a card, we need to implement its

render()

method. Since the card has two distinct states ? face up or down ? we will implement rendering for these tw

Card

class:

```
@override
void
render
(
Canvas
canvas
)
{
if
(
_faceUp
)
{
_renderFront
(
canvas
);
}
else
{
```

_renderBack

(

canvas

);

}

}

void

_renderFront

(

Canvas

canvas

)

{}

void

_renderBack

(

Canvas

canvas

)

{}

renderBack()

¶

Since rendering the back of a card is simpler, we will do it first.

The

render()

method of a

PositionComponent

operates in a local coordinate system, which means we don?t need to worry about where the card is locate

width

and down by

height

pixels.

There is a lot of artistic freedom in how to draw the back of a card, but my implementation contains a solid

void

_renderBack

(

Canvas

canvas

)

{

canvas

.

drawRRect

(

cardRRect

,

backBackgroundPaint

);

canvas

.

drawRRect

(

cardRRect

,

backBorderPaint1

);

canvas

.

drawRRect

(

backRRectInner

,

backBorderPaint2

);

flameSprite

.

render

(

canvas

,

position:

size

/

2

,

anchor:

Anchor

.

center

);

}

The most interesting part here is the rendering of a sprite: we want to render it in the middle (

size/2

), and we use

Anchor.center

to tell the engine that we want the

center

of the sprite to be at that point.

Various properties used in the

_renderBack()

method are defined as follows:

static

final

Paint

backBackgroundPaint

=

Paint

()

..

color

=

const

Color

(

0xff380c02

);

static

final

Paint

backBorderPaint1

=

Paint

()

..

color

=

const

Color

(

0xffdbaf58

)

..

style

=

PaintingStyle

.

stroke

..

strokeWidth

=

10

;

static

final

Paint

backBorderPaint2

=

Paint

()

..

color

=

const

Color

(

0x5CEF971B

)

..

style

=

PaintingStyle

.

stroke

..

strokeWidth

=

35

;

static

final

RRect

cardRRect

=

RRect

.

fromRectAndRadius

(

KlondikeGame

.

cardSize

.

toRect

(),

const

Radius

.

circular

(

KlondikeGame

.

cardRadius

),

);

static

final

RRect

backRRectInner

=

cardRRect

.

deflate

(

40

);

static

final

Sprite

flameSprite

=

klondikeSprite

(

1367

,

6

,

357

,

501

);

I declared these properties as static because they will all be the same across all 52 card objects, so we mig

renderFront()

¶

When rendering the face of a card, we will follow the standard card design: the rank and the suit in two opp

As before, we begin by declaring some constants that will be used for rendering. The background of a card

static

final

Paint

frontBackgroundPaint

=

Paint

()

..

color

=

const

Color

(

0xff000000

);

static

final

Paint

redBorderPaint

=

Paint

()

..

color

=

const

Color

(

0xffece8a3

)

..

style

=

PaintingStyle

.

stroke

..

strokeWidth

=

10

;

static

final

Paint

blackBorderPaint

=

Paint

()

..

color

=

const

Color

(

0xff7ab2e8

)

..

style

=

PaintingStyle

.

stroke

..

strokeWidth

=

10

;

Next, we also need the images for the court cards:

static

final

Sprite

redJack

=

klondikeSprite

(

81

,

565

,

562

,

488

);

static

final

Sprite

redQueen

=

klondikeSprite

(

717

,

541

,

486

,

515

);

static

final

Sprite

redKing

=

klondikeSprite

(

1305

,

532

,

407

,

549

);

Note that I?m calling these sprites

redJack

,

redQueen

, and

redKing

. This is because, after some trial, I found that the images that I have don?t look very well on black-suit car

tint

them with a blueish hue. Tinting of a sprite can be achieved by using a paint with

colorFilter

set to the specified color and the

srcATop

blending mode:

static

final

blueFilter

=

Paint

()

..

colorFilter

=

const

ColorFilter

.

mode

(

Color

(

0x880d8bff

),

BlendMode

.

srcATop

,

);

static

final

Sprite

blackJack

=

klondikeSprite

(

81

,

565

,

562

,

488

)

..

paint

=

blueFilter

;

static

final

Sprite

blackQueen

=

klondikeSprite

(

717

,

541

,

486

,

515

)

..

paint

=

blueFilter

;

static

final

Sprite

blackKing

=

klondikeSprite

(

1305

,

532

,

407

,

549

)

..

paint

=

blueFilter

;

Now we can start coding the render method itself. First, draw the background and the card border:

void

_renderFront

(

Canvas

canvas

)

{

canvas

.

drawRRect

(

cardRRect

,

frontBackgroundPaint

);

canvas

.

drawRRect

(

cardRRect

,

suit

.

isRed

?

redBorderPaint

:

blackBorderPaint

,

);

}

In order to draw the rest of the card, I need one more helper method. This method will draw the provided sp

rotate=true

is passed, the sprite will be drawn as if it was rotated 180º around the center of the card:

void

_drawSprite

(

Canvas

canvas

,

Sprite

sprite

,

double

relativeX

,

double

relativeY

,

{

double

scale

=

1

,

bool

rotate

=

false

,

})

{

if

(

rotate

)

{

canvas

.

save

();

canvas

.

translate

(

size

.

x

/

2

,

size

.

y

/

2

);

canvas

.

rotate

(

pi

);

canvas

.

translate

(

-

size

.

x

/

2

,

-

size

.

y

/

2

);

}

sprite

.

render

(

canvas

,

position:

Vector2

(

relativeX

*

size

.

x

,

relativeY

*

size

.

y

),

anchor:

Anchor

.

center

,

size:

sprite

.

srcSize

.

scaled

(

scale

),

);

if

(

rotate

)

{

canvas

.

restore

();

}

}

Let?s draw the rank and the suit symbols in the corners of the card. Add the following to the

_renderFront()

method:

final

rankSprite

=

suit

.

isBlack

?

rank

.

blackSprite

:

rank

.

redSprite

;

final

suitSprite

=

suit

.

sprite

;

_drawSprite

(

canvas

,

rankSprite

,

0.1

,

0.08

);

_drawSprite

(

canvas

,

rankSprite

,

0.1

,

0.08

,

rotate:

true

);

_drawSprite

(

canvas

,

suitSprite

,

0.1

,

0.18

,

scale:

0.5

);

_drawSprite

(

canvas

,

suitSprite

,

0.1

,

0.18

,

scale:

0.5

,

rotate:

true

);

The middle of the card is rendered in the same manner: we will create a big switch statement on the card?s

switch

(

rank

.

value

)

{

case

1

:

_drawSprite

(

canvas

,

suitSprite

,

0.5

,

0.5

,

scale:

2.5

);

break

;

case

2

:

_drawSprite

(

canvas

,

suitSprite

,

0.5

,

0.25

);

_drawSprite

(

canvas

,

suitSprite

,

0.5

,

0.25

,

rotate:

true

);

break

;

case

3

:

_drawSprite

(

canvas

,

suitSprite

,

0.5

,

0.2

);

_drawSprite

(

canvas

,

suitSprite

,

0.5

,

0.5

);

_drawSprite

(

canvas

,

suitSprite

,

0.5

,

0.2

,

rotate:

true

);

break

;

case

4

:

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.25

);

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.25

);

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.25

,

rotate:

true

);

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.25

,

rotate:

true

);

break

;

case

5

:

```
_drawSprite
(
canvas
,
suitSprite
,
0.3
,
0.25
);
_drawSprite
(
canvas
,
suitSprite
,
0.7
,
0.25
);
_drawSprite
(
canvas
,
suitSprite
,
```

0.3

,

0.25

,

rotate:

true

);

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.25

,

rotate:

true

);

_drawSprite

(

canvas

,

suitSprite

,

0.5

,

0.5

);

break

;

case

6

:

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.25

);

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.25

);

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.5

);

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.5

);

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.25

,

rotate:

true

);

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.25

,

rotate:

true

);

break

;

case

7

:

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.2

);

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.2

);

_drawSprite

(

canvas

,

suitSprite

,

0.5

,

0.35

);

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.5

);

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.5

);

```
_drawSprite
(
canvas
,
suitSprite
,
0.3
,
0.2
,
rotate:
true
);
_drawSprite
(
canvas
,
suitSprite
,
0.7
,
0.2
,
rotate:
true
);
```

break

;

case

8

:

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.2

);

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.2

);

_drawSprite

(

canvas

,

suitSprite

,

0.5

,

0.35

);

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.5

);

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.5

);

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.2

,

rotate:

true

);

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.2

,

rotate:

true

);

_drawSprite

(

canvas

,

suitSprite

,

0.5

,

0.35

,

rotate:

true

);

break

;

case

9

:

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.2

);

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.2

);

_drawSprite

(

canvas

,

suitSprite

,

0.5

,

0.3

);

_drawSprite

```
(
canvas
,
suitSprite
,
0.3
,
0.4
);
_drawSprite
(
canvas
,
suitSprite
,
0.7
,
0.4
);
_drawSprite
(
canvas
,
suitSprite
,
0.3
```

,

0.2

,

rotate:

true

);

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.2

,

rotate:

true

);

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.4

,

rotate:

true

);

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.4

,

rotate:

true

);

break

;

case

10

:

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.2

);

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.2

);

_drawSprite

(

canvas

,

suitSprite

,

0.5

,

0.3

);

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.4

);

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.4

);

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.2

,

rotate:

true

);

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.2

,

rotate:

true

);

_drawSprite

(

canvas

,

suitSprite

,

0.5

,

0.3

,

rotate:

true

);

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.4

,

rotate:

true

);

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.4

,

rotate:

true

);

break

;

case

11

:

_drawSprite

(

canvas

,

suit

.

isRed

?

redJack

:

blackJack

,

0.5

,

0.5

);

break

;

case

12

:

_drawSprite

(

canvas

,

suit

.

isRed

?

redQueen

:

blackQueen

,

0.5

,

0.5

);

break

;

case

13

:

_drawSprite

(

canvas

,

suit

.

isRed

?

redKing

:

blackKing

,

0.5

,

0.5

);

break

;

}

And this is it with the rendering of the

Card

component. If you run the code now, you would see four rows of cards neatly spread on the table. Refreshi

In the next chapter we will discuss how to implement interactions with the cards, that is, how to make them

Run

components/card.dart

1

import

'dart:math'

;

2

import

'dart:ui'

;

3

4

import

'package:flame/components.dart'

;

5

import

'../klondike_game.dart'

;

6

import

'../rank.dart'

;

7

import

'../suit.dart'

;

8

9

class

Card

extends

PositionComponent

{

10

Card

(

int

intRank

,

int

intSuit

)

11

:

rank

=

Rank

.

fromInt

(

intRank

),

12

suit

=

Suit

.

fromInt

(

intSuit

),

13

_faceUp

=

false

,

14

super

(

size:

KlondikeGame

.

cardSize

);

15

16

final

Rank

rank

;

17

final

Suit

suit

;

18

bool

_faceUp

;

19

20

bool

get

isFaceUp

=>

_faceUp

;

21

void

flip

()

=>

_faceUp

=

!

_faceUp

;

22

23

@override

24

String

toString

()

=>

rank

.

label

+

suit

.

label

;

// e.g. "Q?" or "10?"

25

26

@override

27

```
void
render
(
Canvas
canvas
)
{
28
if
(
_faceUp
)
{
29
_renderFront
(
canvas
);
30
}
else
{
31
_renderBack
(
canvas
```

);

32

}

33

}

34

35

static

final

Paint

backBackgroundPaint

=

Paint

()

36

..

color

=

const

Color

(

0xff380c02

);

37

static

final

Paint

backBorderPaint1

=

Paint

()

38

..

color

=

const

Color

(

0xffdbaf58

)

39

..

style

=

PaintingStyle

.

stroke

40

..

strokeWidth

=

10

;

41

static

final

Paint

backBorderPaint2

=

Paint

()

42

..

color

=

const

Color

(

0x5CEF971B

)

43

..

style

=

PaintingStyle

.

stroke

44

..

strokeWidth

=

35

;

45

static

final

RRect

cardRRect

=

RRect

.

fromRectAndRadius

(

46

KlondikeGame

.

cardSize

.

toRect

(),

47

const

Radius

.

circular

(

KlondikeGame

.

cardRadius

),

48

);

49

static

final

RRect

backRRectInner

=

cardRRect

.

deflate

(

40

);

50

static

final

Sprite

flameSprite

=

klondikeSprite

(

1367

,

6

,

357

,

501

);

51

52

void

_renderBack

(

Canvas

canvas

)

{

53

canvas

.

drawRRect

(

cardRRect

,

backBackgroundPaint

);

54

canvas

.

drawRRect

(

cardRRect

,

backBorderPaint1

);

55

canvas

.

drawRRect

(

backRRectInner

,

backBorderPaint2

);

56

flameSprite

.

render

(

canvas

,

position:

size

/

2

,

anchor:

Anchor

.

center

);

57

}

58

59

static

final

Paint

frontBackgroundPaint

=

Paint

()

60

..

color

=

const

Color

(

0xff000000

);

61

static

final

Paint

redBorderPaint

=

Paint

()

62

..

color

=

const

Color

(

0xffece8a3

)

63

..

style

=

PaintingStyle

.

stroke

64

..

strokeWidth

=

10

;

65

static

final

Paint

blackBorderPaint

=

Paint

()

66

..

color

=

const

Color

(

0xff7ab2e8

)

67

..

style

=

PaintingStyle

.

stroke

68

..

strokeWidth

=

10

;

69

static

final

blueFilter

=

Paint

()

70

..

colorFilter

=

const

ColorFilter

.

mode

(

71

Color

(

0x880d8bff

),

72

BlendMode

.

srcATop

,

73

);

74

static

final

Sprite

redJack

=

klondikeSprite

(

81

,

565

,

562

,

488

);

75

static

final

Sprite

redQueen

=

klondikeSprite

(

717

,

541

,

486

,

515

);

76

static

final

Sprite

redKing

=

klondikeSprite

(

1305

,

532

,

407

,

549

);

77

static

final

Sprite

blackJack

=

klondikeSprite

(

81

,

565

,

562

,

488

)

78

..

paint

=

blueFilter

;

79

static

final

Sprite

blackQueen

=

klondikeSprite

(

717

,

541

,

486

,

515

)

80

..

paint

=

blueFilter

;

81

static

final

Sprite

blackKing

=

klondikeSprite

(

1305

,

532

,

407

,

549

)

82

..

paint

=

blueFilter

;

83

```
84
void
_renderFront
(
Canvas
canvas
)
{
85
canvas
.
drawRRect
(
cardRRect
,
frontBackgroundPaint
);
86
canvas
.
drawRRect
(
87
cardRRect
,
88
```

suit

.

isRed

?

redBorderPaint

:

blackBorderPaint

,

89

);

90

91

final

rankSprite

=

suit

.

isBlack

?

rank

.

blackSprite

:

rank

.

redSprite

;

92

final

suitSprite

=

suit

.

sprite

;

93

_drawSprite

(

canvas

,

rankSprite

,

0.1

,

0.08

);

94

_drawSprite

(

canvas

,

suitSprite

,

0.1

,

0.18

,

scale:

0.5

);

95

_drawSprite

(

canvas

,

rankSprite

,

0.1

,

0.08

,

rotate:

true

);

96

_drawSprite

(

canvas

,

suitSprite

,

0.1

,

0.18

,

scale:

0.5

,

rotate:

true

);

97

switch

(

rank

.

value

)

{

98

case

1

:

99

_drawSprite

(

canvas

,

suitSprite

,

0.5

,

0.5

,

scale:

2.5

);

100

break

;

101

case

2

:

102

_drawSprite

(

canvas

,

suitSprite

,

0.5

,

0.25

);

103

_drawSprite

(

canvas

,

suitSprite

,

0.5

,

0.25

,

rotate:

true

);

104

break

;

105

case

3

:

```
106
_drawSprite
(
canvas
,
suitSprite
,
0.5
,
0.2
);
107
_drawSprite
(
canvas
,
suitSprite
,
0.5
,
0.5
);
108
_drawSprite
(
canvas
```

,
suitSprite
,
0.5
,
0.2
,
rotate:
true
);
109
break
;
110
case
4
:
111
_drawSprite
(
canvas
,
suitSprite
,
0.3
,

0.25

);

112

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.25

);

113

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.25

,

rotate:

true

);

114

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.25

,

rotate:

true

);

115

break

;

116

case

5

:

117

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.25

);

118

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.25

);

119

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.25

,

rotate:

true

);

120

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.25

,

rotate:

true

);

121

_drawSprite

(

canvas

,

suitSprite

,

0.5

,

0.5

);

122

break

;

123

case

6

:

124

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.25

);

125

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.25

);

126

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.5

);

127

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.5

);

128

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.25

,

rotate:

true

);

129

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.25

,

rotate:

true

);

130

break

;

131

case

7

:

132

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.2

);

133

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.2

);

134

_drawSprite

(

canvas

,

suitSprite

,

0.5

,

0.35

);

135

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.5

);

136

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.5

);

137

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.2

,

rotate:

true

);

138

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.2

,

rotate:

true

);

139

break

;

140

case

8

:

141

```
_drawSprite
(
canvas
,
suitSprite
,
0.3
,
0.2
);
142
_drawSprite
(
canvas
,
suitSprite
,
0.7
,
0.2
);
143
_drawSprite
(
canvas
,
```

suitSprite

,

0.5

,

0.35

);

144

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.5

);

145

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.5

);

146

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.2

,

rotate:

true

);

147

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.2

,

rotate:

true

);

148

_drawSprite

(

canvas

,

suitSprite

,

0.5

,

0.35

,

rotate:

true

);

149

break

;

150

case

9

:

151

```
_drawSprite
(
canvas
,
suitSprite
,
0.3
,
0.2
);
152
_drawSprite
(
canvas
,
suitSprite
,
0.7
,
0.2
);
153
_drawSprite
(
canvas
,
```

suitSprite

,

0.5

,

0.3

);

154

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.4

);

155

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.4

);

156

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.2

,

rotate:

true

);

157

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.2

,

rotate:

true

);

158

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.4

,

rotate:

true

);

159

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.4

,

rotate:

true

);

160

break

;

161

case

10

:

162

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.2

);

163

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.2

);

164

_drawSprite

(

canvas

,

suitSprite

,

0.5

,

0.3

);

165

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.4

);

166

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.4

);

167

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.2

,

rotate:

true

);

168

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.2

,

rotate:

true

);

169

_drawSprite

(

canvas

,

suitSprite

,

0.5

,

0.3

,

rotate:

true

);

170

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.4

,

rotate:

true

);

171

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.4

,

rotate:

true

);

172

break

;

173

case

11

:

174

_drawSprite

(

canvas

,

suit

.

isRed

?

redJack

:

blackJack

,

0.5

,

0.5

);

175

break

;

176

case

12

:

177

_drawSprite

(

canvas

,

suit

.

isRed

?

redQueen

:

blackQueen

,

0.5

,

0.5

);

178

break

;

179

case

13

:

180

_drawSprite

(

canvas

,

suit

.

isRed

?

redKing

:

blackKing

,

0.5

,

0.5

);

181

break

;

182

}

183

}

184

185

void

_drawSprite

(

186

Canvas

canvas

,

187

Sprite

sprite

,

188

double

relativeX

,

189

double

relativeY

,

{

190

double

scale

=

1

,

191

bool

rotate

=

false

,

192

})

{

193

if

(

rotate

)

{

194

canvas

.

save

();

195

canvas

.

translate

(

size

.

x

/

2

,

size

.

y

/

2

);

196

canvas

.

rotate

(

pi

);

197

canvas

.

translate

(

-

size

.

x

/

2

,

-

size

.

y

/

2

);

198

}

199

sprite

.

render

(

200

canvas

,

201

position:

Vector2

(

relativeX

*

size

.

x

,

relativeY

*

size

.

y

),

202

anchor:

Anchor

.

center

,

203

size:

sprite

.

srcSize

.

scaled

(

scale

),

204

);

205

if

(

rotate

)

{

206

canvas

.

restore

();

207

}

208

}

209

}

components/foundation.dart

1

import

'package:flame/components.dart'

;

2

3

class

Foundation

extends

PositionComponent

{

4

@override

5

bool

get

debugMode

=>

true

;

6

}

components/pile.dart

1

import

'package:flame/components.dart'

;

2

3

class

Pile

extends

PositionComponent

{

4

@override

5

bool

get

debugMode

=>

true

;

6

}

components/stock.dart

1

import

'package:flame/components.dart'

;

2

3

class

Stock

extends

PositionComponent

{

4

@override

5

bool

get

debugMode

=>

true

;

6

}

components/waste.dart

1

import

'package:flame/components.dart'

;

2

3

class

Waste

extends

PositionComponent

{

4

@override

5

bool

get

debugMode

=>

true

;

6

}

klondike_game.dart

1

import

'dart:math'

;

2

3

```
import
'package:flame/components.dart'
;
4
import
'package:flame/flame.dart'
;
5
import
'package:flame/game.dart'
;
6
7
import
'components/card.dart'
;
8
import
'components/foundation.dart'
;
9
import
'components/pile.dart'
;
10
import
```

'components/stock.dart'

;

11

import

'components/waste.dart'

;

12

13

class

KlondikeGame

extends

FlameGame

{

14

static

const

double

cardGap

=

175.0

;

15

static

const

double

cardWidth

=

1000.0

;

16

static

const

double

cardHeight

=

1400.0

;

17

static

const

double

cardRadius

=

100.0

;

18

static

final

Vector2

cardSize

=

Vector2

(

cardWidth

,

cardHeight

);

19

20

@override

21

Future

<

void

>

onLoad

()

async

{

22

await

Flame

.

images

.

load

(

'klondike-sprites.png'

);

23

24

final

stock

=

Stock

()

25

..

size

=

cardSize

26

..

position

=

Vector2

(

cardGap

,

cardGap

);

27

final

waste

=

Waste

()

28

..

size

=

cardSize

29

..

position

=

Vector2

(

cardWidth

+

2

*

cardGap

,

cardGap

);

30

final

foundations

=

List

.

generate

(

31

4

,

32

(

i

)

=>

Foundation

()

33

..

size

=

cardSize

34

..

position

=

35

Vector2

((

i

+

3

)

*

(

cardWidth

+

cardGap

)

+

cardGap

,

cardGap

),

36

);

37

final

piles

=

List

.

generate

(

38

7

,

39

(

i

)

=>

Pile

()

40

..

size

=

cardSize

41

..

position

=

Vector2

(

42

cardGap

+

i

*

(

cardWidth

+

cardGap

),

43

cardHeight

+

2

*

cardGap

,

44

),

45

);

46

47

world

.

add

(

stock

);

48

world

.

```
add
(
waste
);
49
world
.
addAll
(
foundations
);
50
world
.
addAll
(
piles
);
51
52
camera
.
viewfinder
.
visibleGameSize
=
```

53

Vector2

(

cardWidth

*

7

+

cardGap

*

8

,

4

*

cardHeight

+

3

*

cardGap

);

54

camera

.

viewfinder

.

position

=

Vector2

(

cardWidth

*

3.5

+

cardGap

*

4

,

0

);

55

camera

.

viewfinder

.

anchor

=

Anchor

.

topCenter

;

56

57

final

random

=

Random

();

58

for

(

var

i

=

0

;

i

<

7

;

i

++

)

{

59

for

(

var

j

=

0

;

j

<

4

;

j

++

)

{

60

final

card

=

Card

(

random

.

nextInt

(

13

)

+

1

,

random

.

nextInt

(

4

))

61

..

position

=

Vector2

(

100

+

i

*

1150

,

100

+

j

*

1500

)

62

..

addToParent

(

world

);

63

// flip the card face-up with 90% probability

64

if

(

random

.

nextDouble

()

<

0.9

)

{

65

card

.

flip

();

66

}

67

}

68

}

69

}

70

}

71

72

Sprite

klondikeSprite

(

double

x

,

double

y

,

double

width

,

double

height

)

{

73

return

Sprite

(

74

Flame

.

images

.

fromCache

(

'klondike-sprites.png'

),

75

srcPosition:

Vector2

(

x

,

y

),

76

srcSize:

Vector2

(

width

,

height

),

77

);

78

}

main.dart

1

import

'package:flame/game.dart'

;

2

import

'package:flutter/widgets.dart'

;

3

4

import

'klondike_game.dart'

;

5

6

void

main

()

{

7

final

game

=

KlondikeGame

();

8

runApp

(

GameWidget

(

game:

game

));

9

}

rank.dart

1

import

'package:flame/components.dart'

;

2

import

'package:flutter/foundation.dart'

;

3

import

'klondike_game.dart'

;

4

5

@immutable

6

class

Rank

{

7

factory

Rank

.

fromInt

(

int

value

)

{

8

assert

(

9

value

>=

1

&&

value

<=

13

,

10

'value is outside of the bounds of what a rank can be'

,

11

);

12

return

_singletons

[

value

-

1

];

13

}

14

15

Rank

.

_

(

16

this

.

value

,

17

this

.

label

,

18

double

x1

,

19

double

y1

,

20

double

x2

,

21

double

y2

,

22

double

w

,

23

double

h

,

24

)

:

redSprite

=

klondikeSprite

(

x1

,

y1

,

w

,

h

),

25

blackSprite

=

klondikeSprite

(

x2

,

y2

,

w

,

h

);

26

27

final

int

value

;

28

final

String

label

;

29

final

Sprite

redSprite

;

30

final

Sprite

blackSprite

;

31

32

static

final

List

<

Rank

>

_singletons

=

[

33

Rank

.

_

(

1

,

'A'

,

335

,

164

,

789

,

161

,

120

,

129

),

34

Rank

.

–

(

2

,

'2'

,

20

,

19

,

15

,

322

,

83

,

125

),

35

Rank

.

–

(

3

,

'3'

,

122

,

19

,

117

,

322

,

80

,

127

),

36

Rank

.

_

(

4

,

'4'

,

213

,

12

,

208

,

315

,

93

,

132

),

37

Rank

.

_

(

5

,

'5'

,

314

,

21

,

309

,

324

,

85

,

125

),

38

Rank

.

–

(

6

,

'6'

,

419

,

17

,

414

,

320

,

84

,

129

),

39

Rank

.

–

(

7

,

'7'

,

509

,

21

,

505

,

324

,

92

,

128

),

40

Rank

.

–

(

8

,

'8'

,

612

,

19

,

607

,

322

,

78

,

127

),

41

Rank

.

_

(

9

,

'9'

,

709

,

19

,

704

,

322

,

84

,

130

),

42

Rank

.

_

(

10

,

'10'

,

810

,

20

,

805

,

322

,

137

,

127

),

43

Rank

.

–

(

11

,

'J'

,

15

,

170

,

469

,

167

,

56

,

126

),

44

Rank

.

–

(

12

,

'Q'

,

92

,

168

,

547

,

165

,

132

,

128

),

45

Rank

.

–

(

13

,

'K'

,

243

,

170

,

696

,

167

,

92

,

123

),

46

];

47

}

suit.dart

1

import

'package:flame/sprite.dart'

;

2

import

'package:flutter/foundation.dart'

;

3

import

'klondike_game.dart'

;

4

5

@immutable

6

class

Suit

{

7

factory

Suit

.

fromInt

(

int

index

)

{

8

assert

(

9

index

>=

0

&&

index

<=

3

,

10

'index is outside of the bounds of what a suit can be'

,

11

);

12

return

_singletons

[

index

];

13

}

14

15

Suit

.

_

(

this

.

value

,

this

.

label

,

double

x

,

double

y

,

double

w

,

double

h

)

16

:

sprite

=

klondikeSprite

(

x

,

y

,

w

,

h

);

17

18

final

int

value

;

19

final

String

label

;

20

final

Sprite

sprite

;

21

22

static

final

List

<

Suit

>

_singletons

=

[

23

Suit

.

–

(

0

,

'?'

,

1176

,

17

,

172

,

183

),

24

Suit

.

–

(

1

,

'?'

,

973

,

14

,

177

,

182

),

25

Suit

.

–

(

2

,

'?'

,

974

,

226

,

184

,

172

),

26

Suit

.

_

(

3

,

'?'

,

1178

,

220

,

176

,

182

),

27

];

28

29

/// Hearts and Diamonds are red, while Clubs and Spades are black.

30

bool

get

isRed

=>

value

<=

1

;

31

bool

get

isBlack

=>

value

>=

2

;

32

}

Code

CharacterStorage

¶

class

CharacterStorage

The

CharacterStorage

is a cache for all

Character

s defined in your yarn scripts. This container is populated from the

<<character>>

commands as the YarnProject parses the input scripts.

Properties

¶

isEmpty

?

bool

isNotEmpty

?

bool

Methods

¶

contains

(

String

name

)

?

bool

Returns

true

if a character with the given name or alias was defined.

operator[]

(

String

name

)

?

Character?

Retrieves the character with the given name/alias, or returns

null

if such character was not present.

clear

(

)

Clear all characters from storage.

This could be used between scenes in preparation for loading a new set of characters. It would not general

remove

(

String

name

)

Remove a character by name. Its aliases will also be removed.

This could be used if you are certain a character is no longer required. It would not generally be used while

Accessing character storage

¶

Character storage is accessed via the

YarnProject

.

final

characters

=

yarnProject

.

characters

;

Removing characters

¶

There may be situations where characters need to be removed from storage. For example, in a game with

Remove all characters with

clear

.

yarnProject

.

characters

.

clear

();

Use

remove

to remove a single character. Pass in the name of the character or any of its aliases. The character and all

yarnProject

.

characters

.

remove

(

'Jenny'

);

## Supported Platforms

¶

Since Flame runs on top of Flutter, so its supported platforms depend on which platforms that are supporte

At the moment, Flame supports web, mobile(Android and iOS) and desktop (Windows, MacOS and Linux).

## Flutter channels

¶

Flame keeps it support on the stable channel. The dev, beta and master channel should work, but we don?

## Flame web

¶

To use Flame on web you need to make sure your game is using the CanvasKit/

Skia

renderer. This will increase performance on the web, as it will use the

canvas

element instead of separate DOM elements.

To run your game using skia, use the following command:

```
flutter
run
-d
chrome
--web-renderer
canvaskit
```

To build the game for production, using skia, use the following:

```
flutter
build
web
--release
```

--web-renderer

canvaskit

Deploy your game to GitHub Pages

¶

One easy way to deploy your game online, is to use

GitHub Pages

. It is a cool feature from GitHub, by which you can easily host web content from your repository.

Here we will explain the easiest way to get your game hosted using GitHub pages.

First thing, lets create the branch where your deployed files will live:

git

checkout

-b

gh-pages

This branch can be created from

main

or any other place, it doesn?t matter much. After you push that branch go back to your

main

branch.

Now you should add the

flutter-gh-pages

action to your repository, you can do that by creating a file

gh-pages.yaml

under the folder

.github/workflows

.

name

:

Gh-Pages

on

:

push

:

branches

:

[

main

]

jobs

:

build

:

runs-on

:

ubuntu-latest

steps

:

-

uses

:

actions/checkout@v3

-

uses

:

subosito/flutter-action@v2

-

uses

:

bluefireteam/flutter-gh-pages@v8

with

:

baseHref

:

/NAME_OF_YOUR_REPOSITORY/

webRenderer

:

canvaskit

Be sure to change

NAME_OF_YOUR_REPOSITORY

to the name of your GitHub repository.

Now, whenever you push something to the

main

branch, the action will run and update your deployed game.

The game should be available at an URL like this:

https://YOUR_GITHUB_USERNAME.github.io/YOUR_REPO_NAME/

Deploy your game to itch.io

¶

Create a web build, either from your IDE or by running

flutter

build

web

(If it complains about

Missing

index.html

run

flutter

create

.

--platforms=web

)

Go into

index.html

and remove the line that says

<base

href="/">

zip the

build/web

folder and upload to itch.io

Remember that it shouldn?t be the

web

directory in your project?s root, but in

build/web

!

If you are submitting your game to a game jam, remember to make it public and submit it on the game jam

Further instructions can be found on

itch.io

.

Web support

¶

When using Flame on the web some methods may not work. For example

Flame.device.setOrientation

and

Flame.device.fullScreen

won?t work on web, they can be called, but nothing will happen.

Another example: pre caching audio using

flame_audio

package also doesn?t work due to Audioplayers not supporting it on web. This can be worked around by us

http

package, and requesting a get to the audio file, that will make the browser cache the file producing the sam

If you want to create instances of

ui.Image

on the web you can use our

Flame.images.decodeImageFromPixels

method. This wraps the

decodeImageFromPixels

from the

ui

library, but with support for the web platform. If the

runAsWeb

argument is set to

true

(by default it is set to

kIsWeb

) it will decode the image using an internal image method. When the

runAsWeb

is

false

it will use the

decodeImageFromPixels

, which is currently not supported on the web.

Markup

¶

Markup

is a mechanism for annotating fragments of a dialogue line. They are somewhat similar to HTML tags, or y

Syntax

¶

Markup tags are denoted with the name of the tag, placed in square brackets:

[tag_name]

. The corresponding closing tag would be

[/tag_name]

. Every markup tag must have a corresponding closing tag:

Hello,

[wavy]

world

[/wavy]

!

Markup tags may nest within each other, though they must nest properly, in the sense that one markup ran

Lorem

[S]

ipsum dolor

[A]

sit

[/A]

amet

[/S]

, consectetur

[B]

adipiscing

[/B]

elit

The special

close-all

markup tag

[/]

closes all currently opened markup ranges. It is also handy in situations where the name of the markup tag

Lorem ipsum dolor sit amet,

[bold]

consectetur adipiscing elit

[/]

The

self-closing

markup tags have the form

[tag_name/]

. These tags mark a single location within the text. In addition, if such tag is surrounded by spaces on both

Lorem ipsum dolor sit amet,

[wave/]

consectetur adipiscing elit.

Markup tags also accept parameters, which are similar to HTML tag attributes. The names of these parame

Lorem ipsum

[color name=$color]

dolor sit amet

[/color]

Markup tags can surround dynamic text (interpolated expressions), which will cause the length of the mark

Hello,

[b]

{

$player

}

[/b]

!

Lastly, it should be noted that if you want to have an actual text in square brackets within a line, then in ord

\

:

Hello,

\[

world

\]

!

See also

MarkupAttribute

: the runtime representation of a markup attribute within a line.

Examples

¶

Mark a piece of text with a different style

¶

In this example the word ?Voldemort? is rendered with a special ?cursed? markup, indicating that the word

title

:

Scene117_Harry_MrMalfoy

---

Harry

:

I'm not afraid of

[cursed]

Voldemort

[/cursed]

!

MrMalfoy

:

You must be really brave... or really

[i]

stupid

[/i]

?

===

Provide additional information about a text fragment

¶

In this example the word ?Llewellyn? has a tooltip information associated with it. A game might render this

title

:

MonkDialogue

---

Monk

:

Visit

[tooltip place="TS" x=23 y=-74]

Llewellyn

[/]

in Thunderstorm,

\

he will be able to help you.

===

Indicate where special non-text tokens may be inserted

¶

The

[item/]

markup tag will be replaced by the item?s name, which will also be interactive: tapping this name would bri

title

:

BlacksmithQuest

---

<<

local

$reward

=

if

(

$chapter

==

1

,

"

A0325

"

,

"

A1018

"

)>>

Smith

:

Find me my lost ring, and I'll give you this

[item id=$reward/]

.

===

<<wait>>

¶

The

<<wait>>

command forces the dialogue engine to wait for the specified duration (in seconds) before resuming the dia

// Wait for a quarter of a second

<<

wait

0.25

>>

// Wait for the amount of time given by the $delay variable

<<

wait

$delay

>>

<<jump>>

¶

The

<<jump>>

command stops executing the current node, and then immediately starts running the target node. This is si

goto

in many programming languages. For example:

<<

jump

FarewellScene

>>

The argument of this command is the id of the node to jump to. It can be given either as a plain node ID, or

<<

jump

{

"

Ending_

"

+

$ending

}>>

If the expression evaluates at runtime to an unknown name, then a

NameError

exception will be thrown.

See Also

¶

<<visit>>

command, which jumps into the destination node temporarily and then returns to the same place in the dial

Inputs

¶

Tap Events

Drag Events

Gesture Input

Keyboard Input

Other Inputs and Helpers

Hardware Keyboard Detector

Adding Enemies

¶

Now that the starship is able to shoot, we need something for the player to shoot at! So for this step we will

So first things first, let?s create an

Enemy

class that will represent the enemies in game:

class

Enemy

extends

SpriteAnimationComponent

with

HasGameReference

<

SpaceShooterGame

>

{

Enemy

({

super

.

position

,

})

:

super

(

size:

Vector2

.

all

(

enemySize

),

anchor:

Anchor

.

center

,

);

static

const

enemySize

=

50.0

;

@override

Future

<

void

>

onLoad

()

```
async
{
await
super
.
onLoad
();
animation
=
await
game
.
loadSpriteAnimation
(
'enemy.png'
,
SpriteAnimationData
.
sequenced
(
amount:
4
,
stepTime:
.
2
```

,

textureSize:

Vector2

.

all

(

16

),

),

);

}

@override

void

update

(

double

dt

)

{

super

.

update

(

dt

);

position

.

y

+=

dt

*

250

;

if

(

position

.

y

>

game

.

size

.

y

)

{

removeFromParent

();

}

}

}

Note that for now, the

Enemy

class is super similar to the

Bullet

one, the only differences are their sizes, animation information and that bullets travel from bottom to top, w

Next we need to make the enemies spawn in the game, the logic that we will do here will be simple, we wil

x

axis.

Once again, we could manually make all the time based event in the game?s

update

method, maintain a random instance to get the enemy x position and so on and so forth, but Flame provide

SpawnComponent

! So in the

SpaceShooterGame.onLoad

method let?s add the following code:

```
add
(
SpawnComponent
(
factory
:
(
index
)
{
return
Enemy
```

();

},

period:

1

,

area:

Rectangle

.

fromLTWH

(

0

,

0

,

size

.

x

,

-

Enemy

.

enemySize

),

),

);

The

SpawnComponent

will take a couple of arguments, let?s review them as they appear in the code:

factory

receives a function which has the index of the component that should be created. We don?t use the index i

Enemy

.

period

simply define the interval in which a new component will be spawned.

area

defines the possible area where the components can be placed once created. In our case they should be p

And this concludes this short step!

CommandStorage

¶

The

CommandStorage

is a part of

YarnProject

responsible for storing all

user-defined commands

. You can access it as the

YarnProject.commands

property.

The command storage can be used to register any number of custom commands, making them available t

In order to register a function as a yarn command, the function must satisfy several requirements:

The function?s return value must be

void

or

Future<void>

. If the function returns a future, then that future will be awaited before proceeding to the next step of the di

<<walk>>

,

<<moveCamera>>

, or

<<prompt>>

.

The function?s arguments must be of types that are known to Yarn:

String

,

num

,

int

,

double

, or

bool

. All arguments must be positional, with no defaults.

In order to register the function, use methods

addCommand0()

?

addCommand3()

, according to the number of function?s arguments.

If the function?s signature has 1 or more booleans at the end, then those arguments will be considered opt

false

.

Methods

¶

hasCommand

(

String

name

) ?

bool

Returns the status of whether the command

name

has been added to the storage.

addCommand0

(

String

name

,

FutureOr<void>

Function()

fn

)

Registers a no-argument function

fn

as the command

name

.

addCommand1

(

String

name

,

FutureOr<void>

Function(T1)

fn

)

Registers a single-argument function

fn

as the command

name

.

addCommand2

(

String

name

,

FutureOr<void>

Function(T1,

T2)

fn

)

Registers a two-argument function

fn

as the command

name

.

addCommand3

(

String

name

,

FutureOr<void>

Function(T1,

T2,

T3)

fn

)

Registers a three-argument function

fn

as the command

name

.

addOrphanedCommand

(

name

)

Registers a command

name

which is not backed by any Dart function. Such command will still be delivered to

DialogueView

s via the

onCommand()

callback, but its arguments will not be parsed.

clear

Removes all user-defined commands

remove

(

String

name

)

Removes the user-defined command with the specified

name

.

Properties

¶

length

?

int

The number of user-defined commands registered so far.

isEmpty

?

bool

Returns

true

if no user-defined commands were registered.

isNotEmpty

?

bool

Returns

true

if any commands have been registered

Examples

¶

<<StartQuest>>

¶

Suppose we want to have a yarn command

<<StartQuest>>

, which would initiate a quest. The command would take the quest name and quest ID as arguments. Tech

A typical invocation of this command might look like this (note that the name of the quest is in quotes, othe

"Get"

,

"rid"

,

"of"

, and

"bandits"

):

<<

StartQuest

Q037 "Get rid of bandits"

>>

In order to implement this command, we create a Dart function

startQuest()

with two string arguments. The function will do a brief animated ?Started quest X? message, but we don?t

void

, not a future. Finally, we register the command with

commands.addCommand2()

.

class

MyGame

{

late

YarnProject

yarnProject

;

void

startQuest

(

String

questId

,

String

questName

)

{

assert

(

quests

.

containsKey

(

questId

));

assert

(

quests

[

questId

]

!

.

name

==

questName

);

// ...

}

@override

void

onLoad

()

{

yarnProject

=

YarnProject

()

..

commands

.

addCommand2

(

'StartQuest'

,

startQuest

);

}

}

Note that the name of the Dart function is different from the name of the command ? you can choose whate

<<prompt>>

¶

The

<<prompt>>

function will open a modal dialogue and ask the user to enter their response. This command will be waiting

$prompt

, and then the dialogue can access that variable in order to read the result of the prompt.

class

MyGame

{

final

YarnProject

yarnProject

=

YarnProject

();

Future

<

void

>

prompt

(

String

message

)

async

{

// This will wait until the modal dialog is popped from the router stack

final

name

=

await

router

.

pushAndWait

(

KeyboardDialog

(

message

));

yarnProject

.

variables

.

setVariable

(

r'$prompt'

,

name

);

}

@override

void

onLoad

()

{

yarnProject

..

variables

.

setVariable

(

r'$prompt'

,

''

)

..

commands

.

addCommand1

(

'prompt'

,

prompt

);

}

}

Then in a yarn script this command can be used like this:

<<

declare

$name

as

String

>>

title

:

Greeting

---

Guide

:

Hello, my name is Jenny, and you?

<<

prompt

"Enter your name:"

>>

<<

set

$player

=

$prompt

>>

// Store the name for later

Guide

:

Nice to meet you,

{

$player

}

===

<<give>>

¶

Suppose that we want to make a command that will give the player a certain item, or a number of items. Th

<<

give

{

$quest_reward

}

TraderJoe

>>

Note that the quest reward variable will contain both the reward item and its amount, for example it could b

"100

gold"

,

"5

potion_of_healing"

, or

'1

"Sword

of

Darkness"'

. When such variable is substituted into the command at runtime, the command becomes equivalent to

<<

give

100 gold TraderJoe

>>

<<

give

5 potion_of_healing TraderJoe

>>

<<

give

1 "Sword of Darkness" TraderJoe

>>

which will then be parsed as a regular 3-argument command corresponding to the following Dart function:

/// Takes [amount] of [item]s from [source] and gives them to the player.

void

give

(

int

amount

,

String

item

,

String

source

)

{

// ...

}

See also

¶

The description of

user-defined commands

in the YarnSpinner language.

The

UserDefinedCommand

class, which is used to inform a

DialogueView

that a custom command is being executed.

Ember Quest Game Tutorial

¶

In this tutorial, we will follow a step-by-step process for coding a game using the Flame engine.

This tutorial assumes that you have at least some familiarity with common programming concepts, and with

Dart

programming language.

flame_rive

¶

Overview

How to use it

Full Example

6. Adding the HUD

¶

Setting up the HUD

¶

Now that the game is up and running, the rest of the code should come fairly easily. To prepare for the hud

lib/ember_quest.dart

. Add the following to the top of the class:

int

starsCollected

=

0

;

int

health

=

3

;

Start by creating a folder called

lib/overlays

, and in that folder, create a component called

heart.dart

. This is going to be the health monitoring component in the upper left-hand corner of the game. Add the fo

import

'package:ember_quest/ember_quest.dart'

;

import

'package:flame/components.dart'

;

enum

HeartState

{

available

,

unavailable

,

}

class

HeartHealthComponent

extends

SpriteGroupComponent

<

HeartState

>

with

HasGameReference

<

EmberQuestGame

>

{

final

int

heartNumber

;

HeartHealthComponent

({

required

this

.

heartNumber

,

required

super

.

position

,

required

super

.

size

,

super

.

scale

,

super

.

angle

,

super

.

anchor

,

super

.

priority

,

});

@override

Future

<

void

>

onLoad

()

async

{

await

super

.

onLoad

();

final

availableSprite

=

```
await
game
.
loadSprite
(
'heart.png'
,
srcSize:
Vector2
.
all
(
32
),
);
final
unavailableSprite
=
await
game
.
loadSprite
(
'heart_half.png'
,
srcSize:
```

Vector2

.

all

(

32

),

);

sprites

=

{

HeartState

.

available:

availableSprite

,

HeartState

.

unavailable:

unavailableSprite

,

};

current

=

HeartState

.

available

;

}

@override

void

update

(

double

dt

)

{

if

(

game

.

health

<

heartNumber

)

{

current

=

HeartState

.

unavailable

;

}

else

{

current

=

HeartState

.

available

;

}

super

.

update

(

dt

);

}

}

The

HeartHealthComponent

is just a

SpriteGroupComponent

that uses the heart images that were created early on. The unique thing that is being done, is when the cor

heartNumber

, so in the

update

method, we check to see if the

game.health

is less than the

heartNumber

and if so, change the state of the component to unavailable.

To put this all together, create

hud.dart

in the same folder and add the following code:

import

'package:flame/components.dart'

;

import

'package:flutter/material.dart'

;

import

'../ember_quest.dart'

;

import

'heart.dart'

;

class

Hud

extends

PositionComponent

with

HasGameReference

<

EmberQuestGame

>

{

Hud

({

super

.

position

,

super

.

size

,

super

.

scale

,

super

.

angle

,

super

.

anchor

,

super

.

children

,

super

.

priority

=

5

,

});

late

TextComponent

_scoreTextComponent

;

@override

Future

<

void

>

onLoad

()

async

{

_scoreTextComponent

=

TextComponent

(

text:

'

${

game

.

starsCollected

}

'

,

textRenderer:

TextPaint

(

style:

const

TextStyle

(

fontSize:

32

,

color:

Color

.

fromRGBO

(

10

,

10

,

10

,

1

),

),

),

anchor:

Anchor

.

center

,

position:

Vector2

(

game

.

size

.

x

-

60

,

20

),

);

add

(

_scoreTextComponent

);

final

starSprite

=

await

game

.

loadSprite

(

'star.png'

);

add

(

SpriteComponent

(

sprite:

starSprite

,

position:

Vector2

(

game

.

size

.

x

-

100

,

20

),

size:

Vector2

.

all

(

32

),

anchor:

Anchor

.

center

,

),

);

for

(

var

i

=

1

;

i

<=

game

.

health

;

i

++

)

{

final

positionX

=

40

*

i

;

await

add

(

HeartHealthComponent

(

heartNumber:

i

,

position:

Vector2

(

positionX

.

toDouble

(),

20

),

size:

Vector2

.

all

(

32

),

),

);

}

}

@override

void

update

(

double

dt

)

{

_scoreTextComponent

.

text

=

'

${

game

.

starsCollected

}

'

;

}

}

In the

onLoad

method, you can see where we loop from 1 to the

game.health

amount, to create the number of hearts necessary. The last step is to add the hud to the game.

Go to

lib/ember_quest.dart

and add the following code in the

initializeGame

method:

camera

.

viewport

.

add

(

Hud

());

If the auto-import did not occur, you will need to add:

import

'overlays/hud.dart'

;

If you run the game now, you should see:

Updating the HUD Data

¶

The last thing we need to do before closing out the HUD is to update the data. To do this, we need to open

lib/actors/ember.dart

and add the following code:

onCollision

if

(

other

is

Star

)

{

other

.

removeFromParent

();

game

.

starsCollected

++

;

}

void

hit

()

{

if

(

!

hitByEnemy

)

{

game

.

health

--

;

hitByEnemy

=

true

;

}

add

(

OpacityEffect

.

fadeOut

(

EffectController

(

alternate:

true

,

duration:

0.1

,

repeatCount:

5

,

),

)..

onComplete

=

()

{

hitByEnemy

=

false

;

},

);

}

If you run the game now, you will see that your health is updated and the stars are incremented as appropr

7. Adding Menus

, we will finish the game by adding the main menu and the game-over menu.

Operators

¶

The

operators

are special symbols that perform common mathematical operations. For example, operator

+

performs summation, and thus we can write

$x

+

$y

to denote the sum of variables

$x

and

$y

. There are over 20 different operators in YarnSpinner, which can be loosely grouped into the following cat

Operator types

¶

Arithmetic

¶

The

arithmetic

operators, have the same meaning as in regular math. These apply to numeric arguments (with the except

+

which can also be used with strings):

operator

name

notes

+

addition

-

subtraction

Also, a unary minus

*

multiplication

/

division

Division by

0

is not allowed, and will throw a runtime error if it occurs.

%

modulo

This operator can apply to both integer and decimal numbers, and it returns the remainder of integer divisio

%

cannot be zero or a negative number, otherwise a runtime error will be thrown. The result of

x

%

y

is always a number in the range

[0;

y)

, regardless of the sign of

x

.

+

concatenation

When applied to strings, the

+

operator simply glues them together. For example,

"Hello"

+

"World"

produces string

"HelloWorld"

.

Logical

¶

The

logical

operators apply to boolean values. These operators can be written either in symbolic or word form ? both fo

operator

name

notes

!

,

not

logical NOT

This is a unary operator that inverts its operand:

!true

is

false

, and

!false

is

true

.

&&

,

and

logical AND

Returns

true

if both of its arguments are

true

.

||

,

or

logical OR

Returns

true

if at least one of its arguments is

true

.

^

,

xor

logical XOR

Returns

true

if the arguments are different, and

false

if they are the same.

Assignment

¶

The

assignment

operators modify the value of a variable. The left-hand side of such an operator is the variable that shall be

operator

name

notes

=

,

to

assign

$var

=

X

stores the value of

X

into the variable

$var

+=

increase

$var

+=

X

is equivalent to

$var

=

$var

+

X

-=

decrease

$var

-=

X

is equivalent to

$var

=

$var

-

X

*=

multiply

$var

*=

X

is equivalent to

$var

=

$var

*

X

/=

divide

$var

/=

X

is equivalent to

$var

=

$var

/

X

%=

reduce modulo

$var

%=

X

is equivalent to

$var

=

$var

%

X

Unlike all other operators, the assignment operators do not produce a value. This means they cannot be us

3

+

($x

+=

7)

. Instead, the assignment operators are only usable at the top level of commands such as

<<set>>

,

<<declare>>

, and

<<local>>

.

Relational

¶

The

relational

operators compare various values. The first two operators in this list can be applied to operands of any typ

operator

name

notes

==

equality

!=

inequality

<

less than

<=

less than or equal

>

greater than

>=

greater than or equal

Note that operator chaining is not supported. Thus, for example,

$x

==

$y

==

$z

will first compare variables

$x

and

$y

, then the result of that comparison, which is either

true

or

false

, will be compared with variable

$z

. Given that such expressions would be highly confusing to a reader, we recommend against using them. If

$x

,

$y

and

$z

are the same, then you should use the

&&

operator instead:

$x

==

$y

&&

$x

==

$z

.

Precedence

¶

Just as in mathematics, the operators have precedence ordering among them, meaning that some operato

3

+

4

*

5

, then the result will be

23

instead of

35

because multiplication has higher precedence than addition and thus evaluates first.

The precedence order is as follows, from highest to lowest:

*

,

/

,

%

;

-

,

+

;

==

,

!=

,

<

,

<=

,

>=

,

>

;

!

;

&&

,

^

;

||

;

=

,

+=

,

-=

,

*=

,

/=

,

%=

.

You can use parentheses

()

in order to alter the order of evaluation. For example,

(3

+

4)

*

5

is

35

instead of

23

.

Jenny Runtime

¶

# Flame Style Guide

¶

This is a general style guide for writing code within Flame and adjacent projects. We strive to maintain the

This guide extends upon the official

Dart?s style guide

. Please make sure to read that document first, as it is sure to improve your skill in Dart programming.

## Code Formatting

¶

Most of the code formatting rules are enforced automatically via the linter. Run the following commands to

flutter

analyze dart

format

.

## Code Structure

¶

### Imports

¶

If you?re using an external symbol that?s defined in multiple libraries, prefer importing the smallest of them

package:meta/meta.dart

to import annotations like

@protected

, or

dart:ui

to import

Canvas

.

Never import

package:flutter/cupertino.dart

or

package:flutter/material.dart

? prefer a much smaller library

package:flutter/widgets.dart

if you?re working with widgets.

Exports

¶

Strongly prefer to have only one public class per file, and name the file after that class. Having several priv

A possible exception to this rule is if the ?main? class requires some small ?helper? classes that need to b

The ?main? class in a file should be located at the start of the file (right after the imports section), so that it

If multiple public symbols are defined in a file, then they must be exported explicitly using the

export

...

show

...

statement. For example:

export

'src/effects/provider_interfaces.dart'

show

AnchorProvider

,

AngleProvider

,

PositionProvider

,

ScaleProvider

,

SizeProvider

;

Assertions

¶

Use

assert

s to detect contract violations, or pre-condition/post-condition failures. Sometimes, however, using exceptio

Use an assert with a clear error message to check for a condition that is in developers? control. For examp

opacity

level as an input, you should check whether the value is in the range from 0 to 1. Consider also including th

assert

(

0

<=

opacity

&&

opacity

<=

1

,

'The opacity value must be from 0 to 1:

$

opacity

'

);

Always use asserts as early as possible to detect possible violations. For example, check the validity of

opacity

in the constructor/setter, instead of in the render function.

When adding such an assert, also include a test that checks that the assert triggers. This test would verify

Use an assert without an error message to check for a condition that cannot be triggered by the developer

Such asserts serve as ?mini-tests? directly in the code, and protect against future refactorings that could c

Use an explicit if-check with an exception to test for a condition that may be outside of the developer?s co

Class structure

¶

Consider putting all class constructors at the top of the class. This makes it much easier to see how the cla

Try to make as much of your class? API private as possible, do not expose members ?just in case?. This m

Remember to document all your public members! Documenting things is harder than it looks, and one way

If a class exposes a

List<X>

or

Vector2

property ? it is

NOT

an invitation to modify them at will! Consider such properties as read-only, unless the documentation explic

When a class becomes sufficiently big, consider adding

regions

inside it, which help with code navigation and collapsing (note the lack of space after

//

):

//#region Region description

...

//#endregion

If a class has a private member that needs to be exposed via a getter/setter, prefer the following code struc

class

MyClass

{

MyClass

();

...

int

_variable

;

...

/// Docs for both the getter and the setter.

int

get

variable

=>

_variable

;

set

variable

(

int

value

)

{

assert

(

value

>=

0

,

'variable must be non-negative:

$

value

'

);

_variable

=

value

;

}

}

This would gather all private variables in a single block near the top of the class, allowing one to quickly se

Documentation

¶

Use dartdocs

///

to explain the meaning/purpose of a class, method, or a variable.

Use regular comments

//

to explain implementation details of a particular code fragment. That is, these comments explain HOW som

Use markdown documentation in

doc/

folder to give the high-level overview of the functionality, and especially how it fits into the overall Flame fra

Dartdocs

¶

Check the

Flutter Documentation Guide

? it contains lots of great advice on writing good documentation.

However, disregard the advice about writing in a passive voice.

Class documentation should ideally start with the class name itself, and follow a pattern such as:

/// [MyClass] is ...

/// [MyClass] serves as ...

/// [MyClass] does the following ...

The reason for such convention is that often the documentation for a class becomes sufficiently long, and i

Method documentation should start with a verb in the present simple tense, with the method name as an im

/// Adds a new [child] into the container, and becomes the owner of that

/// child.

///

/// The child will be disposed of when this container is destroyed.

/// It is an error to try to add a child that already belongs to another

/// container.

void

addChild

(

T

child

)

{

...

}

Avoid stating the obvious (or at least only the obvious).

Constructor documentation may follow either the style of a method (i.e. ?Creates ??, ?Constructs ??), or of

Do not

use macros to copy the class documentation into the constructor?s dartdoc. Generally, the class document

Main docs

¶

This refers to the docs on the main Flame Documentation website, the one you?re reading right now. The r

When adding the documentation to the main docs site, consider also including an example directly into the

Check the

Documentation

manual about how to work with the docs site.

The following style rules generally apply when writing documentation:

Maximum line length of 100 characters;

Prefer to define external links at the bottom of the document, so as to make reading the plain text of the do

Separate headers from the preceding content with 2 blank lines ? this makes it easier to see the sections w

Lists should start at the beginning of the line and sublists should be indented with 2 spaces.

HardwareKeyboardDetector

¶

class

HardwareKeyboardDetector

extends

Component

The

HardwareKeyboardDetector

component allows you to directly listen to events from a hardware keyboard, bypassing the

Focus

widget in Flutter. It will not listen for events from any on-screen (software) keyboards.

This component can be placed anywhere in the component tree. For example, it can be attached to the roo

HardwareKeyboardDetector

components can coexist within the same game, and they all will receive the key events.

The component provides the

onKeyEvent

event handler, which can either be overridden or passed as a parameter in the constructor. This event han

The stream of key events will be normalized by Flutter, meaning that for every

KeyDownEvent

there will always be the corresponding

KeyUpEvent

, possibly with some

KeyRepeatEvent

s in the middle. Depending on the platform, some of these events may be ?synthesized?, i.e. created by th

HardwareKeyboard

for more details.

Similar normalization guarantee exists when this component is added to or removed from the component t

HardwareKeyboardDetector

was mounted, then artificial

KeyDownEvent

s will be fired; if the user was holding keys when this component was removed, then

KeyUpEvent

s will be synthesized.

Use

pauseKeyEvents

property to temporarily halt/resume the delivery of

onKeyEvent

s. The events will also stop being delivered when the component is removed from the component tree.

Constructors

¶

HardwareKeyboardDetector

(

{

void Function(KeyEvent)?

onKeyEvent

}

)

Properties

¶

physicalKeysPressed

?

List<PhysicalKeyboardKey>

The list of keys that are currently being pressed on the keyboard (or a keyboard-like device). The keys are

logicalKeysPressed

?

Set<LogicalKeyboardKey>

The set of logical keys that are currently being pressed on the keyboard. This set corresponds to the

physicalKeysPressed

list, and can be used to search for keys in a keyboard-layout-independent way.

isControlPressed

?

bool

True if the

Ctrl

key is currently being pressed.

isShiftPressed

?

bool

True if the

Shift

key is currently being pressed.

isAltPressed

?

bool

True if the

Alt

key is currently being pressed.

isNumLockOn

?

bool

True if

Num Lock

currently turned on.

isCapsLockOn

?

bool

True if

Caps Lock

currently turned on.

isScrollLockOn

?

bool

True if

Scroll Lock

currently turned on.

pauseKeyEvents

??

bool

When

true

, delivery of key events will be suspended.

When this property is set to true, the system generates KeyUp events for all keys currently being held, as i

false

, and the user was holding some keys at the time, the system will generate KeyDown events as if the user

Methods

¶

onKeyEvent

(

KeyEvent

event

)

Override this event handler if you want to get notified whenever any key on a keyboard is pressed, held, or

event

will be one of

KeyDownEvent

,

KeyRepeatEvent

, or

KeyUpEvent

, respectively.

Random functions

¶

These functions produce random results each time they run.

Internally, each function uses

YarnSpinner.random

random generator, which can be replaced with a custom generator if you need reproducible draws for debu

dice(n)

¶

Returns a random integer between

1

and

n

, inclusive. For example,

dice(6)

will return a random integer from 1 to 6, as if throwing a regular six-sided die.

The argument

n

must be numeric, and greater or equal than 1. If

n

is a non-integer, then it will be truncated to an integer value at runtime. Thus,

dice(3.5)

is equivalent to

dice(3)

.

<<

set

$roll

=

dice

(

6

)>>

<<

set

$coin_flip

=

if

(

dice

(

2

)

==

1

,

"

H

"

,

"

T

"

)>>

random()

¶

Returns a random floating-point between

0

and

1

.

This function can be used to implement events with a prescribed probability. For example:

<<

if

random

()

<

0.001

>>

// This happens only with 0.1% probability

You found it! The Holy Grail!

<<

endif

>>

random_range(a,

b)

¶

Returns a random integer between

a

and

b

inclusive.

Both arguments

a

and

b

must be numeric, and they will be truncated to integers upon evaluation. The value of

a

must be less than or equal to

b

, or otherwise a runtime exception will be thrown.

The purpose of this function is similar to

dice()

, but it can be used in situations where a custom range is desired.

<<

set

$coin_flip

=

bool

(

random_range

(

0

,

1

))>>

FlameIsolate

¶

The power of

integral_isolates

neatly packaged in

flame_isolate

for your Flame game.

If you?ve ever used the

compute

function before, you will feel right at home. This mixin allows you to run CPU-intensive code in an isolate.

To use it in your game you just need to add

flame_isolate

to your pubspec.yaml.

Usage

¶

Just add the mixin

FlameIsolate

to your component and start utilizing the power of an isolate as simple as running the

compute

function.

Example:

class

MyGame

extends

FlameGame

with

```
FlameIsolate
{
...
@override
void
update
(
double
dt
)
{
if
(
shouldRecalculate
)
{
isolate
(
recalculateWorld
,
worldData
).
then
(
updateWorld
);
```

}

...

}

...

}

Performance note

¶

Keep in mind that every component with

FlameIsolate

mixin that you create and add to your game will create a new isolate. This means you will probably want to

A simple example of this can be found in the example application for the FlameIsolate package.

Backpressure Strategies

¶

Backpressure strategies are a way to cope with the job queue when job items are produced more rapidly th

BackpressureStrategy

.

The ones currently supported are:

NoBackPressureStrategy

that basically does not handle back pressure. It uses a FIFO stack for storing a backlog of unhandled jobs.

ReplaceBackpressureStrategy

that has a job queue with size one, and discards the queue upon adding a new job.

DiscardNewBackPressureStrategy

that has a job queue with size one, and as long as the queue is populated a new job will not be added.

You can specify a backpressure strategy by overriding the

backpressureStrategy

field. This will create the isolate with the specified strategy when the component is mounted.

```
class
MyGame
extends
FlameGame
with
FlameIsolate
{
@override
BackpressureStrategy
get
backpressureStrategy
=>
ReplaceBackpressureStrategy
();
...
}
```

Other Modules

¶

jenny

This module lets you add interactive dialogue into your game. The module itself handles Yarn scripts and t

flame_jenny

in order to add it into a Flame game.

oxygen

Oxygen is a lightweight Entity Component System framework written in Dart, with a focus on performance a

Yarn Project

¶

A

YarnProject

is the central hub for all yarn scripts and the accompanying information. Generally, there would be a single

YarnProject

in a game, though it is also possible to make several yarn projects if their content is completely independer

The standard sequence of initializing a

YarnProject

is the following:

link user-defined functions;

link user-defined commands;

set the locale (if different from

en

);

parse a

.yarn

script containing declarations of global variables and characters;

parse all other

.yarn

scripts;

restore the variables from a save-game storage.

For example:

final

yarn

=

YarnProject

()

..

functions

.

addFunction0

(

'money'

,

player

.

getMoney

)

..

commands

.

addCommand1

(

'achievement'

,

player

.

earnAchievement

)

..

parse

(

readFile

(

'project.yarn'

))

..

parse

(

readFile

(

'chapter1.yarn'

))

..

parse

(

readFile

(

'chapter2.yarn'

));

Properties

¶

locale

String

The language used in this

YarnProject

(the default is

'en'

). Selecting a different language changes the builtin

plural()

function.

random

Random

The random number generator. This can be replaced with a different generator, if, for example, you need t

nodes

Map<String,

Node>

All

Node

s loaded into the project, keyed by their titles.

variables

VariableStorage

The container for all global variables used in this yarn project. There could be several reasons to access th

to change the value of a yarn variable from the game. This enables you to pass the information from the ga

$gold

, which you may want to update whenever the player?s amount of money changes within the game.

to store the values of all yarn variables during the save game, and to restore them when loading the game.

functions

FunctionStorage

The

container

for all user-defined functions linked into the project. The main reason to access this property is to register n

Note that all custom functions must be added to the

YarnProject

before they can be used in a dialogue script ? otherwise a compile error will occur when encountering an u

commands

CommandStorage

The

container

for all user-defined commands linked into the project. The main reason to access this container is to registe

All custom commands must be added before they can be used in the dialogue script.

characters

CharacterStorage

The

container

for all

Character

objects declared in your yarn scripts.

strictCharacterNames

bool

If

true

(default), the validity of character names will be strictly enforced. That is, all characters must be declared b

<<character>>

commands. If this property is set to false, then new

Character

objects will be created automatically as they are encountered in scripts.

trueValues

,

falseValues

Set<String>

The strings that can be recognized as

true

/

false

values respectively.

variables

VariableStorage

The

container

for all variables declared and manipulated in your yarn scripts. This is also used for maintaining the visit co

VariableStorage.variables

and later restore them again.

Methods

¶

parse

(

String

text

)

Parses and compiles the

text

of a yarn script. After this command, the nodes contained within the script will be runnable.

This method can be executed multiple times, and each time the new nodes will be added to the existing on

Game Widget

¶

class

GameWidget

<

T

extends

Game

>

extends

StatefulWidget

The

GameWidget

is a Flutter widget which is used to insert a

Game

instance into the Flutter widget tree.

The

GameWidget

is sufficiently feature-rich to run as the root of your Flutter application. Thus, the simplest way to use

GameWidget

is like this:

void

main

()

{

runApp

(

GameWidget

(

game:

MyGame

()),

);

}

At the same time,

GameWidget

is a regular Flutter widget, and can be inserted arbitrarily deep into the widget tree, including the possibility

GameWidget

s within a single app.

The layout behavior of this widget is that it will expand to fill all available space. Thus, when used as a root

In addition to hosting a

Game

instance, the

GameWidget

also provides some structural support, with the following features:

loadingBuilder

to display something while the game is loading;

errorBuilder

which will be shows if the game throws an error;

backgroundBuilder

to draw some decoration behind the game;

overlayBuilderMap

to draw one or more widgets on top of the game.

It should be noted that

GameWidget

does not clip the content of its canvas, which means the game can potentially draw outside of its boundarie

ClipRect

.

Constructors

¶

GameWidget

(

{

required

this.game

,

this.textDirection

,

this.loadingBuilder

,

this.errorBuilder

,

this.backgroundBuilder

,

this.overlayBuilderMap

,

this.initialActiveOverlays

,

this.focusNode

,

this.autofocus

=

true

,

this.mouseCursor

,

this.addRepaintBoundary

=

true

,

super.key

}

)

Renders the provided

game

instance.

GameWidget.controlled

(

{

required

this.gameFactory

,

this.textDirection

,

this.loadingBuilder

,

this.errorBuilder

,

this.backgroundBuilder

,

this.overlayBuilderMap

,

this.initialActiveOverlays

,

this.focusNode

,

this.autofocus

=

true

,

this.mouseCursor

,

this.addRepaintBoundary

=

true

,

super.key

}

)

A

GameWidget

which will create and own a

Game

instance, using the provided

gameFactory

.

This constructor can be useful when you want to put

GameWidget

into another widget, but would like to avoid the need to store the game?s instance yourself. For example:

class

MyWidget

extends

StatelessWidget

{

@override

Widget

build

(

BuildContext

context

)

{

return

Container

(

padding:

EdgeInsets

.

all

(

20

),

child:

GameWidget

.

controlled

(

gameFactory:

MyGame

.

new

,

),

);

}

}

Properties

¶

game

:

T?

The game instance which this widget will render, if it was provided with the default constructor. Otherwise,

GameWidget.controlled

constructor was used, this will always be

null

.

gameFactory

:

GameFactory<T>?

A function that creates a

Game

that this widget will render.

textDirection

:

TextDirection?

The text direction to be used in text elements in a game.

loadingBuilder

:

GameLoadingWidgetBuilder?

Builder to provide a widget which will be displayed while the game is loading. By default this is an empty

Container

.

errorBuilder

:

GameErrorWidgetBuilder?

If set, errors during the game loading will be caught and this widget will be shown. If not provided, errors ar

backgroundBuilder

:

WidgetBuilder?

Builder to provide a widget tree to be built between the game elements and the background color provided

Game.backgroundColor

.

overlayBuilderMap

:

Map<String, OverlayWidgetBuilder<T>>?

A collection of widgets that can be displayed over the game?s surface. These widgets can be turned on-an

Game.overlays

property.

```
void
main
()
{
runApp
(
GameWidget
(
game:
MyGame
(),
overlayBuilderMap:
{
'PauseMenu'
:
(
```

```
context
,
game
)
{
return
Container
(
color:
const
Color
(
0xFF000000
),
child:
Text
(
'A pause menu'
),
);
},
},
),
);
}
initialActiveOverlays
```

:

List<String>?

The list of overlays that will be shown when the game starts (but after it was loaded).

focusNode

:

FocusNode?

The

FocusNode

to control the games focus to receive event inputs. If omitted, defaults to an internally controlled focus node

autofocus

:

bool

Whether the

focusNode

requests focus once the game is mounted. Defaults to true.

mouseCursor

:

MouseCursor?

The shape of the mouse cursor when it is hovering over the game canvas. This property can be changed d

Game.mouseCursor

.

addRepaintBoundary

:

bool

Whether the game should assume the behavior of a

RepaintBoundary

, defaults to

true

.

Character

¶

class

Character

A

Character

represents a person who is speaking a particular line in a dialogue. This object is available as the

.character

property of a

DialogueLine

delivered to your

DialogueView

.

All characters must be declared with the help of the

<<character>>

command, unless

YarnProject

?s setting

strictCharacterNames

is set to false.

Constructors

¶

Character

(

this.name

,

{

List<String>?

aliases

}

)

Properties

¶

name

:

String

The canonical name of the character, which was the first argument in the

<<character>>

command.

aliases

:

List<String>

Additional names (IDs) that may be used for this character in yarn scripts.

data

?

Map<String, dynamic>

Additional information associated with this character. This may include their short bio, portrait, affiliation, co

DialogueView

s.

You can store any key-value pairs here that you want, or nothing at all.

See Also

¶

CharacterStorage

: the container where all Character objects within a YarnProject are cached.

# Getting Started

¶

## About Flame

¶

Flame is a modular Flutter game engine that provides a complete set of out-of-the-way solutions for games

It provides you with a simple yet effective game loop implementation, and the necessary functionalities that

We also provide stand-alone packages that extend the Flame functionality which can be found in the

Bridge Packages

section.

You can pick and choose whichever parts you want, as they are all independent and modular.

The engine and its ecosystem are constantly being improved by the community, so please feel free to reac

Give us a star if you want to help give the engine exposure and grow the community. :)

## Installation

¶

Add the

flame

package as a dependency in your

pubspec.yaml

by running the following command:

```
flutter pub add flame
```

The latest version can be found on

pub.dev

.

then run

```
flutter
pub
```

get

and you are ready to start using it!

Getting started

¶

There is a set of tutorials that you can follow to get started in the

tutorials folder

.

Simple examples for all features can be found in the

examples folder

.

You can also check out the

awesome flame repository

, it contains quite a lot of good tutorials and articles written by the community to get you started with Flame.

Outside of the scope of the engine

¶

Games sometimes require complex feature sets depending on what the game is all about. Some of these f

Multiplayer (netcode)

¶

Flame doesn?t bundle any network feature, which may be needed to write online multiplayer games.

If you are building a multiplayer game, here are some recommendations of packages/services:

Nakama

: Nakama is an open-source server designed to power modern games and apps.

Firebase

: Provides dozens of services that can be used to write simpler multiplayer experiences.

Supabase

: A cheaper alternative to Firebase, based on Postgres.

AlignComponent

¶

class

AlignComponent

extends

PositionComponent

AlignComponent

is a layout component that positions its child within itself using relative placement. It is similar to Flutter?s

Align

widget.

The component requires a single

child

, which will be the target of this component?s alignment. Of course, other children can be added to this com

child

will be aligned.

The

alignment

parameter describes where the child should be placed within the current component. For example, if the

alignment

is

Anchor.center

, then the child will be centered.

Normally, this component?s size will match the size of its parent. However, if you provide properties

widthFactor

or

heightFactor

, then the size of this component in that direction will be equal to the size of the child times the correspondi

heightFactor

to 1 then the width of this component will be equal to the width of the parent, but the height will match the h

AlignComponent

(

child:

TextComponent

(

'hello'

),

alignment:

Anchor

.

centerLeft

,

);

By default, the child?s anchor is set equal to the

alignment

value. This achieves traditional alignment behavior: for example, the center of the child will be placed at the

keepChildAnchor

to true. For example, if you set

alignment

to

topCenter

, and child?s anchor to

bottomCenter

, then the child will effectively be placed above the current component:

PlayerSprite

().

add

(

AlignComponent

(

child:

HealthBar

()..

anchor

=

Anchor

.

bottomCenter

,

alignment:

Anchor

.

topCenter

,

keepChildAnchor:

true

,

),

);

Constructors

¶

AlignComponent

(

{

PositionComponent?

child

,

Anchor

alignment

=

Anchor.topLeft

,

this.widthFactor

,

this.heightFactor

,

this.keepChildAnchor

=

false

}

)

Creates a component that keeps its

child

positioned according to the

alignment

within this component?s bounding box.

More precisely, the child will be placed at

alignment

relative position within the current component?s bounding box. The child?s anchor will also be set to the

alignment

, unless

keepChildAnchor

parameter is true.

Properties

¶

child

??

PositionComponent?

The component that will be positioned by this component. The

child

will be automatically mounted to the current component.

alignment

??

Anchor

How the

child

will be positioned within the current component.

Note: unlike Flutter?s

Alignment

, the top-left corner of the component has relative coordinates

(0,

0)

, while the bottom-right corner has coordinates

(1,

1)

.

widthFactor

:

double?

If

null

, then the component?s width will be equal to the width of the parent. Otherwise, the width will be equal to

heightFactor

:

double?

If

null

, then the component?s height will be equal to the height of the parent. Otherwise, the height will be equal

keepChildAnchor

:

bool

If

false

(default), then the child?s

anchor

will be kept equal to the

alignment

value. If

true

, then the

child

will be allowed to have its own

anchor

value independent from the parent.

Jenny

¶

The

jenny

library is a toolset for adding

dialogue

into a game. The dialogue may be quite complex, including user-controlled interactions, branching, dynami

jenny

library is an unofficial port of the

Yarn Spinner

library for Unity. The name of the library comes from

spinning jenny

, a kind of yarn-spinning machine.

Adding dialogue into any game generally consists of two stages:

Writing the text of the dialogue;

Interactively displaying it within the game.

With

jenny

, these two tasks are completely separate, allowing the creation of game content and development of the g

Writing dialogue

¶

In

jenny

, the dialogue is written in plain text and stored in

.yarn

files that are added to the game as assets. The

.yarn

file format is developed by the authors of

Yarn Spinner

, and is specifically designed for writing dialogue.

The simplest form of the yarn dialogue looks like a play:

title

:

Scene1_Gregory_and_Sampson

---

Sampson

:

Gregory, on my word, we'll not carry coals.

Gregory

:

No, for then we should be colliers.

Sampson

:

I mean, an we be in choler, we'll draw.

Gregory

:

Ay, while you live, draw your neck out of collar.

Sampson

:

I strike quickly being moved.

Gregory

:

But thou art not quickly moved to strike.

===

This simple exchange, when rendered within a game, will be shown as a sequence of phrases spoken in tu

DialogRunner

will allow you to control whether the dialogue proceeds automatically or requires ?clicking-through? by the

The

.yarn

format supports many more advanced features too, allowing the dialogue to proceed non-linearly, supporti

title

:

Slughorn_encounter

---

<<

if

visited

(

"

Horcrux_question

"

)>>

Slughorn

:

Sorry, Tom, I don't have time right now.

<<

stop

>>

<<

endif

>>

Slughorn

:

Oh hello, Tom, is there anything I can help you with?

Tom

:

Good

{

time_of_day

()}

, Professor.

->

I was curious about the 12 uses of the dragon blood.

Slughorn

:

Such an inquisitive mind! You can read about that in the "Moste

\

Potente Potions" in the Restricted Section of the library.

<<

give

restricted_library_pass

>>

Tom

:

Thank you, Professor, this is very munificent of you.

->

I wanted to ask... about Horcruxes

<<

if

$knows_about_horcruxes

>>

<<

jump

Horcrux_question

>>

->

I just wanted to say how much I always admire your lectures.

Slughorn

:

Thank you, Tom. I do enjoy flattery, even if it is well-deserved.

===

title

:

Horcrux_question

---

Slughorn

:

Where... did you hear that?

->

Tom

:

It was mentioned in an old book in the library...

Slughorn

:

I see that you have read more books from the Restricted Section

\

than is wise.

Slughorn

:

I'm sorry, Tom, I should have seen you'd be tempted...

<<

take

restricted_library_pass

>>

->

But Professor!..

Slughorn

:

This is for your good, Tom. Many of those books are dangerous!

Slughorn

:

Now off you go. And do your best to forget about what you

\

asked...

<<

stop

>>

->

Tom

:

I overheard it... And the word felt sharp and frigid, like it was the

\

embodiment of Dark Art

<<

if

luck

()

>=

80

>>

Slughorn

:

It is a very Dark Art indeed, it is not good for you to know

\

about it...

Tom

:

But if I don't know about this Dark Art, how can I defend myself

\

against it?

Slughorn

:

It is a Ritual, one of the darkest known to wizard-kind ...

...

<<

achievement

"The Darkest Secret"

>>

===

This fragment demonstrates many of the features of the

.yarn

language, including:

ability to divide the text into smaller chunks called

nodes

;

control the flow of the dialog via commands such as

<<if>>

or

<<jump>>

;

different dialogue path depending on player?s choices;

disable certain menu choices dynamically;

keep state information in variables;

user-defined functions (

time_of_day

,

luck

) and commands (

<<give>>

,

<<take>>

).

For more information, see the

Yarn Language

section.

Using the dialogue in a game

¶

By itself, the

jenny

library does not integrate with any game engine. However, it provides a runtime that can be used to build s

YarnProject

? the central repository of information, which knows about all your yarn scripts, variables, custom functions

DialogueRunner

? an executor that can run a specific dialogue node. This executor will send the dialogue lines into one or m

DialogueView

s.

DialogueView

? an abstract interface describing how the dialogue will be presented to the end user. Implementing this int

jenny

into a specific environment.

Components

¶

FlameBlocProvider

¶

FlameBlocProvider is a Component which creates and provides a bloc to its children.

The bloc will only live while this component is alive. It is used as a dependency injection (DI) widget so that

FlameBlocProvider should be used to create new blocs which will be made available to the rest of the subt

FlameBlocProvider

<

BlocA

,

BlocAState

>

(

create:

()

=>

BlocA

(),

children:

[...]

);

FlameBlocProvider can be used to provide an existing bloc to a new portion of the Component tree.

FlameBlocProvider

<

BlocA

,

BlocAState

>

.

value

(

value:

blocA

,

children:

[...],

);

FlameMultiBlocProvider

¶

Similar to FlameBlocProvider, but provides multiples blocs down to the component tree

FlameMultiBlocProvider

(

providers:

[

FlameBlocProvider

<

BlocA

,

BlocAState

>

(

create:

()

=>

BlocA

(),

),

FlameBlocProvider

<

BlocB

,

BlocBState

>

.

value

(

create:

()

=>

BlocB

(),

),

],

children:

[...],

)

FlameBlocListener

¶

FlameBlocListener is Component which can listen to changes in a Bloc state. It invokes the

onNewState

in response to state changes in the bloc. For fine-grained control over when the

onNewState

function is called an optional

listenWhen

can be provided.

listenWhen

takes the previous bloc state and current bloc state and returns a boolean. If

listenWhen

returns true,

onNewState

will be called with

state

. If

listenWhen

returns false,

onNewState

will not be called with

state

.

alternatively you can use

FlameBlocListenable

mixin to listen state changes on Component.

FlameBlocListener

<

GameStatsBloc

,

GameStatsState

>

(

listenWhen:

(

previousState

,

newState

)

{

// return true/false to determine whether or not

// to call listener with state

},

onNewState:

(

state

)

{

// do stuff here based on state

},

)

FlameBlocListenable

¶

FlameBlocListenable is an alternative to FlameBlocListener to listen state changes.

class

ComponentA

extends

Component

with

FlameBlocListenable

<

BlocA

,

BlocAState

>

{

@override

bool

listenWhen

(

PlayerState

previousState

,

PlayerState

newState

)

{

// return true/false to determine whether or not

// to call listener with state

}

@override

void

onNewState

(

PlayerState

state

)

{

super

.

onNewState

(

state

);

// do stuff here based on state

}

}

FlameBlocReader

¶

FlameBlocReader is mixin that allows you to read the current state of bloc on Component. It is Useful for c

class

InventoryReader

extends

Component

with

```
FlameBlocReader
<
InventoryCubit
,
InventoryState
>
{}
/// inside game
final
component
=
InventoryReader
();
// reading current state
var
state
=
component
.
bloc
```

Camera component

¶

Camera-as-a-component is the new way of structuring a game, an approach that allows more flexibility in p

In order to understand how this approach works, imagine that your game world is an entity that exists

somewhere

independently from your application. Imagine that your game is merely a window through which you can lo

With this mindset, we can now understand how camera-as-a-component works.

First, there is the

World

class, which contains all components that are inside your game world. The

World

component can be mounted anywhere, for example at the root of your game class, like the built-in

World

is.

Then, a

CameraComponent

class that ?looks at? the

World

. The

CameraComponent

has a

Viewport

and a

Viewfinder

inside, allowing both the flexibility of rendering the world at any place on the screen, and also control the vi

CameraComponent

also contains a

backdrop

component which is statically rendered below the world.

World

¶

This component should be used to host all other components that comprise your game world. The main pr

World

class is that it does not render through traditional means ? instead it is rendered by one or more

CameraComponent

s to ?look at? the world. In the

FlameGame

class there is one

World

called

world

which is added by default and paired together with the default

CameraComponent

called

camera

.

A game can have multiple

World

instances that can be rendered either at the same time, or at different times. For example, if you have two

Just like with most

Component

s, children can be added to

World

by using the

children

argument in its constructor, or by using the

add

or

addAll

methods.

For many games you want to extend the world and create your logic in there, such a game structure could

```
void
main
()
{
runApp
(
GameWidget
(
FlameGame
(
world:
MyWorld
())));
}
class
MyWorld
extends
```

World

{

@override

Future

<

void

>

onLoad

()

async

{

// Load all the assets that are needed in this world

// and add components etc.

}

}

CameraComponent

¶

This is a component through which a

World

is rendered. It requires a reference to a

World

instance during construction; however later the target world can be replaced with another one. Multiple can

There is a default

CameraComponent

called

camera

on the

FlameGame

class which is paired together with the default

world

, so you don?t need to create or add your own

CameraComponent

if your game doesn?t need to.

A

CameraComponent

has two other components inside: a

Viewport

and a

Viewfinder

. Unlike the

World

object, the camera owns the viewport and the viewfinder, which means those components are children of t

There is also a static property

CameraComponent.currentCamera

which is not null only during the rendering stage, and it returns the camera object that currently performs re

The

FlameGame

class has a

camera

field in its constructor, so you can set what type of default camera that you want like this for example:

void

main

()

{

runApp

(

GameWidget

(

FlameGame

(

camera:

CameraComponent

.

withFixedResolution

(

width:

800

,

height:

600

),

),

),

);

}

CameraComponent.withFixedResolution()

¶

This factory constructor will let you pretend that the user?s device has a fixed resolution of your choice. Fo

final

camera

=

CameraComponent

.

withFixedResolution

(

world:

myWorldComponent

,

width:

800

,

height:

600

,

);

This will create a camera with a viewport centered in the middle of the screen, taking as much space as po

A ?fixed resolution? is very simple to work with, but it will underutilize the user?s available screen space, u

Viewport

¶

The

Viewport

is a window through which the

World

is seen. That window has a certain size, shape, and position on the screen. There are multiple kinds of vie

The

Viewport

is a component, which means you can add other components to it. These children components will be affe

Adding elements to the viewport is a convenient way to implement ?HUD? components.

The following viewports are available:

MaxViewport

(default) ? this viewport expands to the maximum size allowed by the game, i.e. it will be equal to the size

FixedResolutionViewport

? keeps the resolution and aspect ratio fixed, with black bars on the sides if it doesn?t match the aspect ra

FixedSizeViewport

? a simple rectangular viewport with predefined size.

FixedAspectRatioViewport

? a rectangular viewport which expands to fit into the game canvas, but preserving its aspect ratio.

CircularViewport

? a viewport in the shape of a circle, fixed size.

If you add children to the

Viewport

they will appear as static HUDs in front of the world.

Viewfinder

¶

This part of the camera is responsible for knowing which location in the underlying game world we are curr

Viewfinder

also controls the zoom level, and the rotation angle of the view.

The

anchor

property of the viewfinder allows you to designate which point inside the viewport serves as a ?logical cent

anchor

.

If you add children to the

Viewfinder

they will appear will appear in front of the world, but behind the viewport and with the same transformations

You can also add behavioral components as children to the viewfinder, for example

effects

or other controllers. If you for example would add a

ScaleEffect

you would be able to achieve a smooth zoom in your game.

Backdrop

¶

To add static components behind the world you can add them to the

backdrop

component, or replace the

backdrop

component. This is for example useful if you want to have a static

ParallaxComponent

beneath a world that you can move around it.

Example:

camera

.

backdrop

.

add

(

MyStaticBackground

());

or

camera

.

backdrop

=

MyStaticBackground

();

Camera controls

¶

There are several ways to modify camera?s settings at runtime:

Do it manually. You can always override the

CameraComponent.update()

method (or the same method on the viewfinder or viewport) and within it change the viewfinder?s position o

Apply effects and/or behaviors to the camera?s

Viewfinder

or

Viewport

. The effects and behaviors are special kinds of components whose purpose is to modify over time some p

Use special camera functions such as

follow()

,

moveBy()

and

moveTo()

. Under the hood, this approach uses the same effects/behaviors as in (2).

Camera has several methods for controlling its behavior:

Camera.follow()

will force the camera to follow the provided target. Optionally you can limit the maximum speed of moveme

Camera.stop()

will undo the effect of the previous call and stop the camera at its current position.

Camera.moveBy()

can be used to move the camera by the specified offset. If the camera was already following another comp

Camera.moveTo()

can be used to move the camera to the designated point on the world map. If the camera was already follo

Camera.setBounds()

allows you to add limits to where the camera is allowed to go. These limits are in the form of a

Shape

, which is commonly a rectangle, but can also be any other shape.

visibleWorldRect

¶

The camera exposes property

visibleWorldRect

, which is a rect that describes the world?s region which is currently visible through the camera. This regio

The

visibleWorldRect

is a cached property, and it updates automatically whenever the camera moves or the viewport changes its

Check if a component is visible from the camera point of view

¶

The

CameraComponent

has a method called

canSee

which can be used to check if a component is visible from the camera point of view. This is useful for exam

if

(

!

camera

.

canSee

(

component

))

{

component

.

removeFromParent

();

// Cull the component

}

<<stop>>

¶

The

<<stop>>

command: immediately stops evaluating the current node, as if you jumped to its end. This command takes

Normally, the effect of this command is that it stops the dialogue. However, if you?re only visiting the curre

<<stop>>

will only exit the current node, and the execution flow will return to the parent. Thus, the

<<stop>>

command is similar to

return;

in many programming languages.

<<

stop

>>

flame_bloc

¶

flame_bloc

is a bridge library for using

Bloc

in your Flame game.

flame_bloc

offers a simple and natural (as in similar to flutter_bloc) way to use blocs and cubits inside a FlameGame. I

To use it in your game you just need to add

flame_bloc

to your pubspec.yaml, as can be seen in the

Flame Bloc example

and in the pub.dev

installation instructions

.

How to use

¶

Lets assume we have a bloc that handles player inventory, first we need to make it available to our compon

We can do that by using

FlameBlocProvider

component:

class

MyGame

extends

FlameGame

{

```
@override
Future
<
void
>
onLoad
()
async
{
await
add
(
FlameBlocProvider
<
PlayerInventoryBloc
,
PlayerInventoryState
>
(
create:
()
=>
PlayerInventoryBloc
(),
children:
[
```

Player

(),

// ...

],

),

);

}

}

With the above changes, the

Player

component will now have access to our bloc.

If more than one bloc needs to be provided,

FlameMultiBlocProvider

can be used in a similar fashion:

class

MyGame

extends

FlameGame

{

@override

Future

<

void

>

onLoad

()

```
async
{
await
add
(
FlameMultiBlocProvider
(
providers:
[
FlameBlocProvider
<
PlayerInventoryBloc
,
PlayerInventoryState
>
(
create:
()
=>
PlayerInventoryBloc
(),
),
FlameBlocProvider
<
PlayerStatsBloc
,
```

PlayerStatsState

>

(

create:

()

=>

PlayerStatsBloc

(),

),

],

children:

[

Player

(),

// ...

],

),

);

}

}

Listening to states changes at the component level can be done with two approaches:

By using

FlameBlocListener

component:

class

Player

extends

PositionComponent

{

@override

Future

<

void

>

onLoad

()

async

{

await

add

(

FlameBlocListener

<

PlayerInventoryBloc

,

PlayerInventoryState

>

(

listener:

(

state

)

```
{
updateGear
(
state
);
},
),
);
}
}
```

Or by using

```
FlameBlocListenable
```

mixin:

```
class
Player
extends
PositionComponent
with
FlameBlocListenable
<
PlayerInventoryBloc
,
PlayerInventoryState
>
{
@override
```

void

onNewState

(

state

)

{

updateGear

(

state

);

}

}

If all your component need is to simply access a bloc, the

FlameBlocReader

mixin can be applied to a component:

class

Player

extends

PositionComponent

with

FlameBlocReader

<

PlayerStatsBloc

,

PlayerStatsState

>

{

void

takeHit

()

{

bloc

.

add

(

const

PlayerDamaged

());

}

}

Note that one limitation of the mixin is that it can access only a single bloc.

Full Example

¶

You can check an example

here

.

DialogueOption

¶

The

DialogueOption

class represents a single

Option

line in the

.yarn

script. Multiple options will be grouped into

DialogueChoice

objects.

Properties

¶

text

String

The computed text of the option, after evaluating the inline expressions, stripping the markup, and process

tags

List<String>

The list of hashtags for this option. If there are no hashtags, the list will be empty. Each entry in the list will

#

.

attributes

List<MarkupAttribute>

The list of markup spans associated with the option. Each [MarkupAttribute] corresponds to a single span w

text

, delineated with markup tags.

isAvailable

bool

The result of evaluating the

conditional

of this option. If the option has no conditional, this will return

true

.

isDisabled

bool

Same as

!isAvailable

.

DialogueRunner

¶

class

DialogueRunner

The

DialogueRunner

is the engine that executes Jenny?s dialogue at runtime.

If you imagine

YarnProject

as a ?program?, consisting of multiple

Node

s as ?functions?, then

DialogueRunner

is a virtual machine, capable of executing a single ?function? in that ?program?.

A single

DialogueRunner

may only execute one dialogue node at a time. It is an error to try to run another node before the first one c

DialogueRunner

s for the same

YarnProject

, and then they would be able to execute multiple dialogues simultaneously (for example, in a crowded roo

The job of a

DialogueRunner

is to fetch the dialogue lines in the correct order and at the appropriate pace, to execute the logic in dialogu

DialogueChoice

s. The output of a

DialogueRunner

, therefore, is a stream of dialogue statements that need to be presented to the player. Such presentation,

DialogueView

s.

Constructors

¶

DialogueRunner

(

{

required

YarnProject

yarnProject

,

required

List<DialogueView>

dialogueViews

}

)

Creates a

DialogueRunner

for executing the

yarnProject

. The dialogue will be delivered to all the provided

dialogueViews

. Each of these dialogue views may only be assigned to a single

DialogueRunner

at a time.

Properties

¶

project

:

YarnProject

The

YarnProject

that this dialogue runner is executing.

Methods

¶

startDialogue

(

String

nodeName

)

?

Future<void>

Starts the dialogue with the node

nodeName

, and returns a future that completes once the dialogue finishes running. While this future is pending, the

DialogueRunner

cannot start any other dialogue.

sendSignal

(

dynamic

signal

)

Delivers the given

signal

to all dialogue views, in the form of a

DialogueView

method

onLineSignal(line,

signal)

. This can be used, for example, as a means of communication between the dialogue views.

The

signal

object here is completely arbitrary, and it is up to the implementations to decide which signals to send and

stopLine

(

)

Requests (via

onLineStop()

) that the presentation of the current line be finished as quickly as possible. The dialogue will then proceed

Execution model

¶

The

DialogueRunner

uses futures as a main mechanism for controlling the timing of the dialogue progression. For each event, th

DialogueView

s, and each of those callbacks may return a future. The dialogue runner then awaits on all of these futures

For a simple

.yarn

script like this

title

:

main

---

Hello

->

Hi

->

Go away

<<

jump

Away

>>

===

title

:

Away

---

<<

OhNo

>>

===

the sequence of emitted events will be as follows (assuming the second option is selected):

onDialogueStart()

onNodeStart(Node("main"))

onLineStart(Line("Hello"))

onLineFinish(Line("Hello"))

onChoiceStart(Choice(["Hi",

"Go

away"]))

onChoiceFinish(Option("Go

away"))

onNodeFinish(Node("main"))

onNodeStart(Node("Away"))

onCommand(Command("OhNo"))

onNodeFinish(Node("Away"))

onDialogueFinish()

Note

Keep in mind that if a

DialogueError

is thrown while running the dialogue, then the dialogue will terminate immediately and none of the

*Finish

callbacks will run.

<<character>>

¶

The

<<character>>

command declares a character with the given name, and one or more aliases that can be used in the scrip

The command has several purposes:

it protects you from accidentally misspelling a character?s name in your script;

it allows a character to have

full name

, which doesn?t have to be an ID;

it allows declaring multiple aliases for the same character, which can be used in different nodes (an alias m

you can associate additional data with each character, which will then be available at runtime.

The format of this command is the following:

<<

character

"

FULL NAME

"

alias1

alias2

...>>

The

full name

here is optional: if given, it will be considered

the

name of the character. However, if the name is omitted, then the first alias will be considered the true chara

alias

must be a valid ID, and at least one alias must be provided. For example:

// A well-mannered seven-year-old girl, who nevertheless always gets into

// all kinds of zany adventures.

<<

character

Alice

>>

// A magical cat known for his ability to grin majestically, and partially

// vanish. He is mad (by his own admission).

<<

character

"

Cheshire Cat

"

Cat

Cheshire

>>

// A foul-tempered Queen, who is also a playing card. Described as

// "a blind fury", her favorite saying is "Off with their heads!".

// Not to be confused with Red Queen.

<<

character

"

Queen of Hearts

"

Queen

QoH

QH

>>

After a character is declared, any of its aliases can be used in the script: they will all refer to the same

Character

object. At the same time, using a character without declaring it first is not allowed (unless a special flag in

YarnProject

is set to allow this).

title

:

Alice_and_the_Cat

---

Alice

:

But I don't want to go among mad people.

Cat

:

Oh, you can't help that, we're all mad here. I'm mad. You're mad.

Alice

:

How do you know I'm mad?

Cat

:

You must be, or you wouldn't have come here.

Alice

:

And how do you know that you're mad?

Cat

:

To begin with, a dog's not mad. You grant that?

Alice

:

I suppose so.

Cat

:

Well then, you see a dog growls when it's angry, and wags its tail

\

when it's pleased.

Cat

:

Now,

[i]

I

[/i]

growl when I'm pleased, and wag my tail when I'm angry.

\

Therefore, I'm mad.

Alice

:

[i]

I

[/i]

call it purring, not growling.

Cat

:

Call it what you like.

===

Contribution Guidelines

¶

Note:

If these contribution guidelines are not followed your issue or PR might be closed, so please read these ins

Contribution types

¶

Bug Reports

¶

If you find a bug, please first report it using

Github issues

.

First check if there is not already an issue for it; duplicated issues will be closed.

Bug Fix

¶

If you?d like to submit a fix for a bug, please read the

How To

for how to send a Pull Request.

Indicate on the open issue that you are working on fixing the bug and the issue will be assigned to you.

Write

Fixes

#xxxx

in your PR text, where xxxx is the issue number (if there is one).

Include a test that isolates the bug and verifies that it was fixed.

New Features

¶

If you?d like to add a feature to the library that doesn?t already exist, feel free to describe the feature in a n

GitHub issue

.

You can also join us on

Discord

to discuss some initials thoughts.

If you?d like to implement the new feature, please wait for feedback from the project maintainers before sp

Implement the code for the new feature and please read the

How To

.

Documentation & Miscellaneous

¶

If you have suggestions for improvements to the documentation, tutorial or examples (or something else), v

As always first file a

Github issue

.

Implement the changes to the documentation, please read the

How To

.

How To Contribute

¶

Requirements

¶

For a contribution to be accepted:

Follow the

Style Guide

when writing the code;

Format the code using

dart

format

.

;

Lint the code with

melos

run

analyze

;

Check that all tests pass:

melos

run

test

;

Documentation should always be updated or added (if applicable);

Examples should always be updated or added (if applicable);

Tests should always be updated or added (if applicable) ? check the

Test writing guide

for more details;

The PR title should start with a

conventional commit

prefix (

feat:

,

fix:

etc).

If the contribution doesn?t meet these criteria, a maintainer will discuss it with you on the issue or PR. You

Open an issue and fork the repository

¶

If it is a bigger change or a new feature, first of all

file a bug or feature report

, so that we can discuss what direction to follow.

Fork the project

on GitHub.

Clone the forked repository to your local development machine (e.g.

git

clone

git@github.com:<YOUR_GITHUB_USER>/flame.git

).

Environment Setup

¶

Flame is setup to run with the most recent

stable

version of Flutter, so make sure your version matches that:

flutter

channel

stable

Also, Flame uses

Melos

to manage the project and dependencies.

To install Melos, run the following command from your terminal:

flutter

pub

global

activate

melos

Next, at the root of your locally cloned repository bootstrap the projects dependencies:

melos

bootstrap

The bootstrap command locally links all dependencies within the project without having to provide manual

dependency_overrides

. This allows all plugins, examples and tests to build from the local clone project. You should only need to r

You do not need to run

flutter

pub

get

once bootstrap has been completed.

CSpell

¶

If you want to run the spellchecker locally, you will have to install

cspell

; you can do so using npm or yarn:

npm

install

-g

cspell

Then you can run it with the following arguments:

cspell

--no-progress

-c

.github/cspell.json

"**/*.{md,dart}"

Markdown Lint

¶

If you want to lint the markdown files you have to install

markdownlint-cli

and once that is installed you can run

melos

markdown-check

to check if the markdown follows the rules. Some markdown linting errors can be automatically fixed with

melos

markdown-fix

.

Note that, sadly, a particularly laborious rule, MD013,

does not provide an auto-fix option

. However, you can use other tools to circumvent this. For example, the extension

Rewrap

for VSCode, when

configured with

rewrap.wrappingColumn=100

, will do the trick for you.

Performing changes

¶

Create a new local branch from

main

(e.g.

git

checkout

-b

my-new-feature

)

Make your changes (try to split them up with one PR per feature/fix).

When committing your changes, make sure that each commit message is clear (e.g.

git

commit

-m

'Take

in

an

optional

Camera

as

a

parameter

to

FlameGame'

).

Push your new branch to your own fork into the same remote branch (e.g.

git

push

origin

my-username.my-new-feature

, replace

origin

if you use another remote.)

Breaking changes

¶

When doing breaking changes a deprecation tag should be added when possible and contain a message t

Example (if the current version is v1.3.0):

@Deprecated

(

'Will be removed in v1.5.0, use nonDeprecatedFeature() instead'

)

void

deprecatedFeature

()

{}

Open a pull request

¶

Go to the

pull request page of Flame

and in the top of the page it will ask you if you want to open a pull request from your newly created branch.

The title of the pull request should start with a

conventional commit

type.

Allowed types are:

fix:

? patches a bug and is not a new feature;

feat:

? introduces a new feature;

docs:

? updates or adds documentation or examples;

test:

? updates or adds tests;

refactor:

? refactors code but doesn?t introduce any changes or additions to the public API;

perf:

? code change that improves performance;

build:

? code change that affects the build system or external dependencies;

ci:

? changes to the CI configuration files and scripts;

chore:

? other changes that don?t modify source or test files;

revert:

? reverts a previous commit.

If you introduce a

breaking change

the conventional commit type MUST end with an exclamation mark (e.g.

feat!:

Remove

the

position

argument

from

PositionComponent

).

Examples of PR titles:

feat: Component.childrenFactory can be used to set up a global ComponentSet factory

fix: Avoid infinite loop in

FlameGame

docs: Add a

JoystickComponent

example

docs: Improve the Mandarin README

test: Add infinity test for

MoveEffect.to

refactor: Optimize the structure of the game loop

Maintainers

¶

These instructions are for the maintainers of Flame.

Merging a pull request

¶

When merging a pull request, make sure that the title of the merge commit has the correct conventional co

All the default text should be removed from the commit message and the PR description and the instructio

Creating a release

¶

There are a few things to think about when doing a release:

Search through the codebase for

@Deprecated

methods/fields and remove the ones that are marked for removal in the version that you are intending to re

Create a PR containing the changes for removing the deprecated entities.

Run

melos

version

-V

<package1>:<version>

-V

<package2>:<version>

for Melos to generate

CHANGELOG.md

files.

Go through the PRs with breaking changes and add migration documentation to the changelog. There shou

Run

melos

publish

to make sure that there aren?t any problems with any of the packages and make sure that all the versions

Once you are satisfied with the result of the dry run, run

melos

publish

--no-dry-run

Create a PR containing the updated changelog and

pubspec.yaml

files.

flame_tiled

¶

flame_tiled

is the bridge package that connects the flame game engine to

Tiled

maps by parsing TMX (XML) files and accessing the tiles, objects, and everything in there.

To use this,

Create your own map by using

Tiled

.

Create a

TiledComponent

and add it to the component tree as follows:

final

component

=

await

TiledComponent

.

load

(

'my_map.tmx'

,

Vector2

.

all

(

32

),

);

add

(

component

);

TiledComponent

¶

Tiled is a free and open source, full-featured level and map editor for your platformer or RPG game. Curren

tiled.dart

to parse map files and render visible layers using the performant

SpriteBatch

for each layer.

Supported map types include: Orthogonal, Isometric, Hexagonal, and Staggered.

Orthogonal

Hexagonal

Isomorphic

An example of how to use the API can be found

here

.

TileStack

¶

Once a

TiledComponent

is loaded, you can select any column of (x,y) tiles in a

tileStack

to then add animation. Removing the stack will not remove the tiles from the map.

Note

: This currently only supports position based effects.

void

onLoad

()

{

final

stack

=

map

.

tileMap

.

tileStack

(

4

,

0

,

named:

{

'floor_under'

});

stack

.

add

(

SequenceEffect

(

[

MoveEffect

.

by

(

Vector2

(

5

,

0

),

NoiseEffectController

(

duration:

1

,

frequency:

20

),

),

MoveEffect

.

by

(

Vector2

.

zero

(),

LinearEffectController

(

2

)),

],

repeatCount:

3

,

)

..

onComplete

=

()

=>

stack

.

removeFromParent

(),

);

map

.

add

(

stack

);

}

TileAtlas

¶

When a tilemap has multiple images (from multiple tilesets)

TiledComponent

uses a

TileAtlas

to pack all those image into a single big image (a.k.a atlas). This helps in rendering the whole map in a sing

TiledComponent

limits the atlas to

4096x4096

for web and

8192x8192

for all other platforms.

These limits should work well for most cases. But in case you are sure that your target platform can suppor

TiledComponent

you can do so by passing in the

atlasMaxX

and

atlasMaxX

values to

TiledComponent.load

.

NOTE: This is not recommended as such huge sizes might not work with all hardware. Instead consider re

final

component

=

await

TiledComponent

.

load

(

'my_map.tmx'

,

Vector2

.

all

(

32

),

atlasMaxX:

9216

,

atlasMaxY:

9216

,

);

add

(

component

);

Limitations

¶

Flip

¶

Tiled

has a feature that allows you to flip a tile horizontally or vertically, or even rotate it.

flame_tiled

supports this but if you are using a large texture and have flipped tiles there will be a drop in performance.

ignoreFlip

to false in the constructor.

Note

: A large texture in this context means one with multiple tilesets (or a huge tileset) where the sum of their di

final

component

=

await

TiledComponent

.

load

(

'my_map.tmx'

,

Vector2

.

all

(

32

),

ignoreFlip:

true

,

);

Clearing images cache

¶

If you have called

Flame.images.clearCache()

you also need to call

TiledAtlas.clearCache()

to remove disposed images from the tiled cache. It might be useful if your next game map have completely

Troubleshooting

¶

My game shows ?lines? and artifacts between the map tiles

¶

This is caused by the imprecision found in float-pointing numbers in computer science.

Check this

Article

to learn more about the issue and how it can be solved.

flame_isolate

¶

Overview

Usage

Performance note

Backpressure Strategies

Particles

¶

Flame offers a basic, yet robust and extendable particle system. The core concept of this system is the

Particle

class, which is very similar in its behavior to the

ParticleSystemComponent

.

The most basic usage of a

Particle

with

FlameGame

would look as following:

```
import

'package:flame/components.dart'

;

// ...

game

.

add

(

// Wrapping a Particle with ParticleSystemComponent

// which maps Component lifecycle hooks to Particle ones

// and embeds a trigger for removing the component.

ParticleSystemComponent

(

particle:
```

CircleParticle

(),

),

);

When using

Particle

with a custom

Game

implementation, please ensure that both the

update

and

render

methods are called during each game loop tick.

Main approaches to implement desired particle effects:

Composition of existing behaviors.

Use behavior chaining (just a syntactic sugar of the first one).

Using

ComputedParticle

.

Composition works in a similar fashion to those of Flutter widgets by defining the effect from top to bottom.

Random

rnd

=

Random

();

Vector2

randomVector2

()

=>

(

Vector2

.

random

(

rnd

)

-

Vector2

.

random

(

rnd

))

*

200

;

// Composition.

//

// Defining a particle effect as a set of nested behaviors from top to bottom,

// one within another:

//

// ParticleSystemComponent

```
//   > ComposedParticle
//     > AcceleratedParticle
//       > CircleParticle
game
.
add
(
ParticleSystemComponent
(
particle:
Particle
.
generate
(
count:
10
,
generator:
(
i
)
=>
AcceleratedParticle
(
acceleration:
randomVector2
```

```
(),
child:
CircleParticle
(
paint:
Paint
()..
color
=
Colors
.
red
,
),
),
),
),
);
// Chaining.
//
// Expresses the same behavior as above, but with a more fluent API.
// Only Particles with SingleChildParticle mixin can be used as chainable behaviors.
game
.
add
(
```

ParticleSystemComponent

(

particle:

Particle

.

generate

(

count:

10

,

generator:

(

i

)

=>

pt

.

CircleParticle

(

paint:

Paint

()..

color

=

Colors

.

```
red
)
)
)
);
// Computed Particle.
//
// All the behaviors are defined explicitly. Offers greater flexibility
// compared to built-in behaviors.
game
.
add
(
ParticleSystemComponent
(
particle:
Particle
.
generate
(
count:
10
,
generator:
(
i
```

)

{

Vector2

position

=

Vector2

.

zero

();

Vector2

speed

=

Vector2

.

zero

();

final

acceleration

=

randomVector2

();

final

paint

=

Paint

()..

color

=

Colors

.

red

;

return

ComputedParticle

(

renderer:

(

canvas

,

_

)

{

speed

+=

acceleration

;

position

+=

speed

;

canvas

.

drawCircle

(

Offset

(

position

.

x

,

position

.

y

),

1

,

paint

);

}

);

}

)

)

);

You can find more examples of how to use different built-in particles in various combinations

here

.

Lifecycle

¶

A behavior common to all

Particle

s is that all of them accept a

lifespan

argument. This value is used to make the

ParticleSystemComponent

remove itself once its internal

Particle

has reached the end of its life. Time within the

Particle

itself is tracked using the Flame

Timer

class. It can be configured with a

double

, represented in seconds (with microsecond precision) by passing it into the corresponding

Particle

constructor.

Particle

(

lifespan:

.

2

);

// will live for 200ms.

Particle

(

lifespan:

4

);

// will live for 4s.

It is also possible to reset a

Particle

?s lifespan by using the

setLifespan

method, which also accepts a

double

of seconds.

final

particle

=

Particle

(

lifespan:

2

);

// ... after some time.

particle

.

setLifespan

(

2

)

// will live for another 2s.

During its lifetime, a

Particle

tracks the time it was alive and exposes it through the

progress

getter, which returns a value between

0.0

and

1.0

. This value can be used in a similar fashion as the

value

property of the

AnimationController

class in Flutter.

final

particle

=

Particle

(

lifespan:

2.0

);

game

.

add

(

ParticleSystemComponent

(

particle:

particle

));

// Will print values from 0 to 1 with step of .1: 0, 0.1, 0.2 ... 0.9, 1.0.

Timer

.

periodic

(

duration

*

.

1

,

()

=>

print

(

particle

.

progress

));

The

lifespan

is passed down to all the descendants of a given

Particle

, if it supports any of the nesting behaviors.

Built-in particles

¶

Flame ships with a few built-in

Particle

behaviors:

The

TranslatedParticle

translates its

child

by given

Vector2

The

MovingParticle

moves its

child

between two predefined

Vector2

, supports

Curve

The

AcceleratedParticle

allows basic physics based effects, like gravitation or speed dampening

The

CircleParticle

renders circles of all shapes and sizes

The

SpriteParticle

renders Flame

Sprite

within a

Particle

effect

The

ImageParticle

renders

dart:ui

Image

within a

Particle

effect

The

ComponentParticle

renders Flame

Component

within a

Particle

effect

The

FlareParticle

renders Flare animation within a

Particle

effect

More examples of how to use these behaviors together are available

here

. All the implementations are available in the

particles

folder on the Flame repository.

TranslatedParticle

¶

Simply translates the underlying

Particle

to a specified

Vector2

within the rendering

Canvas

. Does not change or alter its position, consider using

MovingParticle

or

AcceleratedParticle

where change of position is required. Same effect could be achieved by translating the

Canvas

layer.

game

.

add

(

ParticleSystemComponent

(

particle:

TranslatedParticle

(

// Will translate the child Particle effect to the center of game canvas.

offset:

game

.

size

/

2

,

child:

Particle

(),

),

),

);

MovingParticle

¶

Moves the child

Particle

between the

from

and

to

Vector2

s during its lifespan. Supports

Curve

via

CurvedParticle

.

game

.

add

(

ParticleSystemComponent

(

particle:

MovingParticle

(

// Will move from corner to corner of the game canvas.

from:

Vector2

.

zero

(),

to:

game

.

size

,

child:

CircleParticle

(

radius:

2.0

,

paint:

Paint

()..

color

=

Colors

.

red

,

),

),

),

);

AcceleratedParticle

¶

A basic physics particle which allows you to specify its initial

position

,

speed

and

acceleration

and lets the

update

cycle do the rest. All three are specified as

Vector2

s, which you can think of as vectors. It works especially well for physics-based ?bursts?, but it is not limited

Vector2

value is

logical px/s

. So a speed of

Vector2(0,

100)

will move a child

Particle

by 100 logical pixels of the device every second of game time.

final

rnd

=

Random

();

Vector2

randomVector2

()

=>

(

Vector2

.

random

(

rnd

)

-

Vector2

.

random

(

rnd

))

*

100

;

game

.

add

(

ParticleSystemComponent

(

particle:

AcceleratedParticle

(

// Will fire off in the center of game canvas

position:

game

.

canvasSize

/

2

,

// With random initial speed of Vector2(-100..100, 0..-100)

speed:

Vector2

(

rnd

.

nextDouble

()

*

200

-

100

,

-

rnd

.

nextDouble

()

*

100

),

// Accelerating downwards, simulating "gravity"

// speed: Vector2(0, 100),

child:

CircleParticle

(

radius:

2.0

,

paint:

Paint

()..

color

=

Colors

.

red

,

),

),

),

);

CircleParticle

¶

A

Particle

which renders a circle with given

Paint

at the zero offset of passed

Canvas

. Use in conjunction with

TranslatedParticle

,

MovingParticle

or

AcceleratedParticle

in order to achieve desired positioning.

game

.

add

(

ParticleSystemComponent

(

particle:

CircleParticle

(

radius:

game

.

size

.

x

/

2

,

paint:

Paint

()..

color

=

Colors

.

red

.

withOpacity

(.

5

),

),

),

);

SpriteParticle

¶

Allows you to embed a

Sprite

into your particle effects.

```
game
.
add
(
ParticleSystemComponent
(
particle:
SpriteParticle
(
sprite:
Sprite
(
'sprite.png'
),
size:
Vector2
(
64
,
64
),
),
),
);
ImageParticle
¶
```

Renders given

dart:ui

image within the particle tree.

// During game initialization

await

Flame

.

images

.

loadAll

(

const

[

'image.png'

,

]);

// ...

// Somewhere during the game loop

final

image

=

await

Flame

.

images

.

```
load
(
'image.png'
);
game
.
add
(
ParticleSystemComponent
(
particle:
ImageParticle
(
size:
Vector2
.
all
(
24
),
image:
image
,
);
),
);
```

ScalingParticle

¶

Scales the child

Particle

between

1

and

to

during its lifespan.

game

.

add

(

ParticleSystemComponent

(

particle:

ScalingParticle

(

lifespan:

2

,

to:

0

,

child:

CircleParticle

(

radius:

2.0

,

paint:

Paint

()..

color

=

Colors

.

red

,

)

);

),

);

SpriteAnimationParticle

¶

A

Particle

which embeds a

SpriteAnimation

. By default, aligns the

SpriteAnimation

?s

stepTime

so that it?s fully played during the

Particle

lifespan. It?s possible to override this behavior with the

alignAnimationTime

argument.

final

spriteSheet

=

SpriteSheet

(

image:

yourSpriteSheetImage

,

srcSize:

Vector2

.

all

(

16.0

),

);

game

.

add

(

ParticleSystemComponent

(

particle:

SpriteAnimationParticle

(

animation:

spriteSheet

.

createAnimation

(

0

,

stepTime:

0.1

),

);

),

);

ComponentParticle

¶

This

Particle

allows you to embed a

Component

within the particle effects. The

Component

could have its own

update

lifecycle and could be reused across different effect trees. If the only thing you need is to add some dynam

Component

, please consider adding it to the

game

directly, without the

Particle

in the middle.

final

longLivingRect

=

RectComponent

();

game

.

add

(

ParticleSystemComponent

(

particle:

ComponentParticle

(

component:

longLivingRect

);

),

);

class

RectComponent

extends

Component

{

void

render

(

Canvas

c

)

{

c

.

drawRect

(

Rect

.

fromCenter

(

center:

Offset

.

zero

,

width:

100

,

height:

100

),

Paint

()..

color

=

Colors

.

red

);

}

void

update

(

double

dt

)

{

/// Will be called by parent [Particle]

}

}

ComputedParticle

¶

A

Particle

which could help you when:

Default behavior is not enough

Complex effects optimization

Custom easings

When created, it delegates all the rendering to a supplied

ParticleRenderDelegate

which is called on each frame to perform necessary computations and render something to the

Canvas

.

game

.

add

(

ParticleSystemComponent

(

// Renders a circle which gradually changes its color and size during the

// particle lifespan.

particle:

ComputedParticle

(

renderer:

(

canvas

,

particle

)

=>

canvas

.

drawCircle

(

Offset

.

zero

,

particle

.

progress

*

10

,

Paint

()

..

color

=

Color

.

lerp

(

Colors

.

red

,

Colors

.

blue

,

particle

.

progress

,

),

),

),

),

)

Nesting behavior

¶

Flame?s implementation of particles follows the same pattern of extreme composition as Flutter widgets. T

Two entities that allow

Particle

s to nest each other are:

SingleChildParticle

mixin and

ComposedParticle

class.

A

SingleChildParticle

may help you with creating

Particles

with a custom behavior. For example, randomly positioning its child during each frame:

The

SingleChildParticle

may help you with creating

Particles

with a custom behavior.

For example, randomly positioning it?s child during each frame:

var

rnd

=

Random

();

class

GlitchParticle

extends

Particle

with

SingleChildParticle

{

Particle

child

;

GlitchParticle

({

required

this

.

child

,

super

.

lifespan

,

});

@override

render

(

Canvas

canvas

)

{

canvas

.

save

();

```
canvas
.
translate
(
rnd
.
nextDouble
()
*
100
,
rnd
.
nextDouble
()
*
100
);
// Will also render the child
super
.
render
();
canvas
.
restore
```

();

}

}

The

ComposedParticle

could be used either as a standalone or within an existing

Particle

tree.

# Getting Started

¶

## About Flame

¶

Flame is a modular Flutter game engine that provides a complete set of out-of-the-way solutions for games

It provides you with a simple yet effective game loop implementation, and the necessary functionalities that

We also provide stand-alone packages that extend the Flame functionality which can be found in the

Bridge Packages

section.

You can pick and choose whichever parts you want, as they are all independent and modular.

The engine and its ecosystem are constantly being improved by the community, so please feel free to reac

Give us a star if you want to help give the engine exposure and grow the community. :)

## Installation

¶

Add the

flame

package as a dependency in your

pubspec.yaml

by running the following command:

```
flutter pub add flame
```

The latest version can be found on

pub.dev

.

then run

```
flutter
pub
```

get

and you are ready to start using it!

Getting started

¶

There is a set of tutorials that you can follow to get started in the

tutorials folder

.

Simple examples for all features can be found in the

examples folder

.

You can also check out the

awesome flame repository

, it contains quite a lot of good tutorials and articles written by the community to get you started with Flame.

Outside of the scope of the engine

¶

Games sometimes require complex feature sets depending on what the game is all about. Some of these f

Multiplayer (netcode)

¶

Flame doesn?t bundle any network feature, which may be needed to write online multiplayer games.

If you are building a multiplayer game, here are some recommendations of packages/services:

Nakama

: Nakama is an open-source server designed to power modern games and apps.

Firebase

: Provides dozens of services that can be used to write simpler multiplayer experiences.

Supabase

: A cheaper alternative to Firebase, based on Postgres.

# Getting Started

¶

## About Flame

¶

Flame is a modular Flutter game engine that provides a complete set of out-of-the-way solutions for games

It provides you with a simple yet effective game loop implementation, and the necessary functionalities that

We also provide stand-alone packages that extend the Flame functionality which can be found in the

Bridge Packages

section.

You can pick and choose whichever parts you want, as they are all independent and modular.

The engine and its ecosystem are constantly being improved by the community, so please feel free to reac

Give us a star if you want to help give the engine exposure and grow the community. :)

## Installation

¶

Add the

flame

package as a dependency in your

pubspec.yaml

by running the following command:

```
flutter pub add flame
```

The latest version can be found on

pub.dev

.

then run

```
flutter
pub
```

get

and you are ready to start using it!

Getting started

¶

There is a set of tutorials that you can follow to get started in the

tutorials folder

.

Simple examples for all features can be found in the

examples folder

.

You can also check out the

awesome flame repository

, it contains quite a lot of good tutorials and articles written by the community to get you started with Flame.

Outside of the scope of the engine

¶

Games sometimes require complex feature sets depending on what the game is all about. Some of these f

Multiplayer (netcode)

¶

Flame doesn?t bundle any network feature, which may be needed to write online multiplayer games.

If you are building a multiplayer game, here are some recommendations of packages/services:

Nakama

: Nakama is an open-source server designed to power modern games and apps.

Firebase

: Provides dozens of services that can be used to write simpler multiplayer experiences.

Supabase

: A cheaper alternative to Firebase, based on Postgres.

Documentation Site

¶

Flame?s documentation is written in

Markdown

. It is then rendered into HTML with the help of the

Sphinx

engine and its

MyST

plugin. The rendered files are then manually (but with the help of a script) published to

flame-docs-site

, where the site is served via

GitHub Pages

.

Markdown

¶

The main documentation site is written in Markdown. We assume that you?re already familiar with the basi

Table of contents

¶

The table of contents for the site must be created manually. This is done using special

{toctree}

blocks, one per each subdirectory:

```{toctree} :hidden:  First Topic  <relative_path/to_topic1.md> Second Topic  <topic2.md> ```

When adding new documents into the documentation site, make sure that they are mentioned in one of the

Admonitions

¶

Admonitions are emphasized blocks of text with a distinct appearance. They are created using the triple-ba
```

```{note} Please note this very important caveat.

```

```{warning}

Don't look down, or you will encounter an error.

```

```{error} I told you so.

```

```{seealso}

Also check out this cool thingy.

```

Note

Please note this very important caveat.

Warning

Don?t look down, or you will encounter an error.

Error

I told you so.

See also

Also check out this cool thingy.

Deprecations

¶

The special

{deprecated}

block can be used to mark some part of documentation or syntax as being deprecated. This block requires

```{deprecated} v1.3.0  Please use this

**other**

thing instead. ```
```

Which would be rendered like this:

Deprecated since version v1.3.0:

Please use this

other

thing instead.

Live examples

¶

Our documentation site includes a custom-built

flutter-app

directive which allows creating Flutter widgets and embedding them alongside the overall documentation c

In Markdown, the code for inserting an embed looks like this:

```{flutter-app} :sources: ../flame/examples :page: tap_events :show: widget code popup :width: 180 :height

Here?s what the different options mean:

sources

: specifies the name of the root directory where the Flutter code that you wish to run is located. This directo

pubspec.yaml

file there. The path is considered relative to the

doc/_sphinx

directory.

page

: a sub-path within the root directory given in

sources

. This option has two effects: first, it is appended to the path of the html page of the widget, like so:

main.dart.html?$page

. Secondly, the button to show the source code of the embed will display the code from the file or directory

page
```

.

The purpose of this option is to be able to bundle multiple examples into a single executable. When using t

main.dart

file of the app should route the execution to the proper widget according to the

page

being passed.

show

: contains a subset of modes:

widget

,

code

,

infobox

, and

popup

. The

widget

mode creates an iframe with the embedded example, directly within the page. The

code

mode will show a button that allows the user to see the code that produced this example. The

popup

mode also shows a button, which displays the example in an overlay window. This is more suitable for dem

infobox

mode will display the result in a floating window ? this mode is best combined with

widget

and

code

.

width

: an integer that defines the width of the embedded application.  If this is not defined, the width will be 100%

height

: an integer that defines the height of the embedded application. If this is not defined, the height will be 350

Run

tap_events.dart

```
1
import
'dart:math'
;
2
3
import
'package:flame/components.dart'
;
4
import
'package:flame/events.dart'
;
5
import
'package:flame/game.dart'
;
6
```

```
import
'package:flutter/rendering.dart'
;
7
8
class
TapEventsGame
extends
FlameGame
{
9
@override
10
Future
<
void
>
onLoad
()
async
{
11
add
(
TapTarget
());
```

12

}

13

}

14

15

/// This component is the tappable blue-ish rectangle in the center of the game.

16

/// It uses the [TapCallbacks] mixin to receive tap events.

17

class

TapTarget

extends

PositionComponent

with

TapCallbacks

{

18

TapTarget

()

:

super

(

anchor:

Anchor

.

center

);

19

20

final

_paint

=

Paint

()..

color

=

const

Color

(

0x448BA8FF

);

21

22

/// We will store all current circles into this map, keyed by the `pointerId`

23

/// of the event that created the circle.

24

final

Map

<

int

,

ExpandingCircle

>

_circles

=

{};

25

26

@override

27

void

onGameResize

(

Vector2

size

)

{

28

super

.

onGameResize

(

size

);

29

this

.

size

=

size

-

Vector2

(

100

,

75

);

30

if

(

this

.

size

.

x

<

100

||

this

.

size

.

y

<

100

)

{

31

this

.

size

=

size

*

0.9

;

32

}

33

position

=

size

/

2

;

34

}

35

```
36
@override
37
void
render
(
Canvas
canvas
)
{
38
canvas
.
drawRect
(
size
.
toRect
(),
_paint
);
39
}
40
41
@override
```

42

void

onTapDown

(

TapDownEvent

event

)

{

43

final

circle

=

ExpandingCircle

(

event

.

localPosition

);

44

_circles

[

event

.

pointerId

]

=

circle

;

45

add

(

circle

);

46

}

47

48

@override

49

void

onLongTapDown

(

TapDownEvent

event

)

{

50

_circles

[

event

.

pointerId

]

!

.

accent

();

51

}

52

53

@override

54

void

onTapUp

(

TapUpEvent

event

)

{

55

_circles

.

remove

(

event

.

pointerId

)

!

.

release

();

56

}

57

58

@override

59

void

onTapCancel

(

TapCancelEvent

event

)

{

60

_circles

.

remove

(

event

.

pointerId

)

!

.

cancel

();

61

}

62

}

63

64

class

ExpandingCircle

extends

Component

{

65

ExpandingCircle

(

this

.

_center

)

66

:

_baseColor

=

67

HSLColor

.

fromAHSL

(

1

,

random

.

nextDouble

()

*

360

,

1

,

0.8

).

toColor

();

68

69

final

Color

_baseColor

;

70

final

Vector2

_center

;

71

double

_outerRadius

=

0

;

72

double

_innerRadius

=

0

;

73

bool

_released

=

false

;

74

bool

_cancelled

=

false

;

75

late

final

_paint

=

Paint

()

76

..

style

=

PaintingStyle

.

stroke

77

..

color

=

_baseColor

;

78

79

/// "Accent" is thin white circle generated by `onLongTapDown`. We use

80

/// negative radius to indicate that the circle should not be drawn yet.

81

double

_accentRadius

=

-

1e10

;

82

late

final

_accentPaint

=

Paint

()

83

..

style

=

PaintingStyle

.

stroke

84

..

strokeWidth

=

0

85

..

color

=

const

Color

(

0xFFFFFFFF

);

86

87

/// At this radius the circle will disappear.

88

static

const

maxRadius

=

175

;

89

static

final

random

=

Random

();

90

91

double

get

radius

=>

(

_innerRadius

+

_outerRadius

)

/

2

;

92

93

void

release

()

=>

_released

=

true

;

94

void

cancel

()

=>

_cancelled

=

true

;

95

void

accent

()

=>

_accentRadius

=

0

;

96

97

@override

98

void

render

(

Canvas

canvas

)

{

99

canvas

.

drawCircle

(

_center

.

toOffset

(),

radius

,

_paint

);

100

if

(

_accentRadius

>=

0

)

{

101

```
canvas
.
drawCircle
(
_center
.
toOffset
(),
_accentRadius
,
_accentPaint
);
102
}
103
}
104
105
@override
106
void
update
(
double
dt
)
```

{

107

if

(

_cancelled

)

{

108

_innerRadius

+=

dt

*

100

;

// implosion

109

}

else

{

110

_outerRadius

+=

dt

*

20

;

111

_innerRadius

+=

dt

*

(

_released

?

20

:

6

);

112

_accentRadius

+=

dt

*

20

;

113

}

114

if

(

radius

>=

maxRadius

||

_innerRadius

>

_outerRadius

)

{

115

removeFromParent

();

116

}

else

{

117

final

opacity

=

1

-

radius

/

maxRadius

;

118

_paint

.

color

=

_baseColor

.

withOpacity

(

opacity

);

119

_paint

.

strokeWidth

=

_outerRadius

-

_innerRadius

;

120

}

121

}

122

}

Code

Standardization and Templates

¶

For every section or package added to the documentation, naming conventions, directory structure, and sta

bridge_packages/package_name/package_name.md

documentation_section/documentation_section.md

Note

Avoid having spaces in the paths to the docs since that will keep you from building the project due to

this bug

.

Building documentation locally

¶

Building the documentation site on your own computer is fairly simple. All you need is the following:

A working

Flutter

installation, accessible from the command line.

Melos

command-line tool, as per the

contributing

guide.

A

Python

environment, with python version 3.8+ or higher. Having a dedicated python virtual environment is recomm

Install the remaining requirements using the command

melos

run

doc-setup

Once these prerequisites are met, you can build the documentation by using the built-in Melos target:

melos

doc-build

The

melos doc-build

command here renders the documentation site into HTML. This command needs to be re-run every time yo

If you want to automatically recompile the docs every time there is a change to one of the files you can use

melos

doc-serve

When using the

melos doc-serve

command, the

melos doc-build

is only needed when there are changes to the sphinx theme. This is because the serve command both auto

http://localhost:8000/

by default.

There are other make commands that you may find occasionally useful too:

melos doc-clean

removes all cached generated files (in case the system gets stuck in a bad state).

melos doc-linkcheck

to check whether there are any broken links in the documentation.

melos doc-kill

removes any orphaned TCP threads running on port 8000.

The generated html files will be in the

doc/_build/html

directory, you can view them directly by opening the file

doc/_build/html/index.html

in your browser. The only drawback is that the browser won?t allow any dynamic content in a file opened fr

melos doc-serve

.

If you ever run the

melos doc-clean

command, the server will need to be restarted, because the clean command deletes the entire

html

directory.

Note

Avoid having spaces in the paths to the docs since that will keep you from building the project due to

this bug

.

Components

¶

%%{init: { 'theme': 'dark' } }%% graph TD %% Config %% classDef default fill:#282828,stroke:#F6BE0

This diagram might look intimidating, but don?t worry, it is not as complex as it looks.

Component

¶

All components inherit from the abstract class

Component

and all components can have other

Component

s as children. This is the base of what we call the Flame Component System, or FCS for short.

Children can be added either with the

add(Component

c)

method or directly in the constructor.

Example:

void

main

()

{

final

component1

=

Component

(

children:

[

Component

(),

Component

()]);

final

component2

=

Component

();

component2

.

add

(

Component

());

component2

.

addAll

([

Component

(),

Component

()]);

}

The

Component()

here could of course be any subclass of

Component

.

Every

Component

has a few methods that you can optionally implement, which are used by the

FlameGame

class.

Component lifecycle

¶

%%{init: { 'theme': 'dark' } }%%  graph TD  %% Node Color %%  classDef default fill:#282828,stroke:#

%%{init: { 'theme': 'dark' } }%%  graph LR  %% Node Color %%  classDef default fill:#282828,stroke:#F

The

onGameResize

method is called whenever the screen is resized, and also when this component gets added into the compo

onMount

.

The

onParentResize

method is similar: it is also called when the component is mounted into the component tree, and also when

The

onRemove

method can be overridden to run code before the component is removed from the game, it is only run once

Component

remove method.

The

onLoad

method can be overridden to run asynchronous initialization code for the component, like loading an image

onGameResize

, but before

onMount

. This method is guaranteed to execute only once during the lifetime of the component, so you can think of

The

onMount

method runs every time when the component is mounted into a game tree. This means that you should not

late

final

variables here, since this method might run several times throughout the component?s lifetime. This metho

The

onChildrenChanged

method can be overridden if it?s needed to detect changes in a parent?s children. This method is called wh

added

or

removed

).

A component lifecycle state can be checked by a series of getters:

isLoaded

: Returns a bool with the current loaded state.

loaded

: Returns a future that will complete once the component has finished loading.

isMounted

: Returns a bool with the current mounted state.

mounted

: Returns a future that will complete once the component has finished mounting.

isRemoved

: Returns a bool with the current removed state.

removed

: Returns a future that will complete once the component has been removed.

Priority

¶

In Flame every

Component

has the

int

priority

property, which determines that component?s sorting order within its parent?s children. This is sometimes

z-index

in other languages and frameworks. The higher the

priority

is set to, the closer the component will appear on the screen, since it will be rendered on top of any compo

If you add two components and set one of their priorities to 1 for example, then that component will be ren

All components take in

priority

as a named argument, so if you know the priority that you want your component at compile time, then you

Example:

class

MyGame

extends

FlameGame

{

@override

void

onLoad

()

{

final

myComponent

=

PositionComponent

(

priority:

5

);

add

(

myComponent

);

}

}

To update the priority of a component you have to set it to a new value, like

component.priority

=

2

, and it will be updated in the current tick before the rendering stage.

In the following example we first initialize the component with priority 1, and then when the user taps the co

```
class
MyComponent
extends
PositionComponent
with
TapCallbacks
{
MyComponent
()
:
super
(
priority:
1
);
@override
void
onTapDown
(
TapDownEvent
event
)
{
priority
```

=

2

;

}

}

Composability of components

¶

Sometimes it is useful to wrap other components inside of your component. For example by grouping visua

PositionComponent

.

When you have child components on a component every time the parent is updated and rendered, all the c

Example of usage, where visibility of two components are handled by a wrapper:

class

GameOverPanel

extends

PositionComponent

{

bool

visible

=

false

;

final

Image

spriteImage

;

```
GameOverPanel
(
this
.
spriteImage
);
@override
void
onLoad
()
{
final
gameOverText
=
GameOverText
(
spriteImage
);
// GameOverText is a Component
final
gameOverButton
=
GameOverButton
(
spriteImage
);
```

```
// GameOverRestart is a SpriteComponent
add
(
gameOverText
);
add
(
gameOverButton
);
}
@override
void
render
(
Canvas
canvas
)
{
if
(
visible
)
{
}
// If not visible none of the children will be rendered
}
```

}

There are two methods for adding children components to your component. First, you have methods

add()

,

addAll()

, and

addToParent()

, which can be used at any time during the game. Traditionally, children will be created and added from the

onLoad()

method, but it is also common to add new children during the course of the game.

The second method is to use the

children:

parameter in the component?s constructor. This approach more closely resembles the standard Flutter AP

class

MyGame

extends

FlameGame

{

@override

void

onLoad

()

{

add

(

PositionComponent

(

position:

Vector2

(

30

,

0

),

children:

[

HighScoreDisplay

(),

HitPointsDisplay

(),

FpsComponent

(),

],

),

);

}

}

The two approaches can be combined freely: the children specified within the constructor will be added firs

Note that the children added via either methods are only guaranteed to be available eventually: after they a

Access to the World from a Component

¶

If a component that has a

World

as an ancestor and requires access to that

World

object, one can use the

HasWorldReference

mixin.

Example:

class

MyComponent

extends

Component

with

HasWorldReference

<

MyWorld

>

,

TapCallbacks

{

@override

void

onTapDown

(

TapDownEvent

info

)

{

// world is of type MyWorld

world

.

add

(

AnotherComponent

());

}

}

If you try to access

world

from a component that doesn?t have a

World

ancestor of the correct type an assertion error will be thrown.

Ensuring a component has a given parent

¶

When a component requires to be added to a specific parent type the

ParentIsA

mixin can be used to enforce a strongly typed parent.

Example:

class

MyComponent

extends

Component

with

ParentIsA

<

MyParentComponent

>

{

@override

void

onLoad

()

{

// parent is of type MyParentComponent

print

(

parent

.

myValue

);

}

}

If you try to add

MyComponent

to a parent that is not

MyParentComponent

, an assertion error will be thrown.

Ensuring a component has a given ancestor

¶

When a component requires to have a specific ancestor type somewhere in the component tree,

HasAncestor

mixin can be used to enforce that relationship.

The mixin exposes the

ancestor

field that will be of the given type.

Example:

```
class
MyComponent
extends
Component
with
HasAncestor
<
MyAncestorComponent
>
{
@override
void
onLoad
()
{
// ancestor is of type MyAncestorComponent.
print
(
ancestor
```

.

myValue

);

}

}

If you try to add

MyComponent

to a tree that does not contain

MyAncestorComponent

, an assertion error will be thrown.

Component Keys

¶

Components can have an identification key that allows them to be retrieved from the component tree, from

To register a component with a key, simply pass a key to the

key

argument on the component?s constructor:

final

myComponent

=

Component

(

key:

ComponentKey

.

named

(

'player'

),

);

Then, to retrieve it in a different point of the component tree:

flameGame

.

findByKey

(

ComponentKey

.

named

(

'player'

));

There are two types of keys,

unique

and

named

. Unique keys are based on equality of the key instance, meaning that:

final

key

=

ComponentKey

.

unique

();

final

key2

=

key

;

print

(

key

==

key2

);

// true

print

(

key

==

ComponentKey

.

unique

());

// false

Named ones are based on the name that it receives, so:

final

key1

=

ComponentKey

.

named

(

'player'

);

final

key2

=

ComponentKey

.

named

(

'player'

);

print

(

key1

==

key2

);

// true

When named keys are used, the

findByKeyName

helper can also be used to retrieve the component.

flameGame

.

findByKeyName

(

'player'

);

Querying child components

¶

The children that have been added to a component live in a

QueryableOrderedSet

called

children

. To query for a specific type of components in the set, the

query<T>()

function can be used. By default

strictMode

is

false

in the children set, but if you set it to true, then the queries will have to be registered with

children.register

before a query can be used.

If you know in compile time that you later will run a query of a specific type it is recommended to register th

strictMode

is set to

true

or

false

, since there are some performance benefits to gain from it. The

register

call is usually done in

onLoad

.

Example:

@override

void

onLoad

()

{

children

.

register

<

PositionComponent

>

();

}

In the example above a query is registered for

PositionComponent

s, and an example of how to query the registered component type can be seen below.

@override

void

update

(

double

dt

)

{

final

allPositionComponents

=

children

.

query

<

PositionComponent

>

();

}

Querying components at a specific point on the screen

¶

The method

componentsAtPoint()

allows you to check which components were rendered at some point on the screen. The returned value is a

List<Vector2>

as a second parameter.

The iterable retrieves the components in the front-to-back order, i.e. first the components in the front, follow

This method can only return components that implement the method

containsLocalPoint()

. The

PositionComponent

(which is the base class for many components in Flame) provides such an implementation. However, if you

Component

, you?d have to implement the

containsLocalPoint()

method yourself.

Here is an example of how

componentsAtPoint()

can be used:

```
void
onDragUpdate
(
DragUpdateInfo
info
)
{
game
.
componentsAtPoint
(
info
.
widget
).
forEach
((
component
```

)

{

if

(

component

is

DropTarget

)

{

component

.

highlight

();

}

});

}

PositionType

¶

Note

If you are using the

CameraComponent

you should not use

PositionType

, but instead adding your components directly to the viewport for example if you want to use them as a HU

If you want to create a HUD (Head-up display) or another component that isn?t positioned in relation to the

PositionType

of the component. The default

PositionType

is

positionType

=

PositionType.game

and that can be changed to either

PositionType.viewport

or

PositionType.widget

depending on how you want to position the component.

PositionType.game

(Default) - Respects camera and viewport.

PositionType.viewport

- Respects viewport only (ignores camera).

PositionType.widget

- Position in relation to the coordinate system of the Flutter game widget (i.e. the raw canvas).

Most of your components will probably be positioned according to

PositionType.game

, since you want them to respect the

Camera

and the

Viewport

. But quite often you want for example buttons and text to always show on the screen, no matter if you mov

PositionType.viewport

. In some rare cases you want to use

PositionType.widget

to position your widgets, when you don?t want the component to respect the camera nor the viewport; this

Do note that this setting is only respected if the component is added directly to the root

FlameGame

and not as a child component of another component.

Visibility of components

¶

The recommended way to hide or show a component is usually to add or remove it from the tree using the

add

and

remove

methods.

However, adding and removing components from the tree will trigger lifecycle steps for that component (su

onRemove

and

onMount

). It is also an asynchronous process and care needs to be taken to ensure the component has finished rer

/// Example of handling the removal and adding of a child component

/// in quick succession

void

show

()

async

{

// Need to await the [removed] future first, just in case the

// component is still in the process of being removed.

await

myChildComponent

.

removed

;

add

(

myChildComponent

);

}

void

hide

()

{

remove

(

myChildComponent

);

}

These behaviors are not always desirable.

An alternative method to show and hide a component is to use the

HasVisibility

mixin, which may be used on any class that inherits from

Component

. This mixin introduces the

isVisible

property. Simply set

isVisible

to

false

to hide the component, and

true

to show it again, without removing it from the tree. This affects the visibility of the component and all it?s de

/// Example that implements HasVisibility

class

MyComponent

extends

PositionComponent

with

HasVisibility

{}

/// Usage of the isVisible property

final

myComponent

=

MyComponent

();

add

(

myComponent

);

myComponent

.

isVisible

=

false

;

The mixin only affects whether the component is rendered, and will not affect other behaviors.

Note

Important! Even when the component is not visible, it is still in the tree and will continue to receive calls to 1

The mixin works by preventing the

renderTree

method, therefore if

renderTree

is being overridden, a manual check for

isVisible

should be included to retain this functionality.

class

MyComponent

extends

PositionComponent

with

HasVisibility

{

@override

void

renderTree

(

Canvas

canvas

)

{

// Check for visibility

if

(

isVisible

)

{

// Custom code here

// Continue rendering the tree

super

.

renderTree

(

canvas

);

}

}

}

PositionComponent

¶

This class represents a positioned object on the screen, being a floating rectangle, a rotating sprite, or anyt

The base of the

PositionComponent

is that it has a

position

,

size

,

scale

,

angle

and

anchor

which transforms how the component is rendered.

Position

¶

The

position

is just a

Vector2

which represents the position of the component?s anchor in relation to its parent; if the parent is a

FlameGame

, it is in relation to the viewport.

Size

¶

The

size

of the component when the zoom level of the camera is 1.0 (no zoom, default). The

size

is

not

in relation to the parent of the component.

Scale

¶

The

scale

is how much the component and its children should be scaled. Since it is represented by a

Vector2

, you can scale in a uniform way by changing

x

and

y

with the same amount, or in a non-uniform way, by change

x

or

y

by different amounts.

Angle

¶

The

angle

is the rotation angle around the anchor, represented as a double in radians. It is relative to the parent?s an

Native Angle

¶

The

nativeAngle

is an angle in radians, measured clockwise, representing the default orientation of the component. It can be

angle

is zero.

It is specially helpful when making a sprite based component look at a specific target. If the original image

nativeAngle

can be used to let the component know what direction the original image is faces.

An example could be a bullet image pointing in east direction. In this case

nativeAngle

can be set to pi/2 radians. Following are some common directions and their corresponding native angle val

Direction

Native Angle

In degrees

Up/North

0

0

Down/South

pi or -pi

180 or -180

Left/West

-pi/2

-90

Right/East

pi/2

90

Anchor

¶

anchor.dart

1

import

'dart:async'

;

2

3

import

'package:flame/components.dart'

;

4

import

'package:flame/events.dart'

;

5

import

'package:flame/game.dart'

;

6

import

'package:flame/palette.dart'

;

7

8

class

AnchorGame

extends

FlameGame

{

9

final

_parentAnchorText

=

TextComponent

(

position:

Vector2

.

all

(

5

));

10

final

_childAnchorText

=

TextComponent

(

position:

Vector2

(

5

,

30

));

11

12

late

_AnchoredRectangle

_redComponent

;

13

late

_AnchoredRectangle

_blueComponent

;

14

15

@override

16

Future

<

void

>

onLoad

()

async

{

17

_redComponent

=

_AnchoredRectangle

(

18

size:

size

/

4

,

19

position:

size

/

2

,

20

paint:

BasicPalette

.

red

.

paint

(),

21

);

22

23

_blueComponent

=

_AnchoredRectangle

(

24

size:

size

/

8

,

25

paint:

BasicPalette

.

blue

.

paint

(),

26

);

27

28

```
await
_redComponent
.
addAll
([
29
_blueComponent
,
30
CircleComponent
(
radius:
2
,
anchor:
Anchor
.
center
),
31
]);
32
33
await
addAll
([
```

34

_redComponent

,

35

_parentAnchorText

,

36

_childAnchorText

,

37

CircleComponent

(

38

radius:

4

,

39

position:

size

/

2

,

40

anchor:

Anchor

.

center

,

41

),

42

]);

43

}

44

45

@override

46

void

update

(

double

dt

)

{

47

_parentAnchorText

.

text

=

'Parent:

${

_redComponent

.

anchor

}

'

;

48

_childAnchorText

.

text

=

'Child:

${

_blueComponent

.

anchor

}

'

;

49

super

.

update

(

dt

);

50

}

51

}

52

53

class

_AnchoredRectangle

extends

RectangleComponent

with

TapCallbacks

{

54

_AnchoredRectangle

({

55

super

.

position

,

56

super

.

size

,

57

super

.

paint

,

58

});

59

60

@override

61

void

onTapDown

(

TapDownEvent

event

)

{

62

var

index

=

Anchor

.

values

.

```
indexOf
(
anchor
)
+
1
;
63
if
(
index
==
Anchor
.
values
.
length
)
{
64
index
=
0
;
65
}
```

66

anchor

=

Anchor

.

values

.

elementAt

(

index

);

67

super

.

onTapDown

(

event

);

68

}

69

}

Code

This example shows effect of changing

anchor

point of parent (red) and child (blue) components. Tap on them to cycle through the anchor points. Note tha

The

anchor

is where on the component that the position and rotation should be defined from (the default is

Anchor.topLeft

). So if you have the anchor set as

Anchor.center

the component?s position on the screen will be in the center of the component and if an

angle

is applied, it is rotated around the anchor, so in this case around the center of the component. You can thin

When

position

or

absolutePosition

of a component is queried, the returned coordinates are that of the

anchor

of the component. In case if you want to find the position of a specific anchor point of a component which is

anchor

of that component, you can use the

positionOfAnchor

and

absolutePositionOfAnchor

method.

final

comp

=

PositionComponent

(

size:

Vector2

.

all

(

20

),

anchor:

Anchor

.

center

,

);

// Returns (0,0)

final

p1

=

component

.

position

;

// Returns (10, 10)

final

p2

=

component

.

positionOfAnchor

(

Anchor

.

bottomRight

);

A common pitfall when using

anchor

is confusing it for as being the attachment point for children components. For example, setting

anchor

to

Anchor.center

for a parent component does not mean that the children components will be placed w.r.t the center of pare

Note

Local origin for a child component is always the top-left corner of its parent component, irrespective of thei

anchor

values.

PositionComponent children

¶

All children of the

PositionComponent

will be transformed in relation to the parent, which means that the

position

,

angle

and

scale

will be relative to the parents state. So if you, for example, wanted to position a child in the center of the pa

@override

void

onLoad

()

{

final

parent

=

PositionComponent

(

position:

Vector2

(

100

,

100

),

size:

Vector2

(

100

,

100

),

);

final

child

=

PositionComponent

(

position:

parent

.

size

/

2

,

anchor:

Anchor

.

center

,

);

parent

.

add

(

child

);

}

Remember that most components that are rendered on the screen are

PositionComponent

s, so this pattern can be used in for example

SpriteComponent

and

SpriteAnimationComponent

too.

Render PositionComponent

¶

When implementing the

render

method for a component that extends

PositionComponent

remember to render from the top left corner (0.0). Your render method should not handle where on the scre

position

,

angle

and

anchor

properties and Flame will automatically handle the rest for you.

If you want to know where on the screen the bounding box of the component is you can use the

toRect

method.

In the event that you want to change the direction of your components rendering, you can also use

flipHorizontally()

and

flipVertically()

to flip anything drawn to canvas during

render(Canvas

canvas)

, around the anchor point. These methods are available on all

PositionComponent

objects, and are especially useful on

SpriteComponent

and

SpriteAnimationComponent

.

In case you want to flip a component around its center without having to change the anchor to

Anchor.center

, you can use

flipHorizontallyAroundCenter()

and

flipVerticallyAroundCenter()

.

SpriteComponent

¶

The most commonly used implementation of

PositionComponent

is

SpriteComponent

, and it can be created with a

Sprite

:

import

'package:flame/components/component.dart'

;

class

MyGame

extends

FlameGame

{

late

final

SpriteComponent

player

;

@override

Future

<

void

>

onLoad

()

async

{

final

sprite

=

await

Sprite

.

load

(

'player.png'

);

final

size

=

Vector2

.

all

(

128.0

);

final

player

=

SpriteComponent

(

size:

size

,

sprite:

sprite

);

// Vector2(0.0, 0.0) by default, can also be set in the constructor

player

.

position

=

Vector2

(

10

,

20

);

// 0 by default, can also be set in the constructor

player

.

angle

=

0

;

// Adds the component

add

(

player

);

}

}

SpriteAnimationComponent

¶

This class is used to represent a Component that has sprites that run in a single cyclic animation.

This will create a simple three frame animation using 3 different images:

@override

Future

<

void

>

onLoad

()

async

{

final

sprites

=

[

0

,

1

,

2

]

.

```
map
((
i
)
=>
Sprite
.
load
(
'player_
$
i
.png'
));
final
animation
=
SpriteAnimation
.
spriteList
(
await
Future
.
wait
(
```

sprites

),

stepTime:

0.01

,

);

this

.

player

=

SpriteAnimationComponent

(

animation:

animation

,

size:

Vector2

.

all

(

64.0

),

);

}

If you have a sprite sheet, you can use the

sequenced

constructor from the

SpriteAnimationData

class (check more details on

Images > Animation

):

@override

Future

<

void

>

onLoad

()

async

{

final

size

=

Vector2

.

all

(

64.0

);

final

data

=

SpriteAnimationData

.

sequenced

(

textureSize:

size

,

amount:

2

,

stepTime:

0.1

,

);

this

.

player

=

SpriteAnimationComponent

.

fromFrameData

(

await

images

.

load

(

'player.png'

),

data

,

);

}

All animation components internally maintains a

SpriteAnimationTicker

which ticks the

SpriteAnimation

. This allows multiple components to share the same animation object.

Example:

final

sprites

=

[

/*You sprite list here*/

];

final

animation

=

SpriteAnimation

.

spriteList

(

```
sprites
,
stepTime:
0.01
);
final
animationTicker
=
SpriteAnimationTicker
(
animation
);
// or alternatively, you can ask the animation object to create one for you.
final
animationTicker
=
animation
.
createTicker
();
// creates a new ticker
animationTicker
.
update
(
dt
```

);

To listen when the animation is done (when it reaches the last frame and is not looping) you can use

animationTicker.completed

.

Example:

await

animationTicker

.

completed

;

doSomething

();

// or alternatively

animationTicker

.

completed

.

whenComplete

(

doSomething

);

Additionally,

SpriteAnimationTicker

also has the following optional event callbacks:

onStart

,

onFrame

, and

onComplete

. To listen to these events, you can do the following:

final

animationTicker

=

SpriteAnimationTicker

(

animation

)

..

onStart

=

()

{

// Do something on start.

};

final

animationTicker

=

SpriteAnimationTicker

(

animation

)

..

onComplete

=

()

{

// Do something on completion.

};

final

animationTicker

=

SpriteAnimationTicker

(

animation

)

..

onFrame

=

(

index

)

{

if

(

index

==

1

)

{

// Do something for the second frame.

}

};

SpriteAnimationGroupComponent

¶

SpriteAnimationGroupComponent

is a simple wrapper around

SpriteAnimationComponent

which enables your component to hold several animations and change the current playing animation at run

SpriteAnimationComponent

.

Its use is very similar to the

SpriteAnimationComponent

but instead of being initialized with a single animation, this component receives a Map of a generic type

T

as key and a

SpriteAnimation

as value, and the current animation.

Example:

enum

RobotState

{

idle

,

running

,

}

final

running

=

await

loadSpriteAnimation

(

/* omitted */

);

final

idle

=

await

loadSpriteAnimation

(

/* omitted */

);

final

robot

=

SpriteAnimationGroupComponent

<

RobotState

>

(

```
animations:
{
RobotState
.
running:
running
,
RobotState
.
idle:
idle
,
},
current:
RobotState
.
idle
,
);
// Changes current animation to "running"
robot
.
current
=
RobotState
.
```

running

;

As this component works with multiple

SpriteAnimation

s, naturally it needs equal number of animation tickers to make all those animation tick. Use

animationsTickers

getter to access a map containing tickers for each animation state. This can be useful if you want to registe

onStart

,

onComplete

and

onFrame

.

Example:

enum

RobotState

{

idle

,

running

,

jump

}

final

running

=

```
await
loadSpriteAnimation
(
/* omitted */
);
final
idle
=
await
loadSpriteAnimation
(
/* omitted */
);
final
robot
=
SpriteAnimationGroupComponent
<
RobotState
>
(
animations:
{
RobotState
.
running:
```

running

,

RobotState

.

idle:

idle

,

},

current:

RobotState

.

idle

,

);

robot

.

animationTickers

?

[

RobotState

.

running

]

?

.

onStart

=

()

{

// Do something on start of running animation.

};

robot

.

animationTickers

?

[

RobotState

.

jump

]

?

.

onStart

=

()

{

// Do something on start of jump animation.

};

robot

.

animationTickers

?

[

RobotState

.

jump

]

?

.

onComplete

=

()

{

// Do something on complete of jump animation.

};

robot

.

animationTickers

?

[

RobotState

.

idle

]

?

.

onFrame

=

(

currentIndex

)

{

// Do something based on current frame index of idle animation.

};

SpriteGroup

¶

SpriteGroupComponent

is pretty similar to its animation counterpart, but especially for sprites.

Example:

class

PlayerComponent

extends

SpriteGroupComponent

<

ButtonState

>

with

HasGameReference

<

SpriteGroupExample

>

,

TapCallbacks

{

```
@override
Future
<
void
>?
onLoad
()
async
{
final
pressedSprite
=
await
gameRef
.
loadSprite
(
/* omitted */
);
final
unpressedSprite
=
await
gameRef
.
loadSprite
```

```
(
/* omitted */
);
sprites
=
{
ButtonState
.
pressed:
pressedSprite
,
ButtonState
.
unpressed:
unpressedSprite
,
};
current
=
ButtonState
.
unpressed
;
}
// tap methods handler omitted...
}
```

SpawnComponent

¶

This component is a non-visual component that spawns other components inside of the parent of the

SpawnComponent

. It?s great if you for example want to spawn enemies or power-ups randomly within an area.

The

SpawnComponent

takes a factory function that it uses to create new components and an area where the components should

For the area, you can use the

Circle

,

Rectangle

or

Polygon

class, and if you want to only spawn components along the edges of the shape set the

within

argument to false (defaults to true).

This would for example spawn new components of the type

MyComponent

every 0.5 seconds randomly within the defined circle:

The

factory

function takes an

int

as an argument, which is the index of the component that is being spawned, so if for example 4 componen

SpawnComponent

(

factory

:

(

i

)

=>

MyComponent

(

size:

Vector2

(

10

,

20

)),

period:

0.5

,

area:

Circle

(

Vector2

(

100

,

200

),

150

),

);

If you don?t want the spawning rate to be static, you can use the

SpawnComponent.periodRange

constructor with the

minPeriod

and

maxPeriod

arguments instead. In the following example the component would be spawned randomly within the circle a

SpawnComponent

.

periodRange

(

factory

:

(

i

)

=>

MyComponent

(

size:

Vector2

(

10

,

20

)),

minPeriod:

0.5

,

maxPeriod:

10

,

area:

Circle

(

Vector2

(

100

,

200

),

150

),

);

If you want to set the position yourself within the

factory

function, you can use set

selfPositioning

=

true

in the constructors and you will be able to set the positions yourself and ignore the

area

argument.

SpawnComponent

(

factory

:

(

i

)

=>

MyComponent

(

position:

Vector2

(

100

,

200

),

size:

Vector2

(

10

,

20

)),

selfPositioning:

true

,

period:

0.5

,

);

SvgComponent

¶

Note

: To use SVG with Flame, use the

flame_svg

package.

This component uses an instance of

Svg

class to represent a Component that has a svg that is rendered in the game:

@override

Future

<

void

>

onLoad

()

async

{

final

svg

=

await

Svg

.

load

(

'android.svg'

);

final

android

=

SvgComponent

.

fromSvg

(

svg

,

position:

Vector2

.

all

(

100

),

size:

Vector2

.

all

(

100

),

);

}

ParallaxComponent

¶

This

Component

can be used to render backgrounds with a depth feeling by drawing several transparent images on top of e

ParallaxRenderer

) is moving with a different velocity.

The rationale is that when you look at the horizon and moving, closer objects seem to move faster than dis

This component simulates this effect, making a more realistic background effect.

The simplest

ParallaxComponent

is created like this:

@override

Future

<
void
>
onLoad
()
async
{
final
parallaxComponent
=
await
loadParallaxComponent
([
ParallaxImageData
(
'bg.png'
),
ParallaxImageData
(
'trees.png'
),
]);
add
(
parallaxComponent
);

}

A ParallaxComponent can also ?load itself? by implementing the

onLoad

method:

class

MyParallaxComponent

extends

ParallaxComponent

<

MyGame

>

{

@override

Future

<

void

>

onLoad

()

async

{

parallax

=

await

gameRef

.

```
loadParallax
([
ParallaxImageData
(
'bg.png'
),
ParallaxImageData
(
'trees.png'
),
]);
}
}
class
MyGame
extends
FlameGame
{
@override
void
onLoad
()
{
add
(
MyParallaxComponent
```

());

}

}

This creates a static background. If you want a moving parallax (which is the whole point of a parallax), you

They simplest way is to set the named optional parameters

baseVelocity

and

velocityMultiplierDelta

in the

load

helper function. For example if you want to move your background images along the X-axis with a faster sp

@override

Future

<

void

>

onLoad

()

async

{

final

parallaxComponent

=

await

loadParallaxComponent

(

_dataList

,

baseVelocity:

Vector2

(

20

,

0

),

velocityMultiplierDelta:

Vector2

(

1.8

,

1.0

),

);

}

You can set the baseSpeed and layerDelta at any time, for example if your character jumps or your game s

@override

void

onLoad

()

{

final

parallax

=

parallaxComponent

.

parallax

;

parallax

.

baseSpeed

=

Vector2

(

100

,

0

);

parallax

.

velocityMultiplierDelta

=

Vector2

(

2.0

,

1.0

);

}

By default, the images are aligned to the bottom left, repeated along the X-axis and scaled proportionally s

repeat

,

alignment

and

fill

parameters for each

ParallaxRenderer

and add them to

ParallaxLayer

s that you then pass in to the

ParallaxComponent

?s constructor.

Advanced example:

final

images

=

[

loadParallaxImage

(

'stars.jpg'

,

repeat:

ImageRepeat

.

repeat

,

alignment:

Alignment

.

center

,

fill:

LayerFill

.

width

,

),

loadParallaxImage

(

'planets.jpg'

,

repeat:

ImageRepeat

.

repeatY

,

alignment:

Alignment

.

bottomLeft

,

fill:

LayerFill

.

none

,

),

loadParallaxImage

(

'dust.jpg'

,

repeat:

ImageRepeat

.

repeatX

,

alignment:

Alignment

.

topRight

,

fill:

LayerFill

.

height

,

),

];

final

layers

=

images

.

map

(

(

image

)

=>

ParallaxLayer

(

await

image

,

velocityMultiplier:

images

.

indexOf

(

image

)

*

2.0

,

)

);

final

parallaxComponent

=

ParallaxComponent

.

fromParallax

(

Parallax

(

await

Future

.

wait

(

layers

),

baseVelocity:

Vector2

(

50

,

0

),

),

);

The stars image in this example will be repeatedly drawn in both axis, align in the center and be scaled to f

The planets image will be repeated in Y-axis, aligned to the bottom left of the screen and not be scaled.

The dust image will be repeated in X-axis, aligned to the top right and scaled to fill the screen height.

Once you are done setting up your

ParallaxComponent

, add it to the game like with any other component (

game.add(parallaxComponent

). Also, don?t forget to add you images to the

pubspec.yaml

file as assets or they wont be found.

The

Parallax

file contains an extension of the game which adds

loadParallax

,

loadParallaxLayer

,

loadParallaxImage

and

loadParallaxAnimation

so that it automatically uses your game?s image cache instead of the global one. The same goes for the

ParallaxComponent

file, but that provides

loadParallaxComponent

.

If you want a fullscreen

ParallaxComponent

simply omit the

size

argument and it will take the size of the game, it will also resize to fullscreen when the game changes size

Flame provides two kinds of

ParallaxRenderer

:

ParallaxImage

and

ParallaxAnimation

,

ParallaxImage

is a static image renderer and

ParallaxAnimation

is, as it?s name implies, an animation and frame based renderer. It is also possible to create custom rende

ParallaxRenderer

class.

Three example implementations can be found in the

examples directory

.

ShapeComponents

¶

A

ShapeComponent

is the base class for representing a scalable geometrical shape. The shapes have different ways of defining

These shapes are meant as a tool for using geometrical shapes in a more general way than together with t

ShapeHitbox

es.

PolygonComponent

¶

A

PolygonComponent

is created by giving it a list of points in the constructor, called vertices. This list will be transformed into a po

For example, this would create a square going from (50, 50) to (100, 100), with it?s center in (75, 75):

```
void
main
()
{
PolygonComponent
([
Vector2
(
100
,
100
),
Vector2
(
100
,
```

50

),

Vector2

(

50

,

50

),

Vector2

(

50

,

100

),

]);

}

A

PolygonComponent

can also be created with a list of relative vertices, which are points defined in relation to the given size, mos

For example you could create a diamond shapes polygon like this:

void

main

()

{

PolygonComponent

.

relative

(

[

Vector2

(

0.0

,

1.0

),

// Middle of top wall

Vector2

(

1.0

,

0.0

),

// Middle of right wall

Vector2

(

0.0

,

-

1.0

),

// Middle of bottom wall

Vector2

(

-

1.0

,

0.0

),

// Middle of left wall

],

size:

Vector2

.

all

(

100

),

);

}

The vertices in the example defines percentages of the length from the center to the edge of the screen in

Vector2(0.0,

1.0)

) we are pointing on the middle of the top wall of the bounding box, since the coordinate system here is def

In the image you can see how the polygon shape formed by the purple arrows is defined by the red arrows

Remember to define the lists in a counter clockwise manner (if you think in the screen coordinate system w

RectangleComponent

¶

A

RectangleComponent

is created very similarly to how a

PositionComponent

is created, since it also has a bounding rectangle.

Something like this for example:

void

main

()

{

RectangleComponent

(

position:

Vector2

(

10.0

,

15.0

),

size:

Vector2

.

all

(

10

),

angle:

pi

/

2

,

anchor:

Anchor

.

center

,

);

}

Dart also already has an excellent way to create rectangles and that class is called

Rect

, you can create a Flame

RectangleComponent

from a

Rect

by using the

Rectangle.fromRect

factory, and just like when setting the vertices of the

PolygonComponent

, your rectangle will be sized according to the

Rect

if you use this constructor.

The following would create a

RectangleComponent

with its top left corner in

(10,

10)

and a size of

(100,

50)

.

void

main

()

{

RectangleComponent

.

fromRect

(

Rect

.

fromLTWH

(

10

,

10

,

100

,

50

),

);

}

You can also create a

RectangleComponent

by defining a relation to the intended parent?s size, you can use the default constructor to build your rectar

relation

is a vector defined in relation to the parent size, for example a

relation

that is

Vector2(0.5,

0.8)

would create a rectangle that is 50% of the width of the parent?s size and 80% of its height.

In the example below a

RectangleComponent

of size

(25.0,

30.0)

positioned at

(100,

100)

would be created.

void

main

()

{

```
RectangleComponent
.
relative
(
Vector2
(
0.5
,
1.0
),
position:
Vector2
.
all
(
100
),
size:
Vector2
(
50
,
30
),
);
}
```

Since a square is a simplified version of a rectangle, there is also a constructor for creating a square

RectangleComponent

, the only difference is that the

size

argument is a

double

instead of a

Vector2

.

void

main

()

{

RectangleComponent

.

square

(

position:

Vector2

.

all

(

100

),

size:

200

,

);

}

CircleComponent

¶

If you know how long your circle?s position and/or how long the radius is going to be from the start you can

radius

and

position

to set those.

The following would create a

CircleComponent

with its center in

(100,

100)

with a radius of 5, and therefore a size of

Vector2(10,

10)

.

void

main

()

{

CircleComponent

(

radius:

5

,

position:

Vector2

.

all

(

100

),

anchor:

Anchor

.

center

);

}

When creating a

CircleComponent

with the

relative

constructor you can define how long the radius is in comparison to the shortest edge of the of the bounding

size

.

The following example would result in a

CircleComponent

that defines a circle with a radius of 40 (a diameter of 80).

void

main

()

{

CircleComponent

.

relative

(

0.8

,

size:

Vector2

.

all

(

100

));

}

IsometricTileMapComponent

¶

This component allows you to render an isometric map based on a cartesian matrix of blocks and an isome

A simple example on how to use it:

// Creates a tileset, the block ids are automatically assigned sequentially

// starting at 0, from left to right and then top to bottom.

final

tilesetImage

=

```
await
images
.
load
(
'tileset.png'
);
final
tileset
=
IsometricTileset
(
tilesetImage
,
32
);
// Each element is a block id, -1 means nothing
final
matrix
=
[[
0
,
1
,
0
```

],

[

1

,

0

,

0

],

[

1

,

1

,

1

]];

add

(

IsometricTileMapComponent

(

tileset

,

matrix

));

It also provides methods for converting coordinates so you can handle clicks, hovers, render entities on top

You can also specify the

tileHeight

, which is the vertical distance between the bottom and top planes of each cuboid in your tile. Basically, it?s

This is an example of how a quarter-length map looks like:

Flame?s Example app contains a more in-depth example, featuring how to parse coordinates to make a se

here

, and a live version can be seen

here

.

NineTileBoxComponent

¶

A Nine Tile Box is a rectangle drawn using a grid sprite.

The grid sprite is a 3x3 grid and with 9 blocks, representing the 4 corners, the 4 sides and the middle.

The corners are drawn at the same size, the sides are stretched on the side direction and the middle is exp

Using this, you can get a box/rectangle that expands well to any sizes. This is useful for making panels, dia

Check the example app

nine_tile_box

for details on how to use it.

CustomPainterComponent

¶

A

CustomPainter

is a Flutter class used with the

CustomPaint

widget to render custom shapes inside a Flutter application.

Flame provides a component that can render a

CustomPainter

called

CustomPainterComponent

, it receives a custom painter and renders it on the game canvas.

This can be used for sharing custom rendering logic between your Flame game, and your Flutter widgets.

Check the example app

custom_painter_component

for details on how to use it.

ComponentsNotifier

¶

Most of the time just accessing children and their attributes is enough to build the logic of your game.

But sometimes, reactivity can help the developer to simplify and write better code, to help with that Flame p

ComponentsNotifier

, which is an implementation of a

ChangeNotifier

that notifies listeners every time a component is added, removed or manually changed.

For example, lets say that we want to show a game over text when the player?s lives reach zero.

To make the component automatically report when new instances are added or removed, the

Notifier

mixin can be applied to the component class:

class

Player

extends

SpriteComponent

with

Notifier

{}

Then to listen to changes on that component the

componentsNotifier

method from

FlameGame

can be used:

class

MyGame

extends

FlameGame

{

int

lives

=

2

;

@override

void

onLoad

()

{

final

playerNotifier

=

componentsNotifier

<

Player

>

()

..

addListener

(()

{

final

player

=

playerNotifier

.

single

;

if

(

player

==

null

)

{

lives

--

;

if

(

lives

==

0

)

{

add

(

GameOverComponent

());

}

else

{

add

(

Player

());

}

}

});

}

}

A

Notifier

component can also manually notify its listeners that something changed. Lets expand the example above

Player

component notify a change manually, example:

class

Player

extends

SpriteComponent

with

Notifier

{

double

health

=

1

;

void

takeHit

()

{

health

-=

.

1

;

if

(

health

==

0

)

{

removeFromParent

();

}

else

if

(

health

<=

.

5

)

{

notifyListeners

();

}

}

}

Then our hud component could look like:

class

Hud

extends

PositionComponent

with

HasGameRef

{

@override

void

onLoad

()

{

final

playerNotifier

=

gameRef

.

componentsNotifier

<

Player

>

()

..

addListener

(()

{

final

player

=

playerNotifier

.

single

;

if

(

player

!=

null

)

{

if

(

player

.

health

<=

.

5

)

{

add

(

BlinkEffect

());

}

}

});

}

}

ComponentsNotifier

s can also come in handy to rebuild widgets when state changes inside a

FlameGame

, to help with that Flame provides a

ComponentsNotifierBuilder

widget.

To see an example of its use check the running example

here

.

ClipComponent

¶

A

ClipComponent

is a component that will clip the canvas to its size and shape. This means that if the component itself or any

ClipComponent

renders outside of the

ClipComponent

?s boundaries, the part that is not inside the area will not be shown.

A

ClipComponent

receives a builder function that should return the

Shape

that will define the clipped area, based on its size.

To make it easier to use that component, there are three factories that offers common shapes:

ClipComponent.rectangle

: Clips the area in the form a rectangle based on its size.

ClipComponent.circle

: Clips the area in the form of a circle based on its size.

ClipComponent.polygon

:  Clips the area in the form of a polygon based on the points received in the constructor.

Check the example app

clip_component

for details on how to use it.

Effects

¶

Flame provides a set of effects that can be applied to a certain type of components, these effects can be us

here

.

Examples of the running effects can be found

here

;

When not using

FlameGame

¶

If you are not using

FlameGame

, don?t forget that all components needs to be updated every time your game updates. This lets componen

For example, the

SpriteAnimationTicker

inside all the

SpriteAnimation

based components needs to tick the animation object to decide which animation frame will be displayed ne

component.update()

when not using

FlameGame

. This also means, if you are implementing your own sprite animation based component, you can directly u

SpriteAnimationTicker

to update the

SpriteAnimation

.

2. Scaffolding

¶

In this section we will use broad strokes to outline the main elements of the game. This includes the main g

KlondikeGame

¶

In Flame universe, the

FlameGame

class is the cornerstone of most games. This class runs the game loop, dispatches events, owns all the co

So, create a new file called

klondike_game.dart

inside the

lib/

folder, and declare the

KlondikeGame

class inside:

```
import
'package:flame/game.dart'
;
class
KlondikeGame
extends
FlameGame
{
@override
Future
<
```

void

>

onLoad

()

async

{

await

Flame

.

images

.

load

(

'klondike-sprites.png'

);

}

}

For now we only declared the

onLoad

method, which is a special handler that is called when the game instance is attached to the Flutter widget t

onLoad

does is that it loads the sprites image into the game; but we will be adding more soon. Any image or other

await

keyword.

I am loading the image into the global

Flame.images

cache here. An alternative approach is to load it into the

Game.images

cache instead, but then it would have been more difficult to access that image from other classes.

Also note that I am

await

ing the image to finish loading before initializing anything else in the game. This is for convenience: it mean

Sprite

klondikeSprite

(

double

x

,

double

y

,

double

width

,

double

height

)

{

return

Sprite

(

Flame

.

images

.

fromCache

(

'klondike-sprites.png'

),

srcPosition:

Vector2

(

x

,

y

),

srcSize:

Vector2

(

width

,

height

),

);

}

This helper function won?t be needed in this chapter, but will be used extensively in the next.

Let?s incorporate this class into the project so that it isn?t orphaned. Open the

main.dart

find the line which says

final

game

=

FlameGame();

and replace the

FlameGame

with

KlondikeGame

. You will need to import the class too. After all is done, the file should look like this:

import

'package:flame/game.dart'

;

import

'package:flutter/widgets.dart'

;

import

'klondike_game.dart'

;

void

main

()

{

final

game

=

KlondikeGame

();

runApp

(

GameWidget

(

game:

game

));

}

Other classes

¶

So far we have the main

KlondikeGame

class, and now we need to create objects that we will add to the game. In Flame these objects are called

components

, and when added to the game they form a ?game component tree?. All entities that exist in the game mus

As we already mentioned in the previous chapter, our game mainly consists of

Card

components. However, since drawing the cards will take some effort, we will defer implementation of that c

For now, let?s create the container classes, as shown on the sketch. These are:

Stock

,

Waste

,

Pile

and

Foundation

. Inside the

lib/

folder create a sub-directory

components

, and then the file

lib/components/stock.dart

. In that file write

import

'package:flame/components.dart'

;

class

Stock

extends

PositionComponent

{

@override

bool

get

debugMode

=>

true

;

}

Here we declare the

Stock

class as a

PositionComponent

(which is a component that has a position and size). We also turn on the debug mode for this class so that

Likewise, create three more classes

Foundation

,

Pile

and

Waste

, each in its corresponding file. For now all four classes will have exactly the same logic inside, we?ll be ad

At this moment the directory structure of your game should look like this:

klondike/ ??assets/ ? ??images/ ? ??klondike-sprites.png ??lib/ ? ??components/ ? ? ??foundatio

Game structure

¶

Once we have some basic components, they need to be added to the game. It is time to make a decision a

There exist multiple approaches here, which differ in their complexity, extendability, and overall philosophy

World

component, together with a

Camera

.

The idea behind this approach is the following: imagine that your game

world

exists independently from the device, that it exists already in our heads, and on the sketch, even though we

All elements that are part of the world will be added to the

World

component, and the

World

component will be then added to the game.

The second part of the overall structure is a

camera

(

CameraComponent

). The purpose of the camera is to be able to look at the world, to make sure that it renders at the right size

Thus, the overall structure of the component tree will look approximately like this:

KlondikeGame ?? World ? ?? Stock ? ?? Waste ? ?? Foundation (×4) ? ?? Pile (×7) ?? CameraC

For this game I?ve been drawing my image assets having in mind the dimension of a single card at 1000×1

cardGap

that can be adjusted later if needed. For simplicity, both the vertical and horizontal inter-card distance will b

cardGap

.

Alright, let?s put all this together and implement our

KlondikeGame

class.

First, we declare several global constants which describe the dimensions of a card and the distance betwe

static

const

double

cardWidth

=

1000.0

;

static

const

double

cardHeight

=

1400.0

;

static

const

double

cardGap

=

175.0

;

static

const

double

cardRadius

=

100.0

;

static

final

Vector2

cardSize

=

Vector2

(

cardWidth

,

cardHeight

);

Next, we will create a

Stock

component, the

Waste

, four

Foundation

s and seven

Pile

s, setting their sizes and positions in the world. The positions are calculated using simple arithmetics. This

onLoad

method, after loading the sprite sheet:

final

stock

=

Stock

()

..

size

=

cardSize

..
position
=
Vector2
(
cardGap
,
cardGap
);
final
waste
=
Waste
()
..
size
=
cardSize
..
position
=
Vector2
(
cardWidth
+
2

*

cardGap

,

cardGap

);

final

foundations

=

List

.

generate

(

4

,

(

i

)

=>

Foundation

()

..

size

=

cardSize

..

position

=

Vector2

((

i

+

3

)

*

(

cardWidth

+

cardGap

)

+

cardGap

,

cardGap

),

);

final

piles

=

List

.

generate

(

7

,

(

i

)

=>

Pile

()

..

size

=

cardSize

..

position

=

Vector2

(

cardGap

+

i

*

(

cardWidth

+

cardGap

),

cardHeight

+

2

*

cardGap

,

),

);

Since Flame version 1.9.0,

FlameGame

sets up default

world

and

camera

objects.

KlondikeGame

is an extension of

FlameGame

, so we can add to that

world

all the components that we just created.

world

.

add

(

stock

);

world

.

add

(

waste

);

world

.

addAll

(

foundations

);

world

.

addAll

(

piles

);

Note

You may be wondering when you need to

await

the result of

add()

, and when you don?t. The short answer is: usually you don?t need to wait, but if you want to, then it won?t

If you check the documentation for

.add()

method, you?ll see that the returned future only waits until the component is finished loading, not until it is a

.add()

if your logic requires that the component is fully loaded before it can proceed. This is not very common.

If you don?t

await

the future from

.add()

, then the component will be added to the game anyways, and in the same amount of time.

Lastly, we use FlameGame?s

camera

object to look at the

world

. Internally, the camera consists of two parts: a

viewport

and a

viewfinder

. The default viewport is

MaxViewport

, which takes up the entire available screen size ? this is exactly what we need for our game, so no need to

We want the entire card layout to be visible on the screen without the need to scroll. In order to accomplish

7*cardWidth

+

8*cardGap

by

4*cardHeight

+

3*cardGap

) to be able to fit into the screen. The

.visibleGameSize

setting ensures that no matter the size of the device, the zoom level will be adjusted such that the specified

The game size calculation is obtained like this: there are 7 cards in the tableau and 6 gaps between them,

7*cardWidth

+

8*cardGap

. Vertically, there are two rows of cards, but in the bottom row we need some extra space to be able to disp

4*cardHeight

+

3*cardGap

.

Next, we specify which part of the world will be in the ?center? of the viewport. In this case I specify that th

[(7*cardWidth

+

8*cardGap)/2,

0]

.

The reason for such choice for the viewfinder?s position and anchor is because of how we want it to respo

camera

.

viewfinder

.

visibleGameSize

=

Vector2

(

cardWidth

*

7

+

cardGap

*

8

,

4

*

cardHeight

+

3

*

cardGap

);

camera

.

viewfinder

.

position

=

Vector2

(

cardWidth

*

3.5

+

cardGap

*

4

,

0

);

camera

.

viewfinder

.

anchor

=

Anchor

.

topCenter

;

If you run the game now, you should see the placeholders for where the various components will be. If you

Run

components/foundation.dart

1

import

'package:flame/components.dart'

;

2

3

class

Foundation

extends

PositionComponent

{

4

@override

5

bool

get

debugMode

=>

true

;

6

}

components/pile.dart

1

import

'package:flame/components.dart'

;

2

3

class

Pile

extends

PositionComponent

{

4

@override

5

bool

get

debugMode

=>

true

;

6

}

components/stock.dart

1

import

'package:flame/components.dart'

;

2

3

class

Stock

extends

PositionComponent

{

4

@override

5

bool

get

debugMode

=>

true

;

6

}

components/waste.dart

1

import

'package:flame/components.dart'

;

2

3

class

Waste

extends

PositionComponent

{

4

@override

5

bool

get

debugMode

=>

true

;

6

}

klondike_game.dart

1

import

'package:flame/components.dart'

;

2

import

'package:flame/flame.dart'

;

3

import

'package:flame/game.dart'

;

4

5

```
import
'components/foundation.dart'
;
6
import
'components/pile.dart'
;
7
import
'components/stock.dart'
;
8
import
'components/waste.dart'
;
9
10
class
KlondikeGame
extends
FlameGame
{
11
static
const
double
```

cardGap

=

175.0

;

12

static

const

double

cardWidth

=

1000.0

;

13

static

const

double

cardHeight

=

1400.0

;

14

static

const

double

cardRadius

=

100.0

;

15

static

final

Vector2

cardSize

=

Vector2

(

cardWidth

,

cardHeight

);

16

17

@override

18

Future

<

void

>

onLoad

()

async

{

```
19
await
Flame
.
images
.
load
(
'klondike-sprites.png'
);
20
21
final
stock
=
Stock
()
22
..
size
=
cardSize
23
..
position
=
```

Vector2

(

cardGap

,

cardGap

);

24

final

waste

=

Waste

()

25

..

size

=

cardSize

26

..

position

=

Vector2

(

cardWidth

+

2

*

cardGap

,

cardGap

);

27

final

foundations

=

List

.

generate

(

28

4

,

29

(

i

)

=>

Foundation

()

30

..

size

=

cardSize

31

..

position

=

32

Vector2

((

i

+

3

)

*

(

cardWidth

+

cardGap

)

+

cardGap

,

cardGap

),

33

);

34

final

piles

=

List

.

generate

(

35

7

,

36

(

i

)

=>

Pile

()

37

..

size

=

cardSize

38

..

position

=

Vector2

(

39

cardGap

+

i

*

(

cardWidth

+

cardGap

),

40

cardHeight

+

2

*

cardGap

,

41

),

42

);

43

44

world

.

add

(

stock

);

45

world

.

add

(

waste

);

46

world

.

addAll

(

foundations

);

47

world

.

addAll

(

piles

);

48

49

camera

.

viewfinder

.

visibleGameSize

=

50

Vector2

(

cardWidth

*

7

+

cardGap

*

8

,

4

*

cardHeight

+

3

*

cardGap

);

51

camera

.

viewfinder

.

position

=

Vector2

(

cardWidth

*

3.5

+

cardGap

*

4

,

0

);

52

camera

.

viewfinder

.

anchor

=

Anchor

.

topCenter

;

53

}

54

}

55

56

Sprite

klondikeSprite

(

double

x

,

double

y

,

double

width

,

double

height

)

{

57

return

Sprite

(

58

Flame

.

images

.

fromCache

(

'klondike-sprites.png'

),

59

srcPosition:

Vector2

(

x

,

y

),

60

srcSize:

Vector2

(

width

,

height

),

61

);

62

}

main.dart

1

import

'package:flame/game.dart'

;

2

import

'package:flutter/widgets.dart'

;

3

4

import

'klondike_game.dart'

;

5

6

void

main

()

{

7

final

game

=

KlondikeGame

();

8

runApp

(

GameWidget

(

game:

game

));

9

}

Code

And this is it with this step ? we?ve created the basic game structure upon which everything else will be bu

Looping Background Music

¶

With the

Bgm

class, you can manage looping of background music tracks with regards to application (or game) lifecycle s

When the application is terminated, or sent to background,

Bgm

will automatically pause the currently playing music track. Similarly, when the application is resumed,

Bgm

will resume the background music. Manually pausing and resuming your tracks is also supported.

For this class to function properly, the observer must be registered by calling the following:

FlameAudio

.

bgm

.

initialize

();

IMPORTANT Note:

The

initialize

function must be called at a point in time where an instance of the

WidgetsBinding

class already exists. Best practice is to put this call inside of your game?s

onLoad

method`.

In cases where you?re done with background music but still want to keep the application/game running, us

dispose

function to remove the observer.

FlameAudio

.

bgm

.

dispose

();

To play a looping background music track, run:

import

'package:flame_audio/flame_audio.dart'

;

FlameAudio

.

bgm

.

play

(

'adventure-track.mp3'

);

You must have an appropriate folder structure and add the files to the

pubspec.yaml

file, as explained in

Flame Audio documentation

.

Caching music files

¶

The

Bgm

class will use the static instance of

FlameAudio

for storing cached music files by default.

So in order to pre-load music, you can use the same recommendations from the

Flame Audio documentation

.

You can optionally create your own

Bgm

instances with different backing

AudioCache

s, if you so desire.

Methods

¶

Play

¶

The

play

function takes in a

String

that should be a path that points to the location of the music file to be played (following the Flame Audio fol

You can pass an additional optional

double

parameter which is the

volume

(defaults to

1.0

).

Examples:

FlameAudio

.

bgm

.

play

(

'music/boss-fight/level-382.mp3'

);

FlameAudio

.

bgm

.

play

(

'music/world-map.mp3'

,

volume:

.

25

);

Stop

¶

To stop a currently playing background music track, just call

stop

.

FlameAudio

.

bgm

.

stop

();

Pause and Resume

¶

To manually pause and resume background music you can use the

pause

and

resume

functions.

FlameAudio.bgm

automatically handles pausing and resuming the currently playing background music track. Manually

pausing

prevents the app/game from auto-resuming when focus is given back to the app/game.

FlameAudio

.

bgm

.

pause

();

FlameAudio

.

bgm

.

resume

();

Components

¶

Components in Oxygen are different than the ones from FCS mainly because instead of containing logic th

PositionComponent

from FCS has. Others are just easy wrappers around certain Flame API functionality, they are often accom

Components can be registered to the world using the

world.registerComponent

method on

OxygenGame

.

PositionComponent

¶

The

PositionComponent

as its name implies is a component that describes the position of an entity. And it is registered to the world

Creating a positioned entity using OxygenGame:

game

.

createEntity

(

position:

Vector2

(

100

,

100

),

size:

// ...

);

Creating a positioned entity using the World:

world

.

createEntity

()

..

add

<

PositionComponent

,

Vector2

>

(

Vector2

(

100

,

100

));

SizeComponent

¶

The

SizeComponent

as its name implies is a component that describes the size of an entity. And it is registered to the world by d

Creating a sized entity using OxygenGame:

```
game
.
createEntity
(
position:
// ...
size:
Vector2
(
50
,
50
),
);
```

Creating a sized entity using the World:

```
world
.
createEntity
()
..
add
<
SizeComponent
```

,

Vector2

>

(

Vector2

(

50

,

50

));

AnchorComponent

¶

The

AnchorComponent

as its name implies is a component that describes the anchor position of an entity. And it is registered to th

This component is especially useful when you are using the

BaseSystem

. But can also be used for your anchoring logic.

Creating an anchored entity using OxygenGame:

game

.

createEntity

(

position:

// ...

size:

// ...

anchor:

Anchor

.

center

,

);

Creating an anchored entity using the World:

world

.

createEntity

()

..

add

<

AnchorComponent

,

Anchor

>

(

Anchor

.

center

);

AngleComponent

¶

The

AngleComponent

as its name implies is a component that describes the angle of an entity and it is registered to the world by

This component is especially useful when you are using the

BaseSystem

. But can also be used for your angle logic.

Creating an angled entity using OxygenGame:

game

.

createEntity

(

position:

// ...

size:

// ...

angle:

1.570796

,

);

Creating an angled entity using the World:

world

.

createEntity

()

..

add

<

AngleComponent

,

double

>

(

1.570796

);

FlipComponent

¶

The

FlipComponent

can be used to flip your rendering on either the X or Y axis. It is registered to the world by default.

This component is especially useful when you are using the

BaseSystem

. But can also be used for your flipping logic.

Creating an entity that is flipped on its X-axis using OxygenGame:

game

.

createEntity

(

position:

// ...

size:

// ...

flipX:

true

);

Creating an entity that is flipped on its X-axis using the World:

world

.

createEntity

()

..

add

<

FlipComponent

,

FlipInit

>

(

FlipInit

(

flipX:

true

));

SpriteComponent

¶

The

SpriteComponent

as its name implies is a component that describes the sprite of an entity and it is registered to the world by

This allows you to assign a Sprite to an Entity.

Creating an entity with a sprite using OxygenGame:

```
game
.
createEntity
(
position:
// ...
size:
// ...
)..
add
<
SpriteComponent
,
SpriteInit
>
(
SpriteInit
(
await
game
.
loadSprite
(
'pizza.png'
)),
```

);

Creating an entity with a sprite using World:

world

.

createEntity

()

..

add

<

SpriteComponent

,

SpriteInit

>

(

SpriteInit

(

await

game

.

loadSprite

(

'pizza.png'

)),

);

TextComponent

¶

The

TextComponent

as its name implies is a component that adds a text component to an entity. And it is registered to the worl

This allows you to add text to your entity, combined with the

PositionComponent

you can use it as a text entity.

Creating an entity with text using OxygenGame:

game

.

createEntity

(

position:

// ...

size:

// ...

)..

add

<

TextComponent

,

TextInit

>

(

TextInit

(

'Your text'

,

config:

const

TextPaintConfig

(),

),

);

Creating an entity with text using World:

world

.

createEntity

()

..

add

<

TextComponent

,

TextInit

>

(

TextInit

(

'Your text'

,

config:

const

TextPaintConfig

(),

),

);

ParticleComponent

¶

The

ParticleComponent

wraps a

Particle

from Flame. Combined with the

ParticleSystem

you can easily add particles to your game without having to worry about how to render a particle or when/h

Creating an entity with a particle using OxygenGame:

game

.

createEntity

(

position:

// ...

size:

// ...

)..

add

<

ParticleComponent

,

Particle

>

(

// Your Particle.

);

Creating an entity with a particle using World:

world

.

createEntity

()

..

add

<

ParticleComponent

,

Particle

>

(

// Your Particle.

);

Miscellaneous functions

¶

if(condition, then, else)

¶

This function implements the ternary-if condition, it is equivalent to the

?:

operator in Dart.

The function evaluates its

condition

(which must be a boolean), and then returns either the value of

then

if the condition was

true

, or the value of

else

if the condition was

false

. The types of arguments

then

and

else

must be the same.

Note: Only one of the

then

/

else

values will be evaluated, depending on the

condition

. This may be important in cases when evaluating those expressions may produce a side-effect.

title

:

Birth

---

Doctor

:

Congratulations, you have a

{

if

(

$gender

==

"

m

"

,

"

boy

"

,

"

girl

"

)

}

!

===

plural(x, words?)

¶

Returns the correct plural form depending on the value of variable

x

.

This function is locale-dependent, and its implementation and signature changes depending on the

locale

property in the

YarnProject

. In all cases, the first argument

x

must be numeric, while all other arguments should be strings.

The purpose of this function is to form correct plural phrases, according to the rules of the current language

{$n}

items

, where

$n

is a variable. If you simply plug in the value of the variable like that, you?ll end up getting phrases like ?23 i

plural()

function can be used, which will select the correct plural form of the word ?item?:

I have

{

plural

(

$n

,

"

% item

"

)}

.

In English locale (

en

), the function

plural()

takes either 1 or 2

word

s after the numeral

$x

. The first word is the singular form, and the second is the plural. The second word can be omitted if the sin

-s

or

-es

. For example:

// Here "foot" is an irregular noun, so its plural form must be specified

// explicitly. At the same time, "inch" is regular, and the function

// plural() will know to add "es" to make its plural form.

The distance is

{

plural

(

$ft

,

"

% foot

"

,

"

% feet

"

)}

and

{

plural

(

$in

,

"

% inch

"

)}

.

In locales other than English, the number of plural words can be anywhere from 1 to 3. Usually, the first wo

uk

) the function

plural()

requires 3 words: the singular form, the ?few? plural form, and the ?many? plural form:

// Assuming locale == 'uk'

? ???? ?

{

plural

(

$coins

,

"

% ??????

"

,

"

% ??????

"

,

"

% ?????

"

)}

.

// Produces phrases like this:

//   ? ???? ? 21 ??????

//   ? ???? ? 23 ??????

//  ? ???? ? 25 ?????

Note that in all examples above the words contain the

%

sign. This is used as a placeholder where the numeral itself should be placed. It is allowed for some (or all)

words

to not contain the

%

sign.

visit_count(node)

¶

Returns the number of times that the

node

was visited.

A node is considered ?visited? if the dialogue enters and then exits that node. The node can be exited eith

<<stop>>

command. However, if a runtime exception occurs while running the node, then the visit will not count.

The

node

argument must be a string, and it must contain a valid node name. If a node with the given name does not

title

:

LuckyWheel

---

<<

if

visit_count

(

"

LuckyWheel

"

)

<

5

>>

Clown

:

Would you like to spin a wheel and get fabulous prizes?

->

I sure do!

<<

jump

SpinLuckyWheel

>>

->

I don't talk to strangers...

<<

stop

>>

<<

else

>>

Clown

:

Sorry kid, we're all out of prizes for now.

<<

endif

>>

===

See also

visited(node)

visited(node)

¶

Returns

true

if the node with the given title was visited, and

false

otherwise.

For a node to be considered ?visited?, the dialogue must enter and then exit the node at least once. For ex

visited("X")

will return

false

during the first run of this node, and

true

upon all subsequent runs.

The

node

argument must be a string, and it must contain a valid node name. If a node with the given name does not

title

:

MerchantDialogue

---

<<

if

not

visited

(

"

MerchantDialogue

"

)>>

// This part of the dialogue will run only during the first interaction

// with the merchant.

Merchant

:

Greetings! My name is Linn.

Merchant

:

I offer exquisite wares for the most fastidious customers!

Player

:

Hi. I'm Bob. I like stuff.

<<

endif

>>

...

===

See also

visit_count(node)

flame_svg

¶

Overview

Installation

How to use flame_svg

DialogueView

¶

class

DialogueView

The

DialogueView

class is the main mechanism for integrating Jenny with a game engine. This class describes how

line

s and

option

s are presented to the user.

There are two ways to use this class:

Extending DialogueView

Adding DialogueView as a mixin

In both cases you will need to create concrete implementations of the abstract event handler methods in or

DialogueView

objects will then be passed to a

DialogueRunner

, which will orchestrate the dialogue?s progression.

The class defines a number of ?event handler? methods, which can be overridden in subclasses in order to

Most of the event handler methods return

FutureOr

, which means they can be implemented either synchronously or asynchronously. In the latter case the dial

Properties

¶

dialogueRunner

?

DialogueRunner?

The owner of this

DialogueView

. This property will be

null

when the dialogue view hasn?t been attached to any

DialogueRunner

yet.

This property can be used in order to access the parent

YarnProject

, or to send signals into the sibling

DialogueView

s.

Methods

¶

onDialogueStart

(

)

?

FutureOr<void>

Called before the start of a new dialogue, i.e. before any lines, options, or commands are delivered.

This method is a good place to prepare the game?s UI, such as fading in/ animating dialogue panels, or loa

onDialogueFinish

(

)

?

FutureOr<void>

Called when the dialogue has ended.

This method can be used to clean up any of the dialogue UI. The returned future will be awaited before the

onNodeStart

(

Node

node

)

?

FutureOr<void>

Called when the dialogue enters a new

node

.

This will be called immediately after the

onDialogueStart

, and then possibly several times more over the course of the dialogue if it jumps to other nodes. This meth

node

?s properties or metadata.

If this method returns a future, then the dialogue runner will wait for it to complete before proceeding with th

onNodeFinish

(

Node

node

)

?

FutureOr<void>

Called when the dialogue exits the

node

.

For example, during a

<<jump>>

this callback will be called with the current node, and then

onNodeStart

will be called with the new node. Similarly, the command

<<stop>>

will trigger this callback too. At the same time, during

<<visit>>

this callback will not be invoked.

This callback can be used to clean up any preparations that were performed in

onNodeStart

.

onLineStart

(

DialogueLine

line

)

?

FutureOr<bool>

Called when the next dialogue

line

should be presented to the user.

The

DialogueView

may decide to present the

line

in whatever way it wants, or to not present the line at all. For example, the dialogue view may: augment the

Some of these methods of delivery can be considered ?primary?, while others are ?auxiliary?. A ?primary?

DialogueView

should return

true

, while all others

false

(especially if a dialogue view ignores the line completely). This is used as a robustness check: if none of th

true

, then a

DialogueError

will be thrown because the line was not shown to the user in a meaningful way.

If this method returns a future, then the dialogue runner will wait for that future to complete before advancin

DialogueView

s return such futures, then the dialogue runner will wait for all of them to complete before proceeding.

Returning a future is quite common for non-trivial

DialogueView

s. After all, if this method were to return immediately, the dialogue runner would immediately advance to th

Completer

-based future that completes based on some user action such as clicking a button or pressing a keyboard k

Note that this method is supposed to only

show

the line to the player, so do not try to hide it at the end ? for that, there is a dedicated method

onLineFinish

.

Also, given that this method may take a significant amount of time, there are two additional methods that m

onLineSignal

and

onLineStop

.

onLineSignal

(

DialogueLine

line

,

dynamic

signal

)

Called when the dialogue runner sends a

signal

to all dialogue views.

The signal will be sent to all views, regardless of whether they have finished running

onLineStart

or not. The interpretation of the signal and the appropriate response is up to each dialogue view.

For example, one possible scenario would be to speed up a typewriter effect and reveal the text immediate

onLineStop

(

DialogueLine

line

)

?

FutureOr<void>

Called when the game demands that the

line

finished presenting as soon as possible.

By itself, the dialogue runner will never call this method. However, it may be invoked as a result of an expli

This method returns a future that will be awaited before continuing to the next line of the dialogue. At the sa

onLineStart

call will be discarded and will no longer be awaited. The

onLineFinish

method will not be called either.

onLineFinish

(

DialogueLine

line

)

?

FutureOr<void>

Called when the

line

has finished presenting in all dialogue views.

Some dialogue views may need to clear their display when this event happens, or make some other prepar

onChoiceStart

(

DialogueChoice

choice

)

?

FutureOr<int?>

Called when the dialogue arrives at an option set, and the player must now make a choice on how to proce

null

(possibly in a

Future

).

The future returned by this method should deliver an integer value of the index of the option that was selec

choice

list, and the indicated option must not be marked as ?unavailable?. If these conditions are violated, an exc

onChoiceFinish

(

DialogueOption

option

)

?

FutureOr<void>

Called when the choice has been made, and the

option

was selected.

The

option

will be the one returned from the

onChoiceStart

method by one of the dialogue views.

onCommand

(

UserDefinedCommand

command

)

?

FutureOr<void>

Called when the dialogue encounters a

user-defined command

.

This method is invoked immediately after the command itself is executed, but before the result of the execu

at the same time

.

In cases when the command?s effect occurs within the game, implementing this method may not be neces

## Joints

¶

Joints are used to connect two different bodies together in various ways. They help to simulate interactions

One

Body

in a joint may be of type

BodyType.static

. Joints between

BodyType.static

and/or

BodyType.kinematic

are allowed, but have no effect and use some processing time.

To construct a

Joint

, you need to create a corresponding subclass of

JointDef

and initialize it with its parameters.

To register a

Joint

use

world.createJoint

and later use

world.destroyJoint

when you want to remove it.

### Built-in joints

¶

Currently, Forge2D supports the following joints:

ConstantVolumeJoint

DistanceJoint

FrictionJoint

GearJoint

MotorJoint

MouseJoint

PrismaticJoint

PulleyJoint

RevoluteJoint

RopeJoint

WeldJoint

WheelJoint

ConstantVolumeJoint

¶

This type of joint connects a group of bodies together and maintains a constant volume within them. Essen

DistanceJoint

s, that connects all bodies one after another.

It can for example be useful when simulating ?soft-bodies?.

final

constantVolumeJoint

=

ConstantVolumeJointDef

()

..

frequencyHz

=

10

..

dampingRatio

=

0.8

;

bodies

.

forEach

((

body

)

{

constantVolumeJoint

.

addBody

(

body

);

});

world

.

createJoint

(

ConstantVolumeJoint

(

world

,

constantVolumeJoint

));

Run

constant_volume_joint.dart

1

import

'dart:math'

;

2

3

import

'package:examples/stories/bridge_libraries/flame_forge2d/utils/balls.dart'

;

4

import

'package:examples/stories/bridge_libraries/flame_forge2d/utils/boundaries.dart'

;

5

import

'package:flame/components.dart'

;

6

import

'package:flame/events.dart'

;

7

import

'package:flame_forge2d/flame_forge2d.dart'

;

8

9

class

ConstantVolumeJointExample

extends

Forge2DGame

{

10

static

const

description

=

'''

11

This example shows how to use a `ConstantVolumeJoint`. Tap the screen to add

12

a bunch off balls, that maintain a constant volume within them.

13

'''

;

14

15

ConstantVolumeJointExample

()

:

super

(

world:

SpriteBodyWorld

());

16

}

17

18

class

SpriteBodyWorld

extends

Forge2DWorld

19

with

TapCallbacks

,

HasGameReference

<

Forge2DGame

>

```
{
20
@override
21
Future
<
void
>
onLoad
()
async
{
22
super
.
onLoad
();
23
addAll
(
createBoundaries
(
game
));
24
}
```

```
25
26
@override
27
Future
<
void
>
onTapDown
(
TapDownEvent
info
)
async
{
28
super
.
onTapDown
(
info
);
29
final
center
=
```

info

.

localPosition

;

30

31

const

numPieces

=

20

;

32

const

radius

=

5.0

;

33

final

balls

=

<

Ball

>

[];

34

35

for

(

var

i

=

0

;

i

<

numPieces

;

i

++

)

{

36

final

x

=

radius

*

cos

(

2

*

pi

*

(

i

/

numPieces

));

37

final

y

=

radius

*

sin

(

2

*

pi

*

(

i

/

numPieces

));

38

39

final

ball

=

Ball

(

Vector2

(

x

+

center

.

x

,

y

+

center

.

y

),

radius:

0.5

);

40

41

add

(

ball

);

42

balls

.

add

(

ball

);

43

}

44

45

await

Future

.

wait

(

balls

.

map

((

e

)

=>

e

.

loaded

));

46

47

final

constantVolumeJoint

=

ConstantVolumeJointDef

()

48

..

frequencyHz

=

10

49

..

dampingRatio

=

0.8

;

50

51

balls

.

forEach

((

ball

)

{

52

constantVolumeJoint

.

addBody

(

ball

.

body

);

53

});

54

55

createJoint

(

56

ConstantVolumeJoint

(

57

physicsWorld

,

58

constantVolumeJoint

,

59

),

60

);

61

}

62

}

Code

ConstantVolumeJointDef

requires at least 3 bodies to be added using the

addBody

method. It also has two optional parameters:

frequencyHz

: This parameter sets the frequency of oscillation of the joint. If it is not set to 0, the higher the value, the les

DistantJoint

s are.

dampingRatio

: This parameter defines how quickly the oscillation comes to rest. It ranges from 0 to 1, where 0 means no

DistanceJoint

¶

A

DistanceJoint

constrains two points on two bodies to remain at a fixed distance from each other.

You can view this as a massless, rigid rod.

final

distanceJointDef

=

DistanceJointDef

()

..

initialize

(

firstBody

,

secondBody

,

firstBody

.

worldCenter

,

secondBody

.

worldCenter

)

..

length

=

10

..

frequencyHz

=

3

..

dampingRatio

=

0.2

;

world

.

createJoint

(

DistanceJoint

(

distanceJointDef

));

Run

distance_joint.dart

1

import

'package:examples/stories/bridge_libraries/flame_forge2d/utils/balls.dart'

;

2

import

'package:examples/stories/bridge_libraries/flame_forge2d/utils/boundaries.dart'

;

3

import

'package:flame/components.dart'

;

4

import

'package:flame/events.dart'

;

5

import

'package:flame_forge2d/flame_forge2d.dart'

;

6

7

class

DistanceJointExample

extends

Forge2DGame

{

8

static

const

description

=

'''

9

This example shows how to use a `DistanceJoint`. Tap the screen to add a

10

pair of balls joined with a `DistanceJoint`.

11

'''

;

12

13

DistanceJointExample

()

:

super

(

world:

DistanceJointWorld

());

14

}

15

16

class

DistanceJointWorld

extends

Forge2DWorld

17

with

TapCallbacks

,

HasGameReference

<

Forge2DGame

>

{

18

@override

19

Future

<

void

>

onLoad

()

async

{

20

super

.

onLoad

();

21

addAll

(

```
createBoundaries
(
game
));
22
}
23
24
@override
25
Future
<
void
>
onTapDown
(
TapDownEvent
info
)
async
{
26
super
.
onTapDown
(
```

info

);

27

final

tap

=

info

.

localPosition

;

28

29

final

first

=

Ball

(

tap

);

30

final

second

=

Ball

(

Vector2

(

tap

.

x

+

3

,

tap

.

y

+

3

));

31

addAll

([

first

,

second

]);

32

33

await

Future

.

wait

([

first

.

loaded

,

second

.

loaded

]);

34

35

final

distanceJointDef

=

DistanceJointDef

()

36

..

initialize

(

37

first

.

body

,

38

second

.

body

,

39

first

.

body

.

worldCenter

,

40

second

.

center

,

41

)

42

..

length

=

10

43

..

frequencyHz

=

3

44

..

dampingRatio

=

0.2

;

45

46

createJoint

(

DistanceJoint

(

distanceJointDef

));

47

}

48

}

Code

To create a

DistanceJointDef

, you can use the

initialize

method, which requires two bodies and a world anchor point on each body. The definition uses local ancho

The

DistanceJointDef

has three optional parameters that you can set:

length

: This parameter determines the distance between the two anchor points and must be greater than 0. The d

frequencyHz

: This parameter sets the frequency of oscillation of the joint. If it is not set to 0, the higher the value, the les

dampingRatio

: This parameter defines how quickly the oscillation comes to rest. It ranges from 0 to 1, where 0 means no

Warning

Do not use a zero or short length.

FrictionJoint

¶

A

FrictionJoint

is used for simulating friction in a top-down game. It provides 2D translational friction and angular friction.

The

FrictionJoint

isn?t related to the friction that occurs when two shapes collide in the x-y plane of the screen. Instead, it?s

The

initialize

method of the

FrictionJointDef

method requires two bodies that will have friction force applied to them, and an anchor.

The third parameter is the

anchor

point in the world coordinates where the friction force will be applied. In most cases, it would be the center

anchor

point to a specific location on one or both of the bodies.

final

frictionJointDef

=

FrictionJointDef

()

..

initialize

(

ballBody

,

floorBody

,

ballBody

.

worldCenter

)

..

maxForce

=

50

..

maxTorque

=

50

;

world

.

createJoint

(

FrictionJoint

(

frictionJointDef

));

Run

friction_joint.dart

1

import

'package:examples/stories/bridge_libraries/flame_forge2d/utils/balls.dart'

;

2

import

'package:examples/stories/bridge_libraries/flame_forge2d/utils/boundaries.dart'

;

3

import

'package:flame/components.dart'

;

4

import

'package:flame/events.dart'

;

5

import

'package:flame_forge2d/flame_forge2d.dart'

;

6

7

class

FrictionJointExample

extends

Forge2DGame

{

8

static

const

description

=

'''

9

This example shows how to use a `FrictionJoint`. Tap the screen to move the

10

ball around and observe it slows down due to the friction force.

11

'''

;

12

13

FrictionJointExample

()

14

:

super

(

gravity:

Vector2

.

all

(

0

),

world:

FrictionJointWorld

());

15

}

16

17

class

FrictionJointWorld

extends

Forge2DWorld

18

with

TapCallbacks

,

HasGameReference

<

Forge2DGame

>

{

19

late

Wall

border

;

20

late

Ball

ball

;

21

22

@override

23

Future

<

void

>

onLoad

()

async

{

24

super

.

onLoad

();

25

final

boundaries

=

createBoundaries

(

game

);

26

border

=

boundaries

.

first

;

27

addAll

(

boundaries

);

28

29

ball

=

Ball

(

Vector2

.

zero

(),

radius:

3

);

30

add

(

ball

);

31

32

await

Future

.

wait

([

ball

.

loaded

,

border

.

loaded

]);

33

34

createFrictionJoint

(

ball

.

body

,

border

.

body

);

35

}

36

37

@override

38

Future

<

void

>

onTapDown

(

TapDownEvent

info

)

async

{

39

super

.

onTapDown

(

info

);

40

ball

.

body

.

applyLinearImpulse

(

Vector2

.

random

()

*

5000

);

41

}

42

43

void

createFrictionJoint

(

Body

first

,

Body

second

)

{

44

final

frictionJointDef

=

FrictionJointDef

()

45

..

initialize

(

first

,

second

,

first

.

worldCenter

)

46

..

collideConnected

=

true

47

..

maxForce

=

500

48

..

maxTorque

=

500

;

49

50

createJoint

(

FrictionJoint

(

frictionJointDef

));

51

}

52

}

Code

When creating a

FrictionJoint

, simulated friction can be applied via maximum force and torque values:

maxForce

: the maximum translational friction which applied to the joined body. A higher value

simulates higher friction.

maxTorque

: the maximum angular friction which may be applied to the joined body. A higher value

simulates higher friction.

In other words, the former simulates the friction, when the body is sliding and the latter simulates the frictio

GearJoint

¶

The

GearJoint

is used to connect two joints together. Joints are required to be a

RevoluteJoint

or a

PrismaticJoint

in any combination.

Warning

The connected joints must attach a dynamic body to a static body. The static body is expected to be a body

final

gearJointDef

=

GearJointDef

()

..

bodyA

=

firstJoint

.

bodyA

..

bodyB

=

secondJoint

.

bodyA

..

joint1

=

firstJoint

..

joint2

=

secondJoint

..

ratio

=

1

;

world

.

createJoint

(

GearJoint

(

gearJointDef

));

Run

gear_joint.dart

1

import

'dart:ui'

;

2

3

import

'package:examples/stories/bridge_libraries/flame_forge2d/utils/balls.dart'

;

4

import

'package:examples/stories/bridge_libraries/flame_forge2d/utils/boxes.dart'

;

5

import

'package:flame/components.dart'

;

6

import

'package:flame_forge2d/flame_forge2d.dart'

;

7

8

class

GearJointExample

extends

Forge2DGame

{

9

static

const

description

=

'''

10

This example shows how to use a `GearJoint`.

11

12

Drag the box along the specified axis and observe gears respond to the

13

translation.

14

'''

;

15

16

GearJointExample

()

:

super

(

world:

GearJointWorld

());

17

}

18

19

class

GearJointWorld

extends

Forge2DWorld

with

HasGameReference

<

Forge2DGame

>

{

20

late

PrismaticJoint

prismaticJoint

;

21

Vector2

boxAnchor

=

Vector2

.

zero

();

22

23

double

boxWidth

=

2

;

24

double

ball1Radius

=

4

;

25

double

ball2Radius

=

2

;

26

27

@override

28

Future

<

void

>

onLoad

()

async

{

29

super

.

onLoad

();

30

31

final

box

=

32

DraggableBox

(

startPosition:

boxAnchor

,

width:

boxWidth

,

height:

20

);

33

add

(

box

);

34

35

final

ball1Anchor

=

boxAnchor

-

Vector2

(

boxWidth

/

2

+

ball1Radius

,

0

);

36

final

ball1

=

Ball

(

ball1Anchor

,

radius:

ball1Radius

);

37

add

(

ball1

);

38

39

final

ball2Anchor

=

ball1Anchor

-

Vector2

(

ball1Radius

+

ball2Radius

,

0

);

40

final

ball2

=

Ball

(

ball2Anchor

,

radius:

ball2Radius

);

41

add

(

ball2

);

42

43

await

Future

.

wait

([

box

.

loaded

,

ball1

.

loaded

,

ball2

.

loaded

]);

44

45

prismaticJoint

=

createPrismaticJoint

(

box

.

body

,

boxAnchor

);

46

final

revoluteJoint1

=

createRevoluteJoint

(

ball1

.

body

,

ball1Anchor

);

47

final

revoluteJoint2

=

createRevoluteJoint

(

ball2

.

body

,

ball2Anchor

);

48

```
49
createGearJoint
(
prismaticJoint
,
revoluteJoint1
,
1
);
50
createGearJoint
(
revoluteJoint1
,
revoluteJoint2
,
0.5
);
51
add
(
JointRenderer
(
joint:
prismaticJoint
,
```

anchor:

boxAnchor

));

52

}

53

54

PrismaticJoint

createPrismaticJoint

(

Body

box

,

Vector2

anchor

)

{

55

final

groundBody

=

createBody

(

BodyDef

());

56

57

final

prismaticJointDef

=

PrismaticJointDef

()

58

..

initialize

(

59

groundBody

,

60

box

,

61

anchor

,

62

Vector2

(

0

,

1

),

63

)

64

..

enableLimit

=

true

65

..

lowerTranslation

=

-

10

66

..

upperTranslation

=

10

;

67

68

final

joint

=

PrismaticJoint

(

prismaticJointDef

);

69

createJoint

(

joint

);

70

return

joint

;

71

}

72

73

RevoluteJoint

createRevoluteJoint

(

Body

ball

,

Vector2

anchor

)

{

74

final

groundBody

=

createBody

(

BodyDef

());

75

76

final

revoluteJointDef

=

RevoluteJointDef

()

77

..

initialize

(

78

groundBody

,

79

ball

,

80

anchor

,

81

);

82

83

final

joint

=

RevoluteJoint

(

revoluteJointDef

);

84

createJoint

(

joint

);

85

return

joint

;

86

}

87

88

void

createGearJoint

(

Joint

first

,

Joint

second

,

double

gearRatio

)

{

89

final

gearJointDef

=

GearJointDef

()

90

..

bodyA

=

first

.

bodyA

91

..

bodyB

=

second

.

bodyA

92

..

joint1

=

first

93

..

joint2

=

second

94

..

ratio

=

gearRatio

;

95

96

final

joint

=

GearJoint

(

gearJointDef

);

97

createJoint

(

joint

);

98

}

99

}

100

101

class

JointRenderer

extends

Component

{

102

JointRenderer

({

required

this

.

joint

,

required

this

.

anchor

});

103

104

final

PrismaticJoint

joint

;

105

final

Vector2

anchor

;

106

final

Vector2

p1

=

Vector2

.

zero

();

107

final

Vector2

p2

=

Vector2

.

zero

();

108

109

@override

110

void

render

(

Canvas

canvas

)

{

111

p1

112

..

setFrom

(

joint

.

getLocalAxisA

())

113

..

scale

(

joint

.

getLowerLimit

())

114

..

add

(

anchor

);

115

p2

116

..

setFrom

(

joint

.

getLocalAxisA

())

117

..

scale

(

joint

.

getUpperLimit

())

118

..

add

(

anchor

);

119

120

canvas

.

drawLine

(

p1

.

toOffset

(),

p2

.

toOffset

(),

debugPaint

);

121

}

122

}

Code

joint1

,

joint2

: Connected revolute or prismatic joints

bodyA

,

bodyB

: Any bodies form the connected joints, as long as they are not the same body.

ratio

: Gear ratio

Similarly to

PulleyJoint

, you can specify a gear ratio to bind the motions together:

coordinate1 + ratio * coordinate2 == constant

The ratio can be negative or positive. If one joint is a

RevoluteJoint

and the other joint is a

PrismaticJoint

, then the ratio will have units of length or units of 1/length.

Since the

GearJoint

depends on two other joints, if these are destroyed, the

GearJoint

needs to be destroyed as well.

Warning

Manually destroy the

GearJoint

if joint1 or joint2 is destroyed

MotorJoint

¶

A

MotorJoint

is used to control the relative motion between two bodies. A typical usage is to control the movement of a d

A

MotorJoint

lets you control the motion of a body by specifying target position and rotation offsets. You can set the max

final

motorJointDef

=

MotorJointDef

()

..

initialize

(

first

,

second

)

..

maxTorque

=

1000

..

maxForce

=

1000

..

correctionFactor

=

0.1

;

world

.

createJoint

(

MotorJoint

(

motorJointDef

));

Run

motor_joint.dart

1

import

'dart:ui'

;

2

3

import

'package:examples/stories/bridge_libraries/flame_forge2d/utils/balls.dart'

;

4

import

'package:examples/stories/bridge_libraries/flame_forge2d/utils/boxes.dart'

;

5

import

'package:flame/components.dart'

;

6

import

'package:flame/events.dart'

;

7

import

'package:flame_forge2d/flame_forge2d.dart'

;

8

9

class

MotorJointExample

extends

Forge2DGame

{

10

static

const

description

=

'''

11

This example shows how to use a `MotorJoint`. The ball spins around the

12

center point. Tap the screen to change the direction.

13

'''

;

14

15

MotorJointExample

()

16

:

super

(

gravity:

Vector2

.

zero

(),

world:

MotorJointWorld

());

17

}

18

19

class

MotorJointWorld

extends

Forge2DWorld

with

TapCallbacks

{

20

late

Ball

ball

;

21

late

MotorJoint

joint

;

22

final

motorSpeed

=

1

;

23

24

bool

clockWise

=

true

;

25

26

@override

27

Future

<

void

>

onLoad

()

async

{

28

super

.

onLoad

();

29

30

final

box

=

Box

(

31

startPosition:

Vector2

.

zero

(),

32

width:

2

,

33

height:

1

,

34

bodyType:

BodyType

.

static

,

35

);

36

add

(

box

);

37

38

ball

=

Ball

(

Vector2

(

0

,

-

5

));

39

add

(

ball

);

40

41

await

Future

.

wait

([

ball

.

loaded

,

box

.

loaded

]);

42

43

joint

=

createMotorJoint

(

ball

.

body

,

box

.

body

);

44

add

(

JointRenderer

(

joint:

joint

));

45

```
}
46
47
@override
48
void
onTapDown
(
TapDownEvent
info
)
{
49
super
.
onTapDown
(
info
);
50
clockWise
=
!
clockWise
;
51
```

}

52

53

MotorJoint

createMotorJoint

(

Body

first

,

Body

second

)

{

54

final

motorJointDef

=

MotorJointDef

()

55

..

initialize

(

first

,

second

)

56

..

maxForce

=

1000

57

..

maxTorque

=

1000

58

..

correctionFactor

=

0.1

;

59

60

final

joint

=

MotorJoint

(

motorJointDef

);

61

createJoint

(

joint

);

62

return

joint

;

63

}

64

65

final

linearOffset

=

Vector2

.

zero

();

66

67

@override

68

void

update

(

double

dt

)

{

69

super

.

update

(

dt

);

70

71

var

deltaOffset

=

motorSpeed

*

dt

;

72

if

(

clockWise

)

{

73

deltaOffset

=

-

deltaOffset

;

74

}

75

76

final

linearOffsetX

=

joint

.

getLinearOffset

().

x

+

deltaOffset

;

77

final

linearOffsetY

=

joint

.

getLinearOffset

().

y

+

deltaOffset

;

78

linearOffset

.

setValues

(

linearOffsetX

,

linearOffsetY

);

79

final

angularOffset

=

joint

.

getAngularOffset

()

+

deltaOffset

;

80

81

joint

.

setLinearOffset

(

linearOffset

);

82

joint

.

setAngularOffset

(

angularOffset

);

83

}

84

}

85

86

class

JointRenderer

extends

Component

{

87

JointRenderer

({

required

this

.

joint

});

88

89

final

MotorJoint

joint

;

90

91

@override

92

void

render

(

Canvas

canvas

)

```
{
93
canvas
.
drawLine
(
94
joint
.
anchorA
.
toOffset
(),
95
joint
.
anchorB
.
toOffset
(),
96
debugPaint
,
97
);
98
```

}

99

}

Code

A

MotorJointDef

has three optional parameters:

maxForce

: the maximum translational force which will be applied to the joined body to reach the target position.

maxTorque

: the maximum angular force which will be applied to the joined body to reach the target rotation.

correctionFactor

: position correction factor in range [0, 1]. It adjusts the joint?s response to deviation from target position. A

The linear and angular offsets are the target distance and angle that the bodies should achieve relative to e

setLinearOffset(Vector2)

and

setLinearOffset(double)

methods of the

MotorJoint

to set the desired relative translation and rotate between the bodies.

For example, this code increments the angular offset of the joint every update cycle, causing the body to ro

@override

void

update

(

double

dt

)

{

super

.

update

(

dt

);

final

angularOffset

=

joint

.

getAngularOffset

()

+

motorSpeed

*

dt

;

joint

.

setAngularOffset

(

angularOffset

);

}

MouseJoint

¶

The

MouseJoint

is used to manipulate bodies with the mouse. It attempts to drive a point on a body towards the current pos

The

MouseJoint

definition has a target point, maximum force, frequency, and damping ratio. The target point initially coincid

Warning

Many users have tried to adapt the mouse joint for game play. Users often want to achieve precise position

final

mouseJointDef

=

MouseJointDef

()

..

maxForce

=

3000

*

ballBody

.

mass

*

10

..

dampingRatio

=

1

..

frequencyHz

=

5

..

target

.

setFrom

(

ballBody

.

position

)

..

collideConnected

=

false

..

bodyA

=

groundBody

..

bodyB

=

ballBody

;

mouseJoint

=

MouseJoint

(

mouseJointDef

);

world

.

createJoint

(

mouseJoint

);

}

Run

mouse_joint.dart

1

import

'package:examples/stories/bridge_libraries/flame_forge2d/revolute_joint_with_motor_example.dart'

;

2

import

'package:examples/stories/bridge_libraries/flame_forge2d/utils/balls.dart'

;

3

import

'package:examples/stories/bridge_libraries/flame_forge2d/utils/boundaries.dart'

;

4

import

'package:flame/components.dart'

;

5

import

'package:flame/events.dart'

;

6

import

'package:flame_forge2d/flame_forge2d.dart'

;

7

8

class

MouseJointExample

extends

Forge2DGame

{

9

static

const

description

=

'''

10

In this example we use a `MouseJoint` to make the ball follow the mouse

11

when you drag it around.

12

'''

;

13

14

MouseJointExample

()

15

:

super

(

gravity:

Vector2

(

0

,

10.0

),

world:

MouseJointWorld

());

16

}

17

18

class

MouseJointWorld

extends

Forge2DWorld

19

with

DragCallbacks

,

HasGameReference

<

Forge2DGame

>

{

20

late

Ball

ball

;

21

late

Body

groundBody

;

22

MouseJoint

?

mouseJoint

;

23

24

@override

25

Future

<

void

>

onLoad

()

async

{

26

super

.

onLoad

();

27

final

boundaries

=

createBoundaries

(

game

);

28

addAll

(

boundaries

);

29

30

final

center

=

Vector2

.

zero

();

31

groundBody

=

createBody

(

BodyDef

());

32

ball

=

Ball

(

center

,

radius:

5

);

33

add

(

ball

);

34

add

(

CornerRamp

(

center

));

35
add
(
CornerRamp
(
center
,
isMirrored:
true
));
36
}
37
38
@override
39
void
onDragStart
(
DragStartEvent
info
)
{
40
super
.

onDragStart

(

info

);

41

final

mouseJointDef

=

MouseJointDef

()

42

..

maxForce

=

3000

*

ball

.

body

.

mass

*

10

43

..

dampingRatio

=

0.1

44

..

frequencyHz

=

5

45

..

target

.

setFrom

(

ball

.

body

.

position

)

46

..

collideConnected

=

false

47

..

bodyA

=

groundBody

48

..

bodyB

=

ball

.

body

;

49

50

if

(

mouseJoint

==

null

)

{

51

mouseJoint

=

MouseJoint

(

mouseJointDef

);

52

createJoint

(

mouseJoint

!

);

53

}

54

}

55

56

@override

57

void

onDragUpdate

(

DragUpdateEvent

info

)

{

58

mouseJoint

?

.

```
setTarget
(
info
.
localEndPosition
);
59
}
60
61
@override
62
void
onDragEnd
(
DragEndEvent
info
)
{
63
super
.
onDragEnd
(
info
);
```

64

destroyJoint

(

mouseJoint

!

);

65

mouseJoint

=

null

;

66

}

67

}

Code

maxForce

: This parameter defines the maximum constraint force that can be exerted to move the candidate body. Us

mass

gravity).

dampingRatio

: This parameter defines how quickly the oscillation comes to rest. It ranges from 0 to 1, where 0 means no

frequencyHz

: This parameter defines the response speed of the body, i.e. how quickly it tries to reach the target positio

target

: The initial world target point. This is assumed to coincide with the body anchor initially.

PrismaticJoint

¶

The

PrismaticJoint

provides a single degree of freedom, allowing for a relative translation of two bodies along an axis fixed in l

PrismaticJointDef

requires defining a line of motion using an axis and an anchor point. The definition uses local anchor points

The joint translation is zero when the local anchor points coincide in world space. Using local anchors and

Warning

At least one body should by dynamic with a non-fixed rotation.

The

PrismaticJoint

definition is similar to the

RevoluteJoint

definition, but instead of rotation, it uses translation.

final

prismaticJointDef

=

PrismaticJointDef

()

..

initialize

(

dynamicBody

,

groundBody

,

dynamicBody

.

worldCenter

,

Vector2

(

1

,

0

),

)

Run

prismatic_joint.dart

1

import

'dart:ui'

;

2

3

import

'package:examples/stories/bridge_libraries/flame_forge2d/utils/boxes.dart'

;

4

import

'package:flame/components.dart'

;

5

import

'package:flame_forge2d/flame_forge2d.dart'

;

6

7

class

PrismaticJointExample

extends

Forge2DGame

{

8

static

const

description

=

'''

9

This example shows how to use a `PrismaticJoint`.

10

11

Drag the box along the specified axis, bound between lower and upper limits.

12

Also, there's a motor enabled that's pulling the box to the lower limit.

13

'''

;

14

15

final

Vector2

anchor

=

Vector2

.

zero

();

16

17

@override

18

Future

<

void

>

onLoad

()

async

{

19

super

.

onLoad

();

20

21

final

box

=

DraggableBox

(

startPosition:

anchor

,

width:

6

,

height:

6

);

22

world

.

add

(

box

);

23

await

Future

.

wait

([

box

.

loaded

]);

24

25

final

joint

=

createJoint

(

box

.

body

,

anchor

);

26

world

.

add

(

JointRenderer

(

joint:

joint

,

anchor:

anchor

));

27

}

28

29

PrismaticJoint

createJoint

(

Body

box

,

Vector2

anchor

)

{

30

final

groundBody

=

world

.

createBody

(

BodyDef

());

31

32

final

prismaticJointDef

=

PrismaticJointDef

()

33

..

initialize

(

34

box

,

35

groundBody

,

36

anchor

,

37

Vector2

(

1

,

0

),

38

)

39

..

enableLimit

=

true

40

..

lowerTranslation

=

-

20

41

..

upperTranslation

=

20

42

..

enableMotor

=

true

43

..

motorSpeed

=

1

44

..

maxMotorForce

=

100

;

45

46

final

joint

=

PrismaticJoint

(

prismaticJointDef

);

47
world
.
createJoint
(
joint
);
48
return
joint
;
49
}
50
}
51
52
class
JointRenderer
extends
Component
{
53
JointRenderer
({
required

this

.

joint

,

required

this

.

anchor

});

54

55

final

PrismaticJoint

joint

;

56

final

Vector2

anchor

;

57

final

Vector2

p1

=

Vector2

.

zero

();

58

final

Vector2

p2

=

Vector2

.

zero

();

59

60

@override

61

void

render

(

Canvas

canvas

)

{

62

p1

63

..

setFrom

(

joint

.

getLocalAxisA

())

64

..

scale

(

joint

.

getLowerLimit

())

65

..

add

(

anchor

);

66

p2

67

..

setFrom

(

joint

.

getLocalAxisA

())

68

..

scale

(

joint

.

getUpperLimit

())

69

..

add

(

anchor

);

70

71

canvas

.

drawLine

(

p1

.

toOffset

(),

p2

.

toOffset

(),

debugPaint

);

72

}

73

}

Code

b1

,

b2

: Bodies connected by the joint.

anchor

: World anchor point, to put the axis through. Usually the center of the first body.

axis

: World translation axis, along which the translation will be fixed.

In some cases you might wish to control the range of motion. For this, the

PrismaticJointDef

has optional parameters that allow you to simulate a joint limit and/or a motor.

Prismatic Joint Limit

¶

You can limit the relative rotation with a joint limit that specifies a lower and upper translation.

jointDef

..

enableLimit

=

true

..

lowerTranslation

=

-

20

..

upperTranslation

=

20

;

enableLimit

: Set to true to enable translation limits

lowerTranslation

: The lower translation limit in meters

upperTranslation

: The upper translation limit in meters

You change the limits after the joint was created with this method:

prismaticJoint

.

setLimits

(

-

10

,

10

);

Prismatic Joint Motor

¶

You can use a motor to drive the motion or to model joint friction. A maximum motor force is provided so th

jointDef

..

enableMotor

=

true

..

motorSpeed

=

1

..

maxMotorForce

=

100

;

enableMotor

: Set to true to enable the motor

motorSpeed

: The desired motor speed in radians per second

maxMotorForce

: The maximum motor torque used to achieve the desired motor speed in N-m.

You change the motor?s speed and force after the joint was created using these methods:

prismaticJoint

.

setMotorSpeed

(

2

);

prismaticJoint

.

setMaxMotorForce

(

200

);

Also, you can get the joint angle and speed using the following methods:

prismaticJoint

.

getJointTranslation

();

prismaticJoint

.

getJointSpeed

();

PulleyJoint

¶

A

PulleyJoint

is used to create an idealized pulley. The pulley connects two bodies to the ground and to each other. As o

length1 + length2 == constant

You can supply a ratio that simulates a block and tackle. This causes one side of the pulley to extend faste

length1 + ratio * length2 == constant

For example, if the ratio is 2, then

length1

will vary at twice the rate of

length2

. Also the force in the rope attached to the first body will have half the constraint force as the rope attached

final

pulleyJointDef

=

PulleyJointDef

()

..

initialize

(

firstBody

,

secondBody

,

firstPulley

.

worldCenter

,

secondPulley

.

worldCenter

,

firstBody

.

worldCenter

,

secondBody

.

worldCenter

,

1

,

);

world

.

createJoint

(

PulleyJoint

(

pulleyJointDef

));

Run

pulley_joint.dart

```
1
import
'dart:ui'
;
2
3
import
'package:examples/stories/bridge_libraries/flame_forge2d/utils/balls.dart'
;
4
import
'package:examples/stories/bridge_libraries/flame_forge2d/utils/boxes.dart'
;
5
import
'package:flame/components.dart'
;
6
import
'package:flame_forge2d/flame_forge2d.dart'
;
7
8
class
```

PulleyJointExample

extends

Forge2DGame

{

9

static

const

description

=

'''

10

This example shows how to use a `PulleyJoint`. Drag one of the boxes and see

11

how the other one gets moved by the pulley

12

'''

;

13

14

@override

15

Future

<

void

>

onLoad

()

async

{

16

super

.

onLoad

();

17

final

distanceFromCenter

=

camera

.

visibleWorldRect

.

width

/

5

;

18

19

final

firstPulley

=

Ball

(

20

Vector2

(

-

distanceFromCenter

,

-

10

),

21

bodyType:

BodyType

.

static

,

22

);

23

final

secondPulley

=

Ball

(

24

Vector2

(

distanceFromCenter

,

-

10

),

25

bodyType:

BodyType

.

static

,

26

);

27

28

final

firstBox

=

DraggableBox

(

29

startPosition:

Vector2

(

-

distanceFromCenter

,

20

),

30

width:

5

,

31

height:

10

,

32

);

33

final

secondBox

=

DraggableBox

(

34

startPosition:

Vector2

(

distanceFromCenter

,

20

),

35

width:

7

,

36

height:

10

,

37

);

38

world

.

addAll

([

firstBox

,

secondBox

,

firstPulley

,

secondPulley

]);

39

40

await

Future

.

wait

([

41

firstBox

.

loaded

,

42

secondBox

.

loaded

,

43

firstPulley

.

loaded

,

44

secondPulley

.

loaded

,

```
45
]);
46
47
final
joint
=
createJoint
(
firstBox
,
secondBox
,
firstPulley
,
secondPulley
);
48
world
.
add
(
PulleyRenderer
(
joint:
joint
```

));

49

}

50

51

PulleyJoint

createJoint

(

52

Box

firstBox

,

53

Box

secondBox

,

54

Ball

firstPulley

,

55

Ball

secondPulley

,

56

)

{

57

final

pulleyJointDef

=

PulleyJointDef

()

58

..

initialize

(

59

firstBox

.

body

,

60

secondBox

.

body

,

61

firstPulley

.

center

,

62

secondPulley

.

center

,

63

firstBox

.

body

.

worldPoint

(

Vector2

(

0

,

-

firstBox

.

height

/

2

)),

64

secondBox

.

body

.

worldPoint

(

Vector2

(

0

,

-

secondBox

.

height

/

2

)),

65

1

,

66

);

67

final

joint

=

PulleyJoint

(

pulleyJointDef

);

68

world

.

createJoint

(

joint

);

69

return

joint

;

70

}

71

}

72

73

class

PulleyRenderer

extends

Component

{

74

PulleyRenderer

({

required

this

.

joint

});

75

76

final

PulleyJoint

joint

;

77

78

@override

79

void

render

(

Canvas

canvas

)

{

80

canvas

.

```
drawLine
(
81
joint
.
anchorA
.
toOffset
(),
82
joint
.
getGroundAnchorA
().
toOffset
(),
83
debugPaint
,
84
);
85
86
canvas
.
drawLine
```

(

87

joint

.

anchorB

.

toOffset

(),

88

joint

.

getGroundAnchorB

().

toOffset

(),

89

debugPaint

,

90

);

91

92

canvas

.

drawLine

(

93

joint

.

getGroundAnchorA

().

toOffset

(),

94

joint

.

getGroundAnchorB

().

toOffset

(),

95

debugPaint

,

96

);

97

}

98

}

Code

The

initialize

method of

PulleyJointDef

requires two ground anchors, two dynamic bodies and their anchor points, and a pulley ratio.

b1

,

b2

: Two dynamic bodies connected with the joint

ga1

,

ga2

: Two ground anchors

anchor1

,

anchor2

: Anchors on the dynamic bodies the joint will be attached to

r

: Pulley ratio to simulate a block and tackle

PulleyJoint

also provides the current lengths:

joint

.

getCurrentLengthA

()

joint

.

getCurrentLengthB

()

Warning

PulleyJoint

can get a bit troublesome by itself. They often work better when combined with prismatic joints. You should

RevoluteJoint

¶

A

RevoluteJoint

forces two bodies to share a common anchor point, often called a hinge point. The revolute joint has a sing

To create a

RevoluteJoint

, provide two bodies and a common point to the

initialize

method. The definition uses local anchor points so that the initial configuration can violate the constraint sli

final

jointDef

=

RevoluteJointDef

()

..

initialize

(

firstBody

,

secondBody

,

firstBody

.

position

);

world

.

createJoint

(

RevoluteJoint

(

jointDef

));

Run

revolute_joint.dart

1

import

'dart:math'

;

2

3

import

'package:examples/stories/bridge_libraries/flame_forge2d/utils/balls.dart'

;

4

import

'package:examples/stories/bridge_libraries/flame_forge2d/utils/boundaries.dart'

;

5

import

'package:flame/components.dart'

;

6

import

'package:flame/events.dart'

;

7

import

'package:flame_forge2d/flame_forge2d.dart'

;

8

9

class

RevoluteJointExample

extends

Forge2DGame

{

10

static

const

description

=

'''

11

In this example we use a joint to keep a body with several fixtures stuck

12

to another body.

13

14

Tap the screen to add more of these combined bodies.

15

'''

;

16

17

RevoluteJointExample

()

18

:

super

(

gravity:

Vector2

(

0

,

10.0

),

world:

RevoluteJointWorld

());

19

}

20

21

class

RevoluteJointWorld

extends

Forge2DWorld

22

with

TapCallbacks

,

HasGameReference

<

Forge2DGame

>

{

23

@override

24

Future

<

void

>

onLoad

()

async

{

25

super

.

onLoad

();

26

addAll

(

createBoundaries

(

game

));

27

}

28

29

@override

30

void

onTapDown

(

TapDownEvent

info

)

{

31

super

.

onTapDown

(

info

);

32

final

ball

=

Ball

(

info

.

localPosition

);

33

add

(

ball

);

34

```
add
(
CircleShuffler
(
ball
));
35
}
36
}
37
38
class
CircleShuffler
extends
BodyComponent
{
39
final
Ball
ball
;
40
41
CircleShuffler
(
```

this

.

ball

);

42

43

@override

44

Body

createBody

()

{

45

final

bodyDef

=

BodyDef

(

46

type:

BodyType

.

dynamic

,

47

position:

ball

.

body

.

position

.

clone

(),

48

);

49

const

numPieces

=

5

;

50

const

radius

=

6.0

;

51

final

body

=

world

.

createBody

(

bodyDef

);

52

53

for

(

var

i

=

0

;

i

<

numPieces

;

i

++

)

{

54

final

xPos

=

radius

*

cos

(

2

*

pi

*

(

i

/

numPieces

));

55

final

yPos

=

radius

*

sin

(

2

*

pi

*

(

i

/

numPieces

));

56

57

final

shape

=

CircleShape

()

58

..

radius

=

1.2

59

..

position

.

setValues

(

xPos

,

yPos

);

60

61

final

fixtureDef

=

FixtureDef

(

62

shape

,

63

density:

50.0

,

64

friction:

0.1

,

65

restitution:

0.9

,

66

);

67

68

body

.

createFixture

(

fixtureDef

);

69

}

70

71

final

jointDef

=

RevoluteJointDef

()

72

..

initialize

(

body

,

ball

.

body

,

body

.

position

);

73

world

.

createJoint

(

RevoluteJoint

(

jointDef

));

74

75

return

body

;

76

}

77

}

Code

In some cases you might wish to control the joint angle. For this, the

RevoluteJointDef

has optional parameters that allow you to simulate a joint limit and/or a motor.

Revolute Joint Limit

¶

You can limit the relative rotation with a joint limit that specifies a lower and upper angle.

jointDef

..

enableLimit

=

true

..

lowerAngle

=

0

..

upperAngle

=

pi

/

2

;

enableLimit

: Set to true to enable angle limits

lowerAngle

: The lower angle in radians

upperAngle

: The upper angle in radians

You change the limits after the joint was created with this method:

revoluteJoint

.

setLimits

(

0

,

pi

);

Revolute Joint Motor

¶

You can use a motor to drive the relative rotation about the shared point. A maximum motor torque is provi

jointDef

..

enableMotor

=

true

..

motorSpeed

=

5

..

maxMotorTorque

=

100

;

enableMotor

: Set to true to enable the motor

motorSpeed

: The desired motor speed in radians per second

maxMotorTorque

: The maximum motor torque used to achieve the desired motor speed in N-m.

You change the motor?s speed and torque after the joint was created using these methods:

revoluteJoint

.

setMotorSpeed

(

2

);

revoluteJoint

.

setMaxMotorTorque

(

200

);

Also, you can get the joint angle and speed using the following methods:

revoluteJoint

.

jointAngle

();

revoluteJoint

.

jointSpeed

();

RopeJoint

¶

A

RopeJoint

restricts the maximum distance between two points on two bodies.

RopeJointDef

requires two body anchor points and the maximum length.

final

ropeJointDef

=

RopeJointDef

()

..

bodyA

=

firstBody

..

localAnchorA

.

setFrom

(

firstBody

.

getLocalCenter

())

..

bodyB

=

secondBody

..

localAnchorB

.

setFrom

(

secondBody

.

getLocalCenter

())

..

maxLength

=

(

secondBody

.

worldCenter

-

firstBody

.

worldCenter

).

length

;

world

.

createJoint

(

RopeJoint

(

ropeJointDef

));

Run

rope_joint.dart

1

import

'package:examples/stories/bridge_libraries/flame_forge2d/utils/balls.dart'

;

2

import

'package:examples/stories/bridge_libraries/flame_forge2d/utils/boxes.dart'

;

3

import

'package:flame/components.dart'

;

4

import

'package:flame/events.dart'

;

5

import

'package:flame_forge2d/flame_forge2d.dart'

;

6

import

'package:flutter/material.dart'

;

7

8

class

RopeJointExample

extends

Forge2DGame

{

9

static

const

description

=

'''

10

This example shows how to use a `RopeJoint`.

11

12

Drag the box handle along the axis and observe the rope respond to the

13

movement.

14

'''

;

15

16

RopeJointExample

()

:

super

(

world:

RopeJointWorld

());

17

}

18

19

class

RopeJointWorld

extends

Forge2DWorld

20

with

DragCallbacks

,

HasGameReference

<

Forge2DGame

>

{

21

double

handleWidth

=

6

;

22

23

@override

24

Future

<

void

>

onLoad

()

async

{

25

super

.

onLoad

();

26

27

final

handleBody

=

await

createHandle

();

28

createRope

(

handleBody

);

29

}

30

31

Future

<

Body

>

createHandle

()

async

{

32

final

anchor

=

game

.

screenToWorld

(

Vector2

(

0

,

100

))..

x

=

0

;

33

34

final

box

=

DraggableBox

(

35

startPosition:

anchor

,

36

width:

handleWidth

,

37

height:

3

,

38

);

39

await

add

(

box

);

40

41

createPrismaticJoint

(

box

.

body

,

anchor

);

42

return

box

.

body

;

43

}

44

45

Future

<

void

>

createRope

(

Body

handle

)

async

{

46

const

length

=

50

;

47

var

prevBody

=

handle

;

48

49

for

(

var

i

=

0

;

i

<

length

;

i

++

)

{

50

final

newPosition

=

prevBody

.

worldCenter

+

Vector2

(

0

,

1

);

51

final

ball

=

Ball

(

newPosition

,

radius:

0.5

,

color:

Colors

.

white

);

52

await

add

(

ball

);

53

54

createRopeJoint

(

ball

.

body

,

prevBody

);

55

prevBody

=

ball

.

body

;

56

}

57

}

58

59

void

createPrismaticJoint

(

Body

box

,

Vector2

anchor

)

{

60

final

groundBody

=

createBody

(

BodyDef

());

61

final

halfWidth

=

game

.

screenToWorld

(

Vector2

.

zero

()).

x

.

abs

();

62

63

final

prismaticJointDef

=

PrismaticJointDef

()

64

..

initialize

(

65

box

,

66

groundBody

,

67

anchor

,

68

Vector2

(

1

,

0

),

69

)

70

..

enableLimit

=

true

71

..

lowerTranslation

=

-

halfWidth

+

handleWidth

/

2

72

..

upperTranslation

=

halfWidth

-

handleWidth

/

2

;

73

74

final

joint

=

PrismaticJoint

(

prismaticJointDef

);

75

createJoint

(

joint

);

76

}

77

78

void

createRopeJoint

(

Body

first

,

Body

second

)

{

79

final

ropeJointDef

=

RopeJointDef

()

80

..

bodyA

=

first

81

..

localAnchorA

.

setFrom

(

first

.

getLocalCenter

())

82

..

bodyB

=

second

83

..

localAnchorB

.

setFrom

(

second

.

getLocalCenter

())

84

..

maxLength

=

(

second

.

worldCenter

-

first

.

worldCenter

).

length

;

85

86

createJoint

(

RopeJoint

(

ropeJointDef

));

87

}

88

}

Code

bodyA

,

bodyB

: Connected bodies

localAnchorA

,

localAnchorB

: Optional parameter, anchor point relative to the body?s origin.

maxLength

: The maximum length of the rope. This must be larger than

linearSlop

, or the joint will have no effect.

Warning

The joint assumes that the maximum length doesn?t change during simulation. See

DistanceJoint

if you want to dynamically control length.

WeldJoint

¶

A

WeldJoint

is used to restrict all relative motion between two bodies, effectively joining them together.

WeldJointDef

requires two bodies that will be connected, and a world anchor:

final

weldJointDef

=

WeldJointDef

()

..

initialize

(

bodyA

,

bodyB

,

anchor

);

world

.

createJoint

(

WeldJoint

(

weldJointDef

));

Run

weld_joint.dart

1

import

'package:examples/stories/bridge_libraries/flame_forge2d/utils/balls.dart'

;

2

import

'package:examples/stories/bridge_libraries/flame_forge2d/utils/boxes.dart'

;

3

import

'package:flame/components.dart'

;

4

import

'package:flame/events.dart'

;

5

import

'package:flame_forge2d/flame_forge2d.dart'

;

6

import

'package:flutter/material.dart'

;

7

8

class

WeldJointExample

extends

Forge2DGame

{

9

static

const

description

=

'''

10

This example shows how to use a `WeldJoint`. Tap the screen to add a

11

ball to test the bridge built using a `WeldJoint`

12

'''

;

13

14

WeldJointExample

()

:

super

(

world:

WeldJointWorld

());

15

}

16

17

class

WeldJointWorld

extends

Forge2DWorld

18

with

TapCallbacks

,

HasGameReference

<

Forge2DGame

>

{

19

final

pillarHeight

=

20.0

;

20

final

pillarWidth

=

5.0

;

21

22

@override

23

Future

<

void

>

onLoad

()

async

{

24

super

.

onLoad

();

25

26

final

leftPillar

=

Box

(

27

startPosition:

game

.

screenToWorld

(

Vector2

(

50

,

game

.

size

.

y

))

28

..

y

-=

pillarHeight

/

2

,

29

width:

pillarWidth

,

30

height:

pillarHeight

,

31

bodyType:

BodyType

.

static

,

32

color:

Colors

.

white

,

33

);

34

final

rightPillar

=

Box

(

35

startPosition:

game

.

screenToWorld

(

Vector2

(

game

.

size

.

x

-

50

,

game

.

size

.

y

))

36

..

y

-=

pillarHeight

/

2

,

37

width:

pillarWidth

,

38

height:

pillarHeight

,

39

bodyType:

BodyType

.

static

,

40

color:

Colors

.

white

,

41

);

42

43

await

addAll

([

leftPillar

,

rightPillar

]);

44

45

createBridge

(

leftPillar

,

rightPillar

);

46

}

47

48

Future

<

void

>

createBridge

(

49

Box

leftPillar

,

50

Box

rightPillar

,

51

)

async

{

52

const

sectionsCount

=

10

;

53

// Vector2.zero is used here since 0,0 is in the middle and 0,0 in the

54

// screen space then gives us the coordinates of the upper left corner in

55

// world space.

56

final

halfSize

=

game

.

screenToWorld

(

Vector2

.

zero

())..

absolute

();

57

final

sectionWidth

=

((

leftPillar

.

center

.

x

.

abs

()

+

58

rightPillar

.

center

.

x

.

abs

()

+

59

pillarWidth

)

/

60

sectionsCount

)

61

.

ceilToDouble

();

62

Body

?

prevSection

;

63

64

for

(

var

i

=

0

;

i

<

sectionsCount

;

i

++

)

{

65

final

section

=

Box

(

66

startPosition:

Vector2

(

67

sectionWidth

*

i

-

halfSize

.

x

+

sectionWidth

/

2

,

68

halfSize

.

y

-

pillarHeight

,

69

),

70

width:

sectionWidth

,

71

height:

1

,

72

);

73

await

add

(

section

);

74

75

if

(

prevSection

!=

null

)

{

76

createWeldJoint

(

77

prevSection

,

78

section

.

body

,

79

Vector2

(

80

sectionWidth

*

i

-

halfSize

.

x

+

sectionWidth

,

81

halfSize

.

y

-

pillarHeight

,

82

),

83

);

84

}

85

86

prevSection

=

section

.

body

;

87

}

88

}

89

90

void

createWeldJoint

(

Body

first

,

Body

second

,

Vector2

anchor

)

{

91

final

weldJointDef

=

WeldJointDef

()..

initialize

(

first

,

second

,

anchor

);

92

93

createJoint

(

WeldJoint

(

weldJointDef

));

94

}

95

96

@override

97

Future

<

void

>

onTapDown

(

TapDownEvent

info

)

async

{

98

super

.

onTapDown

(

info

);

99

final

ball

=

Ball

(

info

.

localPosition

,

radius:

5

);

100

add

(

ball

);

101

}

102

}

Code

bodyA

,

bodyB

: Two bodies that will be connected

anchor

: Anchor point in world coordinates, at which two bodies will be welded together to 0, the higher the value, t

Breakable Bodies and WeldJoint

¶

Since the Forge2D constraint solver is iterative, joints are somewhat flexible. This means that the bodies co

WeldJoint

.

Collision Detection

¶

Collision detection is needed in most games to detect and act upon two components intersecting each othe

In most collision detection systems you use something called hitboxes to create more precise bounding bo

gesture input

) more accurate.

The collision detection system supports three different types of shapes that you can build hitboxes from, th

PositionComponent

s have intersecting hitboxes.

Do note that the built-in collision detection system does not take collisions between two hitboxes that overs

update

being called with a large delta time (for example if your app is not in the foreground). This behavior is calle

Also note that the collision detection system has a limitation that makes it not work properly if you have cer

Mixins

¶

HasCollisionDetection

¶

If you want to use collision detection in your game you have to add the

HasCollisionDetection

mixin to your game so that it can keep track of the components that can collide.

Example:

class

MyGame

extends

FlameGame

with

HasCollisionDetection

{

// ...

}

Now when you add

ShapeHitbox

s to components that are then added to the game, they will automatically be checked for collisions.

You can also add

HasCollisionDetection

directly to another

Component

instead of the

FlameGame

, for example to the

World

that is used for the

CameraComponent

. If that is done, hitboxes that are added in that component?s tree will only be compared to other hitboxes i

FlameGame

.

Example:

class

CollisionDetectionWorld

extends

World

with

HasCollisionDetection

{}

Note

Hitboxes will only be connected to one collision detection system and that is the closest parent that has the

HasCollisionDetection

mixin.

CollisionCallbacks

¶

To react to a collision you should add the

CollisionCallbacks

mixin to your component. Example:

collision_detection.dart

1

import

'package:doc_flame_examples/ember.dart'

;

2

import

'package:flame/collisions.dart'

;

3

import

'package:flame/components.dart'

;

4

import

'package:flame/effects.dart'

;

5

import

'package:flame/game.dart'

;

6

import

'package:flutter/material.dart'

hide

Image

;

7

8

class

CollisionDetectionGame

extends

FlameGame

with

HasCollisionDetection

{

9

@override

10

Future

<

void

>

onLoad

()

async

{

11

final

emberPlayer

=

EmberPlayer

(

12

position:

Vector2

(

10

,

(

size

.

y

/

2

)

-

20

),

13

size:

Vector2

.

all

(

40

),

14

onTap:

(

emberPlayer

)

{

15

emberPlayer

.

add

(

16

MoveEffect

.

to

(

17

Vector2

(

size

.

x

-

40

,

(

size

.

y

/

2

)

-

20

),

18

EffectController

(

19

duration:

5

,

20

reverseDuration:

5

,

21

repeatCount:

1

,

22

curve:

Curves

.

easeOut

,

23

),

24

),

25

);

26

},

27

);

28

add

(

emberPlayer

);

29

add

(

RectangleCollidable

(

canvasSize

/

2

));

30

}

31

}

32

33

class

RectangleCollidable

extends

PositionComponent

with

CollisionCallbacks

{

34

final

_collisionStartColor

=

Colors

.

amber

;

35

final

_defaultColor

=

Colors

.

cyan

;

36

late

ShapeHitbox

hitbox

;

37

38

RectangleCollidable

(

Vector2

position

)

39

:

super

(

40

position:

position

,

41

size:

Vector2

.

all

(

50

),

42

anchor:

Anchor

.

center

,

43

);

44

45

@override

46

Future

<

void

>

onLoad

()

async

{

47

final

defaultPaint

=

Paint

()

48

..

color

=

_defaultColor

49

..

style

=

PaintingStyle

.

stroke

;

50

hitbox

=

RectangleHitbox

()

51

..

paint

=

defaultPaint

52

..

renderShape

=

true

;

53

add

(

hitbox

);

54

}

55

56

@override

57

void

onCollisionStart

(

58

Set

<

Vector2

>

intersectionPoints

,

59

PositionComponent

other

,

60

)

{

61

super

.

onCollisionStart

(

intersectionPoints

,

other

);

62

hitbox

.

paint

.

color

=

_collisionStartColor

;

63

}

64

65

@override

66

void

onCollisionEnd

(

PositionComponent

other

)

{

67

super

.

onCollisionEnd

(

other

);

68

if

(

!

isColliding

)

{

69

hitbox

.

paint

.

color

=

_defaultColor

;

70

}

71

}

72

}

Code

class

MyCollidable

extends

PositionComponent

with

CollisionCallbacks

{

@override

void

onCollision

(

Set

<

Vector2

>

points

,

PositionComponent

other

)

{

if

(

other

is

ScreenHitbox

)

{

//...

}

else

if

(

other

is

YourOtherComponent

)

{

//...

}

}

@override

void

onCollisionEnd

(

PositionComponent

other

)

{

if

(

other

is

ScreenHitbox

)

{

//...

}

else

if

(

other

is

YourOtherComponent

)

{

//...

}

}

}

In this example we use Dart?s

is

keyword to check what kind of component we collided with. The set of points is where the edges of the hitb

Note that the

onCollision

method will be called on both

PositionComponent

s if they have both implemented the

onCollision

method, and also on both hitboxes. The same goes for the

onCollisionStart

and

onCollisionEnd

methods, which are called when two components and hitboxes starts or stops colliding with each other.

When a

PositionComponent

(and hitbox) starts to collide with another

PositionComponent

both

onCollisionStart

and

onCollision

are called, so if you don?t need to do something specific when a collision starts you only need to override

onCollision

, and vice versa.

If you want to check collisions with the screen edges, as we do in the example above, you can use the pre

ScreenHitbox

class.

By default all hitboxes are hollow, this means that one hitbox can be fully enclosed by another hitbox witho

isSolid

=

true

. A hollow hitbox inside of a solid hitbox will trigger a collision, but not the other way around. If there are no

Collision order

¶

If a

Hitbox

collides with more than one other

Hitbox

within a given time step, then the

onCollision

callbacks will be called in an essentially random order. In some cases this can be a problem, such as in a b

collisionsCompletedNotifier

listener can be used - this triggers at the end of the collision detection process.

An example of how this might be used is to add a local variable in your

PositionComponent

to save the other components with which it?s colliding:

List<PositionComponent>

collisionComponents

=

[];

. The

onCollision

callback is then used to save all the other

PositionComponent

s to this list:

```
@override
void
onCollision
(
Set
<
Vector2
>
intersectionPoints
,
PositionComponent
other
)
{
collisionComponents
.
add
(
other
);
super
.
onCollision
(
intersectionPoints
```

,

other

);

}

Finally, one adds a listener to the

onLoad

method of the

PositionComponent

to call a function which will resolve how the collisions should be dealt with:

(

game

as

HasCollisionDetection

)

.

collisionDetection

.

collisionsCompletedNotifier

.

addListener

(()

{

resolveCollisions

();

});

The list

collisionComponents

would need to be cleared in each call to

update

.

ShapeHitbox

¶

The

ShapeHitbox

s are normal components, so you add them to the component that you want to add hitboxes to just like any

class

MyComponent

extends

PositionComponent

{

@override

void

onLoad

()

{

add

(

RectangleHitbox

());

}

}

If you don?t add any arguments to the hitbox, like above, the hitbox will try to fill its parent as much as poss

relative

constructor which defines the hitbox in relation to the size of its intended parent.

In some specific cases you might want to handle collisions only between hitboxes, without propagating

onCollision*

events to the hitbox?s parent component. For example, a vehicle could have a body hitbox to control collisi

triggersParentCollision

variable to

false

:

class

MyComponent

extends

PositionComponent

{

late

final

MySpecialHitbox

utilityHitbox

;

@override

void

onLoad

()

{

utilityHitbox

=

```
MySpecialHitbox
();
add
(
utilityHitbox
);
}
void
update
(
double
dt
)
{
if
(
utilityHitbox
.
isColliding
)
{
// do some specific things if hitbox is colliding
}
}
// component's onCollision* functions, ignoring MySpecialHitbox collisions.
}
```

class

MySpecialHitbox

extends

RectangleHitbox

{

MySpecialHitbox

()

{

triggersParentCollision

=

false

;

}

// hitbox specific onCollision* functions

}

You can read more about how the different shapes are defined in the

ShapeComponents

section.

Remember that you can add as many

ShapeHitbox

s as you want to your

PositionComponent

to make up more complex areas. For example a snowman with a hat could be represented by three

CircleHitbox

s and two

RectangleHitbox

s as its hat.

A hitbox can be used either for collision detection or for making gesture detection more accurate on top of

GestureHitboxes

mixin.

CollisionType

¶

The hitboxes have a field called

collisionType

which defines when a hitbox should collide with another. Usually you want to set as many hitboxes as poss

CollisionType.passive

to make the collision detection more performant. By default the

CollisionType

is

active

.

The

CollisionType

enum contains the following values:

active

collides with other

Hitbox

es of type active or passive

passive

collides with other

Hitbox

es of type active

inactive

will not collide with any other

Hitbox

es

So if you have hitboxes that you don?t need to check collisions against each other you can mark them as p

collisionType:

CollisionType.passive

in the constructor, this could for example be ground components or maybe your enemies don?t need to ch

passive

too.

Imagine a game where there are a lot of bullets, that can?t collide with each other, flying towards the playe

CollisionType.active

and the bullets would be set to

CollisionType.passive

.

Then we have the

inactive

type which simply doesn?t get checked at all in the collision detection. This could be used for example if yo

These are just examples of how you could use these types, there will be a lot more use cases for them so

PolygonHitbox

¶

It should be noted that if you want to use collision detection or

containsPoint

on the

Polygon

, the polygon needs to be convex. So always use convex polygons or you will most likely run into problems

The other hitbox shapes don?t have any mandatory constructor, that is because they can have a default ca

The

PolygonHitbox

has the same constructors as the

PolygonComponent

, see that section for documentation regarding those.

RectangleHitbox

¶

The

RectangleHitbox

has the same constructors as the

RectangleComponent

, see that section for documentation regarding those.

CircleHitbox

¶

The

CircleHitbox

has the same constructors as the

CircleComponent

, see that section for documentation regarding those.

ScreenHitbox

¶

ScreenHitbox

is a component which represents the edges of your viewport/screen. If you add a

ScreenHitbox

to your game your other components with hitboxes will be notified when they collide with the edges. It does

size

of the game that it is added to. To add it you can just do

add(ScreenHitbox())

in your game, if you don?t want the

ScreenHitbox

itself to be notified when something collides with it. Since

ScreenHitbox

has the

CollisionCallbacks

mixin you can add your own

onCollisionCallback

,

onStartCollisionCallback

and

onEndCollisionCallback

functions to that object if needed.

CompositeHitbox

¶

In the

CompositeHitbox

you can add multiple hitboxes so that they emulate being one joined hitbox.

If you want to form a hat for example you might want to use two

RectangleHitbox

s to follow that hat?s edges properly, then you can add those hitboxes to an instance of this class and reac

Broad phase

¶

If your game field isn?t huge and does not have a lot of collidable components - you don?t have to worry a

A broad phase is the first step of collision detection where potential collisions are calculated. Calculating th

The broad phase produces a set of potential collisions (a set of

CollisionProspect

s). This set is then used to check the exact intersections between hitboxes (sometimes called ?narrow pha

By default, Flame?s collision detection is using a sweep and prune broadphase step. If your game requires

Broadphase

and manually setting the collision detection system that should be used.

For example, if you have implemented a broadphase built on a magic algorithm instead of the standard sw

```
class
MyGame
extends
FlameGame
with
HasCollisionDetection
{
MyGame
()
:
super
()
{
collisionDetection
=
StandardCollisionDetection
(
```

broadphase:

MagicAlgorithmBroadphase

());

}

}

Quad Tree broad phase

¶

If your game field is large and the game contains a lot of collidable components (more than a hundred), sta

To do this, add the

HasQuadTreeCollisionDetection

mixin to your game instead of

HasCollisionDetection

and call the

initializeCollisionDetection

function on game load:

class

MyGame

extends

FlameGame

with

HasQuadTreeCollisionDetection

{

@override

void

onLoad

()

{

initializeCollisionDetection

(

mapDimensions:

const

Rect

.

fromLTWH

(

0

,

0

,

mapWidth

,

mapHeight

),

minimumDistance:

10

,

);

}

}

When calling

initializeCollisionDetection

you should pass it the correct map dimensions, to make the quad tree algorithm to work properly. There ar

minimumDistance

: minimum distance between objects to consider them as possibly colliding. If

null

- the check is disabled, it is default behavior

maxObjects

: maximum objects count in one quadrant. Default to 25.

maxDepth

: maximum nesting levels inside quadrant. Default to 10

If you use the quad tree system, you can make it even more efficient by implementing the

onComponentTypeCheck

function of the

CollisionCallbacks

mixin in your components. It is useful if you need to prevent collisions of items of different types. The result

class

Bullet

extends

PositionComponent

with

CollisionCallbacks

{

@override

bool

onComponentTypeCheck

(

PositionComponent

other

)

{

if

(

other

is

Player

||

other

is

Water

)

{

// do NOT collide with Player or Water

return

false

;

}

// Just return true if you're not interested in the parent's type check result.

// Or call super and you will be able to override the result with the parent's

// result.

return

super

.

onComponentTypeCheck

(

```
other
);
}
@override
void
onCollisionStart
(
Set
<
Vector2
>
intersectionPoints
,
PositionComponent
other
,
)
{
// Removes the component when it comes in contact with a Brick.
// Neither Player nor Water would be passed to this function
// because these classes are filtered out by [onComponentTypeCheck]
// in an earlier stage.
if
(
other
is
```

Brick

)

{

removeFromParent

();

}

super

.

onCollisionStart

(

intersectionPoints

,

other

);

}

}

After intensive gameplay a map could become over-clusterized with a lot of empty quadrants. Run

QuadTree.optimize()

to perform a cleanup of empty quadrants:

class

QuadTreeExample

extends

FlameGame

with

HasQuadTreeCollisionDetection

{

/// A function called when intensive gameplay session is over

/// It also might be scheduled, but no need to run it on every update.

/// Use right interval depending on your game circumstances

onGameIdle

()

{

(

collisionDetection

as

QuadTreeCollisionDetection

)

.

quadBroadphase

.

tree

.

optimize

();

}

}

Note

Always experiment with different collision detection approaches and check how they perform on your game

QuadTreeBroadphase

is significantly

slower

than the default. Don?t assume that the more sophisticated approach is always faster.

Ray casting and Ray tracing

¶

Ray casting and ray tracing are methods for sending out rays from a point in your game and being able to s

For all of the following methods, if there are any hitboxes that you wish to ignore, you can add the

ignoreHitboxes

argument which is a list of the hitboxes that you wish to disregard for the call. This can be quite useful for e

ScreenHitbox

.

Ray casting

¶

Ray casting is the operation of casting out one or more rays from a point and see if they hit anything, in Fla

We provide two methods for doing so,

raycast

and

raycastAll

. The first one just casts out a single ray and gets back a result with information about what and where the

raycastAll

, works similarly but sends out multiple rays uniformly around the origin, or within an angle centered at the

By default,

raycast

and

raycastAll

scan for the nearest hit irrespective of how far it lies from the ray origin. But in some use cases, it might be

maxDistance

can be provided.

To use the ray casting functionality you have to have the

HasCollisionDetection

mixin on your game. After you have added that you can call

collisionDetection.raycast(...)

on your game class.

Example:

ray_cast.dart

```
1
import
'package:flame/collisions.dart'
;
2
import
'package:flame/components.dart'
;
3
import
'package:flame/game.dart'
;
4
import
'package:flame/geometry.dart'
;
5
import
'package:flame/palette.dart'
;
```

6

import

'package:flutter/material.dart'

;

7

8

class

RayCastExample

extends

FlameGame

with

HasCollisionDetection

{

9

final

origin

=

Vector2

(

20

,

20

);

10

11

final

direction

=

Vector2

(

1

,

0

);

12

13

final

velocity

=

60

;

14

double

get

resetPosition

=>

-

canvasSize

.

y

;

15

16

Paint

paint

=

Paint

()..

color

=

Colors

.

red

.

withOpacity

(

0.6

);

17

18

RaycastResult

<

ShapeHitbox

>?

result

;

19

20

@override

21

Future

<

void

>

onLoad

()

async

{

22

final

paint

=

BasicPalette

.

gray

.

paint

()

23

..

style

=

PaintingStyle

.

stroke

24

..

strokeWidth

=

2.0

;

25

26

add

(

ScreenHitbox

());

27

28

add

(

29

CircleComponent

(

30

position:

canvasSize

/

2

,

31

radius:

30

,

32

paint:

paint

,

33

children:

[

CircleHitbox

()],

34

),

35

);

36

}

37

38

@override

39

void

update

(

double

dt

)

{

40

super

.

update

(

dt

);

41

final

ray

=

Ray2

(

42

origin:

origin

,

43

direction:

direction

,

44

);

45

result

=

collisionDetection

.

raycast

(

ray

);

46

47

origin

.

y

+=

velocity

*

dt

;

48

49

if

(

origin

.

y

>

canvasSize

.

y

)

{

50

origin

.

y

+=

resetPosition

;

51

}

52

}

53

54

@override

55

void

render

(

Canvas

canvas

)

{

56

super

.

render

(

canvas

);

57

58

if

(

result

!=

null

&&

result

!

.

isActive

)

{

59

final

originOffset

=

origin

.

toOffset

();

60

final

intersectionPoint

=

result

!

.

intersectionPoint

!

.

toOffset

();

61

canvas

.

drawLine

(

62

originOffset

,

```
63
intersectionPoint
,
64
paint
,
65
);
66
67
canvas
.
drawCircle
(
originOffset
,
10
,
paint
);
68
}
69
}
70
}
```

Code

class

MyGame

extends

FlameGame

with

HasCollisionDetection

{

@override

void

update

(

double

dt

)

{

super

.

update

(

dt

);

final

ray

=

Ray2

```
(
origin:
Vector2
(
0
,
100
),
direction:
Vector2
(
1
,
0
),
);
final
result
=
collisionDetection
.
raycast
(
ray
);
}
```

}

In this example one can see that the

Ray2

class is being used, this class defines a ray from an origin position and a direction (which are both defined

Vector2

s). This particular ray starts from

0,

100

and shoots a ray straight to the right.

The result from this operation will either be

null

if the ray didn?t hit anything, or a

RaycastResult

which contains:

Which hitbox the ray hit

The intersection point of the collision

The reflection ray, i.e. how the ray would reflect on the hitbox that it hix

The normal of the collision, i.e. a vector perpendicular to the face of the hitbox that it hits

If you are concerned about performance you can pre create a

RaycastResult

object that you send in to the method with the

out

argument, this will make it possible for the method to reuse this object instead of creating a new one for ea

update

methods.

raycastAll

¶

Sometimes you want to send out rays in all, or a limited range, of directions from an origin. This can have a

Example:

```
class
MyGame
extends
FlameGame
with
HasCollisionDetection
{
@override
void
update
(
double
dt
)
{
super
.
update
(
dt
);
final
origin
```

=

Vector2

(

200

,

200

);

final

result

=

collisionDetection

.

raycastAll

(

origin

,

numberOfRays:

100

,

);

}

}

In this example we would send out 100 rays from (200, 200) uniformly spread in all directions.

If you want to limit the directions you can use the

startAngle

and the

sweepAngle

arguments. Where the

startAngle

(counting from straight up) is where the rays will start and then the rays will end at

startAngle

+

sweepAngle

.

If you are concerned about performance you can re-use the

RaycastResult

objects that are created by the function by sending them in as a list with the

out

argument.

Ray tracing

¶

Ray tracing is similar to ray casting, but instead of just checking what the ray hits you can continue to trace

Example:

ray_trace.dart

```
1
import
'dart:math'
;
2
3
import
'package:flame/collisions.dart'
```

;

4

import

'package:flame/components.dart'

;

5

import

'package:flame/events.dart'

;

6

import

'package:flame/game.dart'

;

7

import

'package:flame/geometry.dart'

;

8

import

'package:flame/palette.dart'

;

9

import

'package:flutter/material.dart'

;

10

11

class

RayTraceExample

extends

FlameGame

12

with

HasCollisionDetection

,

TapDetector

{

13

Paint

paint

=

Paint

()..

color

=

Colors

.

red

.

withOpacity

(

0.6

);

14

bool

isClicked

=

false

;

15

16

Vector2

get

origin

=>

canvasSize

/

2

;

17

18

RaycastResult

<

ShapeHitbox

>?

result

;

19

20

final

Ray2

_ray

=

Ray2

.

zero

();

21

22

final

boxPaint

=

BasicPalette

.

gray

.

paint

()

23

..

style

=

PaintingStyle

.

stroke

24

..

strokeWidth

=

2.0

;

25

26

final

List

<

RaycastResult

<

ShapeHitbox

>>

results

=

[];

27

28

@override

29

Future

<

void

>

onLoad

()

async

{

30

add

(

31

CircleComponent

(

32

radius:

min

(

size

.

x

,

size

.

y

)

/

2

,

33

paint:

boxPaint

,

34

children:

[

CircleHitbox

()],

35

),

36

);

37

}

38

39

var

_timePassed

=

0.0

;

40

41

@override

42

```
void
update
(
double
dt
)
{
43
super
.
update
(
dt
);
44
if
(
isClicked
)
{
45
_timePassed
+=
dt
;
46
```

```
}
47
48
result
=
collisionDetection
.
raycast
(
_ray
);
49
50
_ray
.
origin
.
setFrom
(
origin
);
51
_ray
.
direction
52
```

..

setValues

(

1

,

1

)

53

..

normalize

();

54

collisionDetection

55

.

raytrace

(

56

_ray

,

57

maxDepth:

min

((

_timePassed

*

8

).

ceil

(),

1000

),

58

out:

results

,

59

)

60

.

toList

();

61

}

62

63

@override

64

void

render

(

Canvas

```
canvas
)
{
65
super
.
render
(
canvas
);
66
var
originOffset
=
origin
.
toOffset
();
67
for
(
final
result
in
results
)
```

{

68

if

(

!

result

.

isActive

)

{

69

continue

;

70

}

71

final

intersectionPoint

=

result

.

intersectionPoint

!

.

toOffset

();

```
72
canvas
.
drawLine
(
73
originOffset
,
74
intersectionPoint
,
75
paint
,
76
);
77
originOffset
=
intersectionPoint
;
78
}
79
}
80
```

```
81
@override
82
void
onTap
()
{
83
super
.
onTap
();
84
if
(
!
isClicked
)
{
85
isClicked
=
true
;
86
return
```

;

87

}

88

_timePassed

=

0

;

89

}

90

}

Code

class

MyGame

extends

FlameGame

with

HasCollisionDetection

{

@override

void

update

(

double

dt

)

{

super

.

update

(

dt

);

final

ray

=

Ray2

(

origin:

Vector2

(

0

,

100

),

direction:

Vector2

(

1

,

1

)..

normalize

()

);

final

results

=

collisionDetection

.

raytrace

(

ray

,

maxDepth:

100

,

);

for

(

final

result

in

results

)

{

if

```
(
result
.
intersectionPoint
.
distanceTo
(
ray
.
origin
)
>
300
)
{
break
;
}
}
}
}
```

In the example above we send out a ray from (0, 100) diagonally down to the right and we say that we wan

The method is lazy, which means that it will only do the calculations that you ask for, so you have to loop th

`toList()`

to directly calculate all the results.

In the for-loop it can be seen how this can be used, in that loop we check whether the current reflection ray

If you are concerned about performance you can re-use the

RaycastResult

objects that are created by the function by sending them in as a list with the

out

argument.

Comparison to Forge2D

¶

If you want to have a full-blown physics engine in your game we recommend that you use Forge2D by add

flame_forge2d

as a dependency. But if you have a simpler use-case and just want to check for collisions of components a

If you have the following needs you should at least consider to use

Forge2D

:

Interacting realistic forces

Particle systems that can interact with other bodies

Joints between bodies

It is a good idea to just use the Flame collision detection system if you on the other hand only need some c

The ability to act on some of your components colliding

The ability to act on your components colliding with the screen boundaries

Complex shapes to act as a hitbox for your component so that gestures will be more accurate

Hitboxes that can tell what part of a component that collided with something

Examples

¶

[https://examples.flame-engine.org/#/Collision%20Detection_Collidable%20AnimationComponent]

[https://examples.flame-engine.org/#/Collision%20Detection_Circles]

[https://examples.flame-engine.org/#/Collision%20Detection_Multiple%20shapes]

[https://github.com/flame-engine/flame/tree/main/examples/lib/stories/collision_detection]

User-defined commands

¶

In addition to the built-in commands, you can also declare your own

user-defined commands

for use in your yarn scripts. Typically, these commands would perform some in-game action that can be vie

<<wave>>

,

<<smile>>

,

<<frown>>

,

<<moveCamera>>

,

<<zoom>>

,

<<shakeCamera>>

,

<<fadeOut>>

,

<<walk>>

,

<<give>>

,

<<take>>

,

<<achievement>>

,

<<GainExperience>>

,

<<startQuest>>

,

<<finishQuest>>

,

<<openTrade>>

,

<<drawWeapon>>

, and so on.

In many cases, the commands will need to take arguments. The arguments of a user-defined command ar

First, all content after the command name and until the closing

>>

is parsed according to the rules of regular line parsing, where interpolated expressions are allowed but ma

At runtime, the content of that line is evaluated, meaning that we substitute the values of all expressions.

The evaluated argument string is then broken into individual arguments at whitespace, and the types of the

Then, the backing function is called with the parsed arguments.

Lastly, all dialogue views in the dialogue runner receive the

onCommand()

event.

As a concrete example, consider the following command:

<<

give

Gold

{

round

(

100

*

$multiplier

)}>>

First note that, unlike builtin commands, the arguments of the command are treated as text, and any expre

Then, at runtime the expression is evaluated, and (assuming

$multiplier

is 1.5) the command?s argument string becomes

"Gold

150"

. The string is then broken at white spaces and each argument is parsed according to its type in the backin

void

give(String

item,

int?

amount)

, then it will be invoked as

give("Gold",

150)

. If, on the other hand, the number or types of arguments do not match the expected signature, then a

DialogueException

will be raised.

Nodes

¶

A

node

is a small section of text, that represents a single conversation or interaction with an NPC. Each node has a

title

, which can be used to

run

that node in a

DialogueRunner

, or to

jump

to that node from another node.

You can think of a node as if it was a function in a regular programming language. Running a node is equiv

Each node consists of a

header

and a

body

. The header is separated from the body with 3 (or more) dashes, and the body is terminated with 3 ?=? sig

```
// NODE HEADER
---
// NODE BODY
===
```

In addition, you can use 3 (or more) dashes to separate the header from the previous content, which mean

```
----------
// NODE HEADER
```

---

// NODE BODY

===

A

node

is represented with a

Node

class in Jenny runtime.

Header

¶

The header of a node consists of one or more lines of the form

TAG:

CONTENT

. One of these lines must contain the node?s

title

, which is the name of the node:

title

:

NodeName

The title of a node must be a valid ID (that is, starts with a letter, followed by any number of letters, digits, c

Besides the title, you can add any number of extra tags into the node?s header. Jenny will store these tags

title

:

Alert

colorID

:

0

modal

:

true

---

WARNING

\:

Entering Radioactive Zone!

===

Body

¶

The body of a node is where the dialogue itself is located. The body is just a sequence of statements, whe

Line

, an

Option

, or a

Command

. For example:

title

:

Gloomy_Morning

camera_zoom

:

2

---

You

:

Good morning!

Guard

:

You call this good? 'Tis as crappy as could be

You

:

Why, what happened?

Guard

:

Don't you see the fog? Chills me through to the bones

You

:

Sorry to hear that...

You

:

So, can I pass?

Guard

:

Can I get some exercise cutting you into pieces? Maybe that'll warm me up!

You

:

Ok, I think I'll be going. Hope you feel better soon!

===

7. Adding Menus

¶

To add menus to the game, we will leverage Flame?s built-in

overlay

system.

Main Menu

¶

In the

lib/overlays

folder, create

main_menu.dart

and add the following code:

```
import

'package:flutter/material.dart'

;

import

'../ember_quest.dart'

;

class

MainMenu

extends

StatelessWidget

{

// Reference to parent game.

final

EmberQuestGame
```

game

;

const

MainMenu

({

super

.

key

,

required

this

.

game

});

@override

Widget

build

(

BuildContext

context

)

{

const

blackTextColor

=

Color

.

fromRGBO

(

0

,

0

,

0

,

1.0

);

const

whiteTextColor

=

Color

.

fromRGBO

(

255

,

255

,

255

,

1.0

);

return

Material

(

color:

Colors

.

transparent

,

child:

Center

(

child:

Container

(

padding:

const

EdgeInsets

.

all

(

10.0

),

height:

250

,

width:

300

,

decoration:

const

BoxDecoration

(

color:

blackTextColor

,

borderRadius:

const

BorderRadius

.

all

(

Radius

.

circular

(

20

),

),

),

child:

Column

(

mainAxisAlignment:

MainAxisAlignment

.

center

,

children:

[

const

Text

(

'Ember Quest'

,

style:

TextStyle

(

color:

whiteTextColor

,

fontSize:

24

,

),

),

const

SizedBox

(

height:

40

),

SizedBox

(

width:

200

,

height:

75

,

child:

ElevatedButton

(

onPressed:

()

{

game

.

overlays

.

remove

(

'MainMenu'

);

},

style:

ElevatedButton

.

styleFrom

(

backgroundColor:

whiteTextColor

,

),

child:

const

Text

(

'Play'

,

style:

TextStyle

(

fontSize:

40.0

,

color:

blackTextColor

,

),

),

),

),

const

SizedBox

(

height:

20

),

const

Text

(

'''Use WASD or Arrow Keys for movement.

Space bar to jump.

Collect as many stars as you can and avoid enemies!'''

,

textAlign:

TextAlign

.

center

,

style:

TextStyle

(

color:

whiteTextColor

,

fontSize:

14

,

),

),

],

),

),

),

);

}

}

This is a pretty self-explanatory file that just uses standard Flutter widgets to display information and provic

Play

button. The only Flame-related line is

game.overlays.remove('MainMenu');

which simply removes the overlay so the user can play the game. It should be noted that the user can tech

Game Over Menu

¶

Next, create a file called

lib/overlays/game_over.dart

and add the following code:

import

'package:flutter/material.dart'

;

import

'../ember_quest.dart'

;

class

GameOver

extends

StatelessWidget

{

// Reference to parent game.

final

EmberQuestGame

game

;

const

GameOver

({

super

.

key

,

required

this

.

game

});

@override

Widget

build

(

BuildContext

context

)

{

const

blackTextColor

=

Color

.

fromRGBO

(

0

,

0

,

0

,

1.0

);

const

whiteTextColor

=

Color

.

fromRGBO

(

255

,

255

,

255

,

1.0

);

return

Material

(

color:

Colors

.

transparent

,

child:

Center

(

child:

Container

(

padding:

const

EdgeInsets

.

all

(

10.0

),

height:

200

,

width:

300

,

decoration:

const

BoxDecoration

(

color:

blackTextColor

,

borderRadius:

const

BorderRadius

.

all

(

Radius

.

circular

(

20

),

),

),

child:

Column

(

mainAxisAlignment:

MainAxisAlignment

.

center

,

children:

[

const

Text

(

'Game Over'

,

style:

TextStyle

(

color:

whiteTextColor

,

fontSize:

24

,

),

),

const

SizedBox

(

height:

40

),

SizedBox

(

width:

200

,

height:

75

,

child:

ElevatedButton

(

onPressed:

()

{

game

.

reset

();

game

.

overlays

.

remove

(

'GameOver'

);

},

style:

ElevatedButton

.

styleFrom

(

backgroundColor:

whiteTextColor

,

),

child:

const

Text

(

'Play Again'

,

style:

TextStyle

(

fontSize:

28.0

,

color:

blackTextColor

,

),

),

),

),

],

),

),

),

);

}

}

As with the Main Menu, this is all standard Flutter widgets except for the call to remove the overlay and als

game.reset()

which we will create now.

Open

lib/ember_quest.dart

and add / update the following code:

```
@override
Future
<
void
>
onLoad
()
async
{
await
images
.
loadAll
([
'block.png'
,
'ember.png'
,
'ground.png'
,
'heart_half.png'
,
'heart.png'
```

,

'star.png'

,

'water_enemy.png'

,

]);

camera

.

viewfinder

.

anchor

=

Anchor

.

topLeft

;

initializeGame

(

true

);

}

void

initializeGame

(

bool

loadHud

)

{

// Assume that size.x < 3200

final

segmentsToLoad

=

(

size

.

x

/

640

).

ceil

();

segmentsToLoad

.

clamp

(

0

,

segments

.

length

);

for

```
(
var
i
=
0
;
i
<=
segmentsToLoad
;
i
++
)
{
loadGameSegments
(
i
,
(
640
*
i
).
toDouble
());
}
```

_ember

=

EmberPlayer

(

position:

Vector2

(

128

,

canvasSize

.

y

-

128

),

);

add

(

_ember

);

if

(

loadHud

)

{

add

(

Hud

());

}

}

void

reset

()

{

starsCollected

=

0

;

health

=

3

;

initializeGame

(

false

);

}

You may notice that we have added a parameter to the

initializeGame

method which allows us to bypass adding the HUD to the game. This is because in the coming section, wh

reset()

.

Displaying the Menus

¶

To display the menus, add the following code to

lib/main.dart

:

void

main

()

{

runApp

(

GameWidget

<

EmberQuestGame

>

.

controlled

(

gameFactory:

EmberQuestGame

.

new

,

overlayBuilderMap:

{

'MainMenu'

:

(

_

,

game

)

=>

MainMenu

(

game:

game

),

'GameOver'

:

(

_

,

game

)

=>

GameOver

(

game:

game

),

},

initialActiveOverlays:

const

[

'MainMenu'

],

),

);

}

If the menus did not auto-import, add the following:

import

'overlays/game_over.dart'

;

import

'overlays/main_menu.dart'

;

If you run the game now, you should be greeted with the Main Menu overlay. Pressing play will remove it a

Health Check for Game Over

¶

Our last step to finish Ember Quest is to add a game-over mechanism. This is fairly simple but requires us

In

lib/actors/ember.dart

, in the

update

method, add the following:

// If ember fell in pit, then game over.

if

(

position

.

y

>

game

.

size

.

y

+

size

.

y

)

{

game

.

health

=

0

;

}

if

(

game

.

health

<=

0

)

{

removeFromParent

();

}

In

lib/actors/water_enemy.dart

, in the

update

method update the following code:

if

(

position

.

x

<

-

size

.

x

||

game

.

health

<=

0

)

{

removeFromParent

();

}

In

lib/objects/ground_block.dart

, in the

update

method update the following code:

if

(

game

.

health

<=

0

)

{

removeFromParent

();

}

In

lib/objects/platform_block.dart

, in the

update

method update the following code:

if

(

position

.

x

<

-

size

.

x

||

game

.

health

<=

0

)

{

removeFromParent

();

}

In

lib/objects/star.dart

, in the

update

method update the following code:

if

(

position

.

x

<

-

size

.

x

||

game

.

health

<=

0

)

{

removeFromParent

();

}

Finally, in

lib/ember_quest.dart

, add the following

update

method:

@override

void

update

(

double

dt

)

{

if

(

health

<=

0

)

{

overlays

.

add

(

'GameOver'

);

}

super

.

update

(

dt

);

}

Congratulations

¶

You made it! You have a working Ember Quest. Press the button below to see what the resulting code look

Run

actors/ember.dart

1

import

'package:flame/collisions.dart'

;

2

import

'package:flame/components.dart'

;

3

import

'package:flame/effects.dart'

;

```
4
import
'package:flutter/services.dart'
;
5
6
import
'../ember_quest.dart'
;
7
import
'../objects/ground_block.dart'
;
8
import
'../objects/platform_block.dart'
;
9
import
'../objects/star.dart'
;
10
import
'water_enemy.dart'
;
11
```

12

class

EmberPlayer

extends

SpriteAnimationComponent

13

with

KeyboardHandler

,

CollisionCallbacks

,

HasGameReference

<

EmberQuestGame

>

{

14

EmberPlayer

({

15

required

super

.

position

,

16

})

:

super

(

size:

Vector2

.

all

(

64

),

anchor:

Anchor

.

center

);

17

18

final

Vector2

velocity

=

Vector2

.

zero

();

19

final

Vector2

fromAbove

=

Vector2

(

0

,

-

1

);

20

final

double

gravity

=

15

;

21

final

double

jumpSpeed

=

600

;

22

final

double

moveSpeed

=

200

;

23

final

double

terminalVelocity

=

150

;

24

int

horizontalDirection

=

0

;

25

26

bool

hasJumped

=

false

;

27

bool

isOnGround

=

false

;

28

bool

hitByEnemy

=

false

;

29

30

@override

31

Future

<

void

>

onLoad

()

async

{

32

```
animation
=
SpriteAnimation
.
fromFrameData
(
33
game
.
images
.
fromCache
(
'ember.png'
),
34
SpriteAnimationData
.
sequenced
(
35
amount:
4
,
36
textureSize:
```

Vector2

.

all

(

16

),

37

stepTime:

0.12

,

38

),

39

);

40

41

add

(

42

CircleHitbox

(),

43

);

44

}

45

46

@override

47

bool

onKeyEvent

(

RawKeyEvent

event

,

Set

<

LogicalKeyboardKey

>

keysPressed

)

{

48

horizontalDirection

=

0

;

49

horizontalDirection

+=

(

keysPressed

.

contains

(

LogicalKeyboardKey

.

keyA

)

||

50

keysPressed

.

contains

(

LogicalKeyboardKey

.

arrowLeft

))

51

?

-

1

52

:

0

;

53

horizontalDirection

+=

(

keysPressed

.

contains

(

LogicalKeyboardKey

.

keyD

)

||

54

keysPressed

.

contains

(

LogicalKeyboardKey

.

arrowRight

))

55

?

1

56

:

0

;

57

58

hasJumped

=

keysPressed

.

contains

(

LogicalKeyboardKey

.

space

);

59

return

true

;

60

}

61

62

@override

63

void

update

(

double

dt

)

{

64

velocity

.

x

=

horizontalDirection

*

moveSpeed

;

65

game

.

objectSpeed

=

0

;

66

// Prevent ember from going backwards at screen edge.

67

if

(

position

.

x

-

36

<=

0

&&

horizontalDirection

<

0

)

{

68

velocity

.

x

=

0

;

69

}

70

// Prevent ember from going beyond half screen.

71

if

(

position

.

x

+

64

>=

game

.

size

.

x

/

2

&&

horizontalDirection

>

0

)

{

72

velocity

.

x

=

0

;

73

game

.

objectSpeed

=

-

moveSpeed

;

74

}

75

76

// Apply basic gravity.

77

velocity

.

y

+=

gravity

;

78

79

// Determine if ember has jumped.

80

if

(

hasJumped

)

{

81

if

(

isOnGround

)

{

82

velocity

.

y

=

-

jumpSpeed

;

83

isOnGround

=

false

;

84

}

85

hasJumped

=

false

;

86

}

87

88

// Prevent ember from jumping to crazy fast.

89

velocity

.

y

=

velocity

.

y

.

clamp

(

-

jumpSpeed

,

terminalVelocity

);

90

91

// Adjust ember position.

92

position

+=

velocity

*

dt

;

93

94

// If ember fell in pit, then game over.

95

if

(

position

.

y

>

game

.

size

.

y

+

size

.

y

)

{

96

game

.

health

=

0

;

97

}

98

99

if

(

game

.

health

<=

0

)

{

100

removeFromParent

();

101

}

102

103

// Flip ember if needed.

104

if

(

horizontalDirection

<

0

&&

scale

.

x

>

0

)

{

105

flipHorizontally

();

106

}

else

```
if
(
horizontalDirection
>
0
&&
scale
.
x
<
0
)
{
107
flipHorizontally
();
108
}
109
super
.
update
(
dt
);
110
```

}

111

112

@override

113

void

onCollision

(

Set

<

Vector2

>

intersectionPoints

,

PositionComponent

other

)

{

114

if

(

other

is

GroundBlock

||

other

is

PlatformBlock

)

{

115

if

(

intersectionPoints

.

length

==

2

)

{

116

// Calculate the collision normal and separation distance.

117

final

mid

=

(

intersectionPoints

.

elementAt

(

0

)

+

118

intersectionPoints

.

elementAt

(

1

))

/

119

2

;

120

121

final

collisionNormal

=

absoluteCenter

-

mid

;

122

final

separationDistance

=

(

size

.

x

/

2

)

-

collisionNormal

.

length

;

123

collisionNormal

.

normalize

();

124

125

// If collision normal is almost upwards,

126

// ember must be on ground.

127

if

(

fromAbove

.

dot

(

collisionNormal

)

>

0.9

)

{

128

isOnGround

=

true

;

129

}

130

131

// Resolve collision by moving ember along

132

// collision normal by separation distance.

133

position

+=

collisionNormal

.

scaled

(

separationDistance

);

134

}

135

}

136

137

if

(

other

is

Star

)

{

138

other

.

removeFromParent

();

139

game

.

starsCollected

++

;

140

}

141

142

if

(

other

is

WaterEnemy

)

{

143

hit

();

144

}

145

super

.

onCollision

(

intersectionPoints

,

other

);

146

}

147

148

// This method runs an opacity effect on ember

149

// to make it blink.

150

void

hit

()

{

151

if

(

!

hitByEnemy

)

{

152

game

.

health

--

;

153

hitByEnemy

=

true

;

154

}

155

add

(

156

OpacityEffect

.

fadeOut

(

157

EffectController

(

158

alternate:

true

,

159

duration:

0.1

,

160

repeatCount:

5

,

161

),

162

)..

onComplete

=

()

{

163

hitByEnemy

=

false

;

164

},

165

);

166

}

167

}

actors/water_enemy.dart

```
1
import
'package:flame/collisions.dart'
;
2
import
'package:flame/components.dart'
;
3
import
'package:flame/effects.dart'
;
4
5
import
'../ember_quest.dart'
;
6
7
class
WaterEnemy
extends
SpriteAnimationComponent
8
with
HasGameReference
```

<
EmberQuestGame
>
{
9
final
Vector2
gridPosition
;
10
double
xOffset
;
11
12
final
Vector2
velocity
=
Vector2
.
zero
();
13
14
WaterEnemy

({

15

required

this

.

gridPosition

,

16

required

this

.

xOffset

,

17

})

:

super

(

size:

Vector2

.

all

(

64

),

anchor:

Anchor

.

bottomLeft

);

18

19

@override

20

Future

<

void

>

onLoad

()

async

{

21

animation

=

SpriteAnimation

.

fromFrameData

(

22

game

.

images

.

fromCache

(

'water_enemy.png'

),

23

SpriteAnimationData

.

sequenced

(

24

amount:

2

,

25

textureSize:

Vector2

.

all

(

16

),

26

stepTime:

0.70

,

27

),

28

);

29

position

=

Vector2

(

30

(

gridPosition

.

x

*

size

.

x

)

+

xOffset

,

31

game

.

size

.

y

-

(

gridPosition

.

y

*

size

.

y

),

32

);

33

add

(

RectangleHitbox

(

collisionType:

CollisionType

.

passive

));

34

add

(

35

MoveEffect

.

by

(

36

Vector2

(

-

2

*

size

.

x

,

0

),

37

EffectController

(

38

duration:

3

,

```
39
alternate:
true
,
40
infinite:
true
,
41
),
42
),
43
);
44
}
45
46
@override
47
void
update
(
double
dt
)
```

{

48

velocity

.

x

=

game

.

objectSpeed

;

49

position

+=

velocity

*

dt

;

50

if

(

position

.

x

<

-

size

.

x

||

game

.

health

<=

0

)

{

51

removeFromParent

();

52

}

53

super

.

update

(

dt

);

54

}

55

}

ember_quest.dart

1

import

'package:flame/components.dart'

;

2

import

'package:flame/events.dart'

;

3

import

'package:flame/game.dart'

;

4

import

'package:flutter/material.dart'

;

5

6

import

'actors/ember.dart'

;

7

import

'actors/water_enemy.dart'

;

8

import

'managers/segment_manager.dart'

;

9

import

'objects/ground_block.dart'

;

10

import

'objects/platform_block.dart'

;

11

import

'objects/star.dart'

;

12

import

'overlays/hud.dart'

;

13

14

class

EmberQuestGame

extends

FlameGame

15

with

HasCollisionDetection

,

HasKeyboardHandlerComponents

{

16

EmberQuestGame

();

17

18

late

EmberPlayer

_ember

;

19

late

double

lastBlockXPosition

=

0.0

;

20

late

UniqueKey

lastBlockKey

;

21

22

int

starsCollected

=

0

;

23

int

health

=

3

;

24

double

cloudSpeed

=

0.0

;

25

double

objectSpeed

=

0.0

;

```
26
27
@override
28
Future
<
void
>
onLoad
()
async
{
29
//debugMode = true; // Uncomment to see the bounding boxes
30
await
images
.
loadAll
([
31
'block.png'
,
32
'ember.png'
,
```

33

'ground.png'

,

34

'heart_half.png'

,

35

'heart.png'

,

36

'star.png'

,

37

'water_enemy.png'

,

38

]);

39

camera

.

viewfinder

.

anchor

=

Anchor

.

topLeft

;

40

41

initializeGame

(

loadHud:

true

);

42

}

43

44

@override

45

void

update

(

double

dt

)

{

46

if

(

health

<=

0

)

{

47

overlays

.

add

(

'GameOver'

);

48

}

49

super

.

update

(

dt

);

50

}

51

52

@override

53

```
Color
backgroundColor
()
{
54
return
const
Color
.
fromARGB
(
255
,
173
,
223
,
247
);
55
}
56
57
void
loadGameSegments
(
```

int

segmentIndex

,

double

xPositionOffset

)

{

58

for

(

final

block

in

segments

[

segmentIndex

])

{

59

switch

(

block

.

blockType

)

{

60

case

GroundBlock:

61

world

.

add

(

62

GroundBlock

(

63

gridPosition:

block

.

gridPosition

,

64

xOffset:

xPositionOffset

,

65

),

66

);

67

break

;

68

case

PlatformBlock:

69

world

.

add

(

70

PlatformBlock

(

71

gridPosition:

block

.

gridPosition

,

72

xOffset:

xPositionOffset

,

73

),

74

);

75

break

;

76

case

Star:

77

world

.

add

(

78

Star

(

79

gridPosition:

block

.

gridPosition

,

80

xOffset:

xPositionOffset

,

81

),

82

);

83

break

;

84

case

WaterEnemy:

85

world

.

add

(

86

WaterEnemy

(

87

gridPosition:

block

.

gridPosition

,

88

xOffset:

xPositionOffset

,

89

),

90

);

91

break

;

92

}

93

}

94

}

95

96

void

initializeGame

({

required

bool

loadHud

})

{

97

// Assume that size.x < 3200

98

final

segmentsToLoad

=

(

size

.

x

/

640

).

ceil

();

99

segmentsToLoad

.

clamp

(

0

,

segments

.

length

);

100

101

for

(

var

i

=

0

;

i

<=

segmentsToLoad

;

i

++

)

{

102

loadGameSegments

(

i

,

(

640

*

i

).

toDouble

());

103

}

104

105

_ember

=

EmberPlayer

(

106

position:

Vector2

(

128

,

canvasSize

.

y

-

128

),

107

);

108

world

.

```
add
(
_ember
);
109
if
(
loadHud
)
{
110
camera
.
viewport
.
add
(
Hud
());
111
}
112
}
113
114
void
```

reset

()

{

115

starsCollected

=

0

;

116

health

=

3

;

117

initializeGame

(

loadHud:

false

);

118

}

119

}

main.dart

1

import

'package:flame/game.dart'

;

2

import

'package:flutter/material.dart'

;

3

4

import

'ember_quest.dart'

;

5

import

'overlays/game_over.dart'

;

6

import

'overlays/main_menu.dart'

;

7

8

void

main

()

{

9

runApp

(

10

GameWidget

<

EmberQuestGame

>

.

controlled

(

11

gameFactory:

EmberQuestGame

.

new

,

12

overlayBuilderMap:

{

13

'MainMenu'

:

(

_

,

game

)

=>

MainMenu

(

game:

game

),

14

'GameOver'

:

(

_

,

game

)

=>

GameOver

(

game:

game

),

15

},

16

initialActiveOverlays:

const

[

'MainMenu'

],

17

),

18

);

19

}

managers/segment_manager.dart

1

import

'package:flame/components.dart'

;

2

3

import

'../actors/water_enemy.dart'

;

4

import

'../objects/ground_block.dart'

;

5

import

'../objects/platform_block.dart'

;

6

import

'../objects/star.dart'

;

7

8

class

Block

{

9

// gridPosition position is always segment based X,Y.

10

// 0,0 is the bottom left corner.

11

// 10,10 is the upper right corner.

12

final

Vector2

gridPosition

;

13

final

Type

blockType

;

14

Block

(

this

.

gridPosition

,

this

.

blockType

);

15

}

16

17

final

segments

=

[

18

segment0

,

19

segment1

,

20

segment2

,

21

segment3

,

22

segment4

,

23

];

24

25

final

segment0

=

[

26

Block

(

Vector2

(

0

,

0

),

GroundBlock

),

27

Block

(

Vector2

(

1

,

0

),

GroundBlock

),

28

Block

(

Vector2

(

2

,

0

),

GroundBlock

),

29

Block

(

Vector2

(

3

,

0

),

GroundBlock

),

30

Block

(

Vector2

(

4

,

0

),

GroundBlock

),

31

Block

(

Vector2

(

5

,

0

),

GroundBlock

),

32

Block

(

Vector2

(

5

,

1

),

WaterEnemy

),

33

Block

(

Vector2

(

5

,

3

),

PlatformBlock

),

34

Block

(

Vector2

(

6

,

0

),

GroundBlock

),

35

Block

(

Vector2

(

6

,

3

),

PlatformBlock

),

36

Block

(

Vector2

(

7

,

0

),

GroundBlock

),

37

Block

(

Vector2

(

7

,

3

),

PlatformBlock

),

38

Block

(

Vector2

(

8

,

0

),

GroundBlock

),

39

Block

(

Vector2

(

8

,

3

),

PlatformBlock

),

40

Block

(

Vector2

(

9

,

0

),

GroundBlock

),

41

];

42

43

final

segment1

=

[

44

Block

(

Vector2

(

0

,

0

),

GroundBlock

),

45

Block

(

Vector2

(

1

,

0

),

GroundBlock

),

46

Block

(

Vector2

(

1

,

1

),

PlatformBlock

),

47

Block

(

Vector2

(

1

,

2

),

PlatformBlock

),

48

Block

(

Vector2

(

1

,

3

),

PlatformBlock

),

49

Block

(

Vector2

(

2

,

6

),

PlatformBlock

),

50

Block

(

Vector2

(

3

,

6

),

PlatformBlock

),

51

Block

(

Vector2

(

6

,

5

),

PlatformBlock

),

52

Block

(

Vector2

(

7

,

5

),

PlatformBlock

),

53

Block

(

Vector2

(

7

,

7

),

Star

),

54

Block

(

Vector2

(

8

,

0

),

GroundBlock

),

55

Block

(

Vector2

(

8

,

1

),

PlatformBlock

),

56

Block

(

Vector2

(

8

,

5

),

PlatformBlock

),

57

Block

(

Vector2

(

8

,

6

),

WaterEnemy

),

58

Block

(

Vector2

(

9

,

0

),

GroundBlock

),

59

];

60

61

final

segment2

=

[

62

Block

(

Vector2

(

0

,

0

),

GroundBlock

),

63

Block

(

Vector2

(

1

,

0

),

GroundBlock

),

64

Block

(

Vector2

(

2

,

0

),

GroundBlock

),

65

Block

(

Vector2

(

3

,

0

),

GroundBlock

),

66

Block

(

Vector2

(

3

,

3

),

PlatformBlock

),

67

Block

(

Vector2

(

4

,

0

),

GroundBlock

),

68

Block

(

Vector2

(

4

,

3

),

PlatformBlock

),

69

Block

(

Vector2

(

5

,

0

),

GroundBlock

),

70

Block

(

Vector2

(

5

,

3

),

PlatformBlock

),

71

Block

(

Vector2

(

5

,

4

),

WaterEnemy

),

72

Block

(

Vector2

(

6

,

0

),

GroundBlock

),

73

Block

(

Vector2

(

6

,

3

),

PlatformBlock

),

74

Block

(

Vector2

(

6

,

4

),

PlatformBlock

),

75

Block

(

Vector2

(

6

,

5

),

PlatformBlock

),

76

Block

(

Vector2

(

6

,

7

),

Star

),

77

Block

(

Vector2

(

7

,

0

),

GroundBlock

),

78

Block

(

Vector2

(

8

,

0

),

GroundBlock

),

79

Block

(

Vector2

(

9

,

0

),

GroundBlock

),

80

];

81

82

final

segment3

=

[

83

Block

(

Vector2

(

0

,

0

),

GroundBlock

),

84

Block

(

Vector2

(

1

,

0

),

GroundBlock

),

85

Block

(

Vector2

(

1

,

1

),

WaterEnemy

),

86

Block

(

Vector2

(

2

,

0

),

GroundBlock

),

87

Block

(

Vector2

(

2

,

1

),

PlatformBlock

),

88

Block

(

Vector2

(

2

,

2

),

PlatformBlock

),

89

Block

(

Vector2

(

4

,

4

),

PlatformBlock

),

90

Block

(

Vector2

(

6

,

6

),

PlatformBlock

),

91

Block

(

Vector2

(

7

,

0

),

GroundBlock

),

92

Block

(

Vector2

(

7

,

1

),

PlatformBlock

),

93

Block

(

Vector2

(

8

,

0

),

GroundBlock

),

94

Block

(

Vector2

(

8

,

8

),

Star

),

95

Block

(

Vector2

(

9

,

0

),

GroundBlock

),

96

];

97

98

final

segment4

=

[

99

Block

(

Vector2

(

0

,

0

),

GroundBlock

),

100

Block

(

Vector2

(

1

,

0

),

GroundBlock

),

101

Block

(

Vector2

(

2

,

0

),

GroundBlock

),

102

Block

(

Vector2

(

2

,

3

),

PlatformBlock

),

103

Block

(

Vector2

(

3

,

0

),

GroundBlock

),

104

Block

(

Vector2

(

3

,

1

),

WaterEnemy

),

105

Block

(

Vector2

(

3

,

3

),

PlatformBlock

),

106

Block

(

Vector2

(

4

,

0

),

GroundBlock

),

107

Block

(

Vector2

(

5

,

0

),

GroundBlock

),

108

Block

(

Vector2

(

5

,

5

),

PlatformBlock

),

109

Block

(

Vector2

(

6

,

0

),

GroundBlock

),

110

Block

(

Vector2

(

6

,

5

),

PlatformBlock

),

111

Block

(

Vector2

(

6

,

7

),

Star

),

112

Block

(

Vector2

(

7

,

0

),

GroundBlock

),

113

Block

(

Vector2

(

8

,

0

),

GroundBlock

),

114

Block

(

Vector2

(

8

,

3

),

PlatformBlock

),

115

Block

(

Vector2

(

9

,

0

),

GroundBlock

),

116

Block

(

Vector2

(

9

,

1

),

WaterEnemy

),

117

Block

(

Vector2

(

9

,

3

),

PlatformBlock

),

118

];

objects/ground_block.dart

1

import

'dart:math'

;

2

3

import

'package:flame/collisions.dart'

;

4

import

'package:flame/components.dart'

;

5

import

'package:flutter/material.dart'

;

6

7

import

'../ember_quest.dart'

;

8

import

'../managers/segment_manager.dart'

;

9

10

class

GroundBlock

extends

SpriteComponent

11

with

HasGameReference

<

EmberQuestGame

>

{

12

final

Vector2

gridPosition

;

13

double

xOffset

;

14

15

final

UniqueKey

_blockKey

=

UniqueKey

();

16

final

Vector2

velocity

=

Vector2

.

zero

();

17

18

GroundBlock

({

19

required

this

.

gridPosition

,

20

required

this

.

xOffset

,

21

})

:

super

(

size:

Vector2

.

all

(

64

),

anchor:

Anchor

.

bottomLeft

);

22

23

@override

24

Future

<

void

>

onLoad

()

async

{

25

final

groundImage

=

game

.

images

.

fromCache

(

'ground.png'

);

26

sprite

=

Sprite

(

groundImage

);

27

position

=

Vector2

(

28

(

gridPosition

.

x

*

size

.

x

)

+

xOffset

,

29

game

.

size

.

y

-

(

gridPosition

.

y

*

size

.

y

),

30

);

31

add

(

RectangleHitbox

(

collisionType:

CollisionType

.

passive

));

32

if

(

gridPosition

.

x

==

9

&&

position

.

x

>

game

.

lastBlockXPosition

)

{

33

game

.

lastBlockKey

=

_blockKey

;

34

game

.

lastBlockXPosition

=

position

.

x

+

size

.

x

;

```
35
}
36
}
37
38
@override
39
void
update
(
double
dt
)
{
40
velocity
.
x
=
game
.
objectSpeed
;
41
position
```

+=

velocity

*

dt

;

42

43

if

(

position

.

x

<

-

size

.

x

)

{

44

removeFromParent

();

45

if

(

gridPosition

.

x

==

0

)

{

46

game

.

loadGameSegments

(

47

Random

().

nextInt

(

segments

.

length

),

48

game

.

lastBlockXPosition

,

49

);

50

}

51

}

52

if

(

gridPosition

.

x

==

9

)

{

53

if

(

game

.

lastBlockKey

==

_blockKey

)

{

54

game

.

lastBlockXPosition

=

position

.

x

+

size

.

x

-

10

;

55

}

56

}

57

if

(

game

.

health

<=

0

)

{

58

removeFromParent

();

59

}

60

61

super

.

update

(

dt

);

62

}

63

}

objects/platform_block.dart

1

import

'package:flame/collisions.dart'

;

2

import

'package:flame/components.dart'

;

3

4

import

'../ember_quest.dart'

;

5

6

class

PlatformBlock

extends

SpriteComponent

7

with

HasGameReference

<

EmberQuestGame

>

{

8

final

Vector2

gridPosition

;

9

double

xOffset

;

10

11

final

Vector2

velocity

=

Vector2

.

zero

();

12

13

PlatformBlock

({

14

required

this

.

gridPosition

,

15

required

this

.

xOffset

,

16

})

:

super

(

size:

Vector2

.

all

(

64

),

anchor:

Anchor

.

bottomLeft

);

17

18

@override

19

Future

<

void

>

onLoad

()

async

{

20

final

platformImage

=

game

.

images

.

fromCache

(

'block.png'

);

21

sprite

=

Sprite

(

platformImage

);

22

position

=

Vector2

(

23

(

gridPosition

.

x

*

size

.

x

)

+

xOffset

,

24

game

.

size

.

y

-

(

gridPosition

.

y

*

size

.

y

),

25

);

26

add

(

RectangleHitbox

(

collisionType:

CollisionType

.

passive

));

27

}

28

29

@override

30

void

update

(

double

dt

)

{

31

velocity

.

x

=

game

.

objectSpeed

;

32

position

+=

velocity

*

dt

;

33

if

(

position

.

x

<

-

size

.

x

||

game

.

health

<=

0

)

{

34

removeFromParent

();

35

}

36

super

.

update

(

dt

);

37

}

38

}

objects/star.dart

1

import

'package:flame/collisions.dart'

;

2

import

'package:flame/components.dart'

;

3

import

'package:flame/effects.dart'

;

4

import

'package:flutter/material.dart'

;

5

6

import

'../ember_quest.dart'

;

7

8

class

Star

extends

SpriteComponent

with

HasGameReference

<

EmberQuestGame

>

{

9

final

Vector2

gridPosition

;

10

double

xOffset

;

11

12

final

Vector2

velocity

=

Vector2

.

zero

();

13

14

Star

({

15

required

this

.

gridPosition

,

16

required

this

.

xOffset

,

17

})

:

super

(

size:

Vector2

.

all

(

64

),

anchor:

Anchor

.

center

);

18

19

@override

20

Future

<

void

>

onLoad

()

async

{

21

final

starImage

=

game

.

images

.

fromCache

(

'star.png'

);

22

sprite

=

Sprite

(

starImage

);

23

position

=

Vector2

(

24

(

gridPosition

.

x

*

size

.

x

)

+

xOffset

+

(

size

.

x

/

2

),

25

game

.

size

.

y

-

(

gridPosition

.

y

*

size

.

y

)

-

(

size

.

y

/

2

),

26

);

27

add

(

RectangleHitbox

(

collisionType:

CollisionType

.

passive

));

28

add

(

29

SizeEffect

.

by

(

30

Vector2

(

-

24

,

-

24

),

31

EffectController

(

32

duration:

.

75

,

33

reverseDuration:

.

5

,

34

infinite:

true

,

35

curve:

Curves

.

easeOut

,

36

),

37

),

38

);

39

}

40

41

@override

42

void

update

(

double

dt

)

{

43

velocity

.

x

=

game

.

objectSpeed

;

44

position

+=

velocity

*

dt

;

45

if

(

position

.

x

<

-

size

.

x

||

game

.

health

<=

0

)

{

46

removeFromParent

();

47

}

48

super

.

update

(

dt

);

49

}

50

}

overlays/game_over.dart

1

import

'package:flutter/material.dart'

;

2

3

import

'../ember_quest.dart'

;

4

5

class

GameOver

extends

StatelessWidget

{

6

// Reference to parent game.

7

final

EmberQuestGame

game

;

8

const

GameOver

({

required

this

.

game

,

super

.

key

});

9

10

@override

11

Widget

build

(

BuildContext

context

)

{

12

const

blackTextColor

=

Color

.

fromRGBO

(

0

,

0

,

0

,

1.0

);

13

const

whiteTextColor

=

Color

.

fromRGBO

(

255

,

255

,

255

,

1.0

);

14

15

return

Material

(

16

color:

Colors

.

transparent

,

17

child:

Center

(

18

child:

Container

(

19

padding:

const

EdgeInsets

.

all

(

10.0

),

20

height:

200

,

21

width:

300

,

22

decoration:

const

BoxDecoration

(

23

color:

blackTextColor

,

24

borderRadius:

BorderRadius

.

all

(

25

Radius

.

circular

(

20

),

26

),

27

),

28

child:

Column

(

29

mainAxisAlignment:

MainAxisAlignment

.

center

,

30

children:

[

31

const

Text

(

32

'Game Over'

,

33

style:

TextStyle

(

34

color:

whiteTextColor

,

35

fontSize:

24

,

36

),

37

),

38

const

SizedBox

(

height:

40

),

39

SizedBox

(

40

width:

200

,

41

height:

75

,

42

child:

ElevatedButton

(

43

onPressed:

()

{

44

game

.

reset

();

45

game

.

overlays

.

remove

(

'GameOver'

);

46

},

47

style:

ElevatedButton

.

styleFrom

(

48

backgroundColor:

whiteTextColor

,

49

),

50

child:

const

Text

(

51

'Play Again'

,

52

style:

TextStyle

(

53

fontSize:

28.0

,

54

color:

blackTextColor

,

55

),

56

),

57

),

58

),

59

],

60

),

61

),

62

),

63

);

64

}

65

}

overlays/heart.dart

1

import

'package:flame/components.dart'

;

2

3

import

'../ember_quest.dart'

;

4

5

enum

HeartState

{

6

available

,

7

unavailable

,

8

}

9

10

class

HeartHealthComponent

extends

SpriteGroupComponent

<

HeartState

>

11

with

HasGameReference

<

EmberQuestGame

>

{

12

final

int

heartNumber

;

13

14

HeartHealthComponent

({

15

required

this

.

heartNumber

,

16

required

super

.

position

,

17

required

super

.

size

,

18

super

.

scale

,

19

super

.

angle

,

20

super

.

anchor

,

21

super

.

priority

,

22

});

23

24

@override

25

Future

<

void

>

onLoad

()

async

{

26

await

super

.

onLoad

();

27

final

availableSprite

=

await

game

.

loadSprite

(

28

'heart.png'

,

29

srcSize:

Vector2

.

all

(

32

),

30

);

31

32

final

unavailableSprite

=

await

game

.

loadSprite

(

33

'heart_half.png'

,

34

srcSize:

Vector2

.

all

(

32

),

35

);

36

37

sprites

=

{

38

HeartState

.

available:

availableSprite

,

39

HeartState

.

unavailable:

unavailableSprite

,

40

};

41

42

current

=

HeartState

.

available

;

43

}

44

45

@override

46

void

update

(

double

dt

)

{

47

if

(

game

.

health

<

heartNumber

)

{

48

current

=

HeartState

.

unavailable

;

49

}

else

{

50

current

=

HeartState

.

available

;

51

}

52

super

.

update

(

dt

);

53

}

54

}

overlays/hud.dart

1

import

'package:flame/components.dart'

;

2

import

'package:flutter/material.dart'

;

3

4

import

'../ember_quest.dart'

;

5

import

'heart.dart'

;

6

7

class

Hud

extends

PositionComponent

with

HasGameReference

<

EmberQuestGame

>

{

8

Hud

({

9

super

.

position

,

10

super

.

size

,

11

super

.

scale

,

12

super

.

angle

,

13

super

.

anchor

,

14

super

.

children

,

15

super

.

priority

=

5

,

16

});

17

18

late

TextComponent

_scoreTextComponent

;

19

20

@override

21

Future

<

void

>?

onLoad

()

async

{

22

_scoreTextComponent

=

TextComponent

(

23

text:

'

${

game

.

starsCollected

}

'

,

24

textRenderer:

TextPaint

(

25

style:

const

TextStyle

(

26

fontSize:

32

,

27

color:

Color

.

fromRGBO

(

10

,

10

,

10

,

1

),

28

),

29

),

30

anchor:

Anchor

.

center

,

31

position:

Vector2

(

game

.

size

.

x

-

60

,

20

),

32

);

33

add

(

_scoreTextComponent

);

34

35

final

starSprite

=

await

game

.

loadSprite

(

'star.png'

);

36

add

(

37

SpriteComponent

(

38

sprite:

starSprite

,

39

position:

Vector2

(

game

.

size

.

x

-

100

,

20

),

40

size:

Vector2

.

all

(

32

),

41

anchor:

Anchor

.

center

,

42

),

43

);

44

45

for

(

var

i

=

1

;

i

<=

game

.

health

;

i

++

)

{

46

final

positionX

=

40

*

i

;

47

await

add

(

48

HeartHealthComponent

(

49

heartNumber:

i

,

50

position:

Vector2

(

positionX

.

toDouble

(),

20

),

51

size:

```
Vector2
.
all
(
32
),
52
),
53
);
54
}
55
56
return
super
.
onLoad
();
57
}
58
59
@override
60
void
```

update

(

double

dt

)

{

61

_scoreTextComponent

.

text

=

'

${

game

.

starsCollected

}

'

;

62

super

.

update

(

dt

);

63

}

64

}

overlays/main_menu.dart

1

import

'package:flutter/material.dart'

;

2

3

import

'../ember_quest.dart'

;

4

5

class

MainMenu

extends

StatelessWidget

{

6

// Reference to parent game.

7

final

EmberQuestGame

game

;

8

9

const

MainMenu

({

required

this

.

game

,

super

.

key

});

10

11

@override

12

Widget

build

(

BuildContext

context

)

{

13

const

blackTextColor

=

Color

.

fromRGBO

(

0

,

0

,

0

,

1.0

);

14

const

whiteTextColor

=

Color

.

fromRGBO

(

255

,

255

,

255

,

1.0

);

15

16

return

Material

(

17

color:

Colors

.

transparent

,

18

child:

Center

(

19

child:

Container

(

20

padding:

const

EdgeInsets

.

all

(

10.0

),

21

height:

250

,

22

width:

300

,

23

decoration:

const

BoxDecoration

(

24

color:

blackTextColor

,

25

borderRadius:

BorderRadius

.

all

(

26

Radius

.

circular

(

20

),

27

),

28

),

29

child:

Column

(

30

mainAxisAlignment:

MainAxisAlignment

.

center

,

31

children:

[

32

const

Text

(

33

'Ember Quest'

,

34

style:

TextStyle

(

35

color:

whiteTextColor

,

36

fontSize:

24

,

37

),

38

),

39

const

SizedBox

(

height:

40

),

40

SizedBox

(

41

width:

200

,

42

height:

75

,

43

child:

ElevatedButton

(

44

onPressed:

()

{

45

game

.

overlays

.

remove

(

'MainMenu'

);

46

},

47

style:

ElevatedButton

.

styleFrom

(

48

backgroundColor:

whiteTextColor

,

49

),

50

child:

```
const
Text
(
51
'Play'
,
52
style:
TextStyle
(
53
fontSize:
40.0
,
54
color:
blackTextColor
,
55
),
56
),
57
),
58
),
```

59

const

SizedBox

(

height:

20

),

60

const

Text

(

61

'''Use WASD or Arrow Keys for movement.

62

Space bar to jump.

63

Collect as many stars as you can and avoid enemies!'''

,

64

textAlign:

TextAlign

.

center

,

65

style:

TextStyle

(

66

color:

whiteTextColor

,

67

fontSize:

14

,

68

),

69

),

70

],

71

),

72

),

73

),

74

);

75

}

76

}

Code

Audio

¶

Playing audio is essential for most games, so we made it simple!

First you have to add

flame_audio

to your dependency list in your

pubspec.yaml

file:

dependencies

:

flame_audio

:

VERSION

The latest version can be found on

pub.dev

.

After installing the

flame_audio

package, you can add audio files in the assets section of your

pubspec.yaml

file. Make sure that the audio files exists in the paths that you provide.

The default directory for

FlameAudio

is

assets/audio

(which can be changed by providing your own instance of

AudioCache

).

For the examples below, your

pubspec.yaml

file needs to contain something like this:

flutter

:

assets

:

-

assets/audio/explosion.mp3

-

assets/audio/music.mp3

Then you have the following methods at your disposal:

import

'package:flame_audio/flame_audio.dart'

;

// For shorter reused audio clips, like sound effects

FlameAudio

.

play

(

'explosion.mp3'

);

// For looping an audio file

FlameAudio

```
.
loop
(
'music.mp3'
);
// For playing a longer audio file
FlameAudio
.
playLongAudio
(
'music.mp3'
);
// For looping a longer audio file
FlameAudio
.
loopLongAudio
(
'music.mp3'
);
// For background music that should be paused/played when the pausing/resuming
// the game
FlameAudio
.
bgm
.
play
```

(

'music.mp3'

);

The difference between the

play/loop

and

playLongAudio/loopLongAudio

is that

play/loop

makes use of optimized features that allow sounds to be looped without gaps between their iterations, and

playLongAudio/loopLongAudio

allows for audios of any length to be played, but they do create frame rate drop, and the looped audio will h

You can use

the

Bgm

class

(via

FlameAudio.bgm

) to play looping background music tracks. The

Bgm

class lets Flame automatically manage the pausing and resuming of background music tracks when the ga

You can use

the

AudioPool

class

if you want to fire quick sound effects in a very efficient manner.

AudioPool

will keep a pool of

AudioPlayer

s preloaded with a given sound, and allow you to play them very fast in quick succession.

Some file formats that work across devices and that we recommend are: MP3, OGG and WAV.

This bridge library (flame_audio) uses

audioplayers

in order to allow for playing multiple sounds simultaneously (crucial in a game). You can check the link for a

Both on

play

and

loop

you can pass an additional optional double parameter, the

volume

(defaults to

1.0

).

Both the

play

and

loop

methods return an instance of an

AudioPlayer

from the

audioplayers

lib, that allows you to stop, pause and configure other parameters.

In fact you can always use

AudioPlayer

s directly to gain full control over how your audio is played ? the

FlameAudio

class is just a wrapper for common functionality.

Caching

¶

You can pre-load your assets. Audios need to be stored in the memory the first time they are requested; th

await

FlameAudio

.

audioCache

.

load

(

'explosion.mp3'

);

You can load all your audios in the beginning in your game?s

onLoad

method so that they always play smoothly. To load multiple audio files, use the

loadAll

method:

await

FlameAudio

.

audioCache

.

loadAll

([

'explosion.mp3'

,

'music.mp3'

]);

Finally, you can use the

clear

method to remove a file that has been loaded into the cache:

FlameAudio

.

audioCache

.

clear

(

'explosion.mp3'

);

There is also a

clearCache

method, that clears the whole cache.

This might be useful if, for instance, your game has multiple levels and each has a different set of sounds a

5. Animations, restarting, buttons and a New World

¶

In this chapter we will be showing various ways to make the Klondike game more fun and easier to play. Th

Klondike Draw 1 and Draw 3

Animating and automating moves

Detecting and celebrating a win

Ending and restarting the game

How to implement your own FlameGame world

Simple action-buttons

Anchors and co-ordinates

Random number generation and seeding

Effects and EffectControllers

The Klondike draw

¶

The Klondike patience game (or solitaire game in the USA) has two main variants: Draw 3 and Draw 1. Cu

In Klondike Draw 1 just one card at a time is drawn from the Stock and shown, so every card in it is availab

So how do we implement Klondike Draw 1? Clearly only the Stock and Waste piles are involved, so maybe

klondikeDraw

. In your class declaration add the

HasGameReference<MyGame>

mixin, then write

game.klondikeDraw

wherever you need the value 1 or 3. For class StockPile we will have:

class

StockPile

extends

PositionComponent

with

TapCallbacks

,

HasGameReference

<

KlondikeGame

>

implements

Pile

{

and

@override

void

onTapUp

(

TapUpEvent

event

)

{

final

wastePile

=

parent

!

.

firstChild

<

WastePile

>

()

!

;

if

(

_cards

.

isEmpty

)

{

wastePile

.

removeAllCards

().

reversed

.

forEach

((

card

)

{

card

.

flip

();

acquireCard

(

card

);

});

}

else

{

for

(

var

i

=

0

;

i

<

game

.

klondikeDraw

;

i

++

)

{

if

(

_cards

.

isNotEmpty

)

{

final

card

=

_cards

.

removeLast

();

card

.

flip

();

wastePile

.

acquireCard

(

card

);

}

}

}

}

For class WastePile we will have:

class

WastePile

extends

PositionComponent

with

HasGameReference

<

KlondikeGame

>

implements

Pile

{

and

void

_fanOutTopCards

()

{

if

(

game

.

```
klondikeDraw
==
1
)
{
// No fan-out in Klondike Draw 1.
return
;
}
final
n
=
_cards
.
length
;
for
(
var
i
=
0
;
i
<
n
```

;

i

++

)

{

_cards

[

i

].

position

=

position

;

}

if

(

n

==

2

)

{

_cards

[

1

].

position

.

add

(

_fanOffset

);

}

else

if

(

n

>=

3

)

{

_cards

[

n

-

2

].

position

.

add

(

_fanOffset

);

_cards

[

n

-

1

].

position

.

addScaled

(

_fanOffset

,

2

);

}

}

That makes the Stock and Waste piles play either Klondike Draw 1 or Klondike Draw 3, but how do you tell

// final int klondikeDraw = 3;

final

int

klondikeDraw

=

1

;

This is fine as a temporary measure, when we have not yet decided how to handle some aspect of our des

input

for the player to choose which flavor of Klondike to play, such as a menu screen, a settings screen or a but

Making cards move

¶

In Flame, if we need a component to do something, we use an

Effect

- a special component that can attach to another component, such as a card, and modify its properties. Tha

position

). We also need an

EffectController

, which provides timing for an effect: when to start, how long to go for and what

Curve

to follow. The latter is not a curve in space. It is a time-curve that specifies accelerations and decelerations

To move a card, we will add a

doMove()

method to the

Card

class. It will require a

to

location to go to. Optional parameters are

speed:

(default 10.0),

start:

(default zero),

curve:

(default

Curves.easeOutQuad

) and

onComplete:

(default

null

, i.e. no callback when the move finishes). Speed is in card widths per second. Usually we will provide a ca

after

the animated move. The default

curve:

parameter gives us a fast-in/slow-out move, much as a human player would do. So the following code is ad

Card

class:

void

doMove

(

Vector2

to

,

{

double

speed

=

10.0

,

double

start

=

0.0

,

Curve

curve

=

Curves

.

easeOutQuad

,

VoidCallback

?

onComplete

,

})

{

assert

(

speed

>

0.0

,

'Speed must be > 0 widths per second'

);

final

dt

=

(

to

-

position

).

length

/

(

speed

*

size

.

x

);

assert

(

dt

>

0.0

,

'Distance to move must be > 0'

);

priority

=

100

;

```
add
(
MoveToEffect
(
to
,
EffectController
(
duration:
dt
,
startDelay:
start
,
curve:
curve
),
onComplete:
()
{
onComplete
?
.
call
();
},
```

),

);

}

To make this code compile we need to import

'package:flame/effects.dart'

and

'package:flutter/animation.dart'

at the top of the

components/card.dart

file. That done, we can start using the new method to return the card(s) gracefully to where they came from

bool

_isDragging

=

false

;

Vector2

_whereCardStarted

=

Vector2

(

0

,

0

);

final

List

<
Card
>
attachedCards
=
[];
_isDragging
=
true
;
priority
=
100
;
// Copy each co-ord, else _whereCardStarted changes as the position does.
_whereCardStarted
=
Vector2
(
position
.
x
,
position
.
y

);

if

(

pile

is

TableauPile

)

{

It would be a mistake to write

_whereCardStarted

=

position;

here. In Dart, that would just copy a reference: so

_whereCardStarted

would point to the same data as

position

while the drag occurred and the card?s

position

data changed. We can get around this by copying the card?s

current

X and Y co-ordinates into a

new

Vector2

object.

To animate cards being returned to their original piles after an invalid drag-and-drop, we replace five lines a

onDragEnd()

method with:

// Invalid drop (middle of nowhere, invalid pile or invalid card for pile).

doMove

(

_whereCardStarted

,

onComplete:

()

{

pile

!

.

returnCard

(

this

);

},

);

if

(

attachedCards

.

isNotEmpty

)

{

attachedCards

.

forEach

((

card

)

{

final

offset

=

card

.

position

-

position

;

card

.

doMove

(

_whereCardStarted

+

offset

,

onComplete:

()

{

pile

!

.

returnCard

(

card

);

},

);

});

attachedCards

.

clear

();

}

In each case, we use the default speed of 10 card-widths per second. Notice how the

onComplete:

parameters are used to return each card to the pile where it started. It will then be added back to that pile?

MoveToEffect

and an

EffectController

added to it and these contain all the data needed to get the right card to the right place at the right time. Th

MoveToEffect

and

EffectController

in each moving card automatically get detached and deleted by Flame when the show is over.

Some other automatic and animated moves we can try are dealing the cards, flipping cards from Stock to W

Animating a card-flip

¶

Flutter and Flame do not yet support 3-D effects (as at October 2023), but we can emulate them. To make

void

turnFaceUp

({

double

time

=

0.3

,

double

start

=

0.0

,

VoidCallback

?

onComplete

,

})

{

assert

(

!

_isFaceUpView

,

'Card must be face-down before turning face-up.'

);

assert

(

time

>

0.0

,

'Time to turn card over must be > 0'

);

_isAnimatedFlip

=

true

;

anchor

=

Anchor

.

topCenter

;

position

+=

Vector2

(

width

/

2

,

0

);

priority

=

100

;

add

(

ScaleEffect

.

to

(

Vector2

(

scale

.

x

/

100

,

scale

.

y

),

EffectController

(

startDelay:

start

,

curve:

Curves

.

easeOutSine

,

duration:

time

/

2

,

onMax:

()

{

_isFaceUpView

=

true

;

},

reverseDuration:

time

/

2

,

onMin:

()

{

_isAnimatedFlip

=

false

;

_faceUp

=

true

;

anchor

=

Anchor

.

topLeft

;

position

-=

Vector2

(

width

/

2

,

0

);

},

),

onComplete:

()

{

onComplete

?

.

call

();

},

),

);

}

So how does all this work? We have a default time of 0.3 seconds for the flip to occur, a start time and an o

This is where we use some of the fancier parameters of the

EffectController

class. The

duration:

is set to

time

/

2

and we use an

onMax:

callback, with inline code to change the view to face-up. That callback will happen after

time

/

2

, when the

Effect

(whatever it is) has reached its maximum (i.e. in this case, the view of the card has shrunk to a thin vertical

reverseDuration:

time

/

2

. Everything is reversed: the view of the card expands and the

curve:

of time is applied in reverse order. In total, the timing follows a sine curve from 0 to pi, giving a smooth anir

We are not there yet! If you were to run just the

add()

part of the code, you would see some ugly things happening. Yeah, yeah, been there, done that ? when I w

Anchor

at

topLeft

, which is the point used to set the card?s

position

. We would like the card to flip around its vertical center-line. Easy, just set

anchor

=

Anchor.topCenter

first: that makes the card flip realistically, but it jumps by half a card-width to the left before flipping.

Long story short, see the lines between

assert(

and

add(

and their reversal in the

onMin:

callback, which occurs when the Effect is finished, but before the final

onComplete:

callback. At the beginning, the card?s rendering

priority

is set to 100, so that it will ride above all other cards in the neighborhood. That value cannot always be sav

Pile

is receiving it. So we have made sure that the receiver is always called in the

onComplete:

option, using a method that will adjust the positions and priorities of the cards in the pile.

Last but not least, in the preceding code, notice the use of the variable

_isAnimatedFlip

. This is a

bool

variable defined and initialized near the start of class

Card

in file

components/card.dart

, along with another new

bool

called

_isFaceUpView

. Initially these are set

false

, along with the existing

bool

_faceUp

=

false

variable. What is the significance of these variables? It is

huge

. A few lines further down, we see:

@override

void

render

(

Canvas

canvas

)

{

if

(

_isFaceUpView

)

{

_renderFront

(

canvas

);

}

else

{

_renderBack

(

canvas

);

}

}

This is the code that makes every card visible on the screen, in either face-up or face-down state. At the en

if

statement was

if

(_faceUp)

{

. This was OK because all moves of cards were instantaneous (leaving aside drags and drops): any chang

tick

or soon after. This posed no problem when we started to animate card moves, provided there were no flips

final

card

=

_cards

.

removeLast

();

card

.

flip

;

wastePile

.

acquireCard

(

card

);

And the first thing

wastePile.acquireCard(

does is

assert(card.isFaceUp);

, which fails if an animated flip is keeping the card face-down while the first half of the flip is occurring.

Model and View

¶

Clearly the card cannot be in two states at once: it is not Schrödinger?s cat! We can resolve the dilemma b

_isAnimatedFlip

variable is

true

while there is an animated flip going on, otherwise

false

, and the

Card

class?s

flip()

function is expanded to:

void

flip

()

{

if

(

_isAnimatedFlip

)

{

// Let the animation determine the FaceUp/FaceDown state.

_faceUp

=

_isFaceUpView

;

}

else

{

// No animation: flip and render the card immediately.

_faceUp

=

!

_faceUp

;

_isFaceUpView

=

_faceUp

;

}

}

In the Klondike Tutorial game we are still having to trigger a Model update in the

onComplete:

callback of the flip animation. It might be nice, for impatient or rapid-fingered players, to transfer a card fron

onComplete:

callback. That way, you could flip through the Stock Pile very rapidly, by tapping fast. However, that is beyo

Ending and restarting the game

¶

As it stands, there is no easy way to finish the Klondike Tutorial game and start another, even if you have w

There are various ways to tackle this, depending on the simplicity or complexity of your game and on how l

onLoad()

method is likely to take. They can range from writing your own GameWidget, to doing a few simple re-initia

In the GameWidget case you would supply the Game with a VoidCallback function parameter named

reset

or

restart

. When the callback is invoked, it would use the Flutter conventions of a

StatefulWidget

(e.g.

setState(()

{});)

to force the widget to be rebuilt and replaced, thus releasing all references to the current Game instance, it

Re-initialization should be undertaken only if the operations involved are few and simple. Otherwise coding

Pile

s and then re-shuffle (or not) and re-deal, possibly changing from Klondike Draw 3 to Klondike Draw 1 or vi

Well, that was not as easy as it looked! Re-initializing the

Pile

s and each

Card

was easy enough, but the difficult bit came next? Whether the player wins or restarts without winning, we h

Writing a simple little loop to set each

Card

face-down and use its

doMove

method to make it move independently to the top left fails. It causes one of those ?subtle problems? referre

The problem of the messy Tableau Piles was fixable, but at this point the reviewer of the code and docume

A New World

¶

Start and restart actions

¶

We wish to provide the following actions in the Klondike game:

A first start,

Any number of restarts with a new deal,

Any number of restarts with the same deal as before,

A switch between Klondike Draw 1 and Draw 3 and restart with a new deal, and

Have fun before restarting with a new deal (we?ll keep that as a surprise for later).

The proposal is to have a new KlondikeWorld class, which replaces the default

world

provided by FlameGame. The new world contains (almost) everything we need to play the game and is cre

A stripped-down KlondikeGame class

¶

Here is the new code for the KlondikeGame class (what is left of it).

enum

Action

{

newDeal

,

sameDeal

,

changeDraw

,

haveFun

}

class

KlondikeGame

extends

FlameGame

<

KlondikeWorld

>

{

static

const

double

cardGap

=

175.0

;

static

const

double

topGap

=

500.0

;

static

const

double

cardWidth

=

1000.0

;

static

const

double

cardHeight

=

1400.0

;

static

const

double

cardRadius

=

100.0

;

static

const

double

cardSpaceWidth

=

cardWidth

+

cardGap

;

static

const

double

cardSpaceHeight

=

cardHeight

+

cardGap

;

static

final

Vector2

cardSize

=

Vector2

(

cardWidth

,

cardHeight

);

static

final

cardRRect

=

RRect

.

fromRectAndRadius

(

const

Rect

.

fromLTWH

(

0

,

0

,

cardWidth

,

cardHeight

),

const

Radius

.

circular

(

cardRadius

),

);

// Constant used when creating Random seed.

static

const

int

maxInt

=

0xFFFFFFFE

;

// = (2 to the power 32) - 1

// This KlondikeGame constructor also initiates the first KlondikeWorld.

KlondikeGame

()

:

super

(

world:

KlondikeWorld

());

// These three values persist between games and are starting conditions

// for the next game to be played in KlondikeWorld. The actual seed is

// computed in KlondikeWorld but is held here in case the player chooses

// to replay a game by selecting Action.sameDeal.

int

klondikeDraw

=

1

;

int

seed

=

1

;

Action

action

=

Action

.

newDeal

;

}

Huh! What happened to the

onLoad()

method? And what?s this

seed

thing? And how does KlondikeWorld get into the act? Well, everything that used to be in the

onLoad()

method is now in the

onLoad()

method of KlondikeWorld, which is an extension of the

World

class and is a type of

Component

, so it can have an

onLoad()

method, as can any

Component

type. The content of the method is much the same as before, except that

world.add(

becomes just

add(

. It also brings in some

addButton()

references, but more on these later.

Using a Random Number Generator seed

¶

The

seed

is a common games-programming technique in any programming environment. Usually it allows you to sta

Same

deal

.

Introducing the new KlondikeWorld class

¶

The

class

KlondikeGame

declaration specifies that this extension of the FlameGame class must have a world of type KlondikeWorld

FlameGame<KlondikeWorld>

). Didn?t know we could do that for a game, did we? So how does the first instance of KlondikeWorld get c

KlondikeGame

()

:

super

(

world:

KlondikeWorld

());

The constructor itself is a default constructor, but the colon

:

begins a constructor initialization sequence which creates our world for the first time.

Buttons

¶

We are going to use some buttons to activate the various ways of restarting the Klondike Game. First we e

ButtonComponent

to create class

FlatButton

, adapted from a Flat Button which used to be in Flame?s Examples pages.

ButtonComponent

uses two

PositionComponent

s, one for when the button is in its normal state (up) and one for when it is pressed. The two components a

mounted

and

rendered

alternately as the user presses the button and releases it. To press the button, tap and hold it down.

In our button, the two components are the button?s outlines - the

buttonDown:

one makes the outline of the button turn red when it is pressed, as a warning, because the four button-actio

The four buttons trigger the restart actions described above and are labelled

New

deal

,

Same

deal

,

Draw

1

?

3

and

Have

fun

. Flame also has a

SpriteButtonComponent

, based on two alternating

Sprite

s, a

HudButtonComponent

and an

AdvancedButtonComponent

. For further types of buttons and controllers, it would be best to use a Flutter overlay, menu or settings wid

We use the

addButton()

method, during our world?s

onLoad()

, to set up our four buttons and add them to our

world

.

playAreaSize

=

Vector2

(

7

*

cardSpaceWidth

+

cardGap

,

4

*

cardSpaceHeight

+

topGap

);

final

gameMidX

=

playAreaSize

.

x

/

2

;

addButton

```
(
'New deal'
,
gameMidX
,
Action
.
newDeal
);
addButton
(
'Same deal'
,
gameMidX
+
cardSpaceWidth
,
Action
.
sameDeal
);
addButton
(
'Draw 1 or 3'
,
gameMidX
```

+

2

*

cardSpaceWidth

,

Action

.

changeDraw

);

addButton

(

'Have fun'

,

gameMidX

+

3

*

cardSpaceWidth

,

Action

.

haveFun

);

That places them above our four Foundation piles and centrally aligned with them. The first Foundation pile

Anchors and co-ordinates

¶

The expressions here and in the

addButton()

method may seem odd because the cards and piles all have

Anchor.topLeft

but the buttons have

Anchor.center

. The

position

co-ordinates of a

Card

are where its top-left corner goes, but the

position

co-ordinates of a

FlatButton

are where its

center

goes and the various parts of a

FlatButton

are arranged (internally) around its center. These examples can give us some insight into how co-ordinate

The

deal()

method

¶

The last thing the KlondikeWorld?s

onLoad()

method does is call the

deal()

method to shuffle and deal the cards. This method is now in the KlondikeWorld class and so are the

checkWin()

and

letsCelebrate()

methods, but more about those later. The deal process is the same as before but now includes some anim

void

deal

()

{

assert

(

cards

.

length

==

52

,

'There are

${

cards

.

length

}

cards: should be 52'

);

```
if
(
game
.
action
!=
Action
.
sameDeal
)
{
// New deal: change the Random Number Generator's seed.
game
.
seed
=
Random
().
nextInt
(
KlondikeGame
.
maxInt
);
if
(
```

game

.

action

==

Action

.

changeDraw

)

{

game

.

klondikeDraw

=

(

game

.

klondikeDraw

==

3

)

?

1

:

3

;

}

```
}
// For the "Same deal" option, re-use the previous seed, else use a new one.
cards
.
shuffle
(
Random
(
game
.
seed
));
var
cardToDeal
=
cards
.
length
-
1
;
var
nMovingCards
=
0
;
```

```
for
(
var
i
=
0
;
i
<
7
;
i
++
)
{
for
(
var
j
=
i
;
j
<
7
;
```

j

++

)

{

final

card

=

cards

[

cardToDeal

--

];

card

.

doMove

(

tableauPiles

[

j

].

position

,

start:

nMovingCards

*

0.15

,

onComplete:

()

{

tableauPiles

[

j

].

acquireCard

(

card

);

nMovingCards

--

;

if

(

nMovingCards

==

0

)

{

var

delayFactor

=

0

;

for

(

final

tableauPile

in

tableauPiles

)

{

delayFactor

++

;

tableauPile

.

flipTopCard

(

start:

delayFactor

*

0.15

);

}

}

},

);

nMovingCards

++

;

}

}

for

(

var

n

=

0

;

n

<=

cardToDeal

;

n

++

)

{

stock

.

acquireCard

(

cards

[

n

]);

}

}

First we implement the

Action

value for this game. In the very first game, the KlondikeGame class sets defaults of

Action.newDeal

and

klondikeDraw

=

1

, but after that the player can select an action by pressing and releasing a button and KlondikeWorld saves

Action.newDeal

is selected and saved automatically. The action usually generates and saves a new seed, but that is skippe

Action.sameDeal

. Then we shuffle the cards, using whatever

seed

applies.

The deal logic is the same as we used in Klondike Tutorial Step 4 and the animation is fairly easy. We just

card.doMove(

for each card, with a changing destination and an increasing

start:

value, counting each moving card as it departs. For a few milliseconds after the loops terminate

nMovingCards

will be at a maximum of 28 (i.e. 1 + 2 + 3 + 4 + 5 + 6 + 7) and the remaining 24 cards will go into a properly

Then cards will be arriving over the next second or so and a problem arises. The cards do not necessarily a

j

variable is the Tableau Pile number and

i

is the card?s position in the pile. The King of Hearts for Pile 6 is arriving before the Queen of Clubs that is t

flutter: Move done, i 3, j 6, 6? 5 moving cards.

flutter: Move done, i 4, j 5, 9? 4 moving cards.

flutter: Move done, i 4, j 6, K? 3 moving cards.

flutter: Move done, i 5, j 5, Q? 2 moving cards.

flutter: Move done, i 5, j 6, 2? 1 moving cards.

flutter: Move done, i 6, j 6, 10? 0 moving cards.

flutter: Pile 0 [Q?]

flutter: Pile 1 [J?, Q?]

flutter: Pile 2 [5?, 5?, J?]

flutter: Pile 3 [A?, Q?, A?, 5?]

flutter: Pile 4 [8?, 10?, 7?, 3?, 4?]

flutter: Pile 5 [4?, 8?, 5?, 2?, 9?, Q?]

flutter: Pile 6 [4?, 3?, K?, 6?, K?, 2?, 10?]

So we count off the cards in the

onComplete()

callback code as they arrive. Only when all 28 cards have arrived do we start turning over the last card of e

More animations of moves

¶

The

Card

class?s

doMove()

and

turnFaceUp()

methods have been combined into a doMoveAndFlip() method, which is used to draw cards from the Stock

doMove()

to settle the drop more gracefully. Finally, there is a shortcut to auto-move a card onto its Foundation Pile i

TapCallbacks

to the

Card

class and an

onTapUp()

callback as follows:

```
onTapUp
(
TapUpEvent
event
)
{
if
(
isFaceUp
)
{
final
suitIndex
=
suit
```

.

value

;

if

(

game

.

foundations

[

suitIndex

].

canAcceptCard

(

this

))

{

pile

!

.

removeCard

(

this

);

doMove

(

game

.

foundations

[

suitIndex

].

position

,

onComplete:

()

{

game

.

foundations

[

suitIndex

].

acquireCard

(

this

);

},

);

}

}

else

if

(

pile

is

StockPile

)

{

game

.

stock

.

onTapUp

(

event

);

}

}

If a card is ready to go out, just tap on it and it will move automatically to the correct Foundation Pile for its

Card

object receives the tap first and forwards it on to the

stock

object.

A graphics glitch

¶

If you moved multiple cards from one Tableau Pile to another, the internal code of the

TableauPile

class would formerly (in Tutorial Step 4) move the cards into place abruptly, as soon as the drag-and-drop

acquireCard

call. But this caused some ugly graphics glitches. It appears they were due to

acquireCard

also calling the

layoutCards()

method of

TableauPile

and instantly re-arranging all the cards in the pile, every time a card was acquired. The problem has been s

dropCards

method to

TableauPile

, which mimics some of the existing actions while dovetailing some card animations in as well.

The lesson to be learned is that it is worth giving some attention to animation and time-dependent concerns

Winning the game

¶

You win the game when all cards in all suits, Ace to King, have been moved to the Foundation Piles, 13 ca

isFull

test added to the

FoundationPile

?s

acquireCard()

method, a callback to

KlondikeWorld

and a test as to whether all four Foundations are full. Here is the code:

class

FoundationPile

extends

PositionComponent

implements

Pile

{

FoundationPile

(

int

intSuit

,

this

.

checkWin

,

{

super

.

position

})

:

suit

=

Suit

.

fromInt

(

intSuit

),

super

(

size:

KlondikeGame

.

cardSize

);

final

VoidCallback

checkWin

;

final

Suit

suit

;

final

List

<

Card

>

_cards

=

[];

//#region Pile API

bool

get

isFull

=>

_cards

.

length

==

13

;

void

acquireCard

(

Card

card

)

{

assert

(

card

.

isFaceUp

);

card

.

position

=

position

;

card

.

priority

=

_cards

.

length

;

card

.

pile

=

this

;

_cards

.

add

(

card

);

if

(

isFull

)

{

checkWin

();

// Get KlondikeWorld to check all FoundationPiles.

}

}

void

checkWin

()

{

var

nComplete

=

0

;

for

(

final

f

in

foundations

)

{

if

(

f

.

isFull

)

{

nComplete

++

;

}

}

if

(

nComplete

==

foundations

.

length

)

{

letsCelebrate

();

}

}

It is often possible to calculate whether you can win from a given position of the cards in a Klondike game,

Ending a game and re-starting it

¶

A game ends either after the player wins or they press and release one of the buttons. At that point the Klo

Action

value, a

klondikeDraw

value (1 or 3) and a

seed

from the previous game. Each button has an

onReleased:

callback provided by the

addButton()

method in KlondikeWorld, with code as follows:

onReleased:

()

{

if

(

action

==

Action

.

haveFun

)

{

// Shortcut to the "win" sequence, for Tutorial purposes only.

letsCelebrate

();

}

else

{

// Restart with a new deal or the same deal as before.

game

.

action

=

action

;

game

.

world

=

KlondikeWorld

();

}

},

The

letsCelebrate()

method is normally invoked only when the player wins. The functions of the other three buttons are to set t

Action

value in KlondikeGame and to set

world

in

FlameGame

to refer to a new KlondikeWorld, thus replacing the current one and leaving the former KlondikeWorld?s sto

FlameGame

will continue on to trigger KlondikeWorld?s

onLoad()

method.

The

letsCelebrate()

method ends with similar code, but forces a new deal:

game

.

action

=

Action

.

newDeal

;

game

.

world

=

KlondikeWorld

();

The

Have

fun

button

¶

When you win the Klondike Game, the

letsCelebrate()

method puts on a little display. To save you having to play and win a whole game before you see it ?

and

to test the method, we have provided the

Have

fun

button. Of course a real game could not have such a button?

Well, this is it! The game is now more playable.

We could do more, but this game

is

a Tutorial above all else. Press the buttons below to see what the final code looks like, or to play it live.

But it is also time to have a look at the Ember Tutorial!

Run

components/card.dart

```
1
import
'dart:math'
;
2
import
'dart:ui'
;
3
4
```

```
import
'package:flame/components.dart'
;
5
import
'package:flame/effects.dart'
;
6
import
'package:flame/events.dart'
;
7
import
'package:flutter/animation.dart'
;
8
9
import
'../klondike_game.dart'
;
10
import
'../klondike_world.dart'
;
11
import
```

'../pile.dart'

;

12

import

'../rank.dart'

;

13

import

'../suit.dart'

;

14

import

'foundation_pile.dart'

;

15

import

'stock_pile.dart'

;

16

import

'tableau_pile.dart'

;

17

18

class

Card

extends

PositionComponent

19

with

DragCallbacks

,

TapCallbacks

,

HasWorldReference

<

KlondikeWorld

>

{

20

Card

(

int

intRank

,

int

intSuit

,

{

this

.

isBaseCard

=

false

})

21

:

rank

=

Rank

.

fromInt

(

intRank

),

22

suit

=

Suit

.

fromInt

(

intSuit

),

23

super

(

24

size:

KlondikeGame

.

cardSize

,

25

);

26

27

final

Rank

rank

;

28

final

Suit

suit

;

29

Pile

?

pile

;

30

31

// A Base Card is rendered in outline only and is NOT playable. It can be

32

// added to the base of a Pile (e.g. the Stock Pile) to allow it to handle

33

// taps and short drags (on an empty Pile) with the same behavior and

34

// tolerances as for regular cards (see KlondikeGame.dragTolerance) and using

35

// the same event-handling code, but with different handleTapUp() methods.

36

final

bool

isBaseCard

;

37

38

bool

_faceUp

=

false

;

39

bool

_isAnimatedFlip

=

false

;

40

bool

_isFaceUpView

=

false

;

41

bool

_isDragging

=

false

;

42

Vector2

_whereCardStarted

=

Vector2

(

0

,

0

);

43

44

final

List

<
Card
>
attachedCards
=
[];
45
46
bool
get
isFaceUp
=>
_faceUp
;
47
bool
get
isFaceDown
=>
!
_faceUp
;
48
void
flip
()

{

49

if

(

_isAnimatedFlip

)

{

50

// Let the animation determine the FaceUp/FaceDown state.

51

_faceUp

=

_isFaceUpView

;

52

}

else

{

53

// No animation: flip and render the card immediately.

54

_faceUp

=

!

_faceUp

;

```
55
_isFaceUpView
=
_faceUp
;
56
}
57
}
58
59
@override
60
String
toString
()
=>
rank
.
label
+
suit
.
label
;
// e.g. "Q?" or "10?"
```

```
61
62
//#region Rendering
63
64
@override
65
void
render
(
Canvas
canvas
)
{
66
if
(
isBaseCard
)
{
67
_renderBaseCard
(
canvas
);
68
```

return

;

69

}

70

if

(

_isFaceUpView

)

{

71

_renderFront

(

canvas

);

72

}

else

{

73

_renderBack

(

canvas

);

74

}

75

}

76

77

static

final

Paint

backBackgroundPaint

=

Paint

()

78

..

color

=

const

Color

(

0xff380c02

);

79

static

final

Paint

backBorderPaint1

=

Paint

()

80

..

color

=

const

Color

(

0xffdbaf58

)

81

..

style

=

PaintingStyle

.

stroke

82

..

strokeWidth

=

10

;

83

static

final

Paint

backBorderPaint2

=

Paint

()

84

..

color

=

const

Color

(

0x5CEF971B

)

85

..

style

=

PaintingStyle

.

stroke

86

..

strokeWidth

=

35

;

87

static

final

RRect

cardRRect

=

RRect

.

fromRectAndRadius

(

88

KlondikeGame

.

cardSize

.

toRect

(),

89

const

Radius

.

circular

(

KlondikeGame

.

cardRadius

),

90

);

91

static

final

RRect

backRRectInner

=

cardRRect

.

deflate

(

40

);

92

static

final

Sprite

flameSprite

=

klondikeSprite

(

1367

,

6

,

357

,

501

);

93

94

void

_renderBack

(

Canvas

canvas

)

{

95

canvas

.

drawRRect

(

cardRRect

,

backBackgroundPaint

);

96

canvas

.

drawRRect

(

cardRRect

,

backBorderPaint1

);

97

canvas

.

drawRRect

(

backRRectInner

,

backBorderPaint2

);

98

flameSprite

.

render

(

canvas

,

position:

size

/

2

,

anchor:

Anchor

.

center

);

99

}

100

101

void

_renderBaseCard

(

Canvas

canvas

)

{

102

canvas

.

drawRRect

(

cardRRect

,

backBorderPaint1

);

103

}

104

105

static

final

Paint

frontBackgroundPaint

=

Paint

()

106

..

color

=

const

Color

(

0xff000000

);

107

static

final

Paint

redBorderPaint

=

Paint

()

108

..

color

=

const

Color

(

0xffece8a3

)

109

..

style

=

PaintingStyle

.

stroke

110

..

strokeWidth

=

10

;

111

static

final

Paint

blackBorderPaint

=

Paint

()

112

..

color

=

const

Color

(

0xff7ab2e8

)

113

..

style

=

PaintingStyle

.

stroke

114

..

strokeWidth

=

10

;

115

static

final

blueFilter

=

Paint

()

116

..

colorFilter

=

const

ColorFilter

.

mode

(

117

Color

(

0x880d8bff

),

118

BlendMode

.

srcATop

,

119

);

120

static

final

Sprite

redJack

=

klondikeSprite

(

81

,

565

,

562

,

488

);

121

static

final

Sprite

redQueen

=

klondikeSprite

(

717

,

541

,

486

,

515

);

122

static

final

Sprite

redKing

=

klondikeSprite

(

1305

,

532

,

407

,

549

);

123

static

final

Sprite

blackJack

=

klondikeSprite

(

81

,

565

,

562

,

488

)

124

..

paint

=

blueFilter

;

125

static

final

Sprite

blackQueen

=

klondikeSprite

(

717

,

541

,

486

,

515

)

126

..

paint

=

blueFilter

;

127

static

final

Sprite

blackKing

=

klondikeSprite

(

1305

,

532

,

407

,

549

)

128

..

paint

=

blueFilter

;

129

130

void

_renderFront

(

Canvas

canvas

)

{

131

canvas

.

drawRRect

(

cardRRect

,

frontBackgroundPaint

);

132

canvas

.

drawRRect

(

133

cardRRect

,

134

suit

.

isRed

?

redBorderPaint

:

blackBorderPaint

,

135

);

136

137

final

rankSprite

=

suit

.

isBlack

?

rank

.

blackSprite

:

rank

.

redSprite

;

138

final

suitSprite

=

suit

.

sprite

;

```
139
_drawSprite
(
canvas
,
rankSprite
,
0.1
,
0.08
);
140
_drawSprite
(
canvas
,
suitSprite
,
0.1
,
0.18
,
scale:
0.5
);
141
```

_drawSprite

(

canvas

,

rankSprite

,

0.1

,

0.08

,

rotate:

true

);

142

_drawSprite

(

canvas

,

suitSprite

,

0.1

,

0.18

,

scale:

0.5

,

rotate:

true

);

143

switch

(

rank

.

value

)

{

144

case

1

:

145

_drawSprite

(

canvas

,

suitSprite

,

0.5

,

0.5

,

scale:

2.5

);

146

break

;

147

case

2

:

148

_drawSprite

(

canvas

,

suitSprite

,

0.5

,

0.25

);

149

_drawSprite

(

canvas

,

suitSprite

,

0.5

,

0.25

,

rotate:

true

);

150

break

;

151

case

3

:

152

_drawSprite

(

canvas

,

suitSprite

,

0.5

,

0.2

);

153

_drawSprite

(

canvas

,

suitSprite

,

0.5

,

0.5

);

154

_drawSprite

(

canvas

,

suitSprite

,

0.5

,

0.2

,

rotate:

true

);

155

break

;

156

case

4

:

157

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.25

);

158

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.25

);

159

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.25

,

rotate:

true

);

160

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.25

,

rotate:

true

);

161

break

;

162

case

5

:

163

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.25

);

164

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.25

);

165

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.25

,

rotate:

true

);

166

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.25

,

rotate:

true

);

167

_drawSprite

(

canvas

,

suitSprite

,

0.5

,

0.5

);

168

break

;

169

case

6

:

170

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.25

);

171

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.25

);

172

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.5

);

173

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.5

);

174

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.25

,

rotate:

true

);

175

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.25

,

rotate:

true

);

176

break

;

177

case

7

:

178

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.2

);

179

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.2

);

```
180
_drawSprite
(
canvas
,
suitSprite
,
0.5
,
0.35
);
181
_drawSprite
(
canvas
,
suitSprite
,
0.3
,
0.5
);
182
_drawSprite
(
canvas
```

,

suitSprite

,

0.7

,

0.5

);

183

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.2

,

rotate:

true

);

184

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.2

,

rotate:

true

);

185

break

;

186

case

8

:

187

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.2

);

188

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.2

);

189

_drawSprite

(

canvas

,

suitSprite

,

0.5

,

0.35

);

190

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.5

);

191

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.5

);

192

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.2

,

rotate:

true

);

193

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.2

,

rotate:

true

);

194

_drawSprite

(

canvas

,

suitSprite

,

0.5

,

0.35

,

rotate:

true

);

195

break

;

196

case

9

:

197

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.2

);

198

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.2

);

199

_drawSprite

(

canvas

,

suitSprite

,

0.5

,

0.3

);

200

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.4

);

201

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.4

);

202

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.2

,

rotate:

true

);

203

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.2

,

rotate:

true

);

204

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.4

,

rotate:

true

);

205

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.4

,

rotate:

true

);

206

break

;

207

case

10

:

208

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.2

);

209

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.2

);

```
210
_drawSprite
(
canvas
,
suitSprite
,
0.5
,
0.3
);
211
_drawSprite
(
canvas
,
suitSprite
,
0.3
,
0.4
);
212
_drawSprite
(
canvas
```

,

suitSprite

,

0.7

,

0.4

);

213

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.2

,

rotate:

true

);

214

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.2

,

rotate:

true

);

215

_drawSprite

(

canvas

,

suitSprite

,

0.5

,

0.3

,

rotate:

true

);

216

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.4

,

rotate:

true

);

217

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.4

,

rotate:

true

);

218

break

;

219

case

11

:

220

_drawSprite

(

canvas

,

suit

.

isRed

?

redJack

:

blackJack

,

0.5

,

0.5

);

221

break

;

222

case

12

:

223

_drawSprite

(

canvas

,

suit

.

isRed

?

redQueen

:

blackQueen

,

0.5

,

0.5

);

224

break

;

225

case

13

:

226

_drawSprite

(

canvas

,

suit

.

isRed

?

redKing

:

blackKing

,

0.5

,

0.5

);

227

break

;

228

}

229

}

230

231

void

_drawSprite

(

232

Canvas

canvas

,

233

Sprite

sprite

,

234

double

relativeX

,

235

double

relativeY

,

{

236

double

scale

=

1

,

237

bool

rotate

=

false

,

238

})

{

239

if

(

rotate

)

{

240

canvas

.

save

();

241

canvas

.

translate

(

size

.

x

/

2

,

size

.

y

/

2

);

242

canvas

.

rotate

(

pi

);

243

canvas

.

translate

(

-

size

.

x

/

2

,

-

size

.

y

/

2

);

244

}

245

sprite

.

render

(

246

canvas

,

247

position:

Vector2

(

relativeX

*

size

.

x

,

relativeY

*

size

.

y

),

248

anchor:

Anchor

.

center

,

249

size:

sprite

.

srcSize

.

scaled

(

scale

),

250

);

251

if

(

rotate

)

{

252

canvas

.

restore

();

253

}

254

}

255

256

//#endregion

257

258

//#region Card-Dragging

259

260

@override

261

void

onTapCancel

(

TapCancelEvent

event

)

{

262

if

(

pile

is

StockPile

)

{

263

_isDragging

=

false

;

264

handleTapUp

();

265

}

266

}

267

268

@override

269

void

onDragStart

(

DragStartEvent

event

)

{

270

super

.

onDragStart

(

event

);

271

if

(

pile

is

StockPile

)

{

272

_isDragging

=

false

;

273

return

;

274

}

275

// Clone the position, else _whereCardStarted changes as the position does.

276

_whereCardStarted

=

position

.

clone

();

277

attachedCards

.

clear

();

278

if

(

pile

?

.

canMoveCard

(

this

,

MoveMethod

.

drag

)

??

false

)

{

279

_isDragging

=

true

;

280

priority

=

100

;

281

if

(

pile

is

TableauPile

)

{

282

final

extraCards

=

(

pile

!

as

TableauPile

).

cardsOnTop

(

this

);

283

for

(

final

card

in

extraCards

)

{

284

card

.

priority

=

attachedCards

.

length

+

101

;

285

attachedCards

.

add

(

card

);

286

}

287

}

288

}

289

}

290

291

@override

292

void

onDragUpdate

(

DragUpdateEvent

event

)

{

293

if

(

!

_isDragging

)

{

294

return

;

295

}

296

final

delta

=

event

.

localDelta

;

297

position

.

add

(

delta

);

298

attachedCards

.

forEach

```
((
card
)
=>
card
.
position
.
add
(
delta
));
299
}
300
301
@override
302
void
onDragEnd
(
DragEndEvent
event
)
{
303
```

super

.

onDragEnd

(

event

);

304

if

(

!

_isDragging

)

{

305

return

;

306

}

307

_isDragging

=

false

;

308

309

// If short drag, return card to Pile and treat it as having been tapped.

310

final

shortDrag

=

311

(

position

-

_whereCardStarted

).

length

<

KlondikeGame

.

dragTolerance

;

312

if

(

shortDrag

&&

attachedCards

.

isEmpty

)

{

313

doMove

(

314

_whereCardStarted

,

315

onComplete:

()

{

316

pile

!

.

returnCard

(

this

);

317

// Card moves to its Foundation Pile next, if valid, or it stays put.

318

handleTapUp

();

319

},

320

);

321

return

;

322

}

323

324

// Find out what is under the center-point of this card when it is dropped.

325

final

dropPiles

=

parent

!

326

.

componentsAtPoint

(

position

+

size

/

2

)

327

.

whereType

<

Pile

>

()

328

.

toList

();

329

if

(

dropPiles

.

isNotEmpty

)

{

330

if

(

dropPiles

.

first

.

canAcceptCard

(

this

))

{

331

// Found a Pile: move card(s) the rest of the way onto it.

332

pile

!

.

removeCard

(

this

,

MoveMethod

.

drag

);

333

if

(

dropPiles

.

first

is

TableauPile

)

{

334

// Get TableauPile to handle positions, priorities and moves of cards.

335

(

dropPiles

.

first

as

TableauPile

).

dropCards

(

this

,

attachedCards

);

336

attachedCards

.

clear

();

337

}

else

{

338

// Drop a single card onto a FoundationPile.

339

final

dropPosition

=

(

dropPiles

.

first

as

FoundationPile

).

position

;

340

doMove

(

341

dropPosition

,

342

onComplete:

()

{

343

dropPiles

.

first

.

acquireCard

(

this

);

344

},

345

);

346

}

347

return

;

348

}

349

}

350

351

// Invalid drop (middle of nowhere, invalid pile or invalid card for pile).

352

doMove

(

353

_whereCardStarted

,

354

onComplete:

()

{

355

pile

!

.

returnCard

(

this

);

356

},

357

);

358

if

(

attachedCards

.

isNotEmpty

)

{

359

attachedCards

.

forEach

((

card

)

{

360

final

offset

=

card

.

position

-

position

;

361

card

.

doMove

(

362

_whereCardStarted

+

offset

,

363

onComplete:

()

{

364

pile

!

.

returnCard

(

card

);

365

},

366

);

367

});

368

attachedCards

.

clear

();

369

}

370

}

371

372

//#endregion

373

374

//#region Card-Tapping

375

376

// Tap a face-up card to make it auto-move and go out (if acceptable), but

377

// if it is face-down and on the Stock Pile, pass the event to that pile.

378

379

@override

380

void

onTapUp

(

TapUpEvent

event

)

{

381

handleTapUp

();

382

}

383

384

void

handleTapUp

()

{

385

// Can be called by onTapUp or after a very short (failed) drag-and-drop.

386

// We need to be more user-friendly towards taps that include a short drag.

387

if

(

pile

?

.

canMoveCard

(

this

,

MoveMethod

.

tap

)

??

false

)

{

388

final

suitIndex

=

suit

.

value

;

389

if

(

world

.

foundations

[

suitIndex

].

canAcceptCard

(

this

))

{

390

pile

!

.

removeCard

(

this

,

MoveMethod

.

tap

);

391

doMove

(

392

world

.

foundations

[

suitIndex

].

position

,

393

onComplete:

()

{

394

world

.

foundations

[

suitIndex

].

acquireCard

(

this

);

395

},

396

);

397

}

398

}

```
else
if
(
pile
is
StockPile
)
{
399
world
.
stock
.
handleTapUp
(
this
);
400
}
401
}
402
403
//#endRegion
404
405
```

//#region Effects

406

407

void

doMove

(

408

Vector2

to

,

{

409

double

speed

=

10.0

,

410

double

start

=

0.0

,

411

int

startPriority

=

100

,

412

Curve

curve

=

Curves

.

easeOutQuad

,

413

VoidCallback

?

onComplete

,

414

})

{

415

assert

(

speed

>

0.0

,

'Speed must be > 0 widths per second'

);

416

final

dt

=

(

to

-

position

).

length

/

(

speed

*

size

.

x

);

417

assert

(

dt

>

0

,

'Distance to move must be > 0'

);

418

add

(

419

CardMoveEffect

(

420

to

,

421

EffectController

(

duration:

dt

,

startDelay:

start

,

curve:

curve

),

422

transitPriority:

startPriority

,

423

onComplete:

()

{

424

onComplete

?

.

call

();

425

},

426

),

427

);

428

}

429

430

void

doMoveAndFlip

(

431

Vector2

to

,

{

432

double

speed

=

10.0

,

433

double

start

=

0.0

,

434

Curve

curve

=

Curves

.

easeOutQuad

,

435

VoidCallback

?

whenDone

,

436

})

{

437

assert

(

speed

>

0.0

,

'Speed must be > 0 widths per second'

);

438

final

dt

=

(

to

-

position

).

length

/

(

speed

*

size

.

x

);

439

assert

(

dt

>

0

,

'Distance to move must be > 0'

);

440

priority

=

100

;

441

add

(

442

MoveToEffect

(

443

to

,

444

EffectController

(

duration:

dt

,

startDelay:

start

,

curve:

curve

),

445

onComplete:

()

{

446

turnFaceUp

(

447

onComplete:

whenDone

,

448

);

449

},

450

),

451

);

452

}

453

454

void

turnFaceUp

({

455

double

time

=

0.3

,

456

double

start

=

0.0

,

457

VoidCallback

?

onComplete

,

458

})

{

459

assert

(

!

_isFaceUpView

,

'Card must be face-down before turning face-up.'

);

460

assert

(

time

>

0.0

,

'Time to turn card over must be > 0'

);

461

assert

(

start

>=

0.0

,

'Start tim must be >= 0'

);

462

_isAnimatedFlip

=

true

;

463

anchor

=

Anchor

.

topCenter

;

464

position

+=

Vector2

(

width

/

2

,

0

);

465

priority

=

100

;

466

add

(

467

ScaleEffect

.

to

(

468

Vector2

(

scale

.

x

/

100

,

scale

.

y

),

469

EffectController

(

470

startDelay:

start

,

471

curve:

Curves

.

easeOutSine

,

472

duration:

time

/

2

,

473

onMax:

()

{

474

_isFaceUpView

=

true

;

475

},

476

reverseDuration:

time

/

2

,

477

onMin:

()

{

478

_isAnimatedFlip

=

false

;

479

_faceUp

=

true

;

480

anchor

=

Anchor

.

topLeft

;

481

position

-=

Vector2

(

width

/

2

,

0

);

482

},

483

),

484

onComplete:

()

{

485

onComplete

?

.

call

();

486

},

487

),

488

);

489

}

490

491

//#endregion

492

}

493

494

class

CardMoveEffect

extends

MoveToEffect

{

495

CardMoveEffect

(

496

super

.

destination

,

497

super

.

controller

,

{

498

super

.

onComplete

,

499

this

.

transitPriority

=

100

,

500

});

501

502

final

int

transitPriority

;

503

504

@override

505

void

onStart

()

{

506

super

.

onStart

();

```
// Flame connects MoveToEffect to EffectController.
507
parent
?
.
priority
=
transitPriority
;
508
}
509
}
components/flat_button.dart
1
import
'package:flame/components.dart'
;
2
import
'package:flame/input.dart'
;
3
import
'package:flame/text.dart'
;
```

4

import

'package:flutter/material.dart'

;

5

6

class

FlatButton

extends

ButtonComponent

{

7

FlatButton

(

8

String

text

,

{

9

super

.

size

,

10

super

.

onReleased

,

11

super

.

position

,

12

})

:

super

(

13

button:

ButtonBackground

(

const

Color

(

0xffece8a3

)),

14

buttonDown:

ButtonBackground

(

Colors

.

red

),

15

children:

[

16

TextComponent

(

17

text:

text

,

18

textRenderer:

TextPaint

(

19

style:

TextStyle

(

20

fontSize:

0.5

*

size

!

.

y

,

21

fontWeight:

FontWeight

.

bold

,

22

color:

const

Color

(

0xffdbaf58

),

23

),

24

),

25

position:

size

/

2.0

,

26

anchor:

Anchor

.

center

,

27

),

28

],

29

anchor:

Anchor

.

center

,

30

);

31

}

32

33

class

ButtonBackground

extends

PositionComponent

with

HasAncestor

<

FlatButton

>

{

34

final

_paint

=

Paint

()..

style

=

PaintingStyle

.

stroke

;

35

36

late

double

cornerRadius

;

37

38

ButtonBackground

(

Color

color

)

{

39

_paint

.

color

=

color

;

40

}

41

42

@override

43

void

onMount

()

{

44

super

.

onMount

();

45

size

=

ancestor

.

size

;

46

cornerRadius

=

0.3

*

size

.

y

;

47

_paint

.

strokeWidth

=

0.05

*

size

.

y

;

48

}

49

50

late

final

_background

=

RRect

.

fromRectAndRadius

(

51

size

.

toRect

(),

52

Radius

.

circular

```
(
cornerRadius
),
53
);
54
55
@override
56
void
render
(
Canvas
canvas
)
{
57
canvas
.
drawRRect
(
_background
,
_paint
);
58
```

}

59

}

components/foundation_pile.dart

1

import

'dart:ui'

;

2

3

import

'package:flame/components.dart'

;

4

5

import

'../klondike_game.dart'

;

6

import

'../pile.dart'

;

7

import

'../suit.dart'

;

8

import

'card.dart'

;

9

10

class

FoundationPile

extends

PositionComponent

implements

Pile

{

11

FoundationPile

(

int

intSuit

,

this

.

checkWin

,

{

super

.

position

})

12

:

suit

=

Suit

.

fromInt

(

intSuit

),

13

super

(

size:

KlondikeGame

.

cardSize

);

14

15

final

VoidCallback

checkWin

;

16

17

final

Suit

suit

;

18

final

List

<

Card

>

_cards

=

[];

19

20

//#region Pile API

21

22

bool

get

isFull

=>

_cards

.

length

==

13

;

23

24

@override

25

bool

canMoveCard

(

Card

card

,

MoveMethod

method

)

=>

26

_cards

.

isNotEmpty

&&

card

==

_cards

.

last

&&

method

!=

MoveMethod

.

tap

;

27

28

@override

29

bool

canAcceptCard

(

Card

card

)

{

30

final

topCardRank

=

_cards

.

isEmpty

?

0

:

_cards

.

last

.

rank

.

value

;

31

return

card

.

suit

==

suit

&&

32

card

.

rank

.

value

==

topCardRank

+

1

&&

33

card

.

attachedCards

.

isEmpty

;

34

}

35

36

@override

37

void

removeCard

(

Card

card

,

MoveMethod

method

)

{

38

assert

(

canMoveCard

(

card

,

method

));

39

_cards

.

removeLast

();

40

}

41

42

@override

43

void

returnCard

(

Card

card

)

{

44

card

.

position

=

position

;

45

card

.

priority

=

_cards

.

indexOf

(

card

);

46

}

47

48

@override

```
49
void
acquireCard
(
Card
card
)
{
50
assert
(
card
.
isFaceUp
);
51
card
.
position
=
position
;
52
card
.
priority
```

=

_cards

.

length

;

53

card

.

pile

=

this

;

54

_cards

.

add

(

card

);

55

if

(

isFull

)

{

56

checkWin

();

// Get KlondikeWorld to check all FoundationPiles.

57

}

58

}

59

60

//#endregion

61

62

//#region Rendering

63

64

final

_borderPaint

=

Paint

()

65

..

style

=

PaintingStyle

.

stroke

66

..

strokeWidth

=

10

67

..

color

=

const

Color

(

0x50ffffff

);

68

late

final

_suitPaint

=

Paint

()

69

..

color

=

suit

.

isRed

?

const

Color

(

0x3a000000

)

:

const

Color

(

0x64000000

)

70

..

blendMode

=

BlendMode

.

luminosity

;

71

72

@override

```
73
void
render
(
Canvas
canvas
)
{
74
canvas
.
drawRRect
(
KlondikeGame
.
cardRRect
,
_borderPaint
);
75
suit
.
sprite
.
render
(
```

76

canvas

,

77

position:

size

/

2

,

78

anchor:

Anchor

.

center

,

79

size:

Vector2

.

all

(

KlondikeGame

.

cardWidth

*

0.6

),

80

overridePaint:

_suitPaint

,

81

);

82

}

83

84

//#endregion

85

}

components/stock_pile.dart

1

import

'dart:ui'

;

2

3

import

'package:flame/components.dart'

;

4

5

import

'../klondike_game.dart'

;

6

import

'../pile.dart'

;

7

import

'card.dart'

;

8

import

'waste_pile.dart'

;

9

10

class

StockPile

extends

PositionComponent

11

with

HasGameReference

<

KlondikeGame

>

12

implements

Pile

{

13

StockPile

({

super

.

position

})

:

super

(

size:

KlondikeGame

.

cardSize

);

14

15

/// Which cards are currently placed onto this pile. The first card in the

16

/// list is at the bottom, the last card is on top.

17

final

List

<

Card

>

_cards

=

[];

18

19

//#region Pile API

20

21

@override

22

bool

canMoveCard

(

Card

card

,

MoveMethod

method

)

=>

false

;

23

// Can be moved by onTapUp callback (see below).

24

25

@override

26

bool

canAcceptCard

(

Card

card

)

=>

false

;

27

28

@override

29

void

removeCard

(

Card

card

,

MoveMethod

method

)

=>

30

throw

StateError

(

'cannot remove cards'

);

31

32

@override

33

// Card cannot be removed but could have been dragged out of place.

34

void

returnCard

(

Card

card

)

=>

card

.

priority

=

_cards

.

indexOf

(

card

);

35

36

@override

37

void

acquireCard

(

Card

card

)

{

38

assert

(

card

.

isFaceDown

);

39

card

.

pile

=

this

;

40

card

.

position

=

position

;

41

card

.

priority

=

_cards

.

length

;

42

_cards

.

add

(

card

);

43

}

44

45

//#endregion

46

47

void

handleTapUp

(

Card

card

)

{

48

final

wastePile

=

parent

!

.

firstChild

<

WastePile

>

()

!

;

49

if

(

_cards

.

isEmpty

)

{

50

assert

(

card

.

isBaseCard

,

'Stock Pile is empty, but no Base Card present'

);

51

card

.

position

=

position

;

// Force Base Card (back) into correct position.

52

wastePile

.

removeAllCards

().

reversed

.

forEach

((

card

)

{

53

card

.

flip

();

54

acquireCard

(

card

);

55

});

56

}

else

{

57

for

(

var

i

=

0

;

i

<

game

.

klondikeDraw

;

i

++

)

{

58

if

(

_cards

.

isNotEmpty

)

{

59

final

card

=

_cards

.

removeLast

();

60

card

.

doMoveAndFlip

(

61

wastePile

.

position

,

62

whenDone:

()

{

63

wastePile

.

acquireCard

(

card

);

64

},

65

);

66

}

67

}

68

}

69

}

70

71

//#region Rendering

72

73

final

_borderPaint

=

Paint

()

74

..

style

=

PaintingStyle

.

stroke

75

..

strokeWidth

=

10

76

..

color

=

const

Color

(

0xFF3F5B5D

);

77

final

_circlePaint

=

Paint

()

78

..

style

=

PaintingStyle

.

stroke

79

..

strokeWidth

=

100

80

..

color

=

const

Color

(

0x883F5B5D

);

81

82

@override

83

void

render

(

Canvas

canvas

)

{

84

canvas

.

drawRRect

(

KlondikeGame

.

cardRRect

,

_borderPaint

);

85

canvas

.

drawCircle

(

86

Offset

(

width

/

2

,

height

/

2

),

87

KlondikeGame

.

cardWidth

*

0.3

,

88

_circlePaint

,

89

);

90

}

91

92

//#endregion

93

}

components/tableau_pile.dart

1

import

'dart:ui'

;

2

3

import

'package:flame/components.dart'

;

4

5

import

'../klondike_game.dart'

;

6

import

'../pile.dart'

;

7

import

'card.dart'

;

8

9

class

TableauPile

extends

PositionComponent

implements

Pile

{

10

TableauPile

({

super

.

position

})

:

super

(

size:

KlondikeGame

.

cardSize

);

11

12

/// Which cards are currently placed onto this pile.

13

final

List

<

Card

>

_cards

=

[];

14

final

Vector2

_fanOffset1

=

Vector2

(

0

,

KlondikeGame

.

cardHeight

*

0.05

);

15

final

Vector2

_fanOffset2

=

Vector2

(

0

,

KlondikeGame

.

cardHeight

*

0.20

);

16

17

//#region Pile API

18

19

@override

20

bool

canMoveCard

(

Card

card

,

MoveMethod

method

)

=>

21

card

.

isFaceUp

&&

(

method

==

MoveMethod

.

drag

||

card

==

_cards

.

last

);

22

// Drag can move multiple cards: tap can move last card only (to Foundation).

23

24

@override

25

bool

canAcceptCard

(

Card

card

)

{

26

if

(

_cards

.

isEmpty

)

{

27

return

card

.

rank

.

value

==

13

;

28

}

else

{

29

final

topCard

=

_cards

.

last

;

30

return

card

.

suit

.

isRed

==

!

topCard

.

suit

.

isRed

&&

31

card

.

rank

.

value

==

topCard

.

rank

.

value

-

1

;

32

}

33

}

34

35

@override

36

void

removeCard

(

Card

card

,

MoveMethod

method

)

{

37

assert

(

_cards

.

contains

(

card

)

&&

card

.

isFaceUp

);

38

final

index

=

_cards

.

indexOf

(

card

);

39

_cards

.

removeRange

(

index

,

_cards

.

length

);

40

if

(

_cards

.

isNotEmpty

&&

_cards

.

last

.

isFaceDown

)

{

41

flipTopCard

();

42

return

;

43

}

44

layOutCards

();

45

}

46

47

@override

48
void
returnCard
(
Card
card
)
{
49
card
.
priority
=
_cards
.
indexOf
(
card
);
50
layOutCards
();
51
}
52
53

@override

54

void

acquireCard

(

Card

card

)

{

55

card

.

pile

=

this

;

56

card

.

priority

=

_cards

.

length

;

57

_cards

.

add

(

card

);

58

layOutCards

();

59

}

60

61

//#endregion

62

63

void

dropCards

(

Card

firstCard

,

[

List

<

Card

>

attachedCards

=

const

[]])

{

64

final

cardList

=

[

firstCard

];

65

cardList

.

addAll

(

attachedCards

);

66

Vector2

nextPosition

=

_cards

.

isEmpty

?

position

:

_cards

.

last

.

position

;

67

var

nCardsToMove

=

cardList

.

length

;

68

for

(

final

card

in

cardList

)

{

69

card

.

pile

=

this

;

70

card

.

priority

=

_cards

.

length

;

71

if

(

_cards

.

isNotEmpty

)

{

72

nextPosition

=

73

nextPosition

+

(

card

.

isFaceDown

?

_fanOffset1

:

_fanOffset2

);

74

}

75

_cards

.

add

(

card

);

76

card

.

doMove

(

77

nextPosition

,

78

startPriority:

card

.

priority

,

79

onComplete:

()

{

80

nCardsToMove

--

;

81

if

(

nCardsToMove

==

0

)

{

82

calculateHitArea

();

// Expand the hit-area.

83

}

84

},

85

);

86

}

87

}

88

89

void

flipTopCard

({

double

start

=

0.1

})

{

90

assert

(

_cards

.

last

.

isFaceDown

);

91

_cards

.

last

.

turnFaceUp

(

92

start:

start

,

93

onComplete:

layOutCards

,

94

);

95

}

96

97

void

layOutCards

()

{

98

if

(

_cards

.

isEmpty

)

{

99

calculateHitArea

();

// Shrink hit-area when all cards have been removed.

100

return

;

101

}

102

_cards

[

0

].

position

.

setFrom

(

position

);

103

_cards

[

0

].

priority

=

0

;

104

for

(

var

i

=

1

;

i

<

_cards

.

length

;

i

++

)

{

105

_cards

[

i

].

priority

=

i

;

106

_cards

[

i

].

position

107

..

setFrom

(

_cards

[

i

-

1

].

position

)

108

..

add

(

_cards

[

i

-

1

].

isFaceDown

?

_fanOffset1

:

```
_fanOffset2
);
109
}
110
calculateHitArea
();
// Adjust hit-area to more cards or fewer cards.
111
}
112
113
void
calculateHitArea
()
{
114
height
=
KlondikeGame
.
cardHeight
*
1.5
+
115
```

(

_cards

.

length

<

2

?

0.0

:

_cards

.

last

.

y

-

_cards

.

first

.

y

);

116

}

117

118

List

<
Card
>
cardsOnTop
(
Card
card
)
{
119
assert
(
card
.
isFaceUp
&&
_cards
.
contains
(
card
));
120
final
index
=

_cards

.

indexOf

(

card

);

121

return

_cards

.

getRange

(

index

+

1

,

_cards

.

length

).

toList

();

122

}

123

124

//#region Rendering

125

126

final

_borderPaint

=

Paint

()

127

..

style

=

PaintingStyle

.

stroke

128

..

strokeWidth

=

10

129

..

color

=

const

Color

(

0x50ffffff

);

130

131

@override

132

void

render

(

Canvas

canvas

)

{

133

canvas

.

drawRRect

(

KlondikeGame

.

cardRRect

,

_borderPaint

);

134

}

135

136

//#endregion

137

}

components/waste_pile.dart

1

import

'package:flame/components.dart'

;

2

3

import

'../klondike_game.dart'

;

4

import

'../pile.dart'

;

5

import

'card.dart'

;

6

7

class

WastePile

extends

PositionComponent

8

with

HasGameReference

<

KlondikeGame

>

9

implements

Pile

{

10

WastePile

({

super

.

position

})

:

super

(

size:

KlondikeGame

.

cardSize

);

11

12

final

List

<

Card

>

_cards

=

[];

13

final

Vector2

_fanOffset

=

Vector2

(

KlondikeGame

.

cardWidth

*

0.2

,

0

);

14

15

//#region Pile API

16

17

@override

18

bool

canMoveCard

(

Card

card

,

MoveMethod

method

)

=>

19

_cards

.

isNotEmpty

&&

card

==

_cards

.

last

;

// Tap and drag are both OK.

20

21

@override

22

bool

canAcceptCard

(

Card

card

)

=>

false

;

23

24

@override

25

void

removeCard

(

Card

card

,

MoveMethod

method

)

{

26

assert

(

canMoveCard

(

card

,

method

));

27

_cards

.

removeLast

();

28

_fanOutTopCards

();

29

}

30

```
31
@override
32
void
returnCard
(
Card
card
)
{
33
card
.
priority
=
_cards
.
indexOf
(
card
);
34
_fanOutTopCards
();
35
}
```

```
36
37
@override
38
void
acquireCard
(
Card
card
)
{
39
assert
(
card
.
isFaceUp
);
40
card
.
pile
=
this
;
41
```

card

.

position

=

position

;

42

card

.

priority

=

_cards

.

length

;

43

_cards

.

add

(

card

);

44

_fanOutTopCards

();

45

}

46

47

//#endregion

48

49

List

<

Card

>

removeAllCards

()

{

50

final

cards

=

_cards

.

toList

();

51

_cards

.

clear

();

52

return

cards

;

53

}

54

55

void

_fanOutTopCards

()

{

56

if

(

game

.

klondikeDraw

==

1

)

{

57

// No fan-out in Klondike Draw 1.

58

return

;

59

}

60

final

n

=

_cards

.

length

;

61

for

(

var

i

=

0

;

i

<

n

;

i

++

)

```
{
62
_cards
[
i
].
position
=
position
;
63
}
64
if
(
n
==
2
)
{
65
_cards
[
1
].
position
```

.

add

(

_fanOffset

);

66

}

else

if

(

n

>=

3

)

{

67

_cards

[

n

-

2

].

position

.

add

(

_fanOffset

);

68

_cards

[

n

-

1

].

position

.

addScaled

(

_fanOffset

,

2

);

69

}

70

}

71

}

klondike_game.dart

1

import

'dart:ui'

;

2

3

import

'package:flame/components.dart'

;

4

import

'package:flame/flame.dart'

;

5

import

'package:flame/game.dart'

;

6

7

import

'klondike_world.dart'

;

8

9

enum

Action

{

newDeal

,

sameDeal

,

changeDraw

,

haveFun

}

10

11

class

KlondikeGame

extends

FlameGame

<

KlondikeWorld

>

{

12

static

const

double

cardGap

=

175.0

;

13

static

const

double

topGap

=

500.0

;

14

static

const

double

cardWidth

=

1000.0

;

15

static

const

double

cardHeight

=

1400.0

;

16

static

const

double

cardRadius

=

100.0

;

17

static

const

double

cardSpaceWidth

=

cardWidth

+

cardGap

;

18

static

const

double

cardSpaceHeight

=

cardHeight

+

cardGap

;

19

```
static
final
Vector2
cardSize
=
Vector2
(
cardWidth
,
cardHeight
);
20
static
final
cardRRect
=
RRect
.
fromRectAndRadius
(
21
const
Rect
.
fromLTWH
(
```

0

,

0

,

cardWidth

,

cardHeight

),

22

const

Radius

.

circular

(

cardRadius

),

23

);

24

25

/// Constant used to decide when a short drag is treated as a TapUp event.

26

static

const

double

dragTolerance

=

cardWidth

/

5

;

27

28

/// Constant used when creating Random seed.

29

static

const

int

maxInt

=

0xFFFFFFFE

;

// = (2 to the power 32) - 1

30

31

// This KlondikeGame constructor also initiates the first KlondikeWorld.

32

KlondikeGame

()

:

super

(

world:

KlondikeWorld

());

33

34

// These three values persist between games and are starting conditions

35

// for the next game to be played in KlondikeWorld. The actual seed is

36

// computed in KlondikeWorld but is held here in case the player chooses

37

// to replay a game by selecting Action.sameDeal.

38

int

klondikeDraw

=

1

;

39

int

seed

=

1

;

40

Action

action

=

Action

.

newDeal

;

41

}

42

43

Sprite

klondikeSprite

(

double

x

,

double

y

,

double

width

,

double

height

)

{

44

return

Sprite

(

45

Flame

.

images

.

fromCache

(

'klondike-sprites.png'

),

46

srcPosition:

Vector2

(

x

,

y

),

47

srcSize:

Vector2

(

width

,

height

),

48

);

49

}

klondike_world.dart

1

import

'dart:math'

;

2

3

import

'package:flame/components.dart'

;

4

import

'package:flame/flame.dart'

;

5

6

import

'components/card.dart'

;

```
7
import
'components/flat_button.dart'
;
8
import
'components/foundation_pile.dart'
;
9
import
'components/stock_pile.dart'
;
10
import
'components/tableau_pile.dart'
;
11
import
'components/waste_pile.dart'
;
12
13
import
'klondike_game.dart'
;
14
```

15

class

KlondikeWorld

extends

World

with

HasGameReference

<

KlondikeGame

>

{

16

final

cardGap

=

KlondikeGame

.

cardGap

;

17

final

topGap

=

KlondikeGame

.

topGap

;

18

final

cardSpaceWidth

=

KlondikeGame

.

cardSpaceWidth

;

19

final

cardSpaceHeight

=

KlondikeGame

.

cardSpaceHeight

;

20

21

final

stock

=

StockPile

(

position:

Vector2

(

0.0

,

0.0

));

22

final

waste

=

WastePile

(

position:

Vector2

(

0.0

,

0.0

));

23

final

List

<

FoundationPile

>

foundations

=

[];

24

final

List

<

TableauPile

>

tableauPiles

=

[];

25

final

List

<

Card

>

cards

=

[];

26

late

Vector2

playAreaSize

;

27

28

@override

29

Future

<

void

>

onLoad

()

async

{

30

await

Flame

.

images

.

load

(

'klondike-sprites.png'

);

31

32

stock

.

position

=

Vector2

(

cardGap

,

topGap

);

33

waste

.

position

=

Vector2

(

cardSpaceWidth

+

cardGap

,

topGap

);

34

35

for

(

var

i

=

0

;

i

<

4

;

i

++

)

{

36

foundations

.

add

(

37

FoundationPile

(

38

i

,

39

checkWin

,

40

position:

```
Vector2
((
i
+
3
)
*
cardSpaceWidth
+
cardGap
,
topGap
),
41
),
42
);
43
}
44
for
(
var
i
=
0
```

;

i

<

7

;

i

++

)

{

45

tableauPiles

.

add

(

46

TableauPile

(

47

position:

Vector2

(

48

i

*

cardSpaceWidth

+

cardGap

,

49

cardSpaceHeight

+

topGap

,

50

),

51

),

52

);

53

}

54

55

// Add a Base Card to the Stock Pile, above the pile and below other cards.

56

final

baseCard

=

Card

(

1

,

0

,

isBaseCard:

true

);

57

baseCard

.

position

=

stock

.

position

;

58

baseCard

.

priority

=

-

1

;

59

baseCard

.

pile

=

stock

;

60

stock

.

priority

=

-

2

;

61

62

for

(

var

rank

=

1

;

rank

<=

13

;

rank

++

)

{

63

for

(

var

suit

=

0

;

suit

<

4

;

suit

++

)

{

64

final

card

=

Card

(

rank

,

suit

);

65

card

.

position

=

stock

.

position

;

66

cards

.

add

(

card

);

67

}

68

}

69

70

add

(

stock

);

71

add

(

waste

);

72

addAll

(

foundations

);

73

addAll

(

tableauPiles

);

74

addAll

(

cards

);

75

add

(

baseCard

);

76

77

playAreaSize

=

78

Vector2

(

7

*

cardSpaceWidth

+

cardGap

,

4

*

cardSpaceHeight

+

topGap

);

79

final

gameMidX

=

playAreaSize

.

x

/

2

;

80

81

addButton

(

'New deal'

,

gameMidX

,

Action

.

newDeal

);

82

addButton

(

'Same deal'

,

gameMidX

+

cardSpaceWidth

,

Action

.

sameDeal

);

83

addButton

(

'Draw 1 or 3'

,

gameMidX

+

2

*

cardSpaceWidth

,

Action

.

changeDraw

);

84

addButton

(

'Have fun'

,

gameMidX

+

3

*

cardSpaceWidth

,

Action

.

haveFun

);

85

86

final

camera

=

game

.

camera

;

87

camera

.

viewfinder

.

visibleGameSize

=

playAreaSize

;

88

camera

.

viewfinder

.

position

=

Vector2

(

gameMidX

,

0

);

89

camera

.

viewfinder

.

anchor

=

Anchor

.

topCenter

;

90

91

deal

();

92

}

93

94

void

addButton

(

String

label

,

double

buttonX

,

Action

action

)

{

95

final

button

=

FlatButton

(

96

label

,

97

size:

Vector2

(

KlondikeGame

.

cardWidth

,

0.6

*

topGap

),

98

position:

Vector2

(

buttonX

,

topGap

/

2

),

99

onReleased:

()

```
{
100
if
(
action
==
Action
.
haveFun
)
{
101
// Shortcut to the "win" sequence, for Tutorial purposes only.
102
letsCelebrate
();
103
}
else
{
104
// Restart with a new deal or the same deal as before.
105
game
.
action
```

=

action

;

106

game

.

world

=

KlondikeWorld

();

107

}

108

},

109

);

110

add

(

button

);

111

}

112

113

void

deal

()

{

114

assert

(

cards

.

length

==

52

,

'There are

${

cards

.

length

}

cards: should be 52'

);

115

116

if

(

game

.

action

!=

Action

.

sameDeal

)

{

117

// New deal: change the Random Number Generator's seed.

118

game

.

seed

=

Random

().

nextInt

(

KlondikeGame

.

maxInt

);

119

if

(

game

.

action

==

Action

.

changeDraw

)

{

120

game

.

klondikeDraw

=

(

game

.

klondikeDraw

==

3

)

?

1

:

3

;

121

```
}
122
}
123
// For the "Same deal" option, re-use the previous seed, else use a new one.
124
cards
.
shuffle
(
Random
(
game
.
seed
));
125
126
// Each card dealt must be seen to come from the top of the deck!
127
var
dealPriority
=
1
;
128
```

```
for
(
final
card
in
cards
)
{
129
card
.
priority
=
dealPriority
++
;
130
}
131
132
// Change priority as cards take off: so later cards fly above earlier ones.
133
var
cardToDeal
=
cards
```

.

length

-

1

;

134

var

nMovingCards

=

0

;

135

for

(

var

i

=

0

;

i

<

7

;

i

++

)

```
{
136
for
(
var
j
=
i
;
j
<
7
;
j
++
)
{
137
final
card
=
cards
[
cardToDeal
--
];
```

138

card

.

doMove

(

139

tableauPiles

[

j

].

position

,

140

speed:

15.0

,

141

start:

nMovingCards

*

0.15

,

142

startPriority:

100

+

nMovingCards

,

143

onComplete:

()

{

144

tableauPiles

[

j

].

acquireCard

(

card

);

145

nMovingCards

--

;

146

if

(

nMovingCards

==

0

)

{

147

var

delayFactor

=

0

;

148

for

(

final

tableauPile

in

tableauPiles

)

{

149

delayFactor

++

;

150

tableauPile

.

flipTopCard

(

start:

```
delayFactor
*
0.15
);
151
}
152
}
153
},
154
);
155
nMovingCards
++
;
156
}
157
}
158
for
(
var
n
=
```

0

;

n

<=

cardToDeal

;

n

++

)

{

159

stock

.

acquireCard

(

cards

[

n

]);

160

}

161

}

162

163

void

checkWin

()

{

164

// Callback from a Foundation Pile when it is full (Ace to King).

165

var

nComplete

=

0

;

166

for

(

final

f

in

foundations

)

{

167

if

(

f

.

isFull

)

{

168

nComplete

++

;

169

}

170

}

171

if

(

nComplete

==

foundations

.

length

)

{

172

letsCelebrate

();

173

}

174

```
}
175
176
void
letsCelebrate
({
int
phase
=
1
})
{
177
// Deal won: bring all cards to the middle of the screen (phase 1)
178
// then scatter them to points just outside the screen (phase 2).
179
//
180
// First get the device's screen-size in game co-ordinates, then get the
181
// top-left of the off-screen area that will accept the scattered cards.
182
// Note: The play area is anchored at TopCenter, so topLeft.y is fixed.
183
184
```

final

cameraZoom

=

game

.

camera

.

viewfinder

.

zoom

;

185

final

zoomedScreen

=

game

.

size

/

cameraZoom

;

186

final

screenCenter

=

(

playAreaSize

-

KlondikeGame

.

cardSize

)

/

2

;

187

final

topLeft

=

Vector2

(

188

(

playAreaSize

.

x

-

zoomedScreen

.

x

)

/

2

-

KlondikeGame

.

cardWidth

,

189

-

KlondikeGame

.

cardHeight

,

190

);

191

final

nCards

=

cards

.

length

;

192

final

offscreenHeight

=

zoomedScreen

.

y

+

KlondikeGame

.

cardSize

.

y

;

193

final

offscreenWidth

=

zoomedScreen

.

x

+

KlondikeGame

.

cardSize

.

x

;

194

final

spacing

=

2.0

*

(

offscreenHeight

+

offscreenWidth

)

/

nCards

;

195

196

// Starting points, directions and lengths of offscreen rect's sides.

197

final

corner

=

[

198

Vector2

(

0.0

,

0.0

),

199

Vector2

(

0.0

,

offscreenHeight

),

200

Vector2

(

offscreenWidth

,

offscreenHeight

),

201

Vector2

(

offscreenWidth

,

0.0

),

202

];

203

final

direction

=

[

204

Vector2

(

0.0

,

1.0

),

205

Vector2

(

1.0

,

0.0

),

206

Vector2

(

0.0

,

-

1.0

),

207

```
Vector2
(
-
1.0
,
0.0
),
208
];
209
final
length
=
[
210
offscreenHeight
,
211
offscreenWidth
,
212
offscreenHeight
,
213
offscreenWidth
,
```

```
214
];
215
216
var
side
=
0
;
217
var
cardsToMove
=
nCards
;
218
var
offScreenPosition
=
corner
[
side
]
+
topLeft
;
```

```
219
var
space
=
length
[
side
];
220
var
cardNum
=
0
;
221
222
while
(
cardNum
<
nCards
)
{
223
final
cardIndex
```

=

phase

==

1

?

cardNum

:

nCards

-

cardNum

-

1

;

224

final

card

=

cards

[

cardIndex

];

225

card

.

priority

=

cardIndex

+

1

;

226

if

(

card

.

isFaceDown

)

{

227

card

.

flip

();

228

}

229

// Start cards a short time apart to give a riffle effect.

230

final

delay

=

phase

==

1

?

cardNum

*

0.02

:

0.5

+

cardNum

*

0.04

;

231

final

destination

=

(

phase

==

1

)

?

screenCenter

:

offScreenPosition

;

232

card

.

doMove

(

233

destination

,

234

speed:

(

phase

==

1

)

?

15.0

:

5.0

,

235

start:

delay

,

236

onComplete:

()

{

237

cardsToMove

--

;

238

if

(

cardsToMove

==

0

)

{

239

if

(

phase

==

1

)

{

240

letsCelebrate

(

```
phase:
2
);
241
}
else
{
242
// Restart with a new deal after winning or pressing "Have fun".
243
game
.
action
=
Action
.
newDeal
;
244
game
.
world
=
KlondikeWorld
();
245
```

}

246

}

247

},

248

);

249

cardNum

++

;

250

if

(

phase

==

1

)

{

251

continue

;

252

}

253

254

// Phase 2: next card goes to same side with full spacing, if possible.

255

offScreenPosition

=

offScreenPosition

+

direction

[

side

]

*

spacing

;

256

space

=

space

-

spacing

;

257

if

((

space

<

0.0

)

&&

(

side

<

3

))

{

258

// Out of space: change to the next side and use excess spacing there.

259

side

++

;

260

offScreenPosition

=

corner

[

side

]

+

topLeft

-

direction

[

side

]

*

space

;

261

space

=

length

[

side

]

+

space

;

262

}

263

}

264

}

265

}

main.dart

1

import

'package:flame/game.dart'

;

2

import

'package:flutter/widgets.dart'

;

3

4

import

'klondike_game.dart'

;

5

6

void

main

()

{

7

final

game

=

KlondikeGame

();

8

runApp

(

GameWidget

(

game:

game

));

9

}

pile.dart

1

import

'components/card.dart'

;

2

3

enum

MoveMethod

{

drag

,

tap

}

4

5

abstract

class

Pile

{

6

/// Returns true if the [card] can be taken away from this pile and moved

7

/// somewhere else. A tapping move may need additional validation.

8

bool

canMoveCard

(

Card

card

,

MoveMethod

method

);

9

10

/// Returns true if the [card] can be placed on top of this pile. The [card]

11

/// may have other cards "attached" to it.

12

bool

canAcceptCard

(

Card

card

);

13

14

/// Removes [card] from this pile; this method will only be called for a card

15

/// that both belong to this pile, and for which [canMoveCard] returns true.

16

void

removeCard

(

Card

card

,

MoveMethod

method

);

17

18

/// Places a single [card] on top of this pile. This method will only be

19

/// called for a card for which [canAcceptCard] returns true.

20

void

acquireCard

(

Card

```
card
);
21
22
/// Returns a [card], which already belongs to this pile, to its proper place.
23
void
returnCard
(
Card
card
);
24
}
```

rank.dart

```
1
import
'package:flame/components.dart'
;
2
import
'package:flutter/foundation.dart'
;
3
import
'klondike_game.dart'
```

;

4

5

@immutable

6

class

Rank

{

7

factory

Rank

.

fromInt

(

int

value

)

{

8

assert

(

9

value

>=

1

&&

value

<=

13

,

10

'value is outside of the bounds of what a rank can be'

,

11

);

12

return

_singletons

[

value

-

1

];

13

}

14

15

Rank

.

_

(

16

this

.

value

,

17

this

.

label

,

18

double

x1

,

19

double

y1

,

20

double

x2

,

21

double

y2

,

22

double

w

,

23

double

h

,

24

)

:

redSprite

=

klondikeSprite

(

x1

,

y1

,

w

,

h

),

25

blackSprite

=

klondikeSprite

(

x2

,

y2

,

w

,

h

);

26

27

final

int

value

;

28

final

String

label

;

29

final

Sprite

redSprite

;

30

final

Sprite

blackSprite

;

31

32

static

final

List

<

Rank

>

_singletons

=

[

33

Rank

.

_

(

1

,

'A'

,

335

,

164

,

789

,

161

,

120

,

129

),

34

Rank

.

–

(

2

,

'2'

,

20

,

19

,

15

,

322

,

83

,

125

),

35

Rank

.

–

(

3

,

'3'

,

122

,

19

,

117

,

322

,

80

,

127

),

36

Rank

.

_

(

4

,

'4'

,

213

,

12

,

208

,

315

,

93

,

132

),

37

Rank

.

_

(

5

,

'5'

,

314

,

21

,

309

,

324

,

85

,

125

),

38

Rank

.

–

(

6

,

'6'

,

419

,

17

,

414

,

320

,

84

,

129

),

39

Rank

.

–

(

7

,

'7'

,

509

,

21

,

505

,

324

,

92

,

128

),

40

Rank

.

–

(

8

,

'8'

,

612

,

19

,

607

,

322

,

78

,

127

),

41

Rank

.

_

(

9

,

'9'

,

709

,

19

,

704

,

322

,

84

,

130

),

42

Rank

.

_

(

10

,

'10'

,

810

,

20

,

805

,

322

,

137

,

127

),

43

Rank

.

–

(

11

,

'J'

,

15

,

170

,

469

,

167

,

56

,

126

),

44

Rank

.

–

(

12

,

'Q'

,

92

,

168

,

547

,

165

,

132

,

128

),

45

Rank

.

–

(

13

,

'K'

,

243

,

170

,

696

,

167

,

92

,

123

),

46

];

47

}

suit.dart

1

import

'package:flame/sprite.dart'

;

2

import

'package:flutter/foundation.dart'

;

3

import

'klondike_game.dart'

;

4

5

@immutable

6

class

Suit

{

7

factory

Suit

.

fromInt

(

int

index

)

{

8

assert

(

9

index

>=

0

&&

index

<=

3

,

10

'index is outside of the bounds of what a suit can be'

,

11

);

12

return

_singletons

[

index

];

13

}

14

15

Suit

.

_

(

this

.

value

,

this

.

label

,

double

x

,

double

y

,

double

w

,

double

h

)

16

:

sprite

=

klondikeSprite

(

x

,

y

,

w

,

h

);

17

18

final

int

value

;

19

final

String

label

;

20

final

Sprite

sprite

;

21

22

static

final

List

<

Suit

>

_singletons

=

[

23

Suit

.

–

(

0

,

'?'

,

1176

,

17

,

172

,

183

),

24

Suit

.

–

(

1

,

'?'

,

973

,

14

,

177

,

182

),

25

Suit

.

–

(

2

,

'?'

,

974

,

226

,

184

,

172

),

26

Suit

.

_

(

3

,

'?'

,

1178

,

220

,

176

,

182

),

27

];

28

29

/// Hearts and Diamonds are red, while Clubs and Spades are black.

30

bool

get

isRed

=>

value

<=

1

;

31

bool

get

isBlack

=>

value

>=

2

;

32

}

Code

# Getting Started

¶

## About Flame

¶

Flame is a modular Flutter game engine that provides a complete set of out-of-the-way solutions for games

It provides you with a simple yet effective game loop implementation, and the necessary functionalities that

We also provide stand-alone packages that extend the Flame functionality which can be found in the

Bridge Packages

section.

You can pick and choose whichever parts you want, as they are all independent and modular.

The engine and its ecosystem are constantly being improved by the community, so please feel free to reac

Give us a star if you want to help give the engine exposure and grow the community. :)

## Installation

¶

Add the

flame

package as a dependency in your

pubspec.yaml

by running the following command:

```
flutter pub add flame
```

The latest version can be found on

pub.dev

.

then run

```
flutter
pub
```

get

and you are ready to start using it!

Getting started

¶

There is a set of tutorials that you can follow to get started in the

tutorials folder

.

Simple examples for all features can be found in the

examples folder

.

You can also check out the

awesome flame repository

, it contains quite a lot of good tutorials and articles written by the community to get you started with Flame.

Outside of the scope of the engine

¶

Games sometimes require complex feature sets depending on what the game is all about. Some of these f

Multiplayer (netcode)

¶

Flame doesn?t bundle any network feature, which may be needed to write online multiplayer games.

If you are building a multiplayer game, here are some recommendations of packages/services:

Nakama

: Nakama is an open-source server designed to power modern games and apps.

Firebase

: Provides dozens of services that can be used to write simpler multiplayer experiences.

Supabase

: A cheaper alternative to Firebase, based on Postgres.

# Getting Started

¶

## About Flame

¶

Flame is a modular Flutter game engine that provides a complete set of out-of-the-way solutions for games

It provides you with a simple yet effective game loop implementation, and the necessary functionalities that

We also provide stand-alone packages that extend the Flame functionality which can be found in the

Bridge Packages

section.

You can pick and choose whichever parts you want, as they are all independent and modular.

The engine and its ecosystem are constantly being improved by the community, so please feel free to reac

Give us a star if you want to help give the engine exposure and grow the community. :)

## Installation

¶

Add the

flame

package as a dependency in your

pubspec.yaml

by running the following command:

```
flutter pub add flame
```

The latest version can be found on

pub.dev

.

then run

```
flutter
pub
```

get

and you are ready to start using it!

Getting started

¶

There is a set of tutorials that you can follow to get started in the

tutorials folder

.

Simple examples for all features can be found in the

examples folder

.

You can also check out the

awesome flame repository

, it contains quite a lot of good tutorials and articles written by the community to get you started with Flame.

Outside of the scope of the engine

¶

Games sometimes require complex feature sets depending on what the game is all about. Some of these f

Multiplayer (netcode)

¶

Flame doesn?t bundle any network feature, which may be needed to write online multiplayer games.

If you are building a multiplayer game, here are some recommendations of packages/services:

Nakama

: Nakama is an open-source server designed to power modern games and apps.

Firebase

: Provides dozens of services that can be used to write simpler multiplayer experiences.

Supabase

: A cheaper alternative to Firebase, based on Postgres.

# Keyboard Input

¶

This includes documentation for keyboard inputs.

For other input documents, see also:

Gesture Input

: for mouse and touch pointer gestures

Other Inputs

: For joysticks, game pads, etc.

## Intro

¶

The keyboard API on flame relies on the

Flutter?s Focus widget

.

To customize focus behavior, see

Controlling focus

.

There are two ways a game can react to key strokes; at the game level and at a component level. For each

Game

or

Component

class.

### Receive keyboard events in a game level

¶

To make a

Game

sub class sensitive to key stroke, mix it with

KeyboardEvents

.

After that, it will be possible to override an

onKeyEvent

method.

This method receives two parameters, first the

RawKeyEvent

that triggers the callback in the first place. The second is a set of the currently pressed

LogicalKeyboardKey

.

The return value is a

KeyEventResult

.

KeyEventResult.handled

will tell the framework that the key stroke was resolved inside of Flame and skip any other keyboard handle

GameWidget

.

KeyEventResult.ignored

will tell the framework to keep testing this event in any other keyboard handler widget apart of

GameWidget

. If the event is not resolved by any handler, the framework will trigger

SystemSoundType.alert

.

KeyEventResult.skipRemainingHandlers

is very similar to

.ignored

, apart from the fact that will skip any other handler widget and will straight up play the alert sound.

Minimal example:

```
class
MyGame
extends
FlameGame
with
KeyboardEvents
{
// ...
@override
KeyEventResult
onKeyEvent
(
RawKeyEvent
event
,
Set
<
LogicalKeyboardKey
>
keysPressed
,
)
{
final
```

isKeyDown

=

event

is

RawKeyDownEvent

;

final

isSpace

=

keysPressed

.

contains

(

LogicalKeyboardKey

.

space

);

if

(

isSpace

&&

isKeyDown

)

{

if

(

keysPressed

.

contains

(

LogicalKeyboardKey

.

altLeft

)

||

keysPressed

.

contains

(

LogicalKeyboardKey

.

altRight

))

{

this

.

shootHarder

();

}

else

{

this

.

shoot

();

}

return

KeyEventResult

.

handled

;

}

return

KeyEventResult

.

ignored

;

}

}

Receive keyboard events in a component level

¶

To receive keyboard events directly in components, there is the mixin

KeyboardHandler

.

Similarly to

TapCallbacks

and

DragCallbacks

,

KeyboardHandler

can be mixed into any subclass of

Component

.

KeyboardHandlers must only be added to games that are mixed with

HasKeyboardHandlerComponents

.

?? Note: If

HasKeyboardHandlerComponents

is used, you must remove

KeyboardEvents

from the game mixin list to avoid conflicts.

After applying

KeyboardHandler

, it will be possible to override an

onKeyEvent

method.

This method receives two parameters. First the

RawKeyEvent

that triggered the callback in the first place. The second is a set of the currently pressed

LogicalKeyboardKey

s.

The returned value should be

true

to allow the continuous propagation of the key event among other components. To not allow any other com

false

.

Flame also provides a default implementation called

KeyboardListenerComponent

which can be used to handle keyboard events. Like any other component, it can be added as a child to a

FlameGame

or another

Component

:

For example, imagine a

PositionComponent

which has methods to move on the X and Y axis, then the following code could be used to bind those meth

add

(

KeyboardListenerComponent

(

keyUp:

{

LogicalKeyboardKey

.

keyA:

(

keysPressed

)

{

...

},

LogicalKeyboardKey

.

keyD:

(

keysPressed

)

{

...

},

LogicalKeyboardKey

.

keyW:

(

keysPressed

)

{

...

},

LogicalKeyboardKey

.

keyS:

(

keysPressed

)

{

...

},

},

keyDown:

{

LogicalKeyboardKey

.

keyA:

(

keysPressed

)

{

...

},

LogicalKeyboardKey

.

keyD:

(

keysPressed

)

{

...

},

LogicalKeyboardKey

.

keyW:

(

keysPressed

)

{

...

},

LogicalKeyboardKey

.

keyS:

(

keysPressed

)

{

...

},

},

),

);

Controlling focus

¶

On the widget level, it is possible to use the

FocusNode

API to control whether the game is focused or not.

GameWidget

has an optional

focusNode

parameter that allow its focus to be controlled externally.

By default

GameWidget

has its

autofocus

set to true, which means it will get focused once it is mounted. To override that behavior, set

autofocus

to false.

For a more complete example see

here

.

DialogueChoice

¶

The

DialogueChoice

class represents multiple

Option

lines in the

.yarn

script, which will be presented to the user so that they can make a choice for how the dialogue should proc

DialogueChoice

objects will be delivered to your

DialogueView

with the method

onChoiceStart()

.

Properties

¶

options

List<DialogueOption>

The list of

DialogueOption

s comprising this choice set.

Pointer Events

¶

Note

This document describes the new events API. The old (legacy) approach, which is still supported, is descri

Gesture Input

.

Pointer events

are Flutter?s generalized ?mouse-movement?-type events (for desktop or web).

If you want to interact with mouse movement events within your component or game, you can use the

PointerMoveCallbacks

mixin.

For example:

```
class
MyComponent
extends
PositionComponent
with
PointerMoveCallbacks
{
MyComponent
()
:
super
(
size:
Vector2
```

(

80

,

60

));

@override

void

onPointerMove

(

PointerMoveEvent

event

)

{

// Do something in response to the mouse move (e.g. update coordinates)

}

}

The mixin adds two overridable methods to your component:

onPointerMove

: called when the mouse moves within the component

onPointerMoveStop

: called once if the component was being hovered and the mouse leaves

By default, each of these methods does nothing, they need to be overridden in order to perform any functio

In addition, the component must implement the

containsLocalPoint()

method (already implemented in

PositionComponent

, so most of the time you don?t need to do anything here) ? this method allows Flame to know whether the

Note that only mouse events happening within your component will be proxied along. However,

onPointerMoveStop

will be fired once on the first mouse movement that leaves your component, so you can handle any exit co

HoverCallbacks

¶

If you want to specifically know if your component is being hovered or not, or if you want to hook into hover

HoverCallbacks

.

For example:

class

MyComponent

extends

PositionComponent

with

HoverCallbacks

{

MyComponent

()

:

super

(

size:

Vector2

(

80

,

60

));

@override

void

update

(

double

dt

)

{

// use `isHovered` to know if the component is being hovered

}

@override

void

onHoverEnter

()

{

// Do something in response to the mouse entering the component

}

@override

void

onHoverExit

()

{

// Do something in response to the mouse leaving the component

}

}

Note that you can still listen to the ?raw? onPointerMove methods for additional functionality, just make sur

super

version to enable the

HoverCallbacks

behavior.

Demo

¶

Play with the demo below to see the pointer hover events in action.

pointer_events.dart

1

import

'dart:math'

;

2

3

import

'package:flame/components.dart'

;

4

import

'package:flame/events.dart'

;

5

import

'package:flame/game.dart'

;

6

import

'package:flutter/rendering.dart'

;

7

8

class

PointerEventsGame

extends

FlameGame

with

TapCallbacks

{

9

@override

10

Future

<

void

>

onLoad

()

async

{

11

add

(

HoverTarget

(

Vector2

(

100

,

200

)));

12

add

(

HoverTarget

(

Vector2

(

300

,

300

)));

13

add

(

HoverTarget

(

Vector2

(

400

,

50

)));

14

}

15

16

@override

17

void

onTapDown

(

TapDownEvent

event

)

{

18

add

(

HoverTarget

(

event

.

localPosition

));

19

}

20

}

21

22

class

HoverTarget

extends

PositionComponent

with

HoverCallbacks

{

23

static

final

Random

_random

=

Random

();

24

25

HoverTarget

(

Vector2

position

)

26

:

super

(

27

position:

position

,

28

size:

Vector2

.

all

(

50

),

29

anchor:

Anchor

.

center

,

30

);

31

32

final

_paint

=

Paint

()

33

..

color

=

HSLColor

.

fromAHSL

(

1

,

_random

.

nextDouble

()

*

360

,

1

,

0.8

)

34

.

toColor

()

35

.

withOpacity

(

0.5

);

36

37

@override

38

void

render

(

Canvas

canvas

)

{

39

canvas

.

drawRect

(

size

.

toRect

(),

_paint

);

40

}

41

42

@override

43

void

onHoverEnter

()

{

44

_paint

.

color

=

_paint

.

color

.

withOpacity

(

1

);

45

}

46

47

@override

48

void

onHoverExit

()

{

49

_paint

.

color

=

_paint

.

color

.

withOpacity

(

0.5

);

50

}

51

}

Code

Klondike game tutorial

¶

Klondike

is a popular solitaire card game. In this tutorial we will follow the step-by-step process for coding this game

This tutorial assumes that you have at least some familiarity with common programming concepts, and with

Dart

programming language.

Decorators

¶

Decorators

are classes that can encapsulate certain visual effects and then apply those visual effects to a sequence o

Component

s, but they can be applied to components either manually or via the

HasDecorator

mixin. Likewise, decorators are not

Effect

s, although they can be used to implement certain

Effect

s.

There are a certain number of decorators available in Flame, and it is simple to add one?s own if necessar

Flame built-in decorators

¶

PaintDecorator.blur

¶

decorator_blur.dart

```
1
import
'package:doc_flame_examples/flower.dart'
;
2
import
'package:flame/game.dart'
;
```

3

import

'package:flame/rendering.dart'

;

4

5

class

DecoratorBlurGame

extends

FlameGame

{

6

@override

7

Future

<

void

>

onLoad

()

async

{

8

var

step

=

0

;

9

add

(

10

Flower

(

11

size:

100

,

12

position:

canvasSize

/

2

,

13

onTap:

(

flower

)

{

14

final

decorator

=

flower

.

decorator

;

15

step

++

;

16

if

(

step

==

1

)

{

17

decorator

.

addLast

(

PaintDecorator

.

blur

(

3.0

));

18

}

else

if

(

step

==

2

)

{

19

decorator

.

replaceLast

(

PaintDecorator

.

blur

(

5.0

));

20

}

else

if

(

step

==

3

)

{

21

decorator

.

replaceLast

(

PaintDecorator

.

blur

(

0.0

,

20.0

));

22

}

else

{

23

decorator

.

replaceLast

(

null

);

24

step

=

0

;

25

}

26

},

27

)..

onTapUp

(),

28

);

29

}

30

}

Code

This decorator applies a Gaussian blur to the underlying component. The amount of blur can be different in

final

decorator

=

PaintDecorator

.

blur

(

3.0

);

Possible uses:

soft shadows;

?out-of-focus? objects in the distance or very close to the camera;

motion blur effects;

deemphasize/obscure content when showing a popup dialog;

blurred vision when the character is drunk.

PaintDecorator.grayscale

¶

decorator_grayscale.dart

1

import

'package:doc_flame_examples/flower.dart'

;

2

import

'package:flame/game.dart'

;

3

import

'package:flame/rendering.dart'

;

4

5

class

DecoratorGrayscaleGame

extends

FlameGame

{

6

@override

7

Future

<

void

>

onLoad

()

async

{

8

var

step

=

0

;

9

add

(

10

Flower

(

11

size:

100

,

12

position:

canvasSize

/

2

,

13

onTap:

(

flower

)

{

14

final

decorator

=

flower

.

decorator

;

15

step

++

;

16

if

(

step

==

1

)

{

17

decorator

.

addLast

(

PaintDecorator

.

grayscale

());

18

}

else

if

(

step

==

2

)

{

19

decorator

.

replaceLast

(

PaintDecorator

.

grayscale

(

opacity:

0.5

));

20

}

else

if

(

step

==

3

)

{

21

decorator

.

replaceLast

(

PaintDecorator

.

grayscale

(

opacity:

0.2

));

22

}

else

if

(

step

==

4

)

{

23

decorator

.

replaceLast

(

PaintDecorator

.

grayscale

(

opacity:

0.1

));

24

}

else

{

25

decorator

.

removeLast

();

26

step

=

0

;

27

}

28

},

29

)..

onTapUp

(),

30

);

31

}

32

}

Code

This decorator converts the underlying image into the shades of grey, as if it was a black-and-white photog

opacity

.

final

decorator

=

PaintDecorator

.

grayscale

(

opacity:

0.5

);

Possible uses:

apply to an NPC to turn them into stone, or into a ghost!

apply to a scene to indicate that it is a memory of the past;

black-and-white photos.

PaintDecorator.tint

¶

decorator_tint.dart

1

import

'dart:ui'

;

2

3

import

'package:doc_flame_examples/flower.dart'

;

4

import

'package:flame/game.dart'

;

5

import

'package:flame/rendering.dart'

;

6

7

class

DecoratorTintGame

extends

FlameGame

{

8

@override

9

Future

<

void

>

onLoad

()

async

{

10

var

step

=

0

;

11

add

(

12

Flower

(

13

size:

100

,

14

position:

canvasSize

/

2

,

15

onTap:

(

flower

)

{

16

final

decorator

=

flower

.

decorator

;

17

step

++

;

18

if

(

step

==

1

)

{

19

decorator

.

addLast

(

PaintDecorator

.

tint

(

const

Color

(

0x88FF0000

)));

20

}

else

if

(

step

==

2

)

{

21

decorator

.

replaceLast

(

PaintDecorator

.

tint

(

const

```
Color
(
0x8800FF00
)));
22
}
else
if
(
step
==
3
)
{
23
decorator
.
replaceLast
(
PaintDecorator
.
tint
(
const
Color
(
```

0x88000088

)));

24

}

else

if

(

step

==

4

)

{

25

decorator

.

replaceLast

(

PaintDecorator

.

tint

(

const

Color

(

0x66FFFFFF

)));

```
26
}
else
if
(
step
==
5
)
{
27
decorator
.
replaceLast
(
PaintDecorator
.
tint
(
const
Color
(
0xAA000000
)));
28
}
```

else

{

29

decorator

.

removeLast

();

30

step

=

0

;

31

}

32

},

33

)..

onTapUp

(),

34

);

35

}

36

}

Code

This decorator

tints

the underlying image with the specified color, as if watching it through a colored glass. It is recommended t

color

used by this decorator was semi-transparent, so that you can see the details of the image below.

final

decorator

=

PaintDecorator

.

tint

(

const

Color

(

0xAAFF0000

);

Possible uses:

NPCs affected by certain types of magic;

items/characters in the shadows can be tinted black;

tint the scene red to show bloodlust, or that the character is low on health;

tint green to show that the character is poisoned or sick;

tint the scene deep blue during the night time;

Rotate3DDecorator

¶

decorator_rotate3d.dart

1

import

'package:doc_flame_examples/flower.dart'

;

2

import

'package:flame/game.dart'

;

3

import

'package:flame/rendering.dart'

;

4

5

class

DecoratorRotate3DGame

extends

FlameGame

{

6

@override

7

Future

<

void

>

onLoad

()

async

{

8

var

step

=

0

;

9

final

decorator

=

Rotate3DDecorator

()

10

..

center

=

Vector2

.

all

(

50

)

11

..

perspective

=

0.01

;

12

add

(

13

Flower

(

14

size:

100

,

15

position:

canvasSize

/

2

,

16

decorator:

decorator

,

17

onTap:

(

flower

)

{

18

step

++

;

19

if

(

step

==

1

)

{

20

decorator

.

angleY

=

-

0.8

;

21

}

else

if

(

step

==

2

)

{

22

decorator

.

angleX

=

1.0

;

23

}

else

if

(

step

==

3

)

{

24

decorator

.

angleZ

=

0.2

;

25

}

else

if

(

step

==

4

)

{

26

decorator

.

angleX

=

10

;

27

}

else

if

(

step

==

5

)

{

28

decorator

.

angleY

=

2

;

29

}

else

{

30

decorator

31

..

angleX

=

0

32

..

angleY

=

0

33

..

angleZ

=

0

;

34

step

=

0

;

35

}

36

},

37

)..

onTapUp

(),

38

);

39

}

40

}

Code

This decorator applies a 3D rotation to the underlying component. You can specify the angles of the rotatio

The decorator also supplies the

isFlipped

property, which allows you to determine whether the component is currently being viewed from the front sic

final

decorator

=

Rotate3DDecorator

(

center:

component

.

center

,

angleX:

rotationAngle

,

perspective:

0.002

,

);

Possible uses:

a card that can be flipped over;

pages in a book;

transitions between app routes;

3d falling particles such as snowflakes or leaves.

Shadow3DDecorator

¶

decorator_shadow3d.dart

1

import

'dart:ui'

;

2

3

import

'package:doc_flame_examples/flower.dart'

;

4

import

'package:flame/components.dart'

;

5

import

'package:flame/game.dart'

;

6

import

'package:flame/rendering.dart'

;

7

8

class

DecoratorShadowGame

extends

FlameGame

{

9

@override

10

Color

backgroundColor

()

=>

const

Color

(

0xFFC7C7C7

);

11

12

@override

13

Future

<

void

>

onLoad

()

async

{

14

final

decorator

=

Shadow3DDecorator

(

base:

Vector2

(

0

,

100

));

15

var

step

=

0

;

16

add

(

Grid

());

17

add

(

18

Flower

(

19

size:

100

,

20

position:

canvasSize

/

2

,

21

decorator:

decorator

,

22

onTap:

(

flower

)

{

23

step

++

;

24

if

(

step

==

1

)

{

25

decorator

.

xShift

=

200

;

26

decorator

.

opacity

=

0.5

;

27

}

else

if

(

step

==

2

)

{

28

decorator

.

xShift

=

400

;

29

decorator

.

yScale

=

3

;

30

decorator

.

blur

=

1

;

31

}

else

if

(

step

==

3

)

{

32

decorator

.

angle

=

1.7

;

33

decorator

.

blur

=

2

;

34

}

else

if

(

step

==

4

)

{

35

decorator

.

ascent

=

20

;

36

decorator

.

angle

=

1.7

;

37

decorator

.

blur

=

2

;

38

flower

.

y

-=

20

;

39

}

else

{

40

decorator

.

ascent

=

0

;

41

decorator

.

xShift

=

0

;

42

decorator

.

yScale

=

1

;

43

decorator

.

angle

=

-

1.4

;

44

decorator

.

opacity

=

0.8

;

45

decorator

.

blur

=

0

;

46

flower

.

y

+=

20

;

47

step

=

0

;

48

}

49

},

50

)..

onTapUp

(),

51

);

52

}

53

}

54

55

class

Grid

extends

Component

{

56

final

paint

=

Paint

()

57

..

color

=

const

Color

(

0xffa9a9a9

)

58

..

style

=

PaintingStyle

.

stroke

59

..

strokeWidth

=

1

;

60

@override

61

void

render

(

Canvas

canvas

)

{

62

for

(

var

i

=

0

;

i

<

50

;

i

++

)

```
{
63
canvas
.
drawLine
(
Offset
(
0
,
i
*
25
),
Offset
(
500
,
i
*
25
),
paint
);
64
canvas
```

.

drawLine

(

Offset

(

i

*

25

,

0

),

Offset

(

i

*

25

,

500

),

paint

);

65

}

66

}

67

}

Code

This decorator renders a shadow underneath the component, as if the component was a 3D object standin

The shadow produced by this generator is quite flexible: you can control its angle, length, opacity, blur, etc.

final

decorator

=

Shadow3DDecorator

(

base:

Vector2

(

100

,

150

),

angle:

-

1.4

,

xShift:

200

,

yScale:

1.5

,

opacity:

0.5

,

blur:

1.5

,

);

The primary purpose of this decorator is to add shadows on the ground to your components. The main limi

Using decorators

¶

HasDecorator mixin

¶

This

Component

mixin adds the

decorator

property, which is initially

null

. If you set this property to an actual

Decorator

object, then that decorator will apply its visual effect during the rendering of the component. In order to rem

decorator

property back to

null

.

PositionComponent

¶

PositionComponent

(and all the derived classes) already has a

decorator

property, so for these components the

HasDecorator

mixin is not needed.

In fact, the

PositionComponent

uses its decorator in order to properly position the component on the screen. Thus, any new decorators tha

PositionComponent

will need to be chained (see the

Multiple decorators

section below).

It is also possible to replace the root decorator of the

PositionComponent

, if you want to create an alternative logic for how the component shall be positioned on the screen.

Multiple decorators

¶

It is possible to apply several decorators simultaneously to the same component: the

Decorator

class supports chaining. That is, if you have an existing decorator on a component and you want to add an

component.decorator.addLast(newDecorator)

? this will add the new decorator at the end of the existing chain. The method

removeLast()

can remove that decorator later.

Several decorators can be chain that way. For example, if

A

is an initial decorator, then

A.addLast(B)

can be followed by either

A.addLast(C)

or

B.addLast(C)

? and in both cases the chain

A

->

B

->

C

will be created. In practice, it means that the entire chain can be manipulated from its root, which usually is

component.decorator

.

# Getting Started

¶

## About Flame

¶

Flame is a modular Flutter game engine that provides a complete set of out-of-the-way solutions for games

It provides you with a simple yet effective game loop implementation, and the necessary functionalities that

We also provide stand-alone packages that extend the Flame functionality which can be found in the

Bridge Packages

section.

You can pick and choose whichever parts you want, as they are all independent and modular.

The engine and its ecosystem are constantly being improved by the community, so please feel free to reac

Give us a star if you want to help give the engine exposure and grow the community. :)

## Installation

¶

Add the

flame

package as a dependency in your

pubspec.yaml

by running the following command:

```
flutter pub add flame
```

The latest version can be found on

pub.dev

.

then run

```
flutter
pub
```

get

and you are ready to start using it!

Getting started

¶

There is a set of tutorials that you can follow to get started in the

tutorials folder

.

Simple examples for all features can be found in the

examples folder

.

You can also check out the

awesome flame repository

, it contains quite a lot of good tutorials and articles written by the community to get you started with Flame.

Outside of the scope of the engine

¶

Games sometimes require complex feature sets depending on what the game is all about. Some of these f

Multiplayer (netcode)

¶

Flame doesn?t bundle any network feature, which may be needed to write online multiplayer games.

If you are building a multiplayer game, here are some recommendations of packages/services:

Nakama

: Nakama is an open-source server designed to power modern games and apps.

Firebase

: Provides dozens of services that can be used to write simpler multiplayer experiences.

Supabase

: A cheaper alternative to Firebase, based on Postgres.

AudioPool

¶

An AudioPool is a provider of AudioPlayers that are pre-loaded with local assets to minimize delays.

A single AudioPool always plays the same sound, usually a quick sound effect, like a laser shooting from y

The advantage of using Audio Pool is that by configuring a minimum (starting) size, and a maximum size, t

You can use the helper method

FlameAudio.createPool

to create AudioPool instances using the same global

FlameAudio.audioCache

.

Structure

¶

Flame has a proposed structure for your project that includes the standard Flutter

assets

directory in addition to some children:

audio

,

images

and

tiles

.

If using the following example code:

```
void
main
()
{
FlameAudio
.
play
(
'explosion.mp3'
);
Flame
.
images
.
```

load

(

'player.png'

);

Flame

.

images

.

load

(

'enemy.png'

);

TiledComponent

.

load

(

'level.tmx'

,

tileSize

);

}

The following file structure is where Flame would expect to find the files:

. ??? assets ??? audio ? ??? explosion.mp3 ??? images ? ??? enemy.png ? ??? player.pn

Optionally you can split your

audio

folder into two subfolders, one for

music

and one for

sfx

.

Don?t forget to add these files to your

pubspec.yaml

file:

flutter

:

assets

:

-

assets/audio/explosion.mp3

-

assets/images/player.png

-

assets/images/enemy.png

-

assets/tiles/level.tmx

If you want to change this structure, this is possible by using the

prefix

parameter and creating your instances of

AssetsCache

,

Images

, and

AudioCache

, instead of using the global ones provided by Flame.

Additionally,

AssetsCache

and

Images

can receive a custom

AssetBundle

. This can be used to make Flame look for assets in a different location other the

rootBundle

, like the file system for example.

<<set>>

¶

The

<<set>>

command is used to update the value of an existing variable. The variable must be declared with

<<declare>>

or

<<local>>

before it can be used in

<<set>>

.

The command

<<set>>

allows either regular assignment, or modifying assignment, like follows:

// Regular assignment

<<

set

$VARIABLE

=

EXPRESSION

>>

<<

set

$VARIABLE

to

EXPRESSION

>>

// Modifying assignments

<<

set

$VARIABLE

+=

EXPRESSION

>>

<<

set

$VARIABLE

-=

EXPRESSION

>>

<<

set

$VARIABLE

*=

EXPRESSION

>>

<<

set

$VARIABLE

/=

EXPRESSION

>>

<<

set

$VARIABLE

%=

EXPRESSION

>>

// These modifying assignments are equivalent to the following:

<<

set

$VARIABLE

=

$VARIABLE

+

EXPRESSION

>>

<<

set

$VARIABLE

=

$VARIABLE

-

EXPRESSION

>>

<<

set

$VARIABLE

=

$VARIABLE

*

EXPRESSION

>>

<<

set

$VARIABLE

=

$VARIABLE

/

EXPRESSION

>>

<<

set

$VARIABLE

=

$VARIABLE

%

EXPRESSION

>>

In all cases, the

EXPRESSION

must have the same type as the

$VARIABLE

. If not, a compile-time error will be thrown.

Examples

¶

<<

declare

$favorite_color

as

String

>>

title

:

ColorQuiz

---

What is your favorite color?

->

White

<<

set

$favorite_color

to

"

White

"

>>

->

Red

<<

set

$favorite_color

to

"

Red

"

>>

->

Yellow

<<

set

$favorite_color

=

"

Yellow

"

>>

->

Blue

Oh, Nice! Which shade of blue?

->

Azure

->

Cerulean

->

Lapis Lazuli

Umm, I don't know how to spell that. I'll just put you down as "blue".

<<

set

$favorite_color

=

"

Blue

"

>>

->

Black

<<

set

$favorite_color

=

"

Black

"

>>

That's mine too!

<<

set

$affinity

+=

3

>>

->

Prefer not to tell

Aww... Maybe if I ask again really nicely?

<<

jump

ColorQuiz

>>

===

flame_forge2d

¶

Overview

Forge2DGame

BodyComponent

Contact callbacks

Joints

Built-in joints

ConstantVolumeJoint

DistanceJoint

FrictionJoint

GearJoint

MotorJoint

MouseJoint

PrismaticJoint

Prismatic Joint Limit

Prismatic Joint Motor

PulleyJoint

RevoluteJoint

Revolute Joint Limit

Revolute Joint Motor

RopeJoint

WeldJoint

Breakable Bodies and WeldJoint

Adding animations and depth

¶

We now have something that looks more like a game, having graphics for our spaceship and being able to

But our game so far is too boring, the starship is just a static sprite and the background is just a black scre

In this step we will look at how to improve that, we will replace the static graphics of the player to an anima

So lets start by adding the animation to the player ship! For that, we will something that we call Sprite Anim

To better visualize this, this is the animation that we will be using, note how the image holds 4 individual im

Flame provides us with a specialized classes to deal with such images:

SpriteAnimation

and its component wrapper

SpriteAnimationComponent

and changing our

Player

component to be an animation is quite simple, take a look at how the component will look like now:

class

Player

extends

SpriteAnimationComponent

with

HasGameReference

<

SpaceShooterGame

>

{

Player

()

:

super

(

size:

Vector2

(

100

,

150

),

anchor:

Anchor

.

center

,

);

@override

Future

<

void

>

onLoad

()

async

{

await

super

.

onLoad

();

animation

=

await

game

.

loadSpriteAnimation

(

'player.png'

,

SpriteAnimationData

.

sequenced

(

amount:

4

,

stepTime:

.

2

,

textureSize:

Vector2

(

32

,

48

),

),

);

position

=

game

.

size

/

2

;

}

// Other methods omitted

}

So lets break down the changes:

First we changed our

Player

component to extend from

SpriteAnimationComponent

instead of

SpriteComponent

In the

onLoad

method we are now using the

game.loadSpriteAnimation

helper instead of the

loadSprite

one, and setting the

animation

attribute with its returned value.

The

SpriteAnimationData

class might look complicated at first glance, but it is actually quite simple, note how we used the

sequenced

constructor, which is a helper to load animation images where the frames are already layed down in the se

amount

defines how many frames the animation has, in this case

4

stepTime

is the time in seconds that each frame will be rendered, before it gets replaced with the next one.

textureSize

is the size in pixels which defines each frame of the image.

With all of this information, the

SpriteAnimationComponent

will now automatically play the animation!

Now lets add some depth and energy to our game background. Of course there are many ways of doing so

Wikipedia

.

Flame provides classes to implement parallax scrolling out of the box, these classes are

Parallax

and

ParallaxComponent

, so lets take a look at how we can add that new feature to the game:

class

SpaceShooterGame

extends

FlameGame

with

PanDetector

{

late

Player

player

;

@override

Future

<

void

>

onLoad

()

async

{

final

parallax

=

await

loadParallaxComponent

(

[

ParallaxImageData

(

'stars_0.png'

),

ParallaxImageData

(

'stars_1.png'

),

ParallaxImageData

(

'stars_2.png'

),

],

baseVelocity:

Vector2

(

0

,

-

5

),

repeat:

ImageRepeat

.

repeat

,

velocityMultiplierDelta:

Vector2

(

0

,

5

),

);

add

(

parallax

);

player

=

Player

();

add

(

player

);

}

@override

void

onPanUpdate

(

DragUpdateInfo

info

)

{

player

.

move

(

info

.

delta

.

global

);

}

}

Looking at the code above we notice that we are now using the

loadParallaxComponent

helper method from the

FlameGame

class to directly load a

ParallaxComponent

and add it to our game.

The arguments used there are as follows:

The first argument is a positional one, which should be a list of

ParallaxData

s. There are a couple of types of

ParallaxData

s in Flame, in this tutorial we are using the

ParallaxImageData

which describes a layer in the parallax scrolling effect that is an

image

. This list will tell Flame about all the layers that we want in our parallax.

baseVelocity

is the base value for all the values, so by passing a

Vector2(0,

-5)

to it means that the slower of the layers will move at 0 pixels per second on the

x

axis and

-5

pixels per second on the

y

axis.

Finally

velocityMultiplierDelta

is a vector that is applied to the base value for each layer, and in our example the multiplication rate is

5

on only the

y

axis.

Give it a try by running the game now, you will notice that it looks way more dynamic now, giving a more co

Run

main.dart

1

import

'package:flame/components.dart'

;

2

import

'package:flame/events.dart'

;

3

import

'package:flame/game.dart'

;

4

import

'package:flame/input.dart'

;

5

import

'package:flame/parallax.dart'

;

6

import

'package:flutter/material.dart'

;

7

8

void

main

()

{

9

runApp

(

GameWidget

(

game:

SpaceShooterGame

()));

10

}

11

12

class

SpaceShooterGame

extends

FlameGame

with

PanDetector

{

13

late

Player

player

;

14

15

@override

16

Future

<

void

>

onLoad

()

async

{

17

final

parallax

=

await

```
loadParallaxComponent
(
18
[
19
ParallaxImageData
(
'stars_0.png'
),
20
ParallaxImageData
(
'stars_1.png'
),
21
ParallaxImageData
(
'stars_2.png'
),
22
],
23
baseVelocity:
Vector2
(
0
```

,

-

5

),

24

repeat:

ImageRepeat

.

repeat

,

25

velocityMultiplierDelta:

Vector2

(

0

,

5

),

26

);

27

add

(

parallax

);

28

29
player
=
Player
();
30
add
(
player
);
31
}
32
33
@override
34
void
onPanUpdate
(
DragUpdateInfo
info
)
{
35
player
.

move

(

info

.

delta

.

global

);

36

}

37

}

38

39

class

Player

extends

SpriteAnimationComponent

40

with

HasGameReference

<

SpaceShooterGame

>

{

41

Player

()

42

:

super

(

43

size:

Vector2

(

100

,

150

),

44

anchor:

Anchor

.

center

,

45

);

46

47

@override

48

Future

<

void

>

onLoad

()

async

{

49

await

super

.

onLoad

();

50

51

animation

=

await

game

.

loadSpriteAnimation

(

52

'player.png'

,

53

SpriteAnimationData

.

sequenced

(

54

amount:

4

,

55

stepTime:

.

2

,

56

textureSize:

Vector2

(

32

,

48

),

57

),

58

);

59

60

position

=

game

.

size

/

2

;

61

}

62

63

void

move

(

Vector2

delta

)

{

64

position

.

add

(

delta

);

65

}

66

}

Code

flame_spine

¶

This package allows you to load and add Spine skeletal animations to your Flame game.

Usage

¶

To use it in your game you just need to add

flame_spine

to your pubspec.yaml and your spine assets to your

assets/

directory, and you can add a

SpineComponent

to your

FlameGame

.

Note

Remember to call

await

initSpineFlutter();

in your

main

method, or in

onLoad

.

Example:

void

main

()

async

{

WidgetsFlutterBinding

.

ensureInitialized

();

await

initSpineFlutter

();

runApp

(

const

GameWidget

.

controlled

(

gameFactory:

SpineExample

.

new

));

}

class

FlameSpineExample

extends

```
FlameGame
with
TapDetector
{
late
final
SpineComponent
spineboy
;
@override
Future
<
void
>
onLoad
()
async
{
await
initSpineFlutter
();
// Load the Spineboy atlas and skeleton data from asset files
// and create a SpineComponent from them, scaled down and
// centered on the screen
spineboy
=
```

```
await
SpineComponent
.
fromAssets
(
atlasFile:
'assets/spine/spineboy.atlas'
,
skeletonFile:
'assets/spine/spineboy-pro.skel'
,
scale:
Vector2
(
0.4
,
0.4
),
anchor:
Anchor
.
center
,
position:
size
/
```

```
2
,
);
// Set the "walk" animation on track 0 in looping mode
spineboy
.
animationState
.
setAnimationByName
(
0
,
'walk'
,
true
);
await
add
(
spineboy
);
}
@override
void
onDetach
()
```

{

// Dispose the native resources that have been loaded for spineboy.

spineboy

.

dispose

();

}

}

Adding bullets

¶

For this next step we will add a very important feature to any space shooter game, shooting!

Here is how we will implement it: since we already control our space ship by dragging on the screen with th

So let?s start, to begin let?s first create a

Bullet

component that will represent the shots in the game.

class

Bullet

extends

SpriteAnimationComponent

with

HasGameReference

<

SpaceShooterGame

>

{

Bullet

({

super

.

position

,

})

:

super

(

size:

Vector2

(

25

,

50

),

anchor:

Anchor

.

center

,

);

@override

Future

<

void

>

onLoad

()

async

{

await

super

.

onLoad

();

animation

=

await

game

.

loadSpriteAnimation

(

'bullet.png'

,

SpriteAnimationData

.

sequenced

(

amount:

4

,

stepTime:

.

2

,

textureSize:

Vector2

(

8

,

16

),

),

);

}

}

So far, this does not introduce any new concepts, we just created a component and set up its animations a

The

Bullet

behavior is a simple one, it always moves towards the top of the screen and should be removed from the g

update

method to it and make it happen:

class

Bullet

extends

SpriteAnimationComponent

with

HasGameReference

<

SpaceShooterGame

>

{

Bullet

({

super

.

position

,

})

:

super

(

size:

Vector2

(

25

,

50

),

anchor:

Anchor

.

center

,

);

@override

Future

<

void

>

onLoad

()

async

{

// Omitted

}

@override

void

update

(

double

dt

)

{

super

.

update

(

dt

);

position

.

y

+=

dt

*

-

500

;

if

(

position

.

y

<

-

height

)

{

removeFromParent

();

}

}

}

The above code should be straight forward, but lets break it down:

We add to the bullet?s y axis position at a rate of -500 pixels per second. Remember going up in the y axis

0

since the top left corner of the screen is

0,

0

.

If the y is smaller than the negative value of the bullet?s height, means that the component is completely o

Right, we now have a

Bullet

class ready, so lets start to implement the action of shooting. First thing, let?s create two empty methods in

Player

class,

startShooting

and

stopShooting

.

class

Player

extends

SpriteAnimationComponent

with

HasGameReference

<

SpaceShooterGame

>

{

// Rest of implementation omitted

void

startShooting

()

{

// TODO

}

void

stopShooting

()

{

// TODO

}

}

And let?s hook into those methods from the game class, we will do that by using the

onPanStart

and

onPanEnd

methods from the

PanDetector

mixin that we already have been using for the ship movement:

class

SpaceShooterGame

extends

FlameGame

with

PanDetector

{

late

Player

player

;

// Rest of implementation omitted

@override

```
void
onPanUpdate
(
DragUpdateInfo
info
)
{
player
.
move
(
info
.
delta
.
global
);
}
@override
void
onPanStart
(
DragStartInfo
info
)
{
```

player

.

startShooting

();

}

@override

void

onPanEnd

(

DragEndInfo

info

)

{

player

.

stopShooting

();

}

}

We now have everything set up, so let?s write the shooting routine in our player class.

Remember, the shooting behavior will be adding bullets through time intervals when the player is dragging

We could implement the time interval code and the spawning manually, but Flame provides a component c

SpawnComponent

, so let?s take advantage of it:

class

Player

```
extends
SpriteAnimationComponent
with
HasGameReference
<
SpaceShooterGame
>
{
late
final
SpawnComponent
_bulletSpawner
;
@override
Future
<
void
>
onLoad
()
async
{
// Loading animation omitted
_bulletSpawner
=
SpawnComponent
```

(

period:

.

2

,

selfPositioning:

true

,

factory

:

(

index

)

{

return

Bullet

(

position:

position

+

Vector2

(

0

,

-

height

/

2

,

),

);

return

bullet

;

},

autoStart:

false

,

);

add

(

_bulletSpawner

);

}

void

move

(

Vector2

delta

)

{

position

.

add

(

delta

);

}

void

startShooting

()

{

_bulletSpawner

.

timer

.

start

();

}

void

stopShooting

()

{

_bulletSpawner

.

timer

.

stop

();

}

}

Hopefully the code above speaks for itself, but let?s look at it in more detail:

First we declared a

SpawnComponent

called

_bulletSpawner

in our game class, we needed it to be an variable accessible to the whole component since we will be acce

startShooting

and

stopShooting

methods.

We initialize our

_bulletSpawner

in the

onLoad

method. In the first argument,

period

, we set how much time in seconds it will take between calls, and we choose

.2

seconds for now.

We set

selfPositioning:

true

so the spawn component don?t try to position the created component itself since we want to handle that ou

The

factory

attribute receives a function that will be called every time the

period

is reached. and must return the create component.

Then we set that it should not auto start by default.

Finally we add the

_bulletSpawner

to our component, so it can be processed in the game loop.

With the

_bulletSpawner

all setup, the only missing piece now is to start the

_bulletSpawner.timer

at

startShooting

and stop it in the

stopShooting

!

And that closes this step, putting us real close to a real game!

FlameNetworkAssets

¶

FlameNetworkAssets

is a bridge package focused on providing a solution to fetch, and cache assets from the network.

The

FlameNetworkAssets

class provides an abstraction that should be extended in order to create asset specific handler.

By default, the package relies on the

http

package to make http requests, and

path_provider

to get the place to store the local cache, to use a different approach for those, use the optional arguments i

This package bundles a specific asset handler class for images:

final

networkAssets

=

FlameNetworkImages

();

final

playerSprite

=

await

networkAssets

.

load

(

'https://url.com/image.png'

);

To create a specific asset handler class, you just need to extend the

FlameNetworkAssets

class and define the

decodeAsset

and

encodeAsset

arguments:

class

FlameNetworkCustomAsset

extends

FlameNetworkAssets

<

CustomAsset

>

{

FlameNetworkImages

({

super

.

get

,

super

.

getAppDirectory

,

super

.

cacheInMemory

,

super

.

cacheInStorage

,

})

:

super

(

decodeAsset:

(

bytes

)

=>

CustomAsset

.

decode

(

bytes

),

encodeAsset:

(

CustomAsset

asset

)

=>

asset

.

encode

(),

);

}

Drag Events

¶

Drag events

occur when the user moves their finger across the screen of the device, or when they move the mouse whi

Multiple drag events can occur at the same time, if the user is using multiple fingers. Such cases will be ha

pointerId

property.

For those components that you want to respond to drags, add the

DragCallbacks

mixin.

This mixin adds four overridable methods to your component:

onDragStart

,

onDragUpdate

,

onDragEnd

, and

onDragCancel

. By default, these methods do nothing ? they need to be overridden in order to perform any function.

In addition, the component must implement the

containsLocalPoint()

method (already implemented in

PositionComponent

, so most of the time you don?t need to do anything here) ? this method allows Flame to know whether the

class

MyComponent

```
extends
PositionComponent
with
DragCallbacks
{
MyComponent
()
:
super
(
size:
Vector2
(
180
,
120
));
@override
void
onDragStart
(
DragStartEvent
event
)
{
// Do something in response to a drag event
```

}

}

Demo

¶

In this example you can use drag gestures to either drag star-like shapes across the screen, or to draw cur

drag_events.dart

1

import

'dart:math'

;

2

3

import

'package:flame/components.dart'

;

4

import

'package:flame/events.dart'

;

5

import

'package:flame/game.dart'

;

6

import

'package:flame/geometry.dart'

;

7

import

'package:flutter/rendering.dart'

;

8

9

class

DragEventsGame

extends

FlameGame

{

10

@override

11

Future

<

void

>

onLoad

()

async

{

12

addAll

([

13

DragTarget

(),

14

Star

(

15

n:

5

,

16

radius1:

40

,

17

radius2:

20

,

18

sharpness:

0.2

,

19

color:

const

Color

(

0xffbae5ad

),

20

position:

Vector2

(

70

,

70

),

21

),

22

Star

(

23

n:

3

,

24

radius1:

50

,

25

radius2:

40

,

26

sharpness:

0.3

,

27

color:

const

Color

(

0xff6ecbe5

),

28

position:

Vector2

(

70

,

160

),

29

),

30

Star

(

31

n:

12

,

32

radius1:

10

,

33

radius2:

75

,

34

sharpness:

1.3

,

35

color:

const

Color

(

0xfff6df6a

),

36

position:

Vector2

(

70

,

270

),

37

),

38

Star

(

39

n:

10

,

40

radius1:

20

,

41

radius2:

17

,

42

sharpness:

0.85

,

```
43
color:
const
Color
(
0xfff82a4b
),
44
position:
Vector2
(
110
,
110
),
45
),
46
]);
47
}
48
}
49
50
/// This component is the pink-ish rectangle in the center of the game window.
```

51

/// It uses the [DragCallbacks] mixin in order to receive drag events.

52

class

DragTarget

extends

PositionComponent

with

DragCallbacks

{

53

DragTarget

()

:

super

(

anchor:

Anchor

.

center

);

54

55

final

_rectPaint

=

Paint

()..

color

=

const

Color

(

0x88AC54BF

);

56

57

/// We will store all current circles into this map, keyed by the `pointerId`

58

/// of the event that created the circle.

59

final

Map

<

int

,

Trail

>

_trails

=

{};

60

```
61
@override
62
void
onGameResize
(
Vector2
size
)
{
63
super
.
onGameResize
(
size
);
64
this
.
size
=
size
-
Vector2
(
```

100

,

75

);

65

if

(

this

.

size

.

x

<

100

||

this

.

size

.

y

<

100

)

{

66

this

.

size

=

size

*

0.9

;

67

}

68

position

=

size

/

2

;

69

}

70

71

@override

72

void

render

(

Canvas

canvas

)

{

73

canvas

.

drawRect

(

size

.

toRect

(),

_rectPaint

);

74

}

75

76

@override

77

void

onDragStart

(

DragStartEvent

event

)

```
{
78
super
.
onDragStart
(
event
);
79
final
trail
=
Trail
(
event
.
localPosition
);
80
_trails
[
event
.
pointerId
]
=
```

trail

;

81

add

(

trail

);

82

}

83

84

@override

85

void

onDragUpdate

(

DragUpdateEvent

event

)

{

86

_trails

[

event

.

pointerId

]

!

.

addPoint

(

event

.

localEndPosition

);

87

}

88

89

@override

90

void

onDragEnd

(

DragEndEvent

event

)

{

91

super

.

onDragEnd

(

event

);

92

_trails

.

remove

(

event

.

pointerId

)

!

.

end

();

93

}

94

95

@override

96

void

onDragCancel

(

DragCancelEvent

```
event
)
{
97
super
.
onDragCancel
(
event
);
98
_trails
.
remove
(
event
.
pointerId
)
!
.
cancel
();
99
}
100
```

}

101

102

class

Trail

extends

Component

{

103

Trail

(

Vector2

origin

)

104

:

_paths

=

[

Path

()..

moveTo

(

origin

.

x

,

origin

.

y

)],

105

_opacities

=

[

1

],

106

_lastPoint

=

origin

.

clone

(),

107

_color

=

108

HSLColor

.

fromAHSL

(

1

,

random

.

nextDouble

()

*

360

,

1

,

0.8

).

toColor

();

109

110

final

List

<

Path

>

_paths

;

111

final

List

<

double

>

_opacities

;

112

Color

_color

;

113

late

final

_linePaint

=

Paint

()..

style

=

PaintingStyle

.

stroke

;

114

late

final

_circlePaint

=

Paint

()..

color

=

_color

;

115

bool

_released

=

false

;

116

double

_timer

=

0

;

117

final

_vanishInterval

=

0.03

;

118

final

Vector2

_lastPoint

;

119

120

static

final

random

=

Random

();

121

static

const

lineWidth

=

10.0

;

122

123

@override

124

void

render

(

Canvas

canvas

)

{

125

assert

(

_paths

.

length

==

_opacities

.

length

);

126

for

(

var

i

=

0

;

i

<

_paths

.

length

;

i

++

)

{

127

final

path

=

_paths

[

i

];

128

final

opacity

=

_opacities

[

i

];

129

if

(

opacity

>

0

)

{

130

_linePaint

.

color

=

_color

.

withOpacity

(

opacity

);

131

_linePaint

.

strokeWidth

=

lineWidth

*

opacity

;

```
132
canvas
.
drawPath
(
path
,
_linePaint
);
133
}
134
}
135
canvas
.
drawCircle
(
136
_lastPoint
.
toOffset
(),
137
(
lineWidth
```

-

2

)

*

_opacities

.

last

+

2

,

138

_circlePaint

,

139

);

140

}

141

142

@override

143

void

update

(

double

dt

)

{

144

assert

(

_paths

.

length

==

_opacities

.

length

);

145

_timer

+=

dt

;

146

while

(

_timer

>

_vanishInterval

)

{

147

_timer

-=

_vanishInterval

;

148

for

(

var

i

=

0

;

i

<

_paths

.

length

;

i

++

)

{

149

_opacities

[

i

]

-=

0.01

;

150

if

(

_opacities

[

i

]

<=

0

)

{

151

_paths

[

i

].

reset

();

152

}

153

}

154

if

(

!

_released

)

{

155

_paths

.

add

(

Path

()..

moveTo

(

_lastPoint

.

x

,

_lastPoint

.

y

));

156

_opacities

.

add

(

1

);

157

}

158

}

159

if

(

_opacities

.

last

<

0

)

{

160

removeFromParent

();

161

}

162

}

163

164

void

addPoint

(

Vector2

point

)

{

165

if

(

!

point

.

x

.

isNaN

)

{

166

for

(

final

path

in

_paths

)

{

167

path

.

lineTo

(

point

.

x

,

point

.

y

);

168

}

169

_lastPoint

.

setFrom

(

point

);

```
170
}
171
}
172
173
void
end
()
=>
_released
=
true
;
174
175
void
cancel
()
{
176
_released
=
true
;
177
```

_color

=

const

Color

(

0xFFFFFFFF

);

178

}

179

}

180

181

class

Star

extends

PositionComponent

with

DragCallbacks

{

182

Star

({

183

required

int

n

,

184

required

double

radius1

,

185

required

double

radius2

,

186

required

double

sharpness

,

187

required

this

.

color

,

188

super

.

position

,

189

})

{

190

_path

=

Path

()..

moveTo

(

radius1

,

0

);

191

for

(

var

i

=

0

;

i

<

n

;

i

++

)

{

192

final

p1

=

Vector2

(

radius2

,

0

)..

rotate

(

tau

/

n

*

(

i

+

sharpness

));

193

final

p2

=

Vector2

(

radius2

,

0

)..

rotate

(

tau

/

n

*

(

i

+

1

-

sharpness

));

194

final

p3

=

Vector2

(

radius1

,

0

)..

rotate

(

tau

/

n

*

(

i

+

1

));

195

_path

.

cubicTo

(

p1

.

x

,

p1

.

y

,

p2

.

x

,

p2

.

y

,

p3

.

x

,

p3

.

y

);

196

}

197

_path

.

close

();

198

}

199

200

final

Color

color

;

201

final

Paint

_paint

=

Paint

();

202

final

Paint

_borderPaint

=

Paint

()

203

..

color

=

const

Color

(

0xFFffffff

)

204

..

style

=

PaintingStyle

.

stroke

205

..

strokeWidth

=

3

;

206

final

_shadowPaint

=

Paint

()

207

..

color

=

const

Color

(

0xFF000000

)

208

..

maskFilter

=

const

MaskFilter

.

blur

(

BlurStyle

.

normal

,

4.0

);

209

late

final

Path

_path

;

210

211

@override

212

bool

containsLocalPoint

(

Vector2

point

)

{

213

return

_path

.

contains

(

point

.

toOffset

());

214

}

215

216

@override

217

void

render

(

Canvas

canvas

)

{

218

if

(

isDragged

)

{

219

_paint

.

color

=

color

.

withOpacity

(

0.5

);

220

canvas

.

drawPath

(

_path

,

_paint

);

221

canvas

.

drawPath

(

_path

,

_borderPaint

);

222

}

else

{

```
223
_paint
.
color
=
color
.
withOpacity
(
1
);
224
canvas
.
drawPath
(
_path
,
_shadowPaint
);
225
canvas
.
drawPath
(
_path
```

,

_paint

);

226

}

227

}

228

229

@override

230

void

onDragStart

(

DragStartEvent

event

)

{

231

super

.

onDragStart

(

event

);

232

priority

=

10

;

233

}

234

235

@override

236

void

onDragEnd

(

DragEndEvent

event

)

{

237

super

.

onDragEnd

(

event

);

238

priority

=

0

;

239

}

240

241

@override

242

void

onDragUpdate

(

DragUpdateEvent

event

)

{

243

position

+=

event

.

localDelta

;

244

}

245

}

Code

Drag anatomy

¶

onDragStart

¶

This is the first event that occurs in a drag sequence. Usually, the event will be delivered to the topmost co

DragCallbacks

mixin. However, by setting the flag

event.continuePropagation

to true, you can allow the event to propagate to the components below.

The

DragStartEvent

object associated with this event will contain the coordinate of the point where the event has originated. Th

devicePosition

is given in the coordinate system of the entire device,

canvasPosition

is in the coordinate system of the game widget, and

localPosition

provides the position in the component?s local coordinate system.

Any component that receives

onDragStart

will later be receiving

onDragUpdate

and

onDragEnd

events as well.

onDragUpdate

¶

This event is fired continuously as user drags their finger across the screen. It will not fire if the user is hold

The default implementation delivers this event to all the components that received the previous

onDragStart

with the same pointer id. If the point of touch is still within the component, then

event.localPosition

will give the position of that point in the local coordinate system. However, if the user moves their finger aw

event.localPosition

will return a point whose coordinates are NaNs. Likewise, the

event.renderingTrace

in this case will be empty. However, the

canvasPosition

and

devicePosition

properties of the event will be valid.

In addition, the

DragUpdateEvent

will contain

delta

? the amount the finger has moved since the previous

onDragUpdate

, or since the

onDragStart

if this is the first drag-update after a drag- start.

The

event.timestamp

property measures the time elapsed since the beginning of the drag. It can be used, for example, to compu

onDragEnd

¶

This event is fired when the user lifts their finger and thus stops the drag gesture. There is no position asso

onDragCancel

¶

The precise semantics when this event occurs is not clear, so we provide a default implementation which s

onDragEnd

.

Mixins

¶

DragCallbacks

¶

The

DragCallbacks

mixin can be added to any

Component

in order for that component to start receiving drag events.

This mixin adds methods

onDragStart

,

onDragUpdate

,

onDragEnd

, and

onDragCancel

to the component, which by default don?t do anything, but can be overridden to implement any real functio

Another crucial detail is that a component will only receive drag events that originate

within

that component, as judged by the

containsLocalPoint()

function. The commonly-used

PositionComponent

class provides such an implementation based on its

size

property. Thus, if your component derives from a

PositionComponent

, then make sure that you set its size correctly. If, however, your component derives from the bare

Component

, then the

containsLocalPoint()

method must be implemented manually.

If your component is a part of a larger hierarchy, then it will only receive drag events if its ancestors have a

containsLocalPoint

correctly.

class

MyComponent

extends

PositionComponent

with

DragCallbacks

{

MyComponent

({

super

.

size

});

final

_paint

=

Paint

();

bool

_isDragged

=

false

;

@override

void

onDragStart

(

DragStartEvent

event

)

=>

_isDragged

=

true

;

@override

void

onDragUpdate

(

DragUpdateEvent

event

)

=>

position

+=

event

.

delta

;

@override

void

onDragEnd

(

DragEndEvent

event

)

=>

_isDragged

=

false

;

@override

void

render

(

Canvas

canvas

)

{

_paint

.

color

=

_isDragged

?

Colors

.

red

:

Colors

.

white

;

```
canvas
.
drawRect
(
size
.
toRect
(),
_paint
);
}
}
```

<<if>>

¶

The

<<if>>

command evaluates its condition, and based on that decides which statements to execute next. This is equ

if

keyword in most programming languages. This command may have multiple parts, which look as follows:

<<

if

condition1

>>

statements1...

<<

elseif

condition2

>>

statements2...

<<

else

>>

statementsN...

<<

endif

>>

The conditions within each command must have boolean type.

There could be any number of

<<elseif>>

blocks.

The

<<elseif>>

blocks and

<<else>>

are optional.

The final

<<endif>>

is mandatory.

The statements within each block must be indented.

At runtime, the condition within the

if

block is evaluated first. If it turns out to be

true

, then the dialogue proceeds with executing

statements1

, and no other conditions are evaluated nor other statement blocks executed. However, if

condition1

evaluated to

false

, then

condition2

is calculated. If it is true, then the dialogue runner will execute

statements2

, and if false it will fall-through into the

else

block and execute

statementsN

. In the end, the dialogue will proceed to statements that occur after the final

<<endif>>

.

Example

¶

In this dialogue a

Guard

will greet you differently depending on your reputation with the citizens of the area. If your reputation falls b

title

:

GuardGreeting

---

<<

if

$reputation

>=

100

>>

Guard

:

Hail to the savior of the people!

<<

elseif

$reputation

>=

30

>>

Guard

:

Nice to meet you, sir!

<<

elseif

$reputation

>=

0

>>

Guard

:

Hello

<<

elseif

$reputation

>

-

30

>>

Guard

:

I'm keeping an eye on you...

<<

elseif

$reputation

>

-

100

>>

Guard

:

You filthy scum!

<<

else

>>

Guard

:

You'll pay for your crimes!

#auto

<<

attack

>>

<<

endif

>>

===

VariableStorage

¶

class

VariableStorage

Properties

¶

variables

:

Map<String, dynamic>

length

?

int

isEmpty

?

bool

isNotEmpty

?

bool

Methods

¶

getBooleanValue

(

String

name

)

?

bool

getNumericValue

(

String

name

)

?

num

getStringValue

(

String

name

)

?

String

hasVariable

(

String

name

)

?

bool

getVariable

(

String

name

)

?

dynamic

getVariableAsExpression

(

String

name

)

?

Expression

getVariableType

(

String

name

)

?

ExpressionType

setVariable

(

String

name

,

dynamic

value

)

clear

(

{

bool

clearNodeVisits

=

false

}

)

Clear all variables. By default node visit counts will not be cleared. To remove node visit counts as well, se

clearNodeVisits

to

true

.

Note that node visit variable names are prefixed with an @ symbol. If you have custom variables that start

clearNodeVisits

is

false

. These will need to be removed individually using

remove

.

remove

(

String

name

)

Remove a variable by

name

.

Accessing variable storage

¶

Variable storage is accessed via the

YarnProject

.

final

variables

=

yarnProject

.

variables

;

Removing variables

¶

In most cases variables should be retained for the life of the

YarnProject

. However there may be situations where variables need to be removed from storage. For example, in a ga

Remove all variables with

clear

. By default this will retain node visit counts, which are also stored as variables. Node visit counts are used

clearNodeVisits

to

true

.

/// Clear all variables except node visit counts.

yarnProject

.

variables

.

clear

();

/// Clear all variables including node visit counts.

yarnProject

.

variables

.

clear

(

clearNodeVisits:

true

);

Use

remove

to remove a single variable.

yarnProject

.

variables

.

remove

(

'money'

);

## Layers and Snapshots

¶

Layers and snapshots share some common features, including the ability to pre-render and cache objects f

Snapshot

is a mixin that can be added to any

PositionComponent

. Use this for:

Mixing in to existing game objects (that are

PositionComponents

).

Caching game objects, such as sprites, that are complex to render.

Drawing the same object many times without rendering it each time.

Capturing an image snapshot to save as a screenshot (for example).

Layer

is a class. Use or extend this class for:

Structuring your game with logical layers (e.g. UI, foreground, main, background).

Grouping objects to form a complex scene, and then caching it (e.g. a background layer).

Processor support. Layers allow user-defined processors to run pre- and post- render.

### Layers

¶

Layers allow you to group rendering by context, as well as allow you to pre-render things. This enables, for

There are two types of layers on Flame:

DynamicLayer

: For things that are moving or changing.

PreRenderedLayer

: For things that are static.

DynamicLayer

¶

Dynamic layers are layers that are rendered every time that they are drawn on the canvas. As the name su

Usage example:

```
class
GameLayer
extends
DynamicLayer
{
final
MyGame
game
;
GameLayer
(
this
.
game
);
@override
void
drawLayer
()
{
game
.
```

playerSprite

.

render

(

canvas

,

position:

game

.

playerPosition

,

);

game

.

enemySprite

.

render

(

canvas

,

position:

game

.

enemyPosition

,

);

```
}
}
class
MyGame
extends
Game
{
// Other methods omitted...
@override
void
render
(
Canvas
canvas
)
{
gameLayer
.
render
(
canvas
);
// x and y can be provided as optional position arguments
}
}
PreRenderedLayer
```

¶

Pre-rendered layers are rendered only once, cached in memory and then just replicated on the game canv

Usage example:

class

BackgroundLayer

extends

PreRenderedLayer

{

final

Sprite

sprite

;

BackgroundLayer

(

this

.

sprite

);

@override

void

drawLayer

()

{

sprite

.

render

```
(
canvas
,
position:
Vector2
(
50
,
200
),
);
}
}
class
MyGame
extends
Game
{
// Other methods omitted...
@override
void
render
(
Canvas
canvas
)
```

{

// x and y can be provided as optional position arguments.

backgroundLayer

.

render

(

canvas

);

}

}

Layer Processors

¶

Flame also provides a way to add processors on your layer, which are ways to add effects on the entire lay

ShadowProcessor

is available, this processor renders a back drop shadow on your layer.

To add processors to your layer, just add them to the layer

preProcessors

or

postProcessors

list. For example:

// Works the same for both DynamicLayer and PreRenderedLayer

class

BackgroundLayer

extends

PreRenderedLayer

{

```
final
Sprite
sprite
;
BackgroundLayer
(
this
.
sprite
)
{
preProcessors
.
add
(
ShadowProcessor
());
}
@override
void
drawLayer
()
{
/* omitted */
}
// ...
```

Custom processors can be created by extending the

LayerProcessor

class.

You can check a working example of layers

here

.

Snapshots

¶

Snapshots are an alternative to layers. The

Snapshot

mixin can be applied to any

PositionComponent

.

```
class
SnapshotComponent
extends
PositionComponent
with
Snapshot
{}
class
MyGame
extends
FlameGame
{
late
```

final

SnapshotComponent

root

;

@override

Future

<

void

>

onLoad

()

async

{

// Add a snapshot component.

root

=

SnapshotComponent

();

add

(

root

);

}

}

Render as a snapshot

¶

Setting

renderSnapshot

to

true

(the default) on a snapshot-enabled component behaves similarly to a

PreRenderedLayer

. The component is rendered only once, cached in memory and then just replicated on the game canvas af

class

SnapshotComponent

extends

PositionComponent

with

Snapshot

{}

class

MyGame

extends

FlameGame

{

late

final

SnapshotComponent

root

;

late

final

SpriteComponent

background1

;

late

final

SpriteComponent

background2

;

@override

Future

<

void

>

onLoad

()

async

{

// Add a snapshot component.

root

=

SnapshotComponent

();

add

(

root

);

```
// Add some children.
final
background1Sprite
=
Sprite
(
await
images
.
load
(
'background1.png'
));
background1
=
SpriteComponent
(
sprite:
background1Sprite
);
root
.
add
(
background1
);
```

```
final
background2Sprite
=
Sprite
(
await
images
.
load
(
'background2.png'
));
background2
=
SpriteComponent
(
sprite:
background2Sprite
);
root
.
add
(
background2
);
// root will now render once (itself and all its children) and then cache
```

// the result. On subsequent render calls, root itself, nor any of its

// children, will be rendered. The snapshot will be used instead for

// improved performance.

}

}

Regenerating a snapshot

¶

A snapshot-enabled component will generate a snapshot of its entire tree, including its children. If any of th

takeSnapshot

to update the cached snapshot. If they are changing very frequently, it?s best not to use a

Snapshot

because there will be no performance benefit.

A component rendering a snapshot can still be transformed without incurring any performance cost. Once a

takeSnapshot

.

Taking a snapshot

¶

A snapshot-enabled component can be used to generate a snapshot at any time, even if

renderSnapshot

is set to false. This is useful for taking screen-grabs or any other purpose when it may be useful to have a s

A snapshot is always generated with no transform applied - i.e. as if the snapshot-enabled component is at

A snapshot is saved as a

Picture

, but it can be converted to an

Image

using

snapshotToImage

.

class

SnapshotComponent

extends

PositionComponent

with

Snapshot

{}

class

MyGame

extends

FlameGame

{

late

final

SnapshotComponent

root

;

@override

Future

<

void

>

onLoad

()

```
async
{
// Add a snapshot component, but don't use its render mode.
root
=
SnapshotComponent
()..
renderSnapshot
=
false
;
add
(
root
);
// Other code omitted.
}
// Call something like this to take an image snapshot at any time.
void
takeSnapshot
()
{
root
.
takeSnapshot
();
```

final

image

=

root

.

snapshotToImage

(

200

,

200

);

}

}

Snapshots that are cropped or off-center

¶

Sometimes your snapshot

Image

may appear cropped, or not in the position that is expected.

This is because the contents of a

Picture

can be positioned anywhere with respect to the origin, but when it is converted to an

Image

, the image always starts from

0,0

. This means that anything with a -ve position will be cropped.

The best way to deal with this is to ensure that your

Snapshot

component is always at position

0,0

with respect to your game and you never move it. This means that the image will usually contain what you

However, this is not always possible. To move (or rotate, or scale etc) the snapshot before converting it to

snapshotToImage

.

```
// Call something like this to take an image snapshot at any time.
void
takeSnapshot
()
{
// Prepare a matrix to move the snapshot by 200,50.
final
matrix
=
Matrix4
.
identity
()..
translate
(
200.0
,
50.0
);
```

root

.

takeSnapshot

();

final

image

=

root

.

snapshotToImage

(

200

,

200

,

transform:

matrix

);

}

Numeric functions

¶

These functions are used to manipulate numeric values. Most of them take a single numeric argument and

ceil(x)

¶

Returns the value

x

rounded up towards positive infinity. In other words, this returns the smallest integer value greater than or e

x

.

title

:

ceil

---

{

ceil

(

0

)

}

// 0

{

ceil

(

0.3

)

```
}
// 1
{
ceil
(
5
)
}
// 5
{
ceil
(
5.001
)
}
// 6
{
ceil
(
5.999
)
}
// 6
{
ceil
(
```

-

2.07

)

}

// -2

===

See also

floor(x)

int(x)

dec(x)

¶

Returns the value

x

reduced towards the previous integer. Thus, if

x

is already an integer this returns

x

-

1

, but if

x

is not an integer then this returns

floor(x)

.

title

:

```
dec
---
{
dec
(
0
)
}
// -1
{
dec
(
0.3
)
}
// 0
{
dec
(
5.0
)
}
// 4
{
dec
(
```

5.001

)

}

// 5

{

dec

(

5.999

)

}

// 5

{

dec

(

-

2.07

)

}

// -3

===

See also

inc(x)

decimal(x)

¶

Returns a fractional part of

x

.

If

x

is positive, then the returned value will be between

0

(inclusive) and

1

(exclusive). If

x

is negative, then the returned value will be between

0

and

-1

. In all cases it should hold that

x

==

int(x)

+

decimal(x)

.

title

:

decimal

---

{

decimal

(

0

)

}

// 0

{

decimal

(

0.3

)

}

// 0.3

{

decimal

(

5.0

)

}

// 0

{

decimal

(

5.001

)

}

// 0.001

{

decimal

(

5.999

)

}

// 0.999

{

decimal

(

-

2.07

)

}

// -0.07

===

See also

int(x)

floor(x)

¶

Returns the value

x

rounded down towards negative infinity. In other words, this returns the largest integer value less than or e

x

.

title

:

floor

---

{

floor

(

0

)

}

// 0

{

floor

(

0.3

)

}

// 0

{

floor

(

5

)

}

// 5

{

floor

(

5.001

)

}

// 5

{

floor

(

5.999

)

}

// 5

{

floor

(

-

2.07

)

}

// -3

===

See also

ceil(x)

int(x)

inc(x)

¶

Returns the value

x

increased towards the next integer. Thus, if

x

is already an integer this returns

x

+

1

, but if

x

is not an integer then this returns

ceil(x)

.

title

:

inc

---

{

inc

(

0

)

}

// 1

{

inc

```
(
0.3
)
}
// 1
{
inc
(
5.0
)
}
// 6
{
inc
(
5.001
)
}
// 6
{
inc
(
5.999
)
}
// 6
```

{

inc

(

-

2.07

)

}

// -2

===

See also

dec(x)

int(x)

¶

Truncates the fractional part of

x

, rounding it towards zero, and returns just the integer part of the argument

x

.

title

:

int

---

{

int

(

0

)

}

// 0

{

int

(

0.3

)

}

// 0

{

int

(

5.0

)

}

// 5

{

int

(

5.001

)

}

// 5

{

int

(

5.999

)

}

// 5

{

int

(

-

2.07

)

}

// -2

===

See also

decimal(x)

round(x)

round(x)

¶

Rounds the value

x

towards a nearest integer.

The values that end with

.5

are rounded up if

x

is positive, and down if

x

is negative.

title

:

round

---

{

round

(

0

)

}

// 0

{

round

(

0.3

)

}

// 0

{

round

(

5.0

)

```
}
// 5
{
round
(
5.001
)
}
// 5
{
round
(
5.5
)
}
// 6
{
round
(
5.999
)
}
// 6
{
round
(
```

-

2.07

)

}

// -2

{

round

(

-

2.5

)

}

// -3

===

See also

round_places(x,

n)

round_places(x,

n)

¶

Rounds the value

x

to

n

decimal places.

The value

x

can be either positive, negative, or zero, but it must be an integer. Rounding to

0

decimal places is equivalent to the regular

round(x)

function. If

n

is positive, then the function will attempt to keep that many digits after the decimal point in

x

. If

n

is negative, then

round_places()

will round

x

to nearest tens, hundreds, thousands, etc:

title

:

round_places

---

{

round_places

(

0

,

1

```
)
}
// 0
{
round_places
(
0.3
,
1
)
}
// 0.3
{
round_places
(
5.001
,
1
)
}
// 5.0
{
round_places
(
5.001
,
```

```
2
)
}
// 5.0
{
round_places
(
5.001
,
3
)
}
// 5.001
{
round_places
(
5.5
,
1
)
}
// 5.5
{
round_places
(
5.999
```

,

1

)

}

// 6.0

{

round_places

(

-

2.07

,

1

)

}

// -2.1

{

round_places

(

13

,

-

1

)

}

// 10

{

round_places

(

252

,

-

2

)

}

// 200

===

See also

round(x)

Bare Flame game

¶

This tutorial assumes that you have basic familiarity with using the command line, and the following progra

Flutter

, version 3.13.0 or above.

Android Studio

, or any other IDE for example

Visual Studio Code

.

git

(optional), to save your project on GitHub.

1. Check flutter installation

¶

First, let?s verify that your Flutter SDK was installed correctly and is accessible from the command line:

$

flutter

doctor Doctor

summary

(

to

see

all

details,

run

flutter

doctor

-v

)

:

[

?

]

Flutter

(

Channel

stable,

3

.13.7,

on

macOS

13

.6

22G120

darwin-arm64,

locale

en

)

[

?

]

Android

toolchain

-

develop

for

Android

devices

(

Android

SDK

version

33

.0.0

)

[

?

]

Xcode

-

develop

for

iOS

and

macOS

(

Xcode

15

.0

)

[

?

]

Chrome

-

develop

for

the

web

[

?

]

Android

Studio

(

version

2021

.2

)

[

?

]

IntelliJ

IDEA

Community

Edition

(

version

2022

.2.2

)

[

?

]

VS

Code

(

version

1

.83.0

)

[

?

]

Connected

device

(

2

available

)

[

?

]

Network

resources  ?

No

issues

found!

Your output will be slightly different, but the important thing is to verify that no errors are reported and that y

3.13.0

.

2. Create the Project Directory

¶

Now you need to come up with a name for your project. The name can only use lowercase Latin letters, dig

syzygy

, which is a totally real non-made-up word.

Create the directory for your new project:

mkdir

-p

~/projects/syzygy

cd

~/projects/syzygy

3. Initialize empty Flutter project

¶

To turn this barren directory into an actual Flutter project, run the following command:

flutter

create

.

(I have omitted the output for brevity, but there will be lots of output).

You can verify that the project files were created successfully:

$

ls README.md

android/

lib/

pubspec.yaml

test/ analysis_options.yaml

ios/

pubspec.lock

syzygy.iml

web/

4. Open the project in Android Studio

¶

Launch Android Studio, then in the project selection window choose

[Open]

and navigate to your project directory. With any luck, the project will now look like this:

If you see only the

main.dart

file but not the side panel, then click the vertical

[Project]

button at the left edge of the window.

Before we proceed, let?s fix the view in the left panel. Locate the button in the top left corner that says

[Android]

in the screenshot. In this dropdown select the first option ?Project?. Your project window should now look l

The important part is that you should be able to see all files in your project directory.

5. Clean up the project files

¶

The default project created by Flutter is not very useful for making a Flame game, so we should get rid of it

First, open the file

pubspec.yaml

and replace it with the following code (adjusting the

name

and

description

to match your project):

name

:

syzygy

description

:

Syzygy Flame game

version

:

0.0.0

publish_to

:

none

environment

:

sdk

:

^3.0.0

flutter

:

^3.13.0

dependencies

:

flutter

:

sdk

:

flutter

flame

:

^1.14.0

After that, press the

[Pub

get]

button at the top of the window, or you could run command

flutter

pub

get

from the terminal. This will ?apply? the changes in

pubspec

file to your project, in particular, it will download the Flame library which we have declared as a dependenc

flutter

pub

get

whenever you make changes to this file.

Now, open the file

lib/main.dart

and replace its content with the following:

import

'package:flame/game.dart'

;

import

'package:flutter/widgets.dart'

;

void

main

()

{

final

game

=

FlameGame

();

runApp

(

GameWidget

(

game:

game

));

}

Lastly, remove the file

test/widget_test.dart

completely.

6. Run the project

¶

Let?s verify that everything is working as intended, and the project can run.

In the menu bar at the top of the window find a dropdown that says

<no

device

selected>

. In that dropdown choose

<Chrome

(web)>

instead.

After that open the

main.dart

file and press the green arrow next to the

void

main()

function in line 4. Select

[Run

main.dart]

from the menu.

This should open a new Chrome window (which may take 10-30 seconds) and run your project in that winc

7. Sync to GitHub

¶

The last step is to upload your project to GitHub. This is not required but strongly recommended as it will s

Log into your GitHub account, select

[Your

repositories]

from your profile dropdown, and press the green

[New]

button. In the form, enter the repository name the same as your project name; select type ?private?; and o

README

,

license

, and

.gitignore

.

Now go to your project?s directory in the terminal and execute the following commands (make sure to repla

git

init git

add

--all git

commit

-m

'Initial commit'

git

remote

add

origin

https://github.com/your-github-username/syzygy.git git

branch

-M

main git

push

-u

origin

main

At this point, if you go to your repository page on GitHub, you shall see that all your project files are there.

8. Done

¶

That?s it! By this point you have

Created an initial blank state Flame project;

Set up the Android Studio IDE for that project;

Created a GitHub repository for the project.

Happy coding!

Development

¶

Contributing

Documentation

Style Guide

Tests Guide

flame_splash_screen

¶

Style your flame game with a beautiful splash screen.

flame_splash_screen is a very customizable splash screen package.

FlameSplashScreen

(

theme:

FlameSplashTheme

.

dark

,

onFinish:

(

BuildContext

context

)

=>

Navigator

.

pushNamed

(

context

,

'/your-game-initial-screen'

)

)

Check the

package?s repo

and the

pub page

for more details.

flame_oxygen

¶

flame_oxygen

Commands

¶

The

commands

are special instructions surrounded with double angle-brackets:

<<stop>>

. There are both

built-in

and

user-defined

commands.

The

built-in

commands are those that are supported by the YarnSpinner runtime itself. Typically they would alter the ex

The

user-defined

commands are those that you yourself create and then use within your yarn scripts. For a full description of

user-defined commands

.

Built-in commands

¶

Variables

¶

<<character>>

Declares a character (person).

<<declare>>

Declares a global variable.

<<local>>

Declares a local variable.

<<set>>

Updates the value of a variable (either local or global).

Control flow

¶

<<if>>

Conditionally executes certain statements. This is equivalent to the

if

keyword in most programming languages.

<<jump>>

Switches execution to another node.

<<stop>>

Stops executing the current node.

<<visit>>

Temporarily jumps to another node, and then comes back.

<<wait>>

Pauses the dialogue for the specified amount of time.

## Oxygen

¶

We (the Flame organization) built an ECS (Entity Component System) named Oxygen.

If you want to use Oxygen specifically for Flame as a replacement for the FCS(Flame Component System)

flame_oxygen

and if you just want to use it in a Dart project you can use the

oxygen

library directly.

If you are not familiar with Oxygen yet we recommend you read up on its

documentation

.

To use it in your game you just need to add

flame_oxygen

to your

pubspec.yaml

, as can be seen in the

Oxygen example

and in the

pub.dev

installation instructions

.

## OxygenGame (Game extension)

¶

If you are going to use Oxygen in your project it can be a good idea to use the Oxygen specific extension o

Game

class.

It is called

OxygenGame

and it will give you full access to the Oxygen framework while also having full access to the Flame game lo

Instead of using

onLoad

, as you are used to with Flame,

OxygenGame

comes with the

init

method. This method is called in the

onLoad

but before the world initialization, allowing you to register components and systems and do anything else th

onLoad

.

A simple

OxygenGame

implementation example can be seen in the

example folder

.

The

OxygenGame

also comes with it?s own

createEntity

method that automatically adds certain default components on the entity. This is especially helpful when yo

BaseSystem

as your base.

Systems

¶

Systems define the logic of your game. In FCS you normally would add your logic inside a component with

execute

, which is a method equal to the

update

method in Flame.

On each

execute

Oxygen automatically calls all the systems that were registered in order. But in Flame we can have differen

flame_oxygen

we introduced the

RenderSystem

and

UpdateSystem

mixin. These mixins allow you to add the

render

method and the

update

method respectively to your custom system. For more information see the

RenderSystem

and

UpdateSystem

section.

If you are coming from FCS you might expect certain default functionality that you normally got from the

PositionComponent

. As mentioned before components do not contain any kind of logic, but to give you the same default functi

BaseSystem

. This system acts almost identical to the prerender logic from the

PositionComponent

in FCS. You only have to subclass it to your own system. For more information see the

BaseSystem

section.

Systems can be registered to the world using the

world.registerSystem

method on

OxygenGame

.

mixin GameRef

¶

The

GameRef

mixin allows a system to become aware of the

OxygenGame

instance its attached to. This allows easy access to the methods on the game class.

class

YourSystem

extends

System

with

GameRef

<

YourGame

>

{

@override

void

init

()

{

// Access to game using the .game property

}

// ...

}

mixin RenderSystem

¶

The

RenderSystem

mixin allows a system to be registered for the render loop. By adding a

render

method to the system you get full access to the canvas as you normally would in Flame.

class

SimpleRenderSystem

extends

System

with

RenderSystem

{

Query

?

_query

;

@override

void

init

()

{

_query

=

createQuery

([

/* Your filters */

]);

}

void

render

(

Canvas

canvas

)

{

for

(

final

entity

in

_query

?

.

entities

??

<

Entity

>

[])

{

// Render entity based on components

}

}

}

mixin UpdateSystem

¶

The

MixinSystem

mixin allows a system to be registered for the update loop. By adding a

update

method to the system you get full access to the delta time as you normally would in Flame.

class

SimpleUpdateSystem

extends

System

with

UpdateSystem

{

Query

?

_query

;

@override

void

init

()

{

_query

=

createQuery

([

/* Your filters */

]);

}

void

update

(

double

dt

)

{

for

(

final

entity

in

_query

?

.

entities

??

<

Entity

>

[])

{

// Update components values

}

}

}

BaseSystem

¶

The

BaseSystem

is an abstract class whose logic can be compared to the

PositionComponent

from FCS. The

BaseSystem

automatically filters all entities that have the

PositionComponent

and

SizeComponent

from

flame_oxygen

. On top of that you can add your own filters by defining a getter called

filters

. These filters are then used to filter down the entities you are interested in.

The

BaseSystem

is also fully aware of the game instance. You can access the game instance by using the

game

property. This also gives you access to the

createEntity

helper method on

OxygenGame

.

On each render loop the

BaseSystem

will prepare your canvas the same way the

PositionComponent

from FCS would (translating, rotating and setting the anchor. After that it will call the

renderEntity

method so you can add your own render logic for that entity on a prepared canvas.

The following components will be checked by

BaseSystem

for the preparation of the canvas:

PositionComponent

(required)

SizeComponent

(required)

AnchorComponent

(optional, defaults to

Anchor.topLeft

)

AngleComponent

(optional, defaults to

0

)

class

SimpleBaseSystem

extends

BaseSystem

{

@override

List

<

Filter

<

Component

>>

get

filters

=>

[];

@override

void

renderEntity

(

Canvas

canvas

,

Entity

entity

)

{

// The canvas is translated, rotated and fully prepared for rendering.

}

}

ParticleSystem

¶

The

ParticleSystem

is a simple system that brings the Particle API from Flame to Oxygen. This allows you to use the

ParticleComponent

without having to worry about how it will render or when to update it. As most of that logic is already contain

Simply register the

ParticleSystem

and the

ParticleComponent

to your world like so:

```
world
.
registerSystem
(
ParticleSystem
());
world
.
registerComponent
<
ParticleComponent
,
Particle
>
(()
=>
ParticleComponent
);
```

You can now create a new entity with a

ParticleComponent

. For more info about that see the

ParticleComponent

section.

5. Controlling Movement

¶

If you were waiting for some serious coding, this chapter is it. Prepare yourself as we dive in!

Keyboard Controls

¶

The first step will be to allow control of Ember via the keyboard. We need to start by adding the appropriate

lib/ember_quest.dart

import

'package:flame/events.dart'

;

class

EmberQuestGame

extends

FlameGame

with

HasKeyboardHandlerComponents

{

lib/actors/ember.dart

class

EmberPlayer

extends

SpriteAnimationComponent

with

KeyboardHandler

,

HasGameReference

<

EmberQuestGame

>

{

Now we can add a new method:

@override

bool

onKeyEvent

(

RawKeyEvent

event

,

Set

<

LogicalKeyboardKey

>

keysPressed

)

{

return

true

;

}

Like before, if this did not trigger an auto-import, you will need the following:

import

'package:flutter/services.dart'

;

To control Ember?s movement, it is easiest to set a variable where we think of the direction of movement li

int

horizontalDirection

=

0

;

Now in our

onKeyEvent

method, we can register the key pressed by adding:

@override

bool

onKeyEvent

(

RawKeyEvent

event

,

Set

<

LogicalKeyboardKey

>

keysPressed

)

{

horizontalDirection

=

0

;

horizontalDirection

+=

(

keysPressed

.

contains

(

LogicalKeyboardKey

.

keyA

)

||

keysPressed

.

contains

(

LogicalKeyboardKey

.

arrowLeft

))

?

-

1

:

0

;

horizontalDirection

+=

(

keysPressed

.

contains

(

LogicalKeyboardKey

.

keyD

)

||

keysPressed

.

contains

(

LogicalKeyboardKey

.

arrowRight

))

?

1

:

0

;

return

true

;

}

Let?s make Ember move by adding a few lines of code and creating our

update

method. First, we need to define a velocity variable for Ember. Add the following at the top of the

EmberPlayer

class:

final

Vector2

velocity

=

Vector2

.

zero

();

final

double

moveSpeed

=

200

;

This establishes a base velocity of 0 and stores

moveSpeed

so we can adjust as necessary to suit how the game-play should be. Next, add the

update

method with the following:

```
@override
void
update
(
double
dt
)
{
velocity
.
x
=
horizontalDirection
*
moveSpeed
;
position
+=
velocity
*
dt
;
super
```

.

update

(

dt

);

}

If you run the game now, Ember moves left and right using the arrow keys or the

A

and

D

keys. You may have noticed that Ember doesn?t look back if you are going left, to fix that, add the following

update

method:

if

(

horizontalDirection

<

0

&&

scale

.

x

>

0

)

{

flipHorizontally

();

}

else

if

(

horizontalDirection

>

0

&&

scale

.

x

<

0

)

{

flipHorizontally

();

}

Now Ember looks in the direction they are traveling.

Collisions

¶

It is time to get into the thick of it with collisions. I highly suggest reading the

documentation

to understand how collisions work in Flame. The first thing we need to do is make the game aware that coll

HasCollisionDetection

mixin. Add that to

lib/ember_quest.dart

like:

class

EmberQuestGame

extends

FlameGame

with

HasCollisionDetection

,

HasKeyboardHandlerComponents

{

Next, add the

CollisionCallbacks

mixin to

lib/actors/ember.dart

like:

class

EmberPlayer

extends

SpriteAnimationComponent

with

KeyboardHandler

,

CollisionCallbacks

,

HasGameReference

<

EmberQuestGame

>

{

If it did not auto-import, you will need the following:

import

'package:flame/collisions.dart'

;

Now add the following

onCollision

method:

@override

void

onCollision

(

Set

<

Vector2

>

intersectionPoints

,

PositionComponent

other

)

{

if

(

other

is

GroundBlock

||

other

is

PlatformBlock

)

{

if

(

intersectionPoints

.

length

==

2

)

{

// Calculate the collision normal and separation distance.

final

mid

=

(

intersectionPoints

.

elementAt

(

0

)

+

intersectionPoints

.

elementAt

(

1

))

/

2

;

final

collisionNormal

=

absoluteCenter

-

mid

;

final

separationDistance

=

(

size

.

x

/

2

)

-

collisionNormal

.

length

;

collisionNormal

.

normalize

();

// If collision normal is almost upwards,

// ember must be on ground.

if

(

fromAbove

.

dot

(

collisionNormal

)

>

0.9

)

{

isOnGround

=

true

;

}

// Resolve collision by moving ember along

// collision normal by separation distance.

position

+=

collisionNormal

.

scaled

(

separationDistance

);

}

}

super

.

onCollision

(

intersectionPoints

,

other

);

}

You will need to import the following:

import

'../objects/ground_block.dart'

;

import

'../objects/platform_block.dart'

;

As well as create these class variables:

final

Vector2

fromAbove

=

Vector2

(

0

,

-

1

);

bool

isOnGround

=

false

;

For the collisions to be activated for Ember, we need to add a

CircleHitbox

, so in the

onLoad

method, add the following:

add

(

CircleHitbox

());

Now that we have the basic collisions created, we can add gravity so Ember exists in a game world with ve

final

double

gravity

=

15

;

final

double

jumpSpeed

=

600

;

final

double

terminalVelocity

=

150

;

bool

hasJumped

=

false

;

Now we can add Ember?s ability to jump by adding the following to our

onKeyEvent

method:

hasJumped

=

keysPressed

.

contains

(

LogicalKeyboardKey

.

space

);

Finally, in our

update

method we can tie this all together with:

// Apply basic gravity

velocity

.

y

+=

gravity

;

// Determine if ember has jumped

if

(

hasJumped

)

{

if

(

isOnGround

)

{

velocity

.

y

=

-

jumpSpeed

;

isOnGround

=

false

;

}

hasJumped

=

false

;

}

// Prevent ember from jumping to crazy fast as well as descending too fast and

// crashing through the ground or a platform.

velocity

.

y

=

velocity

.

y

.

clamp

(

-

jumpSpeed

,

terminalVelocity

);

Earlier I mentioned that Ember was in the center of the grass, to solve this and show how collisions and gra

lib/ember_quest.dart

in the

initializeGame

method, change the following:

_ember

=

EmberPlayer

(

position:

Vector2

(

128

,

canvasSize

.

y

-

128

),

);

If you run the game now, Ember should be created and fall to the ground; then you can jump around!

Collisions with Objects

¶

Adding the collisions with the other objects is fairly trivial. All we need to do is add the following to the botto

onCollision

method:

if

(

other

is

Star

)

{

other

.

removeFromParent

();

}

if

(

other

is

WaterEnemy

)

{

hit

();

}

When Ember collides with a star, the game will remove the star, and to implement the

hit

method for when Ember collides with an enemy, we need to do the following:

Add the following variable at the top of the

EmberPlayer

class:

bool

hitByEnemy

=

false

;

Additionally, add this method to the

EmberPlayer

class:

// This method runs an opacity effect on ember

// to make it blink.

void

hit

()

{

if

(

!

hitByEnemy

)

{

hitByEnemy

=

true

;

}

add

(

OpacityEffect

.

fadeOut

(

EffectController

(

alternate:

true

,

duration:

0.1

,

repeatCount:

6

,

),

)..

onComplete

=

()

{

hitByEnemy

=

false

;

},

);

}

If the auto-imports did not occur, you will need to add the following imports to your file:

import

'package:flame/effects.dart'

;

import

'../objects/star.dart'

;

import

'water_enemy.dart'

;

If you run the game now, you should be able to move around, make stars disappear, and if you collide with

Adding the Scrolling

¶

This is our last task with Ember. We need to restrict Ember?s movement because as of now, Ember can g

update

method:

game

.

objectSpeed

=

0

;

// Prevent ember from going backwards at screen edge.

if

(

position

.

x

-

36

<=

0

&&

horizontalDirection

<

0

)

{

velocity

.

x

=

0

;

}

// Prevent ember from going beyond half screen.

if

(

position

.

x

+

64

>=

game

.

size

.

x

/

2

&&

horizontalDirection

>

0

)

{

velocity

.

x

=

0

;

game

.

objectSpeed

=

-

moveSpeed

;

}

position

+=

velocity

*

dt

;

super

.

update

(

dt

);

If you run the game now, Ember can?t move off-screen to the left, and as Ember moves to the right, once t

game.objectSpeed

which we established early on in the series. Additionally, you will see the next random segment be generat

Note

As I mentioned earlier, I would add a section on how this game could be adapted to a traditional level game

3. Building the World

, we could add a segment that has a door or a special block. For every

X

number of segments loaded, we could then add that special segment. When Ember reaches that object, w

We are almost done! In

6. Adding the HUD

, we will add the health system, keep track of the score, and provide a HUD to relay that information to the

MarkupAttribute

¶

A

MarkupAttribute

is a descriptor of a subrange of text in a

line

, demarcated with markup tags. For example, in a

.yarn

line below there are two ranges of text surrounded by markup tags, and therefore there will be two

MarkupAttribute

s associated with this line:

[b]

Jenny

[/b]

is a library based on

\

[link url="docs.yarnspinner.dev"]

YarnSpinner

[/link]

for Unity.

These

MarkupAttribute

s can be found in the

.attributes

property of a

DialogueLine

.

Properties

¶

name

String

The name of the markup tag. In the example above, the name of the first attribute is

"b"

, and the second is

"link"

.

start

,

end

int

The location of the marked-up span within the final text of the line. The first index is inclusive, while the sec

start

may be equal to

end

for a zero-width markup attribute.

length

int

The length of marked-up text. This is always equal to

end

-

start

.

parameters

Map<String,

dynamic>

The set of parameters associated with this markup attribute. In the example above, the first markup attribut

{"url":

"docs.yarnspinner.dev"}

.

The type of each parameter will be either

String

,

num

, or

bool

, depending on the type of expression give in the

.yarn

script. The expressions for parameter values can be dynamic, that is they can be evaluated at runtime. In t

color

will be equal to the value of the variable

$color

, which may change each time the line is run.

My

[i]

favorite

[/i]

color is

[bb color=$color]

{

$color

}

[/bb]

.

Rendering

¶

Colors and Palette

Decorators

Images, Sprites and Animations

Layers

Particles

Text Rendering

FunctionStorage

¶

The

FunctionStorage

is a part of [YarnProject] responsible for storing all

user-defined functions

. You can access it as the

YarnProject.functions

property.

The function storage can be used to register any number of custom functions, making them available to us

A Dart function can be registered as a user-defined function in Jenny, if it satisfies the following requiremer

its return type is one of

int

,

double

,

num

,

bool

, or

String

;

all its arguments are positional, i.e. there are no named arguments;

all its arguments have types

int

,

int?

,

double

,

double?

,

num

,

num?

,

bool

,

bool?

,

String

, or

String?

;

the nullable arguments, if any, must come after the non-nullable ones. These arguments become optional i

null

values;

the first argument in a function can also be

YarnProject

. If such argument is present, then it will be passed automatically. For example, if you have a function

fn(YarnProject,

int)

, then it can be invoked from the yarn script simply as

fn(1)

.

A Dart function can then be added using one of the methods

addFunction0

, ?,

addFunction4

, depending on how many arguments the function has (this is a limitation of Dart?s template language). Wh

name

, under this function will be known in Yarn scripts. This name can be the same, or different from the real fun

hasVisitedTheWizard()

in your game, but you?d register it under the name

has_visited_the_wizard()

in your YarnProject.

Keep in mind that the name of the user-defined function must be:

unique;

a valid ID;

cannot be the same as any built-in function.

Methods

¶

hasFunction

(

String

name

) ?

bool

Returns the status of whether the function

name

has been already registered.

addFunction0

(

String

name

,

T0

Function()

fn

)

Registers a no-argument function

fn

as the user-defined function

name

.

addFunction1

(

String

name

,

T0

Function(T1)

fn1

)

Registers a single-argument function

fn1

under the name

name

.

addFunction2

(

String

name

,

T0

Function(T1,

T2)

fn2

)

Registers a two-argument function

fn2

with the given

name

.

addFunction3

(

String

name

,

T0

Function(T1,

T2,

T3)

fn3

)

Registers a three-argument function

fn3

with the name

name

.

addFunction4

(

String

name

,

T0

Function(T1,

T2,

T3,

T4)

fn4

)

Registers a four-argument function

fn4

as

name

.

clear

Removes all user-defined functions

remove

(

String

name

)

Removes the user-defined function with the specified

name

.

Properties

¶

length

?

int

The number of user-defined functions registered so far.

isEmpty

?

bool

Returns

true

if no user-defined functions were registered.

isNotEmpty

?

bool

Has any functions been registered at all?

Resources

¶

Flame API

Flame Examples

Other

¶

Debugging

Utils

Widgets

Getting Started

¶

This tutorial will guide you on the development of a full Flame game, starting from the ground up, step by s

This first part will introduce you to:

FlameGame

: The base class for games using the Flame Component System.

GameWidget

: The

Widget

that will insert your game into the Flutter widget tree.

PositionComponent

: One of the most basic Flame components holds both a position and dimension in the game space.

Let?s start by creating our game class and the

GameWidget

that will run it.

```
import

'package:flutter/material.dart'

;

import

'package:flame/game.dart'

;

class

SpaceShooterGame

extends

FlameGame

{
```

}

void

main

()

{

runApp

(

GameWidget

(

game:

SpaceShooterGame

()));

}

That is it! If you run this, you will only see an empty black screen for now, from that, we can start implemen

Next, let?s create our player component. To do so, we will create a new class based on Flame?s

PositionComponent

. This component is the base for all components that have a position and a size on the game screen. For n

import

'package:flame/components.dart'

;

class

Player

extends

PositionComponent

{

static

final

_paint

=

Paint

()..

color

=

Colors

.

white

;

@override

void

render

(

Canvas

canvas

)

{

canvas

.

drawRect

(

size

.

toRect

(),

_paint

);

}

}

Now, let?s add our new component to the game. Adding any component on game startup should be done i

onLoad

method, so let?s override

FlameGame.onLoad

and add our logic there. The modified code will look like the following:

class

SpaceShooterGame

extends

FlameGame

{

@override

Future

<

void

>

onLoad

()

async

{

await

super

.

onLoad

();

add

(

Player

()

..

position

=

size

/

2

..

width

=

50

..

height

=

100

..

anchor

=

Anchor

.

center

,

);

}

}

If you run this, you will now see a white rectangle being rendered in the center of the screen.

A couple of points worth commenting:

size

is a

Vector2

variable from the game class and it holds the current dimension of the game area, where

x

is the horizontal dimension or the width, and

y

is the vertical dimension or the height.

By default, Flame follows Flutter?s canvas anchoring, which means that (0, 0) is anchored on the top left c

anchor

attribute to

Anchor.center

, which will make our life way easier if you want to center the component on the screen.

And that is it for this first part! In this first step, we learned the basics of how to create a game class, insert

Run

main.dart

1

import

'package:flame/components.dart'

```
;
2
import
'package:flame/game.dart'
;
3
import
'package:flutter/material.dart'
;
4
5
void
main
()
{
6
runApp
(
GameWidget
(
game:
SpaceShooterGame
()));
7
}
8
```

9

class

Player

extends

PositionComponent

{

10

static

final

_paint

=

Paint

()..

color

=

Colors

.

white

;

11

@override

12

void

render

(

Canvas

canvas

)

{

13

canvas

.

drawRect

(

size

.

toRect

(),

_paint

);

14

}

15

}

16

17

class

SpaceShooterGame

extends

FlameGame

{

18

@override

19

Future

<

void

>

onLoad

()

async

{

20

add

(

21

Player

()

22

..

position

=

size

/

2

23

..

width

=

50

24

..

height

=

100

25

..

anchor

=

Anchor

.

center

,

26

);

27

}

28

}

Code

UserDefinedCommand

¶

The

UserDefinedCommand

class represents a single invocation of a custom (non-built-in) command within a yarn script. Objects of this

DialogueView

in its

.onCommand()

method.

Properties

¶

name

String

The name of the command, without the angle brackets. For example, if the command is

<<smile>>

in the yarn script, then its name will be

"smile"

.

argumentString

String

Command arguments, as a single string. For example, if the command is

<<move

Hippo

{$delta}>>

, and the value of variable

$delta

is

3.17

, then the argument string will be

"Hippo

3.17"

.

The

argumentString

is re-evaluated every time the command is executed, however, it is an error to access this property before

arguments

List<dynamic>?

Command arguments, as a list of parsed values. This property will be null if the command was declared wi

In the same example as above, the

arguments

will be

['Hippo',

3.17]

, assuming the linked Dart function is

move(String

target,

double

distance)

.

See also

¶

The description of

User-defined Commands

in the YarnSpinner language.

The guide on how to register a new custom command in the

CommandStorage

document.

4. Adding the Remaining Components

¶

Star

¶

The star is pretty simple. It is just like the Platform block except we are going to add an effect to make it pu

Anchor

to

center

. This means we will need to adjust the position by half of the image size. For brevity, I am going to add the

import

'package:flame/collisions.dart'

;

import

'package:flame/components.dart'

;

import

'package:flame/effects.dart'

;

import

'package:flutter/material.dart'

;

import

'../ember_quest.dart'

;

class

Star

extends

SpriteComponent

with

HasGameReference

<

EmberQuestGame

>

{

final

Vector2

gridPosition

;

double

xOffset

;

final

Vector2

velocity

=

Vector2

.

zero

();

Star

({

required

this

.

gridPosition

,

required

this

.

xOffset

,

})

:

super

(

size:

Vector2

.

all

(

64

),

anchor:

Anchor

.

center

);

@override

```
void
onLoad
()
{
final
starImage
=
game
.
images
.
fromCache
(
'star.png'
);
sprite
=
Sprite
(
starImage
);
position
=
Vector2
(
(
```

gridPosition

.

x

*

size

.

x

)

+

xOffset

+

(

size

.

x

/

2

),

game

.

size

.

y

-

(

gridPosition

.

y

*

size

.

y

)

-

(

size

.

y

/

2

),

);

add

(

RectangleHitbox

(

collisionType:

CollisionType

.

passive

));

add

(

SizeEffect

.

by

(

Vector2

(

-

24

,

-

24

),

EffectController

(

duration:

.

75

,

reverseDuration:

.

5

,

infinite:

true

,

curve:

Curves

.

easeOut

,

),

),

);

}

@override

void

update

(

double

dt

)

{

velocity

.

x

=

game

.

objectSpeed

;

position

+=

velocity

*

dt

;

if

(

position

.

x

<

-

size

.

x

)

removeFromParent

();

super

.

update

(

dt

);

}

}

So the only change between the Star and the Platform beyond the anchor is simply the following:

```
add
(
SizeEffect
.
by
(
Vector2
(
-
24
,
-
24
),
EffectController
(
duration:
.
75
,
reverseDuration:
.
5
,
infinite:
```

true

,

curve:

Curves

.

easeOut

,

),

),

);

The

SizeEffect

is best explained by going to their

docs

. In short, we simply reduce the size of the star by -24 pixels in both directions and we make it pulse infinite

EffectController

.

Don?t forget to add the star to your

lib/ember_quest.dart

file by doing:

case

Star:

world

.

add

(

Star

(

gridPosition:

block

.

gridPosition

,

xOffset:

xPositionOffset

,

),

);

break

;

If you run your game, you should now see pulsating stars!

Water Enemy

¶

Now that we understand adding effects to our objects, let?s do the same for the water drop enemy. Open

lib/actors/water_enemy.dart

and add the following code:

import

'package:flame/collisions.dart'

;

import

'package:flame/components.dart'

;

```
import
'package:flame/effects.dart'
;
import
'../ember_quest.dart'
;
class
WaterEnemy
extends
SpriteAnimationComponent
with
HasGameReference
<
EmberQuestGame
>
{
final
Vector2
gridPosition
;
double
xOffset
;
final
Vector2
velocity
```

=

Vector2

.

zero

();

WaterEnemy

({

required

this

.

gridPosition

,

required

this

.

xOffset

,

})

:

super

(

size:

Vector2

.

all

(

```
64
),
anchor:
Anchor
.
bottomLeft
);
@override
void
onLoad
()
{
animation
=
SpriteAnimation
.
fromFrameData
(
game
.
images
.
fromCache
(
'water_enemy.png'
),
```

SpriteAnimationData

.

sequenced

(

amount:

2

,

textureSize:

Vector2

.

all

(

16

),

stepTime:

0.70

,

),

);

position

=

Vector2

(

(

gridPosition

.

x

*

size

.

x

)

+

xOffset

,

game

.

size

.

y

-

(

gridPosition

.

y

*

size

.

y

),

);

add

(

RectangleHitbox

(

collisionType:

CollisionType

.

passive

));

add

(

MoveEffect

.

by

(

Vector2

(

-

2

*

size

.

x

,

0

),

EffectController

(

duration:

3

,

alternate:

true

,

infinite:

true

,

),

),

);

}

@override

void

update

(

double

dt

)

{

velocity

.

x

=

game

.

objectSpeed

;

position

+=

velocity

*

dt

;

if

(

position

.

x

<

-

size

.

x

)

removeFromParent

();

super

.

update

(

dt

);

}

}

The water drop enemy is an animation just like Ember, so this class is extending the

SpriteAnimationComponent

class but it uses all of the previous code we have used for the Star and the Platform. The only difference w

SizeEffect

, we are going to use the

MoveEffect

. The best resource for information will be their

help docs

.

In short, the

MoveEffect

will last for 3 seconds, alternate directions, and run infinitely. It will move our enemy to the left, 128 pixels (

Don?t forget to add the water enemy to your

lib/ember_quest.dart

file by doing:

case

WaterEnemy:

world

.

add

(

WaterEnemy

(

gridPosition:

block

.

gridPosition

,

xOffset:

xPositionOffset

,

),

);

break

;

If you run the game now, the Water Enemy should be displayed and moving!

Ground Blocks

¶

Finally, the last component that needs to be displayed is the Ground Block! This component is more compl

When the block is added, if it is the last block in the segment, we need to update a global value as to its po

When the block is removed, if it was the first block in the segment, we need to randomly get the next segm

So let?s start with the basic class which is nothing more than a copy of the Platform Block.

import

'package:flame/collisions.dart'

;

import

'package:flame/components.dart'

;

import

'package:flutter/material.dart'

;

import

'../ember_quest.dart'

;

class

GroundBlock

extends

SpriteComponent

with

HasGameReference

<

EmberQuestGame

>

{

final

Vector2

gridPosition

;

double

xOffset

;

final

Vector2

velocity

=

Vector2

.

zero

();

GroundBlock

({

required

this

.

gridPosition

,

required

this

.

xOffset

,

})

:

super

(

size:

Vector2

.

all

(

64

),

anchor:

Anchor

.

bottomLeft

);

@override

void

onLoad

()

{

final

groundImage

=

game

.

images

.

fromCache

(

'ground.png'

);

sprite

=

Sprite

(

groundImage

);

position

=

Vector2

(

gridPosition

.

x

*

size

.

x

+

xOffset

,

game

.

size

.

y

-

gridPosition

.

y

*

size

.

y

,

);

add

(

RectangleHitbox

(

collisionType:

CollisionType

.

passive

));

}

@override

void

update

(

double

dt

)

{

velocity

.

x

=

game

.

objectSpeed

;

position

+=

velocity

*

dt

;

super

.

update

(

dt

);

}

}

The first thing we will tackle is registering the block globally if it is the absolute last block to be loaded. To d

lib/ember_quest.dart

called:

late

double

lastBlockXPosition

=

0.0

;

late

UniqueKey

lastBlockKey

;

Declare the following variable at the top of your Ground Block class:

final

UniqueKey

_blockKey

=

UniqueKey

();

Now in your Ground Block?s

onLoad

method, add the following at the end of the method:

if

(

gridPosition

.

x

==

9

&&

position

.

x

>

game

.

lastBlockXPosition

)

{

game

.

lastBlockKey

=

_blockKey

;

game

.

lastBlockXPosition

=

position

.

x

+

size

.

x

;

}

All that is happening is if this block is the 10th block (9 as the segment grid is 0 based) AND this block?s po

lastBlockXPosition

, set the global block key to be this block?s key and set the global

lastBlockXPosition

to be this blocks position plus the width of the image (the anchor is bottom left and we want the next block t

Now we can address updating this information, so in the

update

method, add the following code:

@override

void

update

(

double

dt

)

{

velocity

.

x

=

game

.

objectSpeed

;

position

+=

velocity

*

dt

;

if

(

gridPosition

.

x

==

9

)

{

if

(

game

.

lastBlockKey

==

_blockKey

)

{

game

.

lastBlockXPosition

=

position

.

x

+

size

.

x

-

10

;

}

}

super

.

update

(

dt

);

}

game.lastBlockXPosition

is being updated by the block?s current x-axis position plus its width - 10 pixels. This will cause a little over

dt

this prevents gaps in the map as it loads while a player is moving.

Loading the Next Random Segment

¶

To load the next random segment, we will use the

Random()

function that is built-in to

dart:math

. The following line of code gets a random integer from 0 (inclusive) to the max number in the passed para

Random

().

nextInt

(

segments

.

length

),

Back in our Ground Block, we can now add the following to our ?update? method before the other block w

if

(

position

.

x

<

-

size

.

x

)

{

removeFromParent

();

if

(

gridPosition

.

x

==

0

)

{

game

.

loadGameSegments

(

Random

().

nextInt

(

segments

.

length

),

game

.

lastBlockXPosition

,

);

}

}

This simply extends the code that we have in our other objects, where once the block is off the screen and

loadGameSegments

method in our game class, get a random number between 0 and the number of segments and pass in the c

Random()

or

segments.length

does not auto-import, you will need:

import

'dart:math'

;

import

'../managers/segment_manager.dart'

;

So our full Ground Block class should look like this:

import

'dart:math'

;

import

'package:flame/collisions.dart'

;

import

'package:flame/components.dart'

;

import

'package:flutter/material.dart'

;

import

'../ember_quest.dart'

;

import

'../managers/segment_manager.dart'

;

class

GroundBlock

extends

SpriteComponent

with

HasGameReference

<

EmberQuestGame

>

{

final

Vector2

gridPosition

;

double

xOffset

;

final

UniqueKey

_blockKey

=

UniqueKey

();

final

Vector2

velocity

=

Vector2

.

zero

();

GroundBlock

({

required

this

.

gridPosition

,

required

this

.

xOffset

,

})

:

super

(

size:

Vector2

.

all

(

64

),

anchor:

Anchor

.

bottomLeft

);

@override

void

onLoad

()

{

final

groundImage

=

game

.

images

.

fromCache

(

'ground.png'

);

sprite

=

Sprite

(

groundImage

);

position

=

Vector2

(

gridPosition

.

x

*

size

.

x

+

xOffset

,

game

.

size

.

y

-

gridPosition

.

y

*

size

.

y

,

);

add

(

RectangleHitbox

(

collisionType:

CollisionType

.

passive

));

if

(

gridPosition

.

x

==

9

&&

position

.

x

>

game

.

lastBlockXPosition

)

{

game

.

lastBlockKey

=

_blockKey

;

game

.

lastBlockXPosition

=

position

.

x

+

size

.

x

;

}

}

@override

void

update

(

double

dt

)

{

velocity

.

x

=

game

.

objectSpeed

;

position

+=

velocity

*

dt

;

if

(

position

.

x

<

-

size

.

x

)

{

removeFromParent

();

if

(

gridPosition

.

x

==

0

)

{

game

.

loadGameSegments

(

Random

().

nextInt

(

segments

.

length

),

game

.

lastBlockXPosition

,

);

}

}

if

(

gridPosition

.

x

==

9

)

{

if

(

game

.

lastBlockKey

==

_blockKey

)

{

game

.

lastBlockXPosition

=

position

.

x

+

size

.

x

-

10

;

}

}

super

.

update

(

dt

);

}

}

Finally, don?t forget to add your Ground Block to

lib/ember_quest.dart

by adding the following:

case

GroundBlock:

world

.

add

(

GroundBlock

(

gridPosition:

block

.

gridPosition

,

xOffset:

xPositionOffset

,

),

);

break

;

If you run your code, your game should now look like this:

You might say, but wait! Ember is in the middle of the ground and that is correct because Ember?s

Anchor

is set to center. This is ok and we will be addressing this in

5. Controlling Movement

where we will be adding movement and collisions to Ember!

Widget

¶

RiverpodAwareGameWidget

¶

RiverpodAwareGameWidget

is a GameWidget with a

State

object of type

RiverpodAwareGameWidgetState

.

The required

GlobalKey

argument is used to provide

Component

s using

RiverpodComponentMixin

access to

Provider

s via

RiverpodAwareGameWidgetState

.

RiverpodAwareGameWidgetState

¶

RiverpodAwareGameWidgetState

performs the duties associated with the

ConsumerStatefulElement

in

flutter_riverpod

and

GameWidgetState

in

flame

.

Other Inputs and Helpers

¶

This includes documentation for input methods besides keyboard and mouse.

For other input documents, see also:

Gesture Input

: for mouse and touch pointer gestures

Keyboard Input

: for keystrokes

Joystick

¶

Flame provides a component capable of creating a virtual joystick for taking input for your game. To use th

JoystickComponent

, configure it the way you want, and add it to your game.

Check this example to get a better understanding:

class

MyGame

extends

FlameGame

with

HasDraggables

{

MyGame

()

{

joystick

.

```
addObserver
(
player
);
add
(
player
);
add
(
joystick
);
}
@override
Future
<
void
>
onLoad
()
async
{
super
.
onLoad
();
```

final

image

=

await

images

.

load

(

'joystick.png'

);

final

sheet

=

SpriteSheet

.

fromColumnsAndRows

(

image:

image

,

columns:

6

,

rows:

1

,

);

final

joystick

=

JoystickComponent

(

knob:

SpriteComponent

(

sprite:

sheet

.

getSpriteById

(

1

),

size:

Vector2

.

all

(

100

),

),

background:

SpriteComponent

(

sprite:

sheet

.

getSpriteById

(

0

),

size:

Vector2

.

all

(

150

),

),

margin:

const

EdgeInsets

.

only

(

left:

40

,

bottom:

```
40
),
);
final
player
=
Player
(
joystick
);
add
(
player
);
add
(
joystick
);
}
}
class
JoystickPlayer
extends
SpriteComponent
with
HasGameRef
```

{

JoystickPlayer

(

this

.

joystick

)

:

super

(

anchor:

Anchor

.

center

,

size:

Vector2

.

all

(

100.0

),

);

/// Pixels/s

double

maxSpeed

=

300.0

;

final

JoystickComponent

joystick

;

@override

Future

<

void

>

onLoad

()

async

{

sprite

=

await

gameRef

.

loadSprite

(

'layers/player.png'

);

position

=

gameRef

.

size

/

2

;

}

@override

void

update

(

double

dt

)

{

if

(

joystick

.

direction

!=

JoystickDirection

.

idle

)

{

position

.

add

(

joystick

.

velocity

*

maxSpeed

*

dt

);

angle

=

joystick

.

delta

.

screenAngle

();

}

}

}

So in this example, we create the classes

MyGame

and

Player

.

MyGame

creates a joystick which is passed to the

Player

when it is created. In the

Player

class we act upon the current state of the joystick.

The joystick has a few fields that change depending on what state it is in. These are the fields that should b

intensity

: The percentage [0.0, 1.0] that the knob is dragged from the epicenter to the edge of the joystick (or

knobRadius

if that is set).

delta

: The absolute amount (defined as a

Vector2

) that the knob is dragged from its epicenter.

velocity

: The percentage, presented as a

Vector2

, and direction that the knob is currently pulled from its base position to a edge of the joystick.

If you want to create buttons to go with your joystick, check out

HudButtonComponent

.

A full examples of how to use it can be found

here

. And it can be seen running

here

.

There is also a more advanced example

here

which is running

here

.

HudButtonComponent

¶

A

HudButtonComponent

is a button that can be defined with margins to the edge of the

Viewport

instead of with a position. It takes two

PositionComponent

s.

button

and

buttonDown

, the first is used for when the button is idle and the second is shown when the button is being pressed. Th

button

component.

As the name suggests this button is a hud by default, which means that it will be static on your screen ever

hudButtonComponent.respectCamera

=

true;

.

If you want to act upon the button being pressed (which would be the common thing to do) and released, y

onPressed

and

onReleased

arguments, or you can extend the component and override

onTapDown

,

onTapUp

and/or

onTapCancel

and implement your logic there.

SpriteButtonComponent

¶

A

SpriteButtonComponent

is a button that is defined by two

Sprite

s, one that represents when the button is pressed and one that represents when the button is released.

ButtonComponent

¶

A

ButtonComponent

is a button that is defined by two

PositionComponent

s, one that represents when the button is pressed and one that represents when the button is released. If y

SpriteButtonComponent

instead, but this component can be good to use if you for example want to have a

SpriteAnimationComponent

as a button, or anything else which isn?t a pure sprite.

Gamepad

¶

Flame has a separate plugin to support external game controllers (gamepads), check

here

for more information.

AdvancedButtonComponent

¶

The

AdvancedButtonComponent

have separate states for each of the different pointer phases. The skin can be customized for each state ar

PositionComponent

.

These are the fields that can be used to customize the looks of the

AdvancedButtonComponent

:

defaultSkin

: Component that will be displayed by default on the button.

downSkin

: Component displayed when the button is clicked or tapped.

hoverSkin

: Component displayed when the button is hovered. (desktop and web).

defaultLabel

: Component shown on top of skins. Automatically aligned to center.

disabledSkin

: Component displayed when button is disabled.

disabledLabel

: Component shown on top of skins when button is disabled.

ToggleButtonComponent

¶

The [ToggleButtonComponent] is an [AdvancedButtonComponent] that can switch between selected and n

In addition to the already existing skins, the [ToggleButtonComponent] contains the following skins:

defaultSelectedSkin

: The component to display when the button is selected.

downAndSelectedSkin

: The component that is displayed when the selectable button is selected and pressed.

hoverAndSelectedSkin

: Hover on selectable and selected button (desktop and web).

disabledAndSelectedSkin

: For when the button is selected and in the disabled state.

defaultSelectedLabel

: Component shown on top of the skins when button is selected.

IgnoreEvents mixin

¶

If you don?t want a component subtree to receive events, you can use the

IgnoreEvents

mixin. Once you have added this mixin you can turn off events to reach a component and its descendants

ignoreEvents

=

true

(default when the mixin is added), and then set it to

false

when you want to receive events again.

This can be done for optimization purposes, since all events currently go through the whole component tre

# Getting Started

¶

## About Flame

¶

Flame is a modular Flutter game engine that provides a complete set of out-of-the-way solutions for games

It provides you with a simple yet effective game loop implementation, and the necessary functionalities that

We also provide stand-alone packages that extend the Flame functionality which can be found in the

Bridge Packages

section.

You can pick and choose whichever parts you want, as they are all independent and modular.

The engine and its ecosystem are constantly being improved by the community, so please feel free to reac

Give us a star if you want to help give the engine exposure and grow the community. :)

## Installation

¶

Add the

flame

package as a dependency in your

pubspec.yaml

by running the following command:

```
flutter pub add flame
```

The latest version can be found on

pub.dev

.

then run

```
flutter
pub
```

get

and you are ready to start using it!

Getting started

¶

There is a set of tutorials that you can follow to get started in the

tutorials folder

.

Simple examples for all features can be found in the

examples folder

.

You can also check out the

awesome flame repository

, it contains quite a lot of good tutorials and articles written by the community to get you started with Flame.

Outside of the scope of the engine

¶

Games sometimes require complex feature sets depending on what the game is all about. Some of these f

Multiplayer (netcode)

¶

Flame doesn?t bundle any network feature, which may be needed to write online multiplayer games.

If you are building a multiplayer game, here are some recommendations of packages/services:

Nakama

: Nakama is an open-source server designed to power modern games and apps.

Firebase

: Provides dozens of services that can be used to write simpler multiplayer experiences.

Supabase

: A cheaper alternative to Firebase, based on Postgres.

Tap Events

¶

Note

This document describes the new events API. The old (legacy) approach, which is still supported, is descri

Gesture Input

.

Tap events

are one of the most basic methods of interaction with a Flame game. These events occur when the user to

Multiple tap events can occur at the same time, especially if the user has multiple fingers. Such cases will b

pointerId

property.

For those components that you want to respond to taps, add the

TapCallbacks

mixin.

This mixin adds four overridable methods to your component:

onTapDown

,

onTapUp

,

onTapCancel

, and

onLongTapDown

. By default, each of these methods does nothing, they need to be overridden in order to perform any funct

In addition, the component must implement the

containsLocalPoint()

method (already implemented in

PositionComponent

, so most of the time you don?t need to do anything here) ? this method allows Flame to know whether the

class

MyComponent

extends

PositionComponent

with

TapCallbacks

{

MyComponent

()

:

super

(

size:

Vector2

(

80

,

60

));

@override

void

onTapUp

(

TapUpEvent

event

)

{

// Do something in response to a tap event

}

}

Tap anatomy

¶

onTapDown

¶

Every tap begins with a ?tap down? event, which you receive via the

void

onTapDown(TapDownEvent)

handler. The event is delivered to the first component located at the point of touch that has the

TapCallbacks

mixin. Normally, the event then stops propagation. However, you can force the event to also be delivered t

event.continuePropagation

to true.

The

TapDownEvent

object that is passed to the event handler, contains the available information about the event. For example

event.localPosition

will contain the coordinate of the event in the current component?s local coordinate system, whereas

event.canvasPosition

is in the coordinate system of the entire game canvas.

Every component that received an

onTapDown

event will eventually receive either

onTapUp

or

onTapCancel

with the same

pointerId

.

onLongTapDown

¶

If the user holds their finger down for some time (as configured by the

.longTapDelay

property in

MultiTapDispatcher

), ?long tap? will be triggered. This event invokes the

void

onLongTapDown(TapDownEvent)

handler on those components that previously received the

onTapDown

event.

onTapUp

¶

This event indicates successful completion of the tap sequence. It is guaranteed to only be delivered to tho

onTapDown

event with the same pointer id.

The

TapUpEvent

object passed to the event handler contains the information about the event, which includes the coordinate

pointerId

.

Note that the device coordinates of the tap-up event will be the same (or very close) to the device coordina

In extreme case, when the component moves away from the point of touch, the

onTapUp

event will not be generated at all: it will be replaced with

onTapCancel

. Note, however, that in this case the

onTapCancel

will be generated at the moment the user lifts or moves their finger, not at the moment the component mov

onTapCancel

¶

This event occurs when the tap fails to materialize. Most often, this will happen if the user moves their finge

onTapCancel

occurs when another widget pops over the game widget, or when the device turns off, or similar situations.

The

TapCancelEvent

object contains only the

pointerId

of the previous

TapDownEvent

which is now being canceled. There is no position associated with a tap-cancel.

Demo

¶

Play with the demo below to see the tap events in action.

The blue-ish rectangle in the middle is the component that has the

TapCallbacks

mixin. Tapping this component would create circles at the points of touch. Specifically,

onTapDown

event starts making the circle. The thickness of the circle will be proportional to the duration of the tap: afte

onTapUp

the circle?s stroke width will no longer grow. There will be a thin white stripe at the moment the

onLongTapDown

fires. Lastly, the circle will implode and disappear if you cause the

onTapCancel

event by moving the finger.

tap_events.dart

```
1
import
'dart:math'
;
2
3
import
'package:flame/components.dart'
;
4
import
'package:flame/events.dart'
;
```

5

import

'package:flame/game.dart'

;

6

import

'package:flutter/rendering.dart'

;

7

8

class

TapEventsGame

extends

FlameGame

{

9

@override

10

Future

<

void

>

onLoad

()

async

{

11

add

(

TapTarget

());

12

}

13

}

14

15

/// This component is the tappable blue-ish rectangle in the center of the game.

16

/// It uses the [TapCallbacks] mixin to receive tap events.

17

class

TapTarget

extends

PositionComponent

with

TapCallbacks

{

18

TapTarget

()

:

```
super
(
anchor:
Anchor
.
center
);
19
20
final
_paint
=
Paint
()..
color
=
const
Color
(
0x448BA8FF
);
21
22
/// We will store all current circles into this map, keyed by the `pointerId`
23
/// of the event that created the circle.
```

24

final

Map

<

int

,

ExpandingCircle

>

_circles

=

{};

25

26

@override

27

void

onGameResize

(

Vector2

size

)

{

28

super

.

onGameResize

(

size

);

29

this

.

size

=

size

-

Vector2

(

100

,

75

);

30

if

(

this

.

size

.

x

<

100

||

this

.

size

.

y

<

100

)

{

31

this

.

size

=

size

*

0.9

;

32

}

33

position

=

size

/

2

;

34

}

35

36

@override

37

void

render

(

Canvas

canvas

)

{

38

canvas

.

drawRect

(

size

.

toRect

(),

_paint

);

```
39
}
40
41
@override
42
void
onTapDown
(
TapDownEvent
event
)
{
43
final
circle
=
ExpandingCircle
(
event
.
localPosition
);
44
_circles
[
```

```
event
.
pointerId
]
=
circle
;
45
add
(
circle
);
46
}
47
48
@override
49
void
onLongTapDown
(
TapDownEvent
event
)
{
50
```

_circles

[

event

.

pointerId

]

!

.

accent

();

51

}

52

53

@override

54

void

onTapUp

(

TapUpEvent

event

)

{

55

_circles

.

remove

(

event

.

pointerId

)

!

.

release

();

56

}

57

58

@override

59

void

onTapCancel

(

TapCancelEvent

event

)

{

60

_circles

.

remove

(

event

.

pointerId

)

!

.

cancel

();

61

}

62

}

63

64

class

ExpandingCircle

extends

Component

{

65

ExpandingCircle

(

this

.

_center

)

66

:

_baseColor

=

67

HSLColor

.

fromAHSL

(

1

,

random

.

nextDouble

()

*

360

,

1

,

0.8

).

toColor

();

68

69

final

Color

_baseColor

;

70

final

Vector2

_center

;

71

double

_outerRadius

=

0

;

72

double

_innerRadius

=

0

;

73

bool

_released

=

false

;

74

bool

_cancelled

=

false

;

75

late

final

_paint

=

Paint

()

76

..

style

=

PaintingStyle

.

stroke

77

..

color

=

_baseColor

;

78

79

/// "Accent" is thin white circle generated by `onLongTapDown`. We use

80

/// negative radius to indicate that the circle should not be drawn yet.

81

double

_accentRadius

=

-

1e10

;

82

late

final

_accentPaint

=

Paint

()

83

..

style

=

PaintingStyle

.

stroke

84

..

strokeWidth

=

0

85

..

color

=

const

Color

(

0xFFFFFFFF

);

86

87

/// At this radius the circle will disappear.

88

static

const

maxRadius

=

175

;

89

static

final

random

=

Random

();

90

91

double

get

radius

=>

(

_innerRadius

+

_outerRadius

)

/

2

;

92

93

void

release

()

=>

_released

=

true

;

94

void

cancel

()

=>

_cancelled

=

true

;

95

void

accent

()

=>

_accentRadius

=

0

;

96

97

```
@override
98
void
render
(
Canvas
canvas
)
{
99
canvas
.
drawCircle
(
_center
.
toOffset
(),
radius
,
_paint
);
100
if
(
_accentRadius
```

>=

0

)

{

101

canvas

.

drawCircle

(

_center

.

toOffset

(),

_accentRadius

,

_accentPaint

);

102

}

103

}

104

105

@override

106

void

```
update
(
double
dt
)
{
107
if
(
_cancelled
)
{
108
_innerRadius
+=
dt
*
100
;
// implosion
109
}
else
{
110
_outerRadius
```

+=

dt

*

20

;

111

_innerRadius

+=

dt

*

(

_released

?

20

:

6

);

112

_accentRadius

+=

dt

*

20

;

113

}

114

if

(

radius

>=

maxRadius

||

_innerRadius

>

_outerRadius

)

{

115

removeFromParent

();

116

}

else

{

117

final

opacity

=

1

-

radius

/

maxRadius

;

118

_paint

.

color

=

_baseColor

.

withOpacity

(

opacity

);

119

_paint

.

strokeWidth

=

_outerRadius

-

_innerRadius

;

120

}

121

}

122

}

Code

Mixins

¶

This section describes in more details several mixins needed for tap event handling.

TapCallbacks

¶

The

TapCallbacks

mixin can be added to any

Component

in order for that component to start receiving tap events.

This mixin adds methods

onTapDown

,

onLongTapDown

,

onTapUp

, and

onTapCancel

to the component, which by default don?t do anything, but can be overridden to implement any real functio

onTapUp

if you wish to respond to ?real? taps only.

Another crucial detail is that a component will only receive tap events that occur

within

that component, as judged by the

containsLocalPoint()

function. The commonly-used

PositionComponent

class provides such an implementation based on its

size

property. Thus, if your component derives from a

PositionComponent

, then make sure that you set its size correctly. If, however, your component derives from the bare

Component

, then the

containsLocalPoint()

method must be implemented manually.

If your component is a part of a larger hierarchy, then it will only receive tap events if its parent has implem

containsLocalPoint

correctly.

class

MyComponent

extends

Component

with

TapCallbacks

{

final

_rect

=

const

Rect

.

fromLTWH

(

0

,

0

,

100

,

100

);

final

_paint

=

Paint

();

bool

_isPressed

=

false

;

@override

bool

containsLocalPoint

(

Vector2

point

)

=>

_rect

.

contains

(

point

.

toOffset

());

@override

void

onTapDown

(

TapDownEvent

event

)

=>

_isPressed

=

true

;

```
@override
void
onTapUp
(
TapUpEvent
event
)
=>
_isPressed
=
false
;
@override
void
onTapCancel
(
TapCancelEvent
event
)
=>
_isPressed
=
false
;
@override
void
```

render

(

Canvas

canvas

)

{

_paint

.

color

=

_isPressed

?

Colors

.

red

:

Colors

.

white

;

canvas

.

drawRect

(

_rect

,

_paint

);

}

}

DoubleTapCallbacks

¶

Flame also offers a mixin named

DoubleTapCallbacks

to receive a double-tap event from the component. To start receiving double tap events in a component, ad

DoubleTapCallbacks

mixin to your

PositionComponent

.

class

MyComponent

extends

PositionComponent

with

DoubleTapCallbacks

{

@override

void

onDoubleTapUp

(

DoubleTapEvent

event

)

{

/// Do something

}

@override

void

onDoubleTapCancel

(

DoubleTapCancelEvent

event

)

{

/// Do something

}

@override

void

onDoubleTapDown

(

DoubleTapDownEvent

event

)

{

/// Do something

}

Migration

¶

If you have an existing game that uses

Tappable

/

Draggable

mixins, then this section will describe how to transition to the new API described in this document. Here?s w

Take all of your components that uses these mixins, and replace them with

TapCallbacks

/

DragCallbacks

. The methods

onTapDown

,

onTapUp

,

onTapCancel

and

onLongTapDown

will need to be adjusted for the new API:

The argument pair such as

(int

pointerId,

TapDownDetails

details)

was replaced with a single event object

TapDownEvent

event

.

There is no return value anymore, but if you need to make a component to pass-through the taps to the co

event.continuePropagation

to true. This is only needed for

onTapDown

events ? all other events will pass-through automatically.

If your component needs to know the coordinates of the point of touch, use

event.localPosition

instead of computing it manually. Properties

event.canvasPosition

and

event.devicePosition

are also available.

If the component is attached to a custom ancestor then make sure that ancestor also have the correct size

containsLocalPoint()

.

## Forge2D

¶

Blue Fire maintains a ported version of the Box2D physics engine and our version is called Forge2D.

If you want to use Forge2D specifically for Flame you should use our bridge library

flame_forge2d

and if you just want to use it in a Dart project you can use the

forge2d

library directly.

To use it in your game you just need to add

flame_forge2d

to your

pubspec.yaml

, as can be seen in the [Forge2D

example

and the pub. dev

installation instructions

](

https://pub.dev/packages/flame_forge2d

).

## Forge2DGame

¶

If you are going to use Forge2D in your project it can be a good idea to use the Forge2D-specific

FlameGame

class,

Forge2DGame

.

It is called

Forge2DGame

and supports both the special Forge2D components called

BodyComponents

as well as normal Flame components.

Forge2DGame

has a built-in

CameraComponent

and has a zoom level set to 10 by default, so your components will be a lot bigger than in a normal Flame g

Forge2D

world, which you would hit very quickly if you are using it with

zoom

=

1.0

. You can easily change the zoom level either by calling

super(zoom:

yourZoom)

in your constructor or doing

game.cameraComponent.viewfinder.zoom

=

yourZoom;

at a later stage.

If you are previously familiar with Box2D it can be good to know that the whole concept of the Box2d world

world

in the

Forge2DGame

component and every

Body

that you want to use as a component should be wrapped in a

BodyComponent

, and added to the

world

in your

Forge2DGame

.

You can have have non-physics-related components in your

Forge2DGame

world?s component list along with your physical entities. When the update is called, it will use the Forge2D

BodyComponent

and other components in the game will be updated according to the normal

FlameGame

way.

In

Forge2DGame

the gravity is flipped compared to

Forge2D

to keep the same coordinate system as in Flame, so a positive y-axis in the gravity like

Vector2(0,

10)

would be pulling bodies downwards, meanwhile, a negative y-axis would pull them upwards. The gravity ca

Forge2DGame

.

A simple

Forge2DGame

implementation example can be seen in the

examples folder

.

BodyComponent

¶

The

BodyComponent

is a wrapper for the

Forge2D

body, which is the body that the physics engine is interacting with. To create a

BodyComponent

you can either:

override

createBody()

and create and return your created body;

use the default

createBody()

implementation by passing a

BodyDef

instance (and optionally a list of

FixtureDef

instances) to the BodyComponent?s constructor;

use the default

createBody()

implementation and assign a

BodyDef

instance to

this.bodyDef

, and optionally a list of

FixtureDef

instances to

this.fixtureDefs

.

The

BodyComponent

is by default having

renderBody

=

true

, since otherwise, it wouldn?t show anything after you have created a

Body

and added the

BodyComponent

to the game. If you want to turn it off you can just set (or override)

renderBody

to false.

Just like any other Flame component you can add children to the

BodyComponent

, which can be very useful if you want to add for example animations or other components on top of your b

The body that you create should be defined according to Flame?s coordinate system, not according to the

:exclamation: In Forge2D you shouldn?t add any bodies as children to other components, since Forge2D d

Forge2DGame.world

. So instead of

add(Weapon()))

,

world.add(Weapon())

should be used (as below), and the

Player

should also of course initially be added to the world.

class

Weapon

extends

BodyComponent

{

@override

void

onLoad

()

{

...

}

}

class

Player

extends

BodyComponent

{

@override

void

onLoad

()

{

world

.

add

(

Weapon

());

}

}

Later you might want to add bullets coming from your weapon, these are added to the world in the same se

isBullet

=

true

to avoid some tunneling problems.

Contact callbacks

¶

Forge2DGame

provides a simple out-of-the-box solution to propagate contact events.

Contact events occur whenever two

Fixture

s meet each other. These events allow listening when these

Fixture

s begin to come in contact (

beginContact

) and cease being in contact (

endContact

).

There are multiple ways to listen to these events. One common way is to use the

ContactCallbacks

class as a mixin in the

BodyComponent

where you are interested in these events.

class

Ball

extends

BodyComponent

with

ContactCallbacks

{

...

void

beginContact

(

Object

other

,

Contact

contact

)

{

if

(

other

is

Wall

)

{

// Do something here.

}

}

...

}

For the above to work, the

Ball

?s

body.userData

or contacting

fixture.userData

must be set to a

ContactCallback

. And if

Wall

is a

BodyComponent

it?s

body.userData

or contacting

fixture.userData

must be set to

Wall

.

If

userData

is

null

the contact events are ignored, it is

null

by default.

A convenient way of setting

userData

is to assign it when creating the body. For example:

class

Ball

extends

BodyComponent

with

ContactCallbacks

{

...

@override

Body

createBody

()

{

...

final

bodyDef

=

BodyDef

(

userData:

this

,

);

...

}

}

Every time

Ball

and

Wall

begin to come in contact

beginContact

will be called, and once the fixtures cease being in contact,

endContact

will be called.

An implementation example can be seen in the

Flame Forge2D example

.

1. Preparation

¶

Before you begin any kind of game project, you need to give it a

name

. For this tutorial the name will be simply

klondike

.

Having this name in mind, please head over to the

tutorial

and complete the necessary set up steps. When you come back, you should already have the

main.dart

file with the following content:

import

'package:flame/game.dart'

;

import

'package:flutter/widgets.dart'

;

void

main

()

{

final

game

=

FlameGame

();

runApp

(

GameWidget

(

game:

game

));

}

Planning

¶

The start of any project usually feels overwhelming. Where even to begin? I always find it useful to create a

Here you can see both the general layout of the game, as well as names of various objects. These names

standard terminology

for solitaire games. Which is really lucky, because normally figuring out good names for various classes is

Looking at this sketch, we can already imagine the high-level structure of the game. Obviously, there will b

Card

class, but also the

Stock

class, the

Waste

class, a

Tableau

containing seven

Pile

s, and 4

Foundation

s. There may also be a

Deck

. All of these components will be tied together via the

KlondikeGame

derived from the

FlameGame

.

Assets

¶

Another important aspect in any game development is the game?s assets. These includes images, sprites,

In order to prepare the graphic assets, I first took a physical playing card and measured it to be 63mm × 88

10:14

. Thus, I decided that my in-game cards should be rendered at 1000×1400 pixels, and I should draw all my

Note that the exact pixel dimensions are somewhat irrelevant here, since the images will in the end be scal

And now, without further ado, here?s my graphic asset for the Klondike game (I?m not an artist, so don?t ju

Right-click the image, choose ?Save as??, and store it in the

assets/images

folder of the project. At this point our project?s structure looks like this (there are other files too, of course, l

klondike/ ??assets/ ? ??images/ ? ??klondike-sprites.png ??lib/ ? ??main.dart ??pubspec.yaml

By the way, this kind of file is called the

sprite sheet

: it?s just a collection of multiple independent images in a single file. We are using a sprite sheet here for th

drawAtlas

command.

Here are the contents of my sprite sheet:

Numerals 2, 3, 4, ?, K, A. In theory, we could have rendered these in the game as text strings, but then we

Suit marks: ?, ?, ?, ?. Again, we could have used Unicode characters for these, but images are much easie

In case you?re wondering why these are yellow/blue instead of red/black ? turns out, black symbols don?t

Flame logo, for use on the backs of the cards.

Pictures of a Jack, a Queen, and a King. Normally there would be four times more of these, with a different

Also, you need to tell Flutter about this image (just having it inside the

assets

folder is not enough). In order to do this, let?s add the following lines into the

pubspec.yaml

file:

flutter

:

assets

:

-

assets/images/

Alright, enough with preparing ? onward to coding!

flame_fire_atlas

¶

Overview

FireAtlas

Creating Atlas

Texture atlas

Usage

Full Example

flame_audio

¶

General audio

Caching

Background music

Caching music files

Methods

Play

Stop

Pause and Resume

AudioPool

YarnSpinner language

¶

YarnSpinner

is the language in which

.yarn

files are written. You can check out the

official documentation

for the YarnSpinner language, however, here we will be describing the

Jenny

implementation, which may not contain all the original features, but may also contain some that were not in

Yarn files

¶

Any Yarn project will contain one or more

.yarn

files. These are plain text files in UTF-8 encoding. As such, they can be edited in any text editor or IDE.

Having multiple

.yarn

files helps you better organize your project, but Jenny doesn?t impose any requirements on the number of

Each

.yarn

file may contain

comments

,

tags

,

commands

, and

nodes

. For example:

// This is a comment

// The line below, however, is a tag:

# Chapter 1d

<<

declare

$visited_graveyard

=

false

>>

<<

declare

$money

=

25

>>

// is this too much?

title

:

Start

---

// Node content

===

Comments

¶

A comment starts with

//

and continues until the end of the line. All the text inside a comment will be completely ignored by Jenny as

There are no multi-line comments in YarnSpinner.

Tags

¶

File-level tags start with a

#

and continue until the end of the line. A tag can be used to include some per-file custom project metadata.

Commands

¶

The commands are explained in more details

later

, but at this point it is worth pointing out that only a limited number of commands are allowed at the root lev

<<declare>>

<<character>>

The commands outside of nodes are compile-time instructions, that is they are executed during the compila

Nodes

¶

Nodes represent the main bulk of content in a yarn file, and are explained in a dedicated

section

. There could be multiple nodes in a single file, placed one after another. No special separator is needed be

Layers

¶

At its simplest, layers can be retrieved from a Tilemap by invoking:

getLayer

<

ObjectGroup

>

(

"myObjectGroupLayer"

);

getLayer

<

ImageLayer

>

(

"myImageLayer"

);

getLayer

<

TileLayer

>

(

"myTileLayer"

);

getLayer

<

Group

>

(

"myGroupLayer"

);

These methods will either return the requested layer type or null if it does not exist.

Layer properties

¶

The following Tiled properties are supported:

Visible

Opacity

Tint color

Horizontal offset

Vertical offset

Parallax factor

Custom properties

Tiles properties

¶

Tiles can have custom properties accessible at

tile.properties

.

Tiles can have a custom

type

(or

class

starting in Tiled v1.9) accessible at

tile.type

.

Other features

¶

Other advanced features are not yet supported, but you can easily read the objects and other features of th

Full Example

¶

You can check a working example

here

.

flame_bloc

¶

flame_riverpod

¶

Overview

Riverpod

Usage

Component

ComponentRef

RiverpodComponentMixin

RiverpodGameMixin

Widget

RiverpodAwareGameWidget

RiverpodAwareGameWidgetState

Flame

¶

File Structure

Game Widget

Game Loop

Components

Router

Platforms

Collision Detection

Effects

Camera Component

Inputs

Rendering

Layout

Overlays

Other

Options

¶

Options

are special lines that display a menu of choices for the player, and the player must select one of them in or

->

at the start of the line:

title

:

Adventure

---

You arrive at the edge of the forest. The road dives in, but there is another

\

one going around the edge.

->

Go straight ahead, on the beaten path (x)

->

Take the road along the forest's edge

->

Turn back

===

An option is typically followed by an indented list of statements (which may, again, be lines, options, or com

Other than the arrow indicator, an option follows the same syntax as the

line

. Thus, it can have a character name, the main text, interpolated expressions, markup, and hashtags. One

conditional

. A conditional is a short-form

<<if>>

command after the text of an option (but before the hashtags):

title

:

Bridge

---

Guard

:

50 coins and you can cross the bridge.

->

Alright, take the money

<<

if

$gold

>=

50

>>

<<

take

gold 50

>>

<<

grant

bridge_pass

>>

->

I have so much money, here, take a 100

<<

if

$gold

>=

10000

>>

<<

take

gold 100

>>

<<

grant

bridge_pass

>>

Guard

:

Wow, so generous!

Guard

:

But I wouldn't recommend going around telling everyone that you

\

have "so much money"

->

That's too expensive!

Guard

:

Is it? My condolences

->

How about I

[s]

kick your butt

[/s]

instead?

<<

if

$power

<

1000

>>

<<

fight

>>

<<

else

>>

You make a very reasonable point, sir, my apologies.

<<

grant

bridge_pass

>>

<<

endif

>>

===

When the conditional evaluates to

true

, the option is delivered to the game normally. If the conditional turns out to be false, the option is still delive

unavailable

. It is up to the game whether to display such option as greyed out, or crossed, or not show it at all; howeve

As you have noticed, options always come in groups: after all, the player must select among several possib

choice set

.

Component

¶

ComponentRef

¶

ComponentRef

exposes Riverpod functionality to individual

Component

s, and is comparable to

flutter_riverpod

?s

WidgetRef

.

RiverpodComponentMixin

¶

RiverpodComponentMixin

manages the lifecycle of listeners on behalf of individual

Component

s.

Component

s using this mixin must use

addToGameWidgetBuild

in their

onMount

method to add listeners (e.g.

ref.watch

or

ref.listen

)

prior to

calling

super.onMount

, which manages the staged listeners and disposes of them on the user?s behalf inside

onRemove

.

class

RiverpodAwareTextComponent

extends

PositionComponent

with

RiverpodComponentMixin

{

late

TextComponent

textComponent

;

int

currentValue

=

0

;

@override

void

onMount

()

{

addToGameWidgetBuild

((()

{

ref

.

listen

(

countingStreamProvider

,

(

p0

,

p1

)

{

if

(

p1

.

hasValue

)

{

currentValue

=

p1

.

value

!

;

textComponent

.

text

=

'

$

currentValue

'

;

}

});

});

super

.

onMount

();

add

(

textComponent

=

TextComponent

(

position:

position

+

Vector2

(

0

,

27

)));

}

}

RiverpodGameMixin

¶

RiverpodGameMixin

provides listeners from all components to the build method of the

RiverpodAwareGameWidget

.

Bridge Packages

¶

flame_audio

Play multiple audio files simultaneously (bridge package for

AudioPlayers

).

flame_bloc

A predictable state management library (bridge package for

Bloc

).

flame_fire_atlas

Create texture atlases for games (bridge package for

FireAtlas

).

flame_forge2d

A Box2D physics engine (bridge package for

Forge2D

).

flame_isolate

Use isolates to offload heavy computations to another thread.

flame_lottie

Use Lottie animations in Flame (bridge package for

Lottie

).

flame_network_assets

Fetch assets over the network.

flame_oxygen

Replace FCS with the Oxygen Entity Component System.

flame_rive

Create interactive animations (bridge package for

Rive

).

flame_riverpod

A reactive caching and data-binding framework (bridge package for

Riverpod

).

flame_spine

Use Spine skeletal animations (bridge package for

Spine

).

flame_splash_screen

Add the ?Powered by Flame? splash screen.

flame_svg

Draw SVG files in Flutter (bridge package for

flutter_svg

).

flame_tiled

2D tilemap level editor (bridge package for

Tiled

).

Type conversion functions

¶

These functions convert values of one type into another type, if possible. All of these functions take a single

bool(x)

¶

Converts its argument into a boolean value.

If

x

is already a boolean, then it returns the argument as-is.

If

x

is numeric, then the result is

false

when

x

is

0

, and

true

for all other values of

x

.

If

x

is string, then the function will check whether that string can be found within

YarnProject.trueValues

or

YarnProject.falseValues

sets. If yes, then it will return the

true

/

false

value respectively. Otherwise, an error will be thrown.

number(x)

¶

Converts its argument

x

into a numeric value.

If

x

is boolean, then it returns

1

for

true

and

0

for

false

.

If

x

is numeric, then it is returned unmodified.

If

x

is string, then the function attempts to parse that string as a number. A runtime exception will be raised if

x

does not have a valid format for a number. The following formats are recognized:

integer:

"-3"

,

"214"

decimal:

"0.745"

,

"3.14159"

,

".1"

,

"-3."

scientific:

"2e5"

,

"3.11e-05"

hexadecimal:

"0xDEAD"

,

"0x7F"

string(x)

¶

Converts its argument

x

into a string value.

If

x

is boolean, returns strings

"true"

or

"false"

.

If

x

is numeric, converts it into a string representation using the standard Dart?s

.toString()

method, which attempts to produce the shortest string that can represent the number

x

. In particular,

if

x

is integer-valued, returns its decimal representation without a decimal point;

if

x

is a double in the range

1e-6

to

1e21

, returns its decimal representation with a decimal point;

for all other doubles, returns

x

written in the scientific (exponential) format.

If

x

is a string, then it is returned as-is.

Layout

¶

Align Component

1. Preparation

¶

Before you begin any kind of game project, you need an idea of what you want to make and I like to then g

name

. For this tutorial and game, Ember will be on a quest to gather as many (GitHub) stars as possible and I w

Ember

Quest

.

Now it is time to get started, but first you need to go to the

bare flame game tutorial

and complete the necessary setup steps. When you come back, you should already have the

main.dart

file with the following content:

```
import

'package:flame/game.dart'

;

import

'package:flutter/widgets.dart'

;

void

main

()

{

final

game

=
```

FlameGame

();

runApp

(

GameWidget

(

game:

game

));

}

Planning

¶

Like in the

klondike

tutorial, starting a new game can feel overwhelming. I like to first decide what platform I am trying to target.

Starting with a simple sketch (it doesn?t have to be perfect as mine is very rough) is the best way to get an

Player Class

Enemy Class

Star Class

Platform Class

Ground Class

HUD Class (health and stars collected)

All of these will be brought together in

EmberQuestGame

derived from

FlameGame

.

Assets

¶

Every game needs assets. Assets are images, sprites, animations, sounds, etc. Now, I am not an artist, bu

Right-click the images below, choose ?Save as??, and store them in the

assets/images

folder of the project. At this point our project?s structure looks like this:

emberquest/ ??assets/ ? ??images/ ? ??block.png ? ??ember.png ? ??ground.png ? ??hea

Note

You may ask, why are the images different sizes?

As I was using the online tool to make the assets, I had trouble getting the detail I desired for the game in a

Also, you need to tell Flutter about these images (just having them inside the

assets

folder is not enough). To do this, let?s add the following lines into the

pubspec.yaml

file:

flutter

:

assets

:

-

assets/images/

Alright, enough with preparing ? onward to coding!

# Text Rendering

¶

Flame has some dedicated classes to help you render text.

## Text Components

¶

The simplest way to render text with Flame is to leverage one of the provided text-rendering components:

TextComponent

for rendering a single line of text

TextBoxComponent

for bounding multi-line text within a sized box, including the possibility of a typing effect

ScrollTextBoxComponent

enhances the functionality of

TextBoxComponent

by adding scrolling capability when the text exceeds the boundaries of the enclosing box.

Use the

onFinished

callback to get notified when the text is completely printed.

All components are showcased in

this example

.

### TextComponent

¶

TextComponent

is a simple component that renders a single line of text.

Simple usage:

class

```
MyGame
extends
FlameGame
{
@override
void
onLoad
()
{
add
(
TextComponent
(
text:
'Hello, Flame'
,
position:
Vector2
.
all
(
16.0
),
),
);
}
```

}

In order to configure aspects of the rendering like font family, size, color, etc, you need to provide (or amen

TextRenderer

with such information; while you can read more details about this interface below, the simplest implementa

TextPaint

, which takes a Flutter

TextStyle

:

final

regular

=

TextPaint

(

style:

TextStyle

(

fontSize:

48.0

,

color:

BasicPalette

.

white

.

color

,

),

);

class

MyGame

extends

FlameGame

{

@override

void

onLoad

()

{

add

(

TextComponent

(

text:

'Hello, Flame'

,

textRenderer:

regular

,

anchor:

Anchor

.

topCenter

,

position:

Vector2

(

size

.

width

/

2

,

32.0

),

),

);

}

}

You can find all the options under

TextComponent?s API

.

TextBoxComponent

¶

TextBoxComponent

is very similar to

TextComponent

, but as its name suggest it is used to render text inside a bounding box, creating line breaks according to t

You can decide if the box should grow as the text is written or if it should be static by the

growingBox

variable in the

TextBoxConfig

. A static box could either have a fixed size (setting the

size

property of the

TextBoxComponent

), or to automatically shrink to fit the text content.

In addition, the

align

property allows you to control the the horizontal and vertical alignment of the text content. For example, se

align

to

Anchor.center

will center the text within its bounding box both vertically and horizontally.

If you want to change the margins of the box use the

margins

variable in the

TextBoxConfig

.

Finally, if you want to simulate a ?typing? effect, by showing each character of the string one by one as if b

boxConfig.timePerChar

parameter.

Example usage:

class

MyTextBox

extends

TextBoxComponent

{

MyTextBox

(

String

text

)

:

super

(

text:

text

,

textRenderer:

tiny

,

boxConfig:

TextBoxConfig

(

timePerChar:

0.05

),

);

final

bgPaint

=

Paint

()..

color

=

Color

(

0xFFFF00FF

);

final

borderPaint

=

Paint

()..

color

=

Color

(

0xFF000000

)..

style

=

PaintingStyle

.

stroke

;

```
@override
void
render
(
Canvas
canvas
)
{
Rect
rect
=
Rect
.
fromLTWH
(
0
,
0
,
width
,
height
);
canvas
.
drawRect
```

(
rect
,
bgPaint
);
canvas
.
drawRect
(
rect
.
deflate
(
boxConfig
.
margin
),
borderPaint
);
super
.
render
(
canvas
);
}

}

You can find all the options under

TextBoxComponent?s API

.

ScrollTextBoxComponent

¶

The

ScrollTextBoxComponent

is an advanced version of the

TextBoxComponent

, designed for displaying scrollable text within a defined area. This component is particularly useful for crea

Note that the

align

property of

TextBoxComponent

is not available.

Example usage:

class

MyScrollableText

extends

ScrollTextBoxComponent

{

MyScrollableText

(

Vector2

frameSize

,

String

text

)

:

super

(

size:

frameSize

,

text:

text

,

textRenderer:

regular

,

boxConfig:

TextBoxConfig

(

timePerChar:

0.05

),

);

}

TextElementComponent

¶

If you want to render an arbitrary TextElement, ranging from a single InlineTextElement to a formatted Doc

TextElementComponent

.

A simple example is to create a DocumentRoot to render a sequence of block elements (think of an HTML

final

document

=

DocumentRoot

([

HeaderNode

.

simple

(

'1984'

,

level:

1

),

ParagraphNode

.

simple

(

'Anything could be true. The so-called laws of nature were nonsense.'

,

),

// ...

```
]);
final
element
=
TextElementComponent
.
fromDocument
(
document:
document
,
position:
Vector2
(
100
,
50
),
size:
Vector2
(
400
,
200
),
);
```

Note that the size can be specified in two ways; either via:

the size property common to all

PositionComponents

; or

the width/height included within the

DocumentStyle

applied.

An example applying a style to the document (which can include the size but other parameters as well):

final

style

=

DocumentStyle

(

width:

400

,

height:

200

,

padding:

const

EdgeInsets

.

symmetric

(

vertical:

10

,

horizontal:

14

),

background:

BackgroundStyle

(

color:

const

Color

(

0xFF4E322E

),

borderColor:

const

Color

(

0xFF000000

),

borderWidth:

2.0

,

),

);

final

```
document
=
DocumentRoot
([
...
]);
final
element
=
TextElementComponent
.
fromDocument
(
document:
document
,
style:
style
,
position:
Vector2
(
100
,
50
),
```

);

For a more elaborate example of rich-text, formatted text blocks rendering, check

this example

.

For more details about the underlying mechanics of the text rendering pipeline, see ?Text Elements, Text N

Flame Markdown

¶

In order to more easily create rich-text-based DocumentRoots, from simple strings with bold/italics to comp

flame_markdown

bridge package that connects the

markdown

library with Flame?s text rendering infrastructure.

Just use the

FlameMarkdown

helper class and the

toDocument

method to convert a markdown string into a DocumentRoot (which can then be used to create a

TextElementComponent

):

import

'package:flame/text.dart'

;

import

'package:flame_markdown/flame_markdown.dart'

;

// ...

```
final
component
=
await
TextElementComponent
.
fromDocument
(
document:
FlameMarkdown
.
toDocument
(
'# Header
\n
'
'
\n
'
'This is a **bold** text, and this is *italic*.
\n
'
'
\n
'
'This is a second paragraph.
```

\n

'

,

),

style:

...,

position:

...,

size:

...,

);

Infrastructure

¶

If you are not using the Flame Component System, want to understand the infrastructure behind text rende

TextRenderer

: renderers know ?how? to render text; in essence they contain the style information to render any string

TextElement

: an element is formatted, ?laid-out? piece of text, include the string (?what?) and the style (?how?)

The following diagram showcases the class and inheritance structure of the text rendering pipeline:

%%{init: { 'theme': 'dark' } }%% classDiagram %% renderers note for TextRenderer "This just the style

TextRenderer

¶

TextRenderer

is the abstract class used by Flame to render text. Implementations of

TextRenderer

must include the information about the ?how? the text is rendered. Font style, size, color, etc. It should be a

format

method, to generate a

TextElement

.

Flame provides two concrete implementations:

TextPaint

: most used, uses Flutter

TextPainter

to render regular text

SpriteFontRenderer

: uses a

SpriteFont

(a sprite sheet-based font) to render bitmap text

DebugTextRenderer

: only intended to be used for Golden Tests

But you can also provide your own if you want to extend to other customized forms of text rendering.

The main job of a

TextRenderer

is to format a string of text into a

TextElement

, that then can be rendered onto the screen:

final

textElement

=

textRenderer

.

format

(

"Flame is awesome"

)

textElement

.

render

(...)

However the renderer provides a helper method to directly create the element and render it:

textRenderer

.

render

(

canvas

,

'Flame is awesome'

,

Vector2

(

10

,

10

),

anchor:

Anchor

.

topCenter

,

);

TextPaint

¶

TextPaint

is the built-in implementation of text rendering in Flame. It is based on top of Flutter?s

TextPainter

class (hence the name), and it can be configured by the style class

TextStyle

, which contains all typographical information required to render text; i.e., font size and color, font family, et

Outside of the style you can also optionally provide one extra parameter which is the

textDirection

(but that is typically already set to

ltr

or left-to-right).

Example usage:

const

TextPaint

textPaint

=

TextPaint

(

style:

TextStyle

(

fontSize:

48.0

,

fontFamily:

'Awesome Font'

,

),

);

Note: there are several packages that contain the class

TextStyle

. We export the right one (from Flutter) via the

text

module:

import

'package:flame/text.dart'

;

But if you want to import it explicitly, make sure that you import it from

package:flutter/painting.dart

(or from material or widgets). If you also need to import

dart:ui

, you might need to hide its version of

TextStyle

, since that module contains a different class with the same name:

import

'package:flutter/painting.dart'

;

import

'dart:ui'

hide

TextStyle

;

Some common properties of

TextStyle

are the following (here is the

full list

):

fontFamily

: a commonly available font, like Arial (default), or a custom font added in your pubspec (see

here

how to do it).

fontSize

: font size, in pts (default

24.0

).

height

: height of text line, as a multiple of font size (default

null

).

color

: the color, as a

ui.Color

(default white).

For more information regarding colors and how to create then, see the

Colors and the Palette

guide.

SpriteFontRenderer

¶

The other renderer option provided out of the box is

SpriteFontRenderer

, which allows you to provide a

SpriteFont

based off of a sprite sheet. TODO

DebugTextRenderer

¶

This renderer is intended to be used for Golden Tests. Rendering normal font-based text in Golden Tests is

Inline Text Elements

¶

A

TextElement

is a ?pre-compiled?, formatted and laid-out piece of text with a specific styling applied, ready to be rendere

A

InlineTextElement

implements the

TextElement

interface and must implement their two methods, one that teaches how to translate it around and another c

void

translate

(

double

dx

,

double

dy

);

void

draw

(

Canvas

canvas

);

These methods are intended to be overwritten by the implementations of

InlineTextElement

, and probably will not be called directly by users; because a convenient

render

method is provided:

void

render

(

Canvas

canvas

,

Vector2

position

,

{

Anchor

anchor

=

Anchor

.

topLeft

,

})

That allows the element to be rendered at a specific position, using a given anchor.

The interface also mandates (and provides) a getter for the

LineMetrics

object associated with that

InlineTextElement

, which allows you (and the

render

implementation) to access sizing information related to the element (width, height, ascend, etc).

LineMetrics

get

metrics

;

Text Elements, Text Nodes, and Text Styles

¶

While normal renderers always work with a

InlineTextElement

directly, there is a bigger underlying infrastructure that can be used to render more rich or formatter text.

Text Elements are a superset of Inline Text Elements that represent an arbitrary rendering block within a ri

This property distinguishes them from Text Nodes, which are structured pieces of text, and from Text Style

FlameTextStyle

in code to make it easier to work alongside Flutter?s

TextStyle

), which are descriptors for how arbitrary pieces of text ought to be rendered.

So, in the most general case, a user would use a

TextNode

to describe a desired piece of rich text; define a

FlameTextStyle

to apply to it; and use that to generate a

TextElement

. Depending on the type of rendering, the

TextElement

generated will be an

InlineTextElement

, which brings us back to the normal flow of the rendering pipeline. The unique property of the Inline-Text-ty

draw

method which is unaware of sizing and positioning.

However, the other types of Text Elements, Text Nodes, and Text Styles must be used if the intent is to cre

TextElementComponent

(see above).

An example of such usages can be seen in

this example

.

Text Nodes and the Document Root

¶

A

DocumentRoot

is not a

TextNode

(inheritance-wise) in itself but represents a grouping of

BlockNodes

that layout a ?page? or ?document? of rich text laid out in multiple blocks or paragraphs. It represents the c

The first step to define your rich-text document is to create a Node, which will likely be a

DocumentRoot

.

It will first contain the top-most list of Block Nodes that can define headers, paragraphs or columns.

Then each of those blocks can contain other blocks or the Inline Text Nodes, either Plain Text Nodes or so

Note that the hierarchy defined by the node structure is also used for styling purposes as per defined in the

FlameTextStyle

class.

The actual nodes all inherit from

TextNode

and are broken down by the following diagram:

%%{init: { 'theme': 'dark' } }%% graph TD %% Config %% classDef default fill:#282828,stroke:#F6BE0

(Flame) Text Styles

¶

Text Styles can be applied to nodes to generate elements. They all inherit from

FlameTextStyle

abstract class (which is named as is to avoid confusion with Flutter?s

TextStyle

).

They follow a tree-like structure, always having

DocumentStyle

as the root; this structure is leveraged to apply cascading style to the analogous Node structure. In fact, the

The full inheritance chain can be seen on the following diagram:

%%{init: { 'theme': 'dark' } }%% classDiagram %% Nodes %% class FlameTextStyle { copyWith()

Text Elements

¶

Finally, we have the elements, that represent a combination of a node (?what?) with a style (?how?), and t

Inline Text Elements specifically can alternatively be thought of as a combination of a

TextRenderer

(simplified ?how?) and a string (single line of ?what?).

That is because an

InlineTextStyle

can be converted to a specific

TextRenderer

via the

asTextRenderer

method, which is then used to lay out each line of text into a unique

InlineTextElement

.

When using the renderer directly, the entire layout process is skipped, and a single

TextPainterTextElement

or

SpriteFontTextElement

is returned.

As you can see, both definitions of an Element are, essentially, equivalent, all things considered. But it still

When in doubt, the following guidelines can help you picking the best path for you:

for the simplest way to render text, use

TextPaint

(basic renderer implementation)

you can use the FCS provided component

TextComponent

for that.

for rendering Sprite Fonts, you must use

SpriteFontRenderer

(a renderer implementation that accepts a

SpriteFont

);

for rendering multiple lines of text, with automatic line breaks, you have two options:

use the FCS

TextBoxComponent

, which uses any text renderer to draw each line of text as an Element, and does its own layout and line bre

use the Text Node & Style system to create your pre-laid-out Elements. Note: there is no current FCS com

finally, in order to have formatted (or rich) text, you must use Text Nodes & Styles.

Lines

¶

A

line

is the most common element of the Yarn dialogue. It?s just a single phrase that a character in the game sa

.yarn

file, a

line

is represented by a single line of text in a

node body

. A line may contain the following elements:

A character ID;

Normal text;

Escaped text;

Interpolated expressions;

Markup;

Hashtags;

A comment at the end of the line;

(a line, however, cannot contain commands).

A

line

is represented with the

DialogueLine

class in Jenny runtime.

Character ID

¶

If a line starts with a single word followed by a

:

, then that word is presumed to be the name of the character who is speaking that line. In the following exa

title

:

Bulldozer_Conversation

---

Prosser

:

You want me to come and lie there...

Ford

:

Yes

Prosser

:

In front of the bulldozer?

Ford

:

Yes

Prosser

:

In the mud.

Ford

:

In, as you say, the mud. (low rumbling noise...)

===

It is worth emphasizing that a character ID must be a valid ID ? that is, it cannot contain spaces or other sp

title

:

Hello

---

Harry Potter: Hello, Hermione!

Harry_Potter

:

Hello, Hermione!

HarryPotter

:

Hello, Hermione!

Harry

:

Hello, Hermione!

===

If you want to have a line that starts with a

WORD

+

':'

, but you don?t want that word to be interpreted as a character name, then the colon can be

escaped

:

title

:

Warning

---

Attention

\:

The cake is NOT a lie

===

Note

All characters must be

declared

using the [<<character>>] command before they can be used in a script.

Interpolated expressions

¶

You can insert dynamic text into a line with the help of

interpolated expression

s. These expressions are surrounded with curly braces

{}

, and everything inside the braces will be evaluated, and then the result of the evaluation will be inserted in

title

:

Greeting

---

Trader

:

Hello,

{

$player_name

}

! Would you like to see my wares?

Player

:

I have only

{

plural

(

$money

,

"

% coin

"

)}

, do you have anything I can afford?

===

The expressions will be evaluated at runtime when the line is delivered, which means it can produce differe

title

:

Exam_Greeting

---

<<

if

$n_attempts

==

0

>>

Professor

:

Welcome to the exam!

<<

jump

Exam

>>

<<

elseif

$n_attempts

<

5

>>

Professor

:

You have tried

{

plural

(

$n_attempts

,

"

% time

"

)}

already, but I

\

can give you another try.

<<

jump

Exam

>>

<<

else

>>

Professor

:

You've failed 5 times in a row! How is this even possible?

<<

endif

>>

===

After evaluation, the text of the expression will be inserted into the line as-is, without any further processing

[

,

]

,

{

,

}

,

\

, etc), and they don?t need to be escaped. It also means that the expression cannot contain markup, or pro

Read more about expressions in the

Expressions

section.

Markup

¶

The

markup

is a mechanism for text annotation. It is somewhat similar to HTML tags, except that it uses square bracket

[]

instead of angular ones:

title

:

Markup

---

Wizard

:

No, no, no!

[em]

This is insanity!

[/em]

===

The markup tags do not alter the text of the line, they merely insert annotations in it. Thus, the line above w

em

tag.

Markup tags can be nested, or be zero-width, they can also include parameters whose values can be dyna

Markup

document.

Hashtags

¶

Hashtags may appear at the end of the line, and take the following form:

#text

. That is, a hashtag is a

#

symbol followed by any text that doesn?t contain whitespace.

Hashtags are used to add line-level metadata. There can be no line content after a hashtag (though comm

title

:

Hashtags

---

Harry

:

There is no justice in the laws of Nature, Headmaster, no term for

\

fairness in the equations of motion.

#sad

// HPMOR.39

Harry

:

The universe is neither evil, nor good, it simply does not care.

Harry

:

The stars don't care, or the Sun, or the sky.

Harry

:

But they don't have to! We care!

#elated

#volume:+1

Harry

:

There is light in the world, and it is us!

#volume:+2

===

In most cases the Jenny engine does not interpret the tags, but merely stores them as part of the line infor

Escaped text

¶

Whenever you have a line that needs to include a character that would normally be interpreted as one of th

escaped

with a backslash

\

.

The following escape sequences are recognized:

\\

,

\/

,

\#

,

\<

,

\>

,

\[

,

\]

,

\{

,

\}

,

\:

,

\-

,

\n

. In addition, there is also

\?

(i.e. backslash followed immediately by a newline).

title

:

Escapes

---

\/

/

This is not a comment

// but this is

This is not a

\#

hashtag This is not a

\<

<command>>

\{

This line

\}

does not contain an expression Not a

\[

markup

\]

===

The

\?

escape can be used to split a single long line into multiple physical lines, that would still be treated by Jenn

title

:

One_long_line

---

This line is so long that it becomes uncomfortable to read in a text editor.

\

Therefore, we use the backslash-newline escape sequence to split it into

\

several physical lines. The indentation at the start of the continuation

\

lines is for convenience only, and will be removed from the resulting

\

text.

===

FlameGame

¶

The

FlameGame

class implements a

Component

based

Game

. It has a tree of components and calls the

update

and

render

methods of all components that have been added to the game.

We refer to this component-based system as the Flame Component System (FCS). Throughout the docum

Components can be added to the

FlameGame

directly in the constructor with the named

children

argument, or from anywhere else with the

add

/

addAll

methods. Most of the time however, you want to add your children to a

World

, the default world exist under

FlameGame.world

and you add components to it just like you would to any other component.

A simple

FlameGame

implementation that adds two components, one in

onLoad

and one directly in the constructor can look like this:

```
import

'package:flame/components.dart'

;

import

'package:flame/game.dart'

;

import

'package:flutter/widgets.dart'

;

/// A component that renders the crate sprite, with a 16 x 16 size.

class

MyCrate

extends

SpriteComponent

{

MyCrate

()

:

super

(
```

size:

Vector2

.

all

(

16

));

@override

Future

<

void

>

onLoad

()

async

{

sprite

=

await

Sprite

.

load

(

'crate.png'

);

}

```
}
class
MyWorld
extends
World
{
@override
Future
<
void
>
onLoad
()
async
{
await
add
(
MyCrate
());
}
}
void
main
()
{
```

final

myGame

=

FlameGame

(

world:

MyWorld

());

runApp

(

GameWidget

(

game:

myGame

),

);

}

Note

If you instantiate your game in a build method your game will be rebuilt every time the Flutter tree gets rebu

GameWidget.controlled

constructor.

To remove components from the list on a

FlameGame

the

remove

or

removeAll

methods can be used. The first can be used if you just want to remove one component, and the second ca

Component

s, including the world.

Game Loop

¶

The

GameLoop

module is a simple abstraction of the game loop concept. Basically, most games are built upon two method

The render method takes the canvas for drawing the current state of the game.

The update method receives the delta time in microseconds since the last update and allows you to move t

The

GameLoop

is used by all of Flame?s

Game

implementations.

Resizing

¶

Every time the game needs to be resized, for example when the orientation is changed,

FlameGame

will call all of the

Component

s

onGameResize

methods and it will also pass this information to the camera and viewport.

The

FlameGame.camera

controls which point in the coordinate space that should be at the anchor of your viewfinder, [0,0] is in the c

Anchor.center

) of the viewport by default.

Lifecycle

¶

The

FlameGame

lifecycle callbacks,

onLoad

,

render

, etc. are called in the following sequence:

%%{init: { 'theme': 'dark' } }%% graph TD %% Node Color %% classDef default fill:#282828,stroke:#

%%{init: { 'theme': 'dark' } }%% graph LR %% Node Color %% classDef default fill:#282828,stroke:#F

When a

FlameGame

is first added to a

GameWidget

the lifecycle methods

onGameResize

,

onLoad

and

onMount

will be called in that order. Then

update

and

render

are called in sequence for every game tick.  If the

FlameGame

is removed from the

GameWidget

then

onRemove

is called. If the

FlameGame

is added to a new

GameWidget

the sequence repeats from

onGameResize

.

Note

The order of

onGameResize

and

onLoad

are reversed from that of other

Component

s. This is to allow game element sizes to be calculated before resources are loaded or generated.

The

onRemove

callback can be used to clean up children and cached data:

```
@override
void
onRemove
()
{
// Optional based on your game needs.
removeAll
(
children
);
processLifecycleEvents
();
Flame
.
images
.
clearCache
();
Flame
.
assets
.
clearCache
();
// Any other code that you want to run when the game is removed.
```

}

Note

Clean-up of children and resources in a

FlameGame

is not done automatically and must be explicitly added to the

onRemove

call.

Debug mode

¶

Flame?s

FlameGame

class provides a variable called

debugMode

, which by default is

false

. It can, however, be set to

true

to enable debug features for the components of the game.

Be aware

that the value of this variable is passed through to its components when they are added to the game, so if y

debugMode

at runtime, it will not affect already added components by default.

To read more about the

debugMode

on Flame, please refer to the

Debug Docs

Change background color

¶

To change the background color of your

FlameGame

you have to override

backgroundColor()

.

In the following example, the background color is set to be fully transparent, so that you can see the widget

GameWidget

. The default is opaque black.

```
class
MyGame
extends
FlameGame
{
@override
Color
backgroundColor
()
=>
const
Color
(
0x00000000
);
}
```

Note that the background color can?t change dynamically while the game is running, but you could just dra

SingleGameInstance mixin

¶

An optional mixin

SingleGameInstance

can be applied to your game if you are making a single-game application. This is a common scenario wher

GameWidget

that hosts a single

Game

instance.

Adding this mixin provides performance advantages in certain scenarios. In particular, a component?s

onLoad

method is guaranteed to start when that component is added to its parent, even if the parent is not yet mou

await

-ing on

parent.add(component)

is guaranteed to always finish loading the component.

Using this mixin is simple:

class

MyGame

extends

FlameGame

with

SingleGameInstance

{

// ...

}

Low-level Game API

¶

%%{init: { 'theme': 'dark' } }%%  graph TD  %% Node Color %%  classDef default fill:#282828,strok

%%{init: { 'theme': 'dark' } }%%  graph BT  %% Node Color %%  classDef default fill:#282828,strok

The abstract

Game

class is a low-level API that can be used when you want to implement the functionality of how the game en

Game

does not implement any

update

or

render

function for example.

The class also has the lifecycle methods

onLoad

,

onMount

and

onRemove

in it, which are called from the

GameWidget

(or another parent) when the game is loaded + mounted, or removed.

onLoad

is only called the first time the class is added to a parent, but

onMount

(which is called after

onLoad

) is called every time it is added to a new parent.

onRemove

is called when the class is removed from a parent.

Note

The

Game

class allows for more freedom of how to implement things, but you are also missing out on all of the built-in

An example of how a

Game

implementation could look like is:

```
class

MyGameSubClass

extends

Game

{

@override

void

render

(

Canvas

canvas

)

{

// ...
```

```
}
@override
void
update
(
double
dt
)
{
// ...
}
}
void
main
()
{
final
myGame
=
MyGameSubClass
();
runApp
(
GameWidget
(
game:
```

myGame

,

)

);

}

Pause/Resuming/Stepping game execution

¶

A Flame

Game

can be paused and resumed in two ways:

With the use of the

pauseEngine

and

resumeEngine

methods.

By changing the

paused

attribute.

When pausing a

Game

, the

GameLoop

is effectively paused, meaning that no updates or new renders will happen until it is resumed.

While the game is paused, it is possible to advanced it frame by frame using the

stepEngine

method. It might not be much useful in the final game, but can be very helpful in inspecting game state step

Backgrounding

¶

The game will be automatically paused when the app is sent to the background, and resumed when it com

pauseWhenBackgrounded

to

false

.

class

MyGame

extends

FlameGame

{

MyGame

()

{

pauseWhenBackgrounded

=

false

;

}

}

On the current Flutter stable (3.13), this flag is effectively ignored on non-mobile platforms including the we

3. Building the World

¶

Creating Segments

¶

For this world to be infinite, the best way to approach this is to create segments that can be reloaded over a

Each segment is a 10x10 grid and each block is 64 pixels x 64 pixels. This means Ember Quest has a heig

Segment Manager

¶

To get started, we have to understand that we will be referencing our blocks in the segment manager, so fi

lib/objects

. In that folder, create 3 files called

ground_block.dart

,

platform_block.dart

, and

star.dart

. Those files just need basic boilerplate code for the class, so create the following in their respective files:

class

GroundBlock

{}

class

PlatformBlock

{}

class

Star

{}

Also, create

water_enemy.dart

in the

lib/actors

folder using this boilerplate code:

class

WaterEnemy

{}

Now we can create a file called

segment_manager.dart

which will be placed in a new folder called

lib/managers

. The segment manager is the heart and soul, if you will, of Ember Quest. This is where you can get as cre

segment_manager.dart

:

class

Block

{

// gridPosition position is always segment based X,Y.

// 0,0 is the bottom left corner.

// 10,10 is the upper right corner.

final

Vector2

gridPosition

;

final

Type

blockType

;

Block

(

this

.

gridPosition

,

this

.

blockType

);

}

final

segments

=

[

segment0

,

];

final

segment0

=

[

];

So what this does, is allows us to create segments (segment0, segment1, etc) in a list format that gets add

segments

list. The individual segments will be made up of multiple entries of the

Block

class. This information will allow us to translate the block position from a 10x10 grid to the actual pixel posi

To understand each segment, if we start in the bottom left corner of the grid in the sketch, we see that we s

Block()

in the

segment0

list with a first parameter

gridPosition

of a

Vector2(0,0)

and a

blockType

of the

GroundBlock

class that we created earlier. Remember, the very bottom left cell is x=0 and y=0 thus the

Vector2(x,y)

is

Vector2(0,0)

.

The full segment would look like this:

final

segment0

=

[

Block

(

Vector2

(

0

,

0

),

GroundBlock

),

Block

(

Vector2

(

1

,

0

),

GroundBlock

),

Block

(

Vector2

(

2

,

0

),

GroundBlock

),

Block

(

Vector2

(

3

,

0

),

GroundBlock

),

Block

(

Vector2

(

4

,

0

),

GroundBlock

),

Block

(

Vector2

(

5

,

0

),

GroundBlock

),

Block

(

Vector2

(

5

,

1

),

WaterEnemy

),

Block

(

Vector2

(

5

,

3

),

PlatformBlock

),

Block

(

Vector2

(

6

,

0

),

GroundBlock

),

Block

(

Vector2

(

6

,

3

),

PlatformBlock

),

Block

(

Vector2

(

7

,

0

),

GroundBlock

),

Block

(

Vector2

(

7

,

3

),

PlatformBlock

),

Block

(

Vector2

(

8

,

0

),

GroundBlock

```
),
Block
(
Vector2
(
8
,
3
),
PlatformBlock
),
Block
(
Vector2
(
9
,
0
),
GroundBlock
),
];
```

Proceed to build the remaining segments. The full segment manager should look like this:

```
import
'package:flame/components.dart'
;
```

```
import
'../actors/water_enemy.dart'
;
import
'../objects/ground_block.dart'
;
import
'../objects/platform_block.dart'
;
import
'../objects/star.dart'
;
class
Block
{
// gridPosition position is always segment based X,Y.
// 0,0 is the bottom left corner.
// 10,10 is the upper right corner.
final
Vector2
gridPosition
;
final
Type
blockType
;
```

```
Block
(
this
.
gridPosition
,
this
.
blockType
);
}
final
segments
=
[
segment0
,
segment1
,
segment2
,
segment3
,
segment4
,
];
```

final

segment0

=

[

Block

(

Vector2

(

0

,

0

),

GroundBlock

),

Block

(

Vector2

(

1

,

0

),

GroundBlock

),

Block

(

Vector2

(

2

,

0

),

GroundBlock

),

Block

(

Vector2

(

3

,

0

),

GroundBlock

),

Block

(

Vector2

(

4

,

0

),

GroundBlock

),

Block

(

Vector2

(

5

,

0

),

GroundBlock

),

Block

(

Vector2

(

5

,

1

),

WaterEnemy

),

Block

(

Vector2

(

5

,

3

),

PlatformBlock

),

Block

(

Vector2

(

6

,

0

),

GroundBlock

),

Block

(

Vector2

(

6

,

3

),

PlatformBlock

),

Block

(

Vector2

(

7

,

0

),

GroundBlock

),

Block

(

Vector2

(

7

,

3

),

PlatformBlock

),

Block

(

Vector2

(

8

,

0

),

GroundBlock

),

Block

(

Vector2

(

8

,

3

),

PlatformBlock

),

Block

(

Vector2

(

9

,

0

),

GroundBlock

),

];

final

segment1

=

[

Block

(

Vector2

(

0

,

0

),

GroundBlock

),

Block

(

Vector2

(

1

,

0

),

GroundBlock

),

Block

(

Vector2

(

1

,

1

),

PlatformBlock

),

Block

(

Vector2

(

1

,

2

),

PlatformBlock

),

Block

(

Vector2

(

1

,

3

),

PlatformBlock

),

Block

(

Vector2

(

2

,

6

),

PlatformBlock

),

Block

(

Vector2

(

3

,

6

),

PlatformBlock

),

Block

(

Vector2

(

6

,

5

),

PlatformBlock

),

Block

(

Vector2

(

7

,

5

),

PlatformBlock

),

Block

(

Vector2

(

7

,

7

),

Star

),

Block

(

Vector2

(

8

,

0

),

GroundBlock

),

Block

(

Vector2

(

8

,

1

),

PlatformBlock

),

Block

(

Vector2

(

8

,

5

),

PlatformBlock

),

Block

(

Vector2

(

8

,

6

),

WaterEnemy

),

Block

(

Vector2

(

9

,

0

),

GroundBlock

),

];

final

segment2

=

[

Block

(

Vector2

(

0

,

0

),

GroundBlock

),

Block

(

Vector2

(

1

,

0

),

GroundBlock

),

Block

(

Vector2

(

2

,

0

),

GroundBlock

),

Block

(

Vector2

(

3

,

0

),

GroundBlock

),

Block

(

Vector2

(

3

,

3

),

PlatformBlock

),

Block

(

Vector2

(

4

,

0

),

GroundBlock

),

Block

(

Vector2

(

4

,

3

),

PlatformBlock

),

Block

(

Vector2

(

5

,

0

),

GroundBlock

),

Block

(

Vector2

(

5

,

3

),

PlatformBlock

),

Block

(

Vector2

(

5

,

4

),

WaterEnemy

),

Block

(

Vector2

(

6

,

0

),

GroundBlock

),

Block

(

Vector2

(

6

,

3

),

PlatformBlock

),

Block

(

Vector2

(

6

,

4

),

PlatformBlock

),

Block

(

Vector2

(

6

,

5

),

PlatformBlock

),

Block

(

Vector2

(

6

,

7

),

Star

),

Block

(

Vector2

(

7

,

0

),

GroundBlock

),

Block

(

Vector2

(

8

,

0

),

GroundBlock

),

Block

(

Vector2

(

9

,

0

),

GroundBlock

),

];

final

segment3

=

[

Block

(

Vector2

(

0

,

0

),

GroundBlock

),

Block

(

Vector2

(

1

,

0

),

GroundBlock

),

Block

(

Vector2

(

1

,

1

),

WaterEnemy

),

Block

(

Vector2

(

2

,

0

),

GroundBlock

),

Block

(

Vector2

(

2

,

1

),

PlatformBlock

),

Block

(

Vector2

(

2

,

2

),

PlatformBlock

),

Block

(

Vector2

(

4

,

4

),

PlatformBlock

),

Block

(

Vector2

(

6

,

6

),

PlatformBlock

),

Block

(

Vector2

(

7

,

0

),

GroundBlock

),

Block

(

Vector2

(

7

,

1

),

PlatformBlock

),

Block

(

Vector2

(

8

,

0

),

GroundBlock

),

Block

(

Vector2

(

8

,

8

),

Star

),

Block

(

Vector2

(

9

,

0

),

GroundBlock

),

];

final

segment4

=

[

Block

(

Vector2

(

0

,

0

),

GroundBlock

),

Block

(

Vector2

(

1

,

0

),

GroundBlock

),

Block

(

Vector2

(

2

,

0

),

GroundBlock

),

Block

(

Vector2

(

2

,

3

),

PlatformBlock

),

Block

(

Vector2

(

3

,

0

),

GroundBlock

),

Block

(

Vector2

(

3

,

1

),

WaterEnemy

),

Block

(

Vector2

(

3

,

3

),

PlatformBlock

),

Block

(

Vector2

(

4

,

0

),

GroundBlock

),

Block

(

Vector2

(

5

,

0

),

GroundBlock

),

Block

(

Vector2

(

5

,

5

),

PlatformBlock

),

Block

(

Vector2

(

6

,

0

),

GroundBlock

),

Block

(

Vector2

(

6

,

5

),

PlatformBlock

),

Block

(

Vector2

(

6

,

7

),

Star

),

Block

(

Vector2

(

7

,

0

),

GroundBlock

),

Block

(

Vector2

(

8

,

0

),

GroundBlock

),

Block

(

Vector2

(

8

,

3

),

PlatformBlock

),

Block

(

Vector2

(

9

,

0

),

GroundBlock

),

Block

(

Vector2

(

9

,

1

),

WaterEnemy

),

Block

(

Vector2

(

9

,

3

),

PlatformBlock

),

];

Loading the Segments into the World

¶

Now that our segments are defined, we need to create a way to load these blocks into our world. To do tha

ember_quest.dart

file. We will create a

loadSegments

method that when given an index for the segments list, will then loop through that segment from our

segment_manager

and we will add the appropriate blocks later. It should look like this:

void

loadGameSegments

(

int

segmentIndex

,

double

xPositionOffset

)

{

for

(

final

block

in

segments

[

segmentIndex

])

{

switch

(

block

.

```
blockType
)
{
case
GroundBlock:
break
;
case
PlatformBlock:
break
;
case
Star:
break
;
case
WaterEnemy:
break
;
}
}
}
```

You will need to add the following imports if they were not auto-imported:

```
import
'actors/water_enemy.dart'
;
```

import

'managers/segment_manager.dart'

;

import

'objects/ground_block.dart'

;

import

'objects/platform_block.dart'

;

import

'objects/star.dart'

;

Now we can refactor our game a bit and create an

initializeGame()

method which will call our

loadGameSegments

method.

void

initializeGame

()

{

// Assume that size.x < 3200

final

segmentsToLoad

=

(

size

.

x

/

640

).

ceil

();

segmentsToLoad

.

clamp

(

0

,

segments

.

length

);

for

(

var

i

=

0

;

i

<=

segmentsToLoad

;

i

++

)

{

loadGameSegments

(

i

,

(

640

*

i

).

toDouble

());

}

_ember

=

EmberPlayer

(

position:

Vector2

(

128

,

canvasSize

.

y

-

70

),

);

world

.

add

(

_ember

);

}

We simply are taking the width of the game screen, divide that by 640 (10 blocks in a segment times 64 pix

segmentsToLoad

and call

loadGameSegments

with the integer to load and then calculate the offset.

Additionally, I have moved the Ember-related code from the

onLoad

method to our new

initializeGame

method. This means I can now make the call in

onLoad

to

initializeGame

such as:

@override

Future

<

void

>

onLoad

()

async

{

await

images

.

loadAll

([

'block.png'

,

'ember.png'

,

'ground.png'

,

'heart_half.png'

,

'heart.png'

,

'star.png'

,

'water_enemy.png'

,

]);

camera

.

viewfinder

.

anchor

=

Anchor

.

topLeft

;

initializeGame

();

}

At this point, you probably have errors for all the object classes and the enemy class, but don?t worry, we v

The Platform Block

¶

One of the easiest blocks to start with is the Platform Block. There are two things that we need to develop l

Open the

lib/objects/platform_block.dart

file and add the following code:

```
import
'package:flame/collisions.dart'
;
import
'package:flame/components.dart'
;
import
'../ember_quest.dart'
;
class
PlatformBlock
extends
SpriteComponent
with
HasGameReference
<
EmberQuestGame
>
{
final
Vector2
gridPosition
;
double
xOffset
```

;

PlatformBlock

({

required

this

.

gridPosition

,

required

this

.

xOffset

,

})

:

super

(

size:

Vector2

.

all

(

64

),

anchor:

Anchor

.

bottomLeft

);

@override

void

onLoad

()

{

}

@override

void

update

(

double

dt

)

{

super

.

update

(

dt

);

}

}

We are going to extend the Flame

SpriteComponent

and we will need the

HasGameRef

mixin to access our game class just like we did before. We are starting with the empty

onLoad

and

update

methods and we will begin adding code to create the functionality that is necessary for the game.

The secret to any gaming engine is the game loop. This is an infinite loop that calls all the objects in your g

update

method is the hook into this and it uses a

double

dt

to pass to your method the amount of time in seconds since it was last called. This

dt

variable then allows you to calculate how far your component needs to move on-screen.

All components in our game will need to move at the same speed, so to do this, open

lib/ember_quest.dart

, and let?s define a global variable called

objectSpeed

. At the top of the

EmberQuestGame

class, add:

late

EmberPlayer

_ember

;

double

objectSpeed

=

0.0

;

So to implement that movement, declare a variable at the top of the

PlatformBlock

class and make your

update

method look like this:

final

Vector2

velocity

=

Vector2

.

zero

();

@override

void

update

(

double

dt

)

{

velocity

.

x

=

game

.

objectSpeed

;

position

+=

velocity

*

dt

;

if

(

position

.

x

<

-

size

.

x

)

removeFromParent

();

super

.

update

(

dt

);

}

All that is happening is we define a base

velocity

that is instantiated at 0 on both axes and then we update

velocity

using the global

objectSpeed

variable for the x-axis. As this is our platform block, it will only scroll left and right, so our y-axis in the

velocity

will always be 0 as do not want our blocks jumping.

Next, we update the

position

which is a special variable built into the Flame engine components. By multiplying the

velocity

vector by the

dt

we can move our component to the required amount.

Finally, if

x

value of position is

-size.x

(this means off the left side of the screen by the width of the image) then remove this platform block from th

Now we just need to finish the

onLoad

method. So make your

onLoad

method look like this:

@override

void

onLoad

()

{

final

platformImage

=

game

.

images

.

fromCache

(

'block.png'

);

sprite

=

Sprite

(

platformImage

);

position

=

Vector2

((

gridPosition

.

x

*

size

.

x

)

+

xOffset

,

game

.

size

.

y

-

(

gridPosition

.

y

*

size

.

y

),

);

add

(

RectangleHitbox

(

collisionType:

CollisionType

.

passive

));

}

First, we retrieve the image from cache as we did before, and because this is a

SpriteComponent

we can use the built-in

sprite

variable to assign the image to the component. Next, we need to calculate its starting position. This is whe

Just like in the

update

method we will be setting the

position

variable to a

Vector2

. To determine where it needs to be, we need to calculate the x and y positions. Focusing on the x first, we

gridPosition.x

times the width of the image and then we will add that to the

xOffset

that we pass in. With the y-axis, we will take the height of the game and we will subtract the

gridPosition.y

times the height of the image.

Lastly, as we want Ember to be able to interact with the platform, we will add a

RectangleHitbox

with a

passive

CollisionType

. Collisions will be explained more in a later chapter.

Display the Platform

¶

In our

loadGameSegments

method from earlier, we will need to add the call to add our block. We will need to define

gridPosition

and

xOffset

to be passed in.

gridPosition

will be a

Vector2

and

xOffset

is a double as that will be used to calculate the x-axis offset for the block in a

Vector2

. So add the following to your

loadGameSegments

method:

case

PlatformBlock:

add

(

PlatformBlock

(

gridPosition:

block

.

gridPosition

,

xOffset:

xPositionOffset

,

));

break

;

If you run your code, you should now see:

While this does run, the black just makes it look like Ember is in a dungeon. Let?s change that background

lib/ember_quest.dart

:

import

'package:flutter/material.dart'

;

@override

Color

backgroundColor

()

{

return

const

Color

.

fromARGB

(

255

,

173

,

223

,

247

);

}

Excellent! Ember is now in front of a blue sky.

On to

4. Adding the Remaining Components

, where we will add the rest of the components now that we have a basic understanding of what we are goi

router.dart

1

import

'package:flame/components.dart'

;

2

import

'package:flame/effects.dart'

;

3

import

'package:flame/events.dart'

;

4

import

'package:flame/game.dart'

;

5

import

'package:flame/geometry.dart'

;

6

import

'package:flame/rendering.dart'

;

7

import

'package:flutter/rendering.dart'

;

8

9

class

RouterGame

extends

FlameGame

{

10

late

final

RouterComponent

router

;

11

12

@override

13

Future

<

void

>

onLoad

()

```
async
{
14
add
(
15
router
=
RouterComponent
(
16
routes:
{
17
'splash'
:
Route
(
SplashScreenPage
.
new
),
18
'home'
:
Route
```

(

StartPage

.

new

),

19

'level1'

:

Route

(

Level1Page

.

new

),

20

'level2'

:

Route

(

Level2Page

.

new

),

21

'pause'

:

PauseRoute

(),

22

},

23

initialRoute:

'splash'

,

24

),

25

);

26

}

27

}

28

29

class

SplashScreenPage

extends

Component

30

with

TapCallbacks

,

HasGameReference

<

RouterGame

>

{

31

@override

32

Future

<

void

>

onLoad

()

async

{

33

addAll

([

34

Background

(

const

Color

(

0xff282828

)),

35

TextBoxComponent

(

36

text:

'[Router demo]'

,

37

textRenderer:

TextPaint

(

38

style:

const

TextStyle

(

39

color:

Color

(

0x66ffffff

),

40

fontSize:

16

,

41

),

42

),

43

align:

Anchor

.

center

,

44

size:

game

.

canvasSize

,

45

),

46

]);

47

}

48

49

@override

50

bool

containsLocalPoint

(

Vector2

point

)

=>

true

;

51

52

@override

53

void

onTapUp

(

TapUpEvent

event

)

=>

game

.

router

.

pushNamed

```
(
'home'
);
54
}
55
56
class
StartPage
extends
Component
with
HasGameReference
<
RouterGame
>
{
57
StartPage
()
{
58
addAll
([
59
_logo
```

=

TextComponent

(

60

text:

'Syzygy'

,

61

textRenderer:

TextPaint

(

62

style:

const

TextStyle

(

63

fontSize:

64

,

64

color:

Color

(

0xFFC8FFF5

),

65

fontWeight:

FontWeight

.

w800

,

66

),

67

),

68

anchor:

Anchor

.

center

,

69

),

70

_button1

=

RoundedButton

(

71

text:

'Level 1'

,

72

action:

()

=>

game

.

router

.

pushNamed

(

'level1'

),

73

color:

const

Color

(

0xffadde6c

),

74

borderColor:

const

Color

(

0xffedffab

),

75

),

76

_button2

=

RoundedButton

(

77

text:

'Level 2'

,

78

action:

()

=>

game

.

router

.

pushNamed

(

'level2'

),

79

color:

```
const
Color
(
0xffdebe6c
),
80
borderColor:
const
Color
(
0xfffff4c7
),
81
),
82
]);
83
}
84
85
late
final
TextComponent
_logo
;
86
```

late

final

RoundedButton

_button1

;

87

late

final

RoundedButton

_button2

;

88

89

@override

90

void

onGameResize

(

Vector2

size

)

{

91

super

.

onGameResize

(

size

);

92

_logo

.

position

=

Vector2

(

size

.

x

/

2

,

size

.

y

/

3

);

93

_button1

.

position

=

Vector2

(

size

.

x

/

2

,

_logo

.

y

+

80

);

94

_button2

.

position

=

Vector2

(

size

.

x

/

2

,

_logo

.

y

+

140

);

95

}

96

}

97

98

class

Background

extends

Component

{

99

Background

(

this

.

color

);

100

final

Color

color

;

101

102

@override

103

void

render

(

Canvas

canvas

)

{

104

canvas

.

drawColor

(

color

,

BlendMode

.

srcATop

);

105

}

106

}

107

108

class

RoundedButton

extends

PositionComponent

with

TapCallbacks

{

109

RoundedButton

({

110

required

this

.

text

,

111

required

this

.

action

,

112

required

Color

color

,

113

required

Color

borderColor

,

114

super

.

anchor

=

Anchor

.

center

,

115

})

:

_textDrawable

=

TextPaint

(

116

style:

const

TextStyle

(

117

fontSize:

20

,

118

color:

Color

(

0xFF000000

),

119

fontWeight:

FontWeight

.

w800

,

120

),

121

).

toTextPainter

(

text

)

{

122

size

=

Vector2

(

150

,

40

);

123

_textOffset

=

Offset

(

124

(

size

.

x

-

_textDrawable

.

width

)

/

2

,

125

(

size

.

y

-

_textDrawable

.

height

)

/

2

,

126

);

127

_rrect

=

RRect

.

fromLTRBR

(

0

,

0

,

size

.

x

,

size

.

y

,

Radius

.

circular

(

size

.

y

/

2

));

128

_bgPaint

=

Paint

()..

color

=

color

;

129

_borderPaint

=

Paint

()

130

..

style

=

PaintingStyle

.

stroke

131

..

strokeWidth

=

2

132

..

color

=

borderColor

;

133

}

134

135

final

String

text

;

136

final

void

Function

()

action

;

137

final

TextPainter

_textDrawable

;

138

late

final

Offset

_textOffset

;

139

late

final

RRect

_rrect

;

140

late

final

Paint

_borderPaint

;

141

late

final

Paint

_bgPaint

;

142

143

```
@override
144
void
render
(
Canvas
canvas
)
{
145
canvas
.
drawRRect
(
_rrect
,
_bgPaint
);
146
canvas
.
drawRRect
(
_rrect
,
_borderPaint
```

);

147

_textDrawable

.

paint

(

canvas

,

_textOffset

);

148

}

149

150

@override

151

void

onTapDown

(

TapDownEvent

event

)

{

152

scale

=

Vector2

.

all

(

1.05

);

153

}

154

155

@override

156

void

onTapUp

(

TapUpEvent

event

)

{

157

scale

=

Vector2

.

all

(

1.0

);

158

action

();

159

}

160

161

@override

162

void

onTapCancel

(

TapCancelEvent

event

)

{

163

scale

=

Vector2

.

all

(

1.0

);

164

}

165

}

166

167

abstract

class

SimpleButton

extends

PositionComponent

with

TapCallbacks

{

168

SimpleButton

(

this

.

_iconPath

,

{

super

.

position

})

:

super

(

size:

Vector2

.

all

(

40

));

169

170

final

Paint

_borderPaint

=

Paint

()

171

..

style

=

PaintingStyle

.

stroke

172

..

color

=

const

Color

(

0x66ffffff

);

173

final

Paint

_iconPaint

=

Paint

()

174

..

style

=

PaintingStyle

.

stroke

175

..

color

=

const

Color

(

0xffaaaaaa

)

176

..

strokeWidth

=

7

;

177

final

Path

_iconPath

;

178

179

void

action

();

180

181

@override

182

```
void
render
(
Canvas
canvas
)
{
183
canvas
.
drawRRect
(
184
RRect
.
fromRectAndRadius
(
size
.
toRect
(),
const
Radius
.
circular
(
```

8

)),

185

_borderPaint

,

186

);

187

canvas

.

drawPath

(

_iconPath

,

_iconPaint

);

188

}

189

190

@override

191

void

onTapDown

(

TapDownEvent

```
event
)
{
192
_iconPaint
.
color
=
const
Color
(
0xffffffff
);
193
}
194
195
@override
196
void
onTapUp
(
TapUpEvent
event
)
{
```

197

_iconPaint

.

color

=

const

Color

(

0xffaaaaaa

);

198

action

();

199

}

200

201

@override

202

void

onTapCancel

(

TapCancelEvent

event

)

{

203

_iconPaint

.

color

=

const

Color

(

0xffaaaaaa

);

204

}

205

}

206

207

class

BackButton

extends

SimpleButton

with

HasGameReference

<

RouterGame

>

{

208

BackButton

()

209

:

super

(

210

Path

()

211

..

moveTo

(

22

,

8

)

212

..

lineTo

(

10

,

20

)

213

..

lineTo

(

22

,

32

)

214

..

moveTo

(

12

,

20

)

215

..

lineTo

(

34

,

20

),

216

position:

Vector2

.

all

(

10

),

217

);

218

219

@override

220

void

action

()

=>

game

.

router

.

pop

();

221

}

222

223

class

PauseButton

extends

SimpleButton

with

HasGameReference

<

RouterGame

>

{

224

PauseButton

()

225

:

super

(

226

Path

()

227

..

moveTo

(

14

,

10

)

228

..

lineTo

(

14

,

30

)

229

..

moveTo

(

26

,

10

)

230

..

lineTo

(

26

,

30

),

231

position:

Vector2

(

60

,

10

),

232

);

233

@override

234

void

action

()

=>

game

.

router

.

pushNamed

(

'pause'

);

235

}

236

237

class

Level1Page

extends

Component

{

238

@override

239

Future

<

void

>

onLoad

()

async

{

240

final

game

=

findGame

()

!

;

241

addAll

([

242

Background

(

const

Color

(

0xbb2a074f

)),

243

BackButton

(),

244

PauseButton

(),

245

Planet

(

246

radius:

25

,

247

color:

const

Color

(

0xfffff188

),

248

position:

game

.

size

/

2

,

249

children:

[

250

Orbit

(

251

radius:

110

,

252

revolutionPeriod:

6

,

253

planet:

Planet

(

254

radius:

10

,

255

color:

const

Color

(

0xff54d7b1

),

256

children:

[

257

Orbit

(

258

radius:

25

,

259

revolutionPeriod:

5

,

260

planet:

Planet

(

radius:

3

,

color:

const

Color

(

0xFFcccccc

)),

261

),

262

],

263

),

264

),

265

],

266

),

267

]);

268

}

269

}

270

271

class

Level2Page

extends

Component

{

272

@override

273

Future

<

void

>

onLoad

()

async

{

274

final

game

=

findGame

()

!

;

275

addAll

([

276

Background

(

const

Color

(

0xff052b44

)),

277

BackButton

(),

278

PauseButton

(),

279

Planet

(

280

radius:

30

,

281

color:

const

Color

(

0xFFFFFFff

),

282

position:

game

.

size

/

2

,

283

children:

[

284

Orbit

(

285

radius:

60

,

286

revolutionPeriod:

5

,

287

planet:

Planet

(

radius:

10

,

color:

const

Color

(

0xffc9ce0d

)),

288

),

289

Orbit

(

290

radius:

110

,

291

revolutionPeriod:

10

,

292

planet:

Planet

(

293

radius:

14

,

294

color:

const

Color

(

0xfff32727

),

295

children:

[

296

Orbit

(

297

radius:

26

,

298

revolutionPeriod:

3

,

299

planet:

Planet

(

radius:

5

,

color:

const

Color

(

0xffffdb00

)),

300

),

301

Orbit

(

302

radius:

35

,

303

revolutionPeriod:

4

,

304

planet:

Planet

(

radius:

3

,

color:

const

Color

(

0xffdc00ff

)),

305

),

306

],

307

),

308

),

309

],

310

),

311

]);

312

}

313

}

314

315

class

Planet

extends

PositionComponent

{

316

Planet

({

317

required

this

.

radius

,

318

required

this

.

color

,

319

super

.

position

,

320

super

.

children

,

321

})

:

_paint

=

Paint

()..

color

=

color

;

322

323

final

double

radius

;

324

final

Color

color

;

325

final

Paint

_paint

;

```
326
327
@override
328
void
render
(
Canvas
canvas
)
{
329
canvas
.
drawCircle
(
Offset
.
zero
,
radius
,
_paint
);
330
}
```

331

}

332

333

class

Orbit

extends

PositionComponent

{

334

Orbit

({

335

required

this

.

radius

,

336

required

this

.

planet

,

337

required

this

.

revolutionPeriod

,

338

double

initialAngle

=

0

,

339

})

:

_paint

=

Paint

()

340

..

style

=

PaintingStyle

.

stroke

341

..

color

=

const

Color

(

0x888888aa

),

342

_angle

=

initialAngle

{

343

add

(

planet

);

344

}

345

346

final

double

radius

;

347

final

double

revolutionPeriod

;

348

final

Planet

planet

;

349

final

Paint

_paint

;

350

double

_angle

;

351

352

@override

353

void

render

(

Canvas

canvas

)

{

354

canvas

.

drawCircle

(

Offset

.

zero

,

radius

,

_paint

);

355

}

356

357

@override

358

void

update

(

double

dt

)

{

359

_angle

+=

dt

/

revolutionPeriod

*

tau

;

360

planet

.

position

=

Vector2

(

radius

,

0

)..

rotate

(

_angle

);

361

}

362

}

363

364

class

PauseRoute

extends

Route

{

365

PauseRoute

()

:

super

(

PausePage

.

new

,

transparent:

true

);

366

367

@override

368

void

onPush

(

Route

?

previousRoute

)

{

369

previousRoute

!

370

..

stopTime

()

371

..

addRenderEffect

(

372

PaintDecorator

.

grayscale

(

opacity:

0.5

)..

addBlur

(

3.0

),

373

);

374

}

375

376

@override

377

void

onPop

(

Route

nextRoute

)

{

378

nextRoute

379

..

resumeTime

()

380

..

removeRenderEffect

();

381

}

382

}

383

384

class

PausePage

extends

Component

385

with

TapCallbacks

,

HasGameReference

<

RouterGame

>

{

386

@override

387

Future

<

void

>

onLoad

()

async

{

388

final

game

=

findGame

()

!

;

389

addAll

([

390

TextComponent

(

391

text:

'PAUSED'

,

392

position:

game

.

canvasSize

/

2

,

393

anchor:

Anchor

.

center

,

394

children:

[

395

ScaleEffect

.

to

(

396

Vector2

.

all

(

1.1

),

397

EffectController

(

398

duration:

0.3

,

399

alternate:

true

,

400

infinite:

true

,

401

),

402

),

403

```
],
404
),
405
]);
406
}
407
408
@override
409
bool
containsLocalPoint
(
Vector2
point
)
=>
true
;
410
411
@override
412
void
onTapUp
```

(

TapUpEvent

event

)

=>

game

.

router

.

pop

();

413

}

Code

This example app shows the use of the

RouterComponent

to move across multiple screens within the game. In addition, the ?pause? button stops time and applies vi

RouterComponent

¶

The

RouterComponent

?s job is to manage navigation across multiple screens within the game. It is similar in spirit to Flutter?s

Navigator

class, except that it works with Flame components instead of Flutter widgets.

A typical game will usually consist of multiple pages: the splash screen, the starting menu page, the setting

Internally, the

RouterComponent

contains a stack of routes. When you request it to show a route, it will be placed on top of all other pages in

pop()

to remove the topmost page from the stack. The pages of the router are addressed by their unique names.

Each page in the router can be either transparent or opaque. If a page is opaque, then the pages below it i

Usage example:

```
class
MyGame
extends
FlameGame
{
late
final
RouterComponent
router
;
@override
void
onLoad
()
{
add
(
router
=
RouterComponent
```

(

routes:

{

'home'

:

Route

(

HomePage

.

new

),

'level-selector'

:

Route

(

LevelSelectorPage

.

new

),

'settings'

:

Route

(

SettingsPage

.

new

,

transparent:

true

),

'pause'

:

PauseRoute

(),

'confirm-dialog'

:

OverlayRoute

.

existing

(),

},

initialRoute:

'home'

,

),

);

}

}

class

PauseRoute

extends

Route

{

...

}

Note

Use

hide

Route

if any of your imported packages export another class called

Route

eg:

import

'package:flutter/material.dart'

hide

Route;

Route

¶

The

Route

component holds information about the content of a particular page.

Route

s are mounted as children to the

RouterComponent

.

The main property of a

Route

is its

builder

? the function that creates the component with the content of its page.

In addition, the routes can be either transparent or opaque (default). An opaque prevents the route below it

By default, routes maintain the state of the page component after being popped from the stack and the

builder

function is only called the first time a route is activated. Setting

maintainState

to

false

drops the page component after the route is popped from the route stack and the

builder

function is called each time the route is activated.

The current route can be replaced using

pushReplacementNamed

or

pushReplacement

. Each method simply executes

pop

on the current route and then

pushNamed

or

pushRoute

.

OverlayRoute

¶

The

OverlayRoute

is a special route that allows adding game overlays via the router. These routes are transparent by default.

There are two constructors for the

OverlayRoute

. The first constructor requires a builder function that describes how the overlay?s widget is to be built. The

GameWidget

:

final

router

=

RouterComponent

(

routes:

{

'ok-dialog'

:

OverlayRoute

(

(

context

,

game

)

{

return

Center

(

child:

DecoratedContainer

(...),

);

},

),

// OverlayRoute

'confirm-dialog'

:

OverlayRoute

.

existing

(),

},

);

Overlays that were defined within the

GameWidget

don?t even need to be declared within the routes map beforehand: the

RouterComponent.pushOverlay()

method can do it for you. Once an overlay route was registered, it can be activated either via the regular

.pushNamed()

method, or via the

.pushOverlay()

? the two methods will do exactly the same, though you can use the second one to make it more clear in yo

The current overlay can be replaced using

pushReplacementOverlay

. This method executes

pushReplacementNamed

or

pushReplacement

based on the status of the overlay being pushed.

ValueRoute

¶

value_route.dart

```
1
import
'dart:math'
;
2
import
'dart:ui'
;
3
4
import
'package:doc_flame_examples/router.dart'
;
5
import
'package:flame/components.dart'
;
```

6

import

'package:flame/events.dart'

;

7

import

'package:flame/game.dart'

;

8

import

'package:flame/geometry.dart'

;

9

10

class

ValueRouteExample

extends

FlameGame

{

11

late

final

RouterComponent

router

;

12

13

@override

14

Future

<

void

>

onLoad

()

async

{

15

router

=

RouterComponent

(

16

routes:

{

'home'

:

Route

(

HomePage

.

new

)},

17

initialRoute:

'home'

,

18

);

19

add

(

router

);

20

}

21

}

22

23

class

HomePage

extends

Component

with

HasGameReference

<

ValueRouteExample

>

{

24

@override

25

Future

<

void

>

onLoad

()

async

{

26

add

(

27

RoundedButton

(

28

text:

'Rate me'

,

29

action:

()

async

{

30

final

score

=

await

game

.

router

.

pushAndWait

(

RateRoute

());

31

firstChild

<

TextComponent

>

()

!

.

text

=

'Score:

$

score

'

;

32

},

33

color:

const

Color

(

0xff758f9a

),

34

borderColor:

const

Color

(

0xff60d5ff

),

35

)..

position

=

game

.

size

/

2

,

36

);

37

add

(

38

TextComponent

(

39

text:

'Score: ?'

,

40

anchor:

Anchor

.

topCenter

,

41

position:

game

.

size

/

2

+

Vector2

(

0

,

30

),

42

scale:

Vector2

.

all

(

0.7

),

43

),

44

);

45

}

46

}

47

48

class

RateRoute

extends

ValueRoute

<

int

>

49

with

HasGameReference

<

ValueRouteExample

>

{

50

RateRoute

()

:

super

(

value:

-

1

,

transparent:

true

);

51

52

@override

53

Component

build

()

{

54

final

size

=

Vector2

(

250

,

130

);

55

const

radius

=

18.0

;

56

final

starGap

=

(

size

.

x

-

5

*

2

*

radius

)

/

6

;

57

return

RectangleComponent

(

58

position:

game

.

size

/

2

,

59

size:

size

,

60

anchor:

Anchor

.

center

,

61

paint:

Paint

()..

color

=

const

Color

(

0xee858585

),

62

children:

[

63

RoundedButton

(

64

text:

'Ok'

,

65

action:

()

{

66

completeWith

(

67

descendants

().

where

((

c

)

=>

c

is

Star

&&

c

.

active

).

length

,

68

);

69

},

70

color:

const

Color

(

0xFFFFFFFF

),

71

borderColor:

const

Color

(

0xFF000000

),

72

)..

position

=

Vector2

(

size

.

x

/

2

,

100

),

73

for

(

var

i

=

0

;

i

<

5

;

i

++

)

74

Star

(

75

value:

i

+

1

,

76

radius:

radius

,

77

position:

Vector2

(

starGap

*

(

i

+

1

)

+

radius

*

(

2

*

i

+

1

),

40

),

78

),

79

],

80

);

81

}

82

}

83

84

class

Star

extends

PositionComponent

with

TapCallbacks

{

85

Star

({

required

this

.

value

,

required

this

.

radius

,

super

.

position

})

86

:

super

(

size:

Vector2

.

all

(

2

*

radius

),

anchor:

Anchor

.

center

);

87

88

final

int

value

;

89

final

double

radius

;

90

final

Path

path

=

Path

();

91

final

Paint

borderPaint

=

Paint

()

92

..

style

=

PaintingStyle

.

stroke

93

..

color

=

```
const
Color
(
0xffffe395
)
94
..
strokeWidth
=
2
;
95
final
Paint
fillPaint
=
Paint
()..
color
=
const
Color
(
0xffffe395
);
96
```

bool

active

=

false

;

97

98

@override

99

Future

<

void

>

onLoad

()

async

{

100

path

.

moveTo

(

radius

,

0

);

101

for

(

var

i

=

0

;

i

<

5

;

i

++

)

{

102

path

.

lineTo

(

103

radius

+

0.6

*

radius

*

sin

(

tau

/

5

*

(

i

+

0.5

)),

104

radius

-

0.6

*

radius

*

cos

(

tau

/

5

*

(

i

+

0.5

)),

105

);

106

path

.

lineTo

(

107

radius

+

radius

*

sin

(

tau

/

5

*

(

i

+

1

)),

108

radius

-

radius

*

cos

(

tau

/

5

*

(

i

+

1

)),

109

);

110

}

111

path

.

close

();

112

}

113

114

@override

115

void

render

(

Canvas

canvas

)

{

116

if

(

active

)

{

117

canvas

.

drawPath

(

path

,

fillPaint

);

118

}

119

canvas

.

drawPath

(

path

,

borderPaint

);

120

}

121

122

@override

123

void

onTapDown

(

TapDownEvent

event

)

{

124

var

on

=

true

;

125

for

(

final

star

in

parent

!

.

children

.

whereType

<

Star

>

())

{

126

star

.

active

=

on

;

127

if

(

star

==

this

)

{

128

on

=

false

;

129

}

130

}

131

}

132

}

Code

A

ValueRoute

is a route that will return a value when it is eventually popped from the stack. Such routes can be used, for

In order to use

ValueRoute

s, two steps are required:

Create a route derived from the

ValueRoute<T>

class, where

T

is the type of the value that your route will return. Inside that class override the

build()

method to construct the component that will be displayed. The component should use the

completeWith(value)

method to pop the route and return the specified value.

class

YesNoDialog

extends

ValueRoute

<

bool

>

{

YesNoDialog

(

this

.

text

)

:

super

(

value:

false

);

final

String

text

;

@override

Component

build

()

{

return

PositionComponent

(

children:

[

RectangleComponent

(),

```
TextComponent
(
text:
text
),
Button
(
text:
'Yes'
,
action:
()
=>
completeWith
(
true
),
),
Button
(
text:
'No'
,
action:
()
=>
```

completeWith

(

false

),

),

],

);

}

}

Display the route using

Router.pushAndWait()

, which returns a future that resolves with the value returned from the route.

Future

<

void

>

foo

()

async

{

final

result

=

await

game

.

```
router
.
pushAndWait
(
YesNoDialog
(
'Are you sure?'
));
if
(
result
)
{
// ... the user is sure
}
else
{
// ... the user was not so sure
}
}
```

# Util

¶

On this page you can find documentation for some utility classes and methods.

## Device Class

¶

This class can be accessed from

Flame.device

and it has some methods that can be used to control the state of the device, for instance you can change t

### Flame.device.fullScreen()

¶

When called, this disables all

SystemUiOverlay

making the app full screen. When called in the main method, it makes your app full screen (no top nor botto

Note:

It has no effect when called on the web.

### Flame.device.setLandscape()

¶

This method sets the orientation of the whole application (effectively, also the game) to landscape and dep

Flame.device.setLandscapeLeftOnly

or

Flame.device.setLandscapeRightOnly

.

Note:

It has no effect when called on the web.

### Flame.device.setPortrait()

¶

This method sets the orientation of the whole application (effectively, also the game) to portrait and depend

Flame.device.setPortraitUpOnly

or

Flame.device.setPortraitDownOnly

.

Note:

It has no effect when called on the web.

Flame.device.setOrientation()

and

Flame.device.setOrientations()

¶

If a finer control of the allowed orientations is required (without having to deal with

SystemChrome

directly),

setOrientation

(accepts a single

DeviceOrientation

as a parameter) and

setOrientations

(accepts a

List<DeviceOrientation>

for possible orientations) can be used.

Note:

It has no effect when called on the web.

Timer

¶

Flame provides a simple utility class to help you handle countdowns and timer state changes like events.

Countdown example:

```
import

'package:flame/components.dart'

;

import

'package:flame/game.dart'

;

import

'package:flame/input.dart'

;

import

'package:flutter/material.dart'

;

class

MyGame

extends

Game

{

final

TextPaint

textPaint

=

TextPaint

(

style:
```

```
const
TextStyle
(
color:
Colors
.
white
,
fontSize:
20
),
);
final
countdown
=
Timer
(
2
);
@override
void
update
(
double
dt
)
```

```
{
countdown
.
update
(
dt
);
if
(
countdown
.
finished
)
{
// Prefer the timer callback, but this is better in some cases
}
}
@override
void
render
(
Canvas
canvas
)
{
textPaint
```

.

render

(

canvas

,

"Countdown:

${

countdown

.

current

.

toString

()

}

"

,

Vector2

(

10

,

100

),

);

}

}

Interval example:

```
import
'package:flame/components.dart'
;
import
'package:flame/game.dart'
;
import
'package:flame/input.dart'
;
import
'package:flutter/material.dart'
;
class
MyGame
extends
Game
{
final
TextPaint
textPaint
=
TextPaint
(
style:
const
TextStyle
```

(

color:

Colors

.

white

,

fontSize:

20

),

);

Timer

interval

;

int

elapsedSecs

=

0

;

MyGame

()

{

interval

=

Timer

(

1

,

onTick:

()

=>

elapsedSecs

+=

1

,

repeat:

true

,

);

}

@override

void

update

(

double

dt

)

{

interval

.

update

(

dt

```
);
}
@override
void
render
(
Canvas
canvas
)
{
textPaint
.
render
(
canvas
,
"Elapsed time:
$
elapsedSecs
"
,
Vector2
(
10
,
150
```

));

}

}

Timer

instances can also be used inside a

FlameGame

game by using the

TimerComponent

class.

TimerComponent

example:

import

'package:flame/timer.dart'

;

import

'package:flame/components.dart'

;

import

'package:flame/game.dart'

;

class

MyFlameGame

extends

FlameGame

{

MyFlameGame

```
()
{
add
(
TimerComponent
(
period:
10
,
repeat:
true
,
onTick:
()
=>
print
(
'10 seconds elapsed'
),
)
);
}
}
```

Time Scale

¶

In many games it is often desirable to create  slow-motion or fast-forward effects based on some in game e

To make this manipulation easier, Flame provides a

HasTimeScale

mixin. This mixin can be attached to any Flame

Component

and exposes a simple get/set API for

timeScale

. The default value of

timeScale

is

1

, implying in-game time of the component is running at the same speed as real life time. Setting it to

2

will make the component tick twice as fast and setting it to

0.5

will make it tick at half the speed as compared to real life time.

Since

FlameGame

is a

Component

too, this mixin can be attached to the

FlameGame

as well. Doing so will allow controlling time scale for all the component of the game from a single place.

time_scale.dart

1

import

'dart:async'

;

2

3

import

'package:doc_flame_examples/ember.dart'

;

4

import

'package:flame/components.dart'

;

5

import

'package:flame/game.dart'

;

6

7

class

TimeScaleGame

extends

FlameGame

with

HasTimeScale

{

8

final

_timeScales

=

[

0.5

,

1.0

,

2.0

];

9

var

_index

=

1

;

10

11

@override

12

Future

<

void

>

onLoad

()

async

{

13

await

add

(

14

EmberPlayer

(

15

position:

size

/

2

,

16

size:

size

/

4

,

17

onTap:

(

p0

)

=>

timeScale

=

getNextTimeScale

(),

18

),

19

);

20

return

super

.

onLoad

();

21

}

22

23

double

getNextTimeScale

()

{

24

++

_index

;

25

```
if
(
_index
>=
_timeScales
.
length
)
{
26
_index
=
0
;
27
}
28
return
_timeScales
[
_index
];
29
}
30
}
```

Code

```
import
'package:flame/components.dart'
;
import
'package:flame/game.dart'
;
class
MyFlameGame
extends
FlameGame
with
HasTimeScale
{
void
speedUp
(){
timeScale
=
2.0
;
}
void
slowDown
(){
timeScale
```

=

1.0

;

}

}

Extensions

¶

Flame bundles a collection of utility extensions, these extensions are meant to help the developer with sho

They can all be imported from

package:flame/extensions.dart

Canvas

¶

Methods:

scaleVector

: Just like

canvas

scale

method, but takes a

Vector2

as an argument.

translateVector

: Just like

canvas

translate

method, but takes a

Vector2

as an argument.

renderPoint

: renders a single point on the canvas (mostly for debugging purposes).

renderAt

and

renderRotated

: if you are directly rendering to the

Canvas

, you can use these functions to easily manipulate coordinates to render things on the correct places. They

Canvas

transformation matrix but reset afterwards.

Color

¶

Methods:

darken

: Darken the shade of the color by an amount between 0 to 1.

brighten

: Brighten the shade of the color by an amount between 0 to 1.

Factories:

ColorExtension.fromRGBHexString

: Parses an RGB color from a valid hex string (e.g. #1C1C1C).

ColorExtension.fromARGBHexString

: Parses an ARGB color from a valid hex string (e.g. #FF1C1C1C).

Image

¶

Methods:

pixelsInUint8

: Retrieves the pixel data as a

Uint8List

, in the

ImageByteFormat.rawRgba

pixel format, for the image.

getBoundingRect

: Get the bounding rectangle of the

Image

as a

Rect

.

size

: The size of an

Image

as

Vector2

.

darken

: Darken each pixel of the

Image

by an amount between 0 to 1.

brighten

: Brighten each pixel of the

Image

by an amount between 0 to 1.

Offset

¶

Methods;

toVector2

; Creates an

Vector2

from the

Offset

.

toSize

: Creates a

Size

from the

Offset

.

toPoint

: Creates a

Point

from the

Offset

.

toRect

: Creates a

Rect

starting in (0,0) and its bottom right corner is the [Offset].

Rect

¶

Methods:

toOffset

: Creates an

Offset

from the

Rect

.

toVector2

: Creates a

Vector2

starting in (0,0) and goes to the size of the

Rect

.

containsPoint

Whether this

Rect

contains a

Vector2

point or not.

intersectsSegment

; Whether the segment formed by two

Vector2

s intersects this

Rect

.

intersectsLineSegment

: Whether the

LineSegment

intersects the

Rect

.

toVertices

: Turns the four corners of the

Rect

into a list of

Vector2

.

toFlameRectangle

: Converts this

Rect

into a Flame

Rectangle

.

toMathRectangle

: Converts this

Rect

into a

math.Rectangle

.

toGeometryRectangle

: Converts this

Rect

into a

Rectangle

from flame-geom.

transform

: Transforms the

Rect

using a

Matrix4

.

Factories:

RectExtension.getBounds

: Construct a

Rect

that represents the bounds of a list of

Vector2

s.

RectExtension.fromCenter

: Construct a

Rect

from a center point (using

Vector2

).

math.Rectangle

¶

Methods:

toRect

: Converts this math

Rectangle

into an ui

Rect

.

Size

¶

Methods:

toVector2

; Creates an

Vector2

from the

Size

.

toOffset

: Creates a

Offset

from the

Size

.

toPoint

: Creates a

Point

from the

Size

.

toRect

: Creates a

Rect

starting in (0,0) with the size of

Size

.

Vector2

¶

This class comes from the

vector_math

package and we have some useful extension methods on top of what is offered by that package.

Methods:

toOffset

: Creates a

Offset

from the

Vector2

.

toPoint

: Creates a

Point

from the

Vector2

.

toRect

: Creates a

Rect

starting in (0,0) with the size of

Vector2

.

toPositionedRect

: Creates a

Rect

starting from [x, y] in the

Vector2

and has the size of the

Vector2

argument.

lerp

: Linearly interpolates the

Vector2

towards another Vector2.

rotate

: Rotates the

Vector2

with an angle specified in radians, it rotates around the optionally defined

Vector2

, otherwise around the center.

scaleTo

: Changes the length of the

Vector2

to the length provided, without changing direction.

moveToTarget

: Smoothly moves a Vector2 in the target direction by a given distance.

Factories:

Vector2Extension.fromInts

: Create a

Vector2

with ints as input.

Operators:

&

: Combines two

Vector2

s to form a Rect, the origin should be on the left and the size on the right.

%

: Modulo/Remainder of x and y separately of two

Vector2

s.

Matrix4

¶

This class comes from the

vector_math

package. We have created a few extension methods on top of what is already offered by

vector_math

.

Methods:

translate2

: Translate the

Matrix4

by the given

Vector2

.

transform2

: Create a new

Vector2

by transforming the given

Vector2

using the

Matrix4

.

transformed2

: Transform the input

Vector2

into the output

Vector2

.

Getters:

m11

: The first row and first column.

m12

: The first row and second column.

m13

: The first row and third column.

m14

: The first row and fourth column.

m21

: The second row and first column.

m22

: The second row and second column.

m23

: The second row and third column.

m24

: The second row and fourth column.

m31

: The third row and first column.

m32

: The third row and second column.

m33

: The third row and third column.

m34

: The third row and fourth column.

m41

: The fourth row and first column.

m42

: The fourth row and second column.

m43

: The fourth row and third column.

m44

: The fourth row and fourth column.

Factories:

Matrix4Extension.scale

: Create a scaled

Matrix4

. Either by passing a

Vector4

or

Vector2

as it?s first argument, or by passing x y z doubles.

# 4. Gameplay

¶

In this chapter we will be implementing the core of Klondike?s gameplay: how the cards move between the

Before we begin though, let?s clean up all those cards that we left scattered across the table in the previou

KlondikeGame

class and erase the loop at the bottom of

onLoad()

that was adding 28 cards onto the table.

## The piles

¶

Another small refactoring that we need to do is to rename our components:

Stock

?

StockPile

,

Waste

?

WastePile

,

Foundation

?

FoundationPile

, and

Pile

?

TableauPile

. This is because these components have some common features in how they handle interactions with the

Pile

class.

Note

Refactors and changes in architecture happen during development all the time: it?s almost impossible to g

After such a rename, we can begin implementing each of these components.

Stock pile

¶

The

stock

is a place in the top-left corner of the playing field which holds the cards that are not currently in play. We w

Ability to hold cards that are not currently in play, face down;

Tapping the stock should reveal top 3 cards and move them to the

waste

pile;

When the cards run out, there should be a visual indicating that this is the stock pile;

When the cards run out, tapping the empty stock should move all the cards from the waste pile into the sto

The first question that needs to be decided here is this: who is going to own the

Card

components? Previously we have been adding them directly to the game field, but now wouldn?t it be bette

Stock

component, or to the waste, or piles, or foundations? While this approach is tempting, I believe it would ma

So, I decided to stick with my first approach: the

Card

components are owned directly by the

KlondikeGame

itself, whereas the

StockPile

and other piles are merely aware of which cards are currently placed there.

Having this in mind, let?s start implementing the

StockPile

component:

class

StockPile

extends

PositionComponent

{

StockPile

({

super

.

position

})

:

super

(

size:

KlondikeGame

.

cardSize

);

/// Which cards are currently placed onto this pile. The first card in the

/// list is at the bottom, the last card is on top.

final

List

<

Card

>

_cards

=

[];

void

acquireCard

(

Card

card

)

{

assert

(

!

card

.

isFaceUp

);

card

.

position

=

position

;

card

.

priority

=

_cards

.

length

;

_cards

.

add

(

card

);

}

}

Here the

acquireCard()

method stores the provided card into the internal list

_cards

; it also moves that card to the

StockPile

?s position and adjusts the cards priority so that they are displayed in the right order. However, this metho

StockPile

component ? it remains belonging to the top-level game.

Speaking of the game class, let?s open the

KlondikeGame

and add the following lines to create a full deck of 52 cards and put them onto the stock pile (this should be

onLoad

method):

final

cards

=

[

for

(

var

rank

=

1

;

rank

<=

13

;

rank

++

)

for

(

var

suit

=

0

;

suit

<

4

;

suit

++

)

Card

(

rank

,

suit

)

];

world

.

addAll

(

cards

);

cards

.

forEach

(

stock

.

acquireCard

);

This concludes the first step of our short plan at the beginning of this section. For the second step, though,

WastePile

class.

Waste pile

¶

The

waste

is a pile next to the stock. During the course of the game we will be taking the cards from the top of the sto

Let?s start implementing the

WastePile

class same way as we did with the

StockPile

class, only now the cards are expected to be face up:

class

WastePile

extends

PositionComponent

{

WastePile

({

super

.

position

})

:

super

(

size:

KlondikeGame

.

cardSize

);

final

List

<

Card

>

_cards

=

[];

void

acquireCard

(

Card

card

)

{

assert

(

card

.

isFaceUp

);

card

.

position

=

position

;

card

.

priority

=

_cards

.

length

;

_cards

.

add

(

card

);

}

}

So far, this puts all cards into a single neat pile, whereas we wanted a fan-out of top three. So, let?s add a

_fanOutTopCards()

for this, which we will call at the end of each

acquireCard()

:

void

_fanOutTopCards

()

{

final

n

=

_cards

.

length

;

for

(

var

i

=

0

;

i

<

n

;

i

++

)

{

_cards

[

i

].

position

=

position

;

}

if

(

n

==

2

)

{

_cards

[

1

].

position

.

add

(

_fanOffset

);

}

else

if

(

n

>=

3

)

{

_cards

[

n

-

2

].

position

.

add

(

_fanOffset

);

_cards

[

n

-

1

].

position

.

addScaled

(

_fanOffset

,

2

);

}

}

The

_fanOffset

variable here helps determine the shift between cards in the fan, which I decided to be about 20% of the ca

final

Vector2

_fanOffset

=

Vector2

(

KlondikeGame

.

cardWidth

*

0.2

,

0

);

Now that the waste pile is ready, let?s get back to the

StockPile

.

Stock pile ? tap to deal cards

¶

The second item on our todo list is the first interactive functionality in the game: tap the stock pile to deal 3

Adding tap functionality to the components in Flame is quite simple: we just add the mixin

TapCallbacks

to the component that we want to be tappable:

class

StockPile

extends

PositionComponent

with

TapCallbacks

{

...

}

Oh, and we also need to say what we want to happen when the tap occurs. Here we want the top 3 cards t

StockPile

class:

@override

void

onTapUp

(

TapUpEvent

event

)

{

final

wastePile

=

parent

!

.

firstChild

<

WastePile

>

()

!

;

for

(

var

i

=

0

;

i

<

3

;

i

++

)

{

if

(

_cards

.

isNotEmpty

)

{

final

card

=

_cards

.

removeLast

();

card

.

flip

();

wastePile

.

acquireCard

(

card

);

}

}

}

You have probably noticed that the cards move from one pile to another immediately, which looks very unn

Also, the cards are organized in a well-defined order right now, starting from Kings and ending with Aces. T

cards

.

shuffle

();

in the

KlondikeGame

class right after the list of cards is created.

See also

For more information about tap functionality, see

Tap Events

.

Stock pile ? visual representation

¶

Currently, when the stock pile has no cards, it simply shows an empty space ? there is no visual cue that th

In our case, the empty stock pile will have a card-like border, and a circle in the middle:

@override

void

render

(

Canvas

canvas

)

{

canvas

.

drawRRect

(

KlondikeGame

.

cardRRect

,

_borderPaint

);

canvas

.

drawCircle

(

Offset

(

width

/

2

,

height

/

2

),

KlondikeGame

.

cardWidth

*

0.3

,

_circlePaint

,

);

}

where the paints are defined as

final

_borderPaint

=

Paint

()

..

style

=

PaintingStyle

.

stroke

..

strokeWidth

=

10

..

color

=

const

Color

(

0xFF3F5B5D

);

final

_circlePaint

=

Paint

()

..

style

=

PaintingStyle

.

stroke

..

strokeWidth

=

100

..

color

=

const

Color

(

0x883F5B5D

);

and the

cardRRect

in the

KlondikeGame

class as

static

final

cardRRect

=

RRect

.

fromRectAndRadius

(

const

Rect

.

fromLTWH

(

0

,

0

,

cardWidth

,

cardHeight

),

const

Radius

.

circular

(

cardRadius

),

);

Now when you click through the stock pile till the end, you should be able to see the placeholder for the sto

Stock pile ? refill from the waste

¶

The last piece of functionality to add, is to move the cards back from the waste pile into the stock pile when

onTapUp()

method like so:

@override

void

onTapUp

(

TapUpEvent

event

)

{

final

wastePile

=

parent

!

.

firstChild

<

WastePile

>

()

!

;

if

(

_cards

.

isEmpty

)

{

wastePile

.

removeAllCards

().

reversed

.

forEach

((

card

)

{

card

.

flip

();

acquireCard

(

card

);

});

}

else

{

for

(

var

i

=

0

;

i

<

3

;

i

++

)

{

if

(

_cards

.

```
isNotEmpty
)
{
final
card
=
_cards
.
removeLast
();
card
.
flip
();
wastePile
.
acquireCard
(
card
);
}
}
}
}
```

If you?re curious why we needed to reverse the list of cards removed from the waste pile, then it is becaus

The method

WastePile.removeAllCards()

still needs to be implemented though:

List

<

Card

>

removeAllCards

()

{

final

cards

=

_cards

.

toList

();

_cards

.

clear

();

return

cards

;

}

This pretty much concludes the

StockPile

functionality, and we already implemented the

WastePile

? so the only two components remaining are the

FoundationPile

and the

TableauPile

. We?ll start with the first one because it looks simpler.

Foundation piles

¶

The

foundation

piles are the four piles in the top right corner of the game. This is where we will be building the ordered runs

StockPile

and the

WastePile

: it has to be able to hold cards face up, and there has to be some visual to show where the foundation is w

First, let?s implement the card-holding logic:

class

FoundationPile

extends

PositionComponent

{

FoundationPile

({

super

.

position

})

:

super

(

size:

KlondikeGame

.

cardSize

);

final

List

<

Card

>

_cards

=

[];

void

acquireCard

(

Card

card

)

{

assert

(

card

.

isFaceUp

);

card

.

position

=

position

;

card

.

priority

=

_cards

.

length

;

_cards

.

add

(

card

);

}

}

For visual representation of a foundation, I?ve decided to make a large icon of that foundation?s suit, in gre

class

FoundationPile

extends

PositionComponent

{

FoundationPile

(

int

intSuit

,

{

super

.

position

})

:

suit

=

Suit

.

fromInt

(

intSuit

),

super

(

size:

KlondikeGame

.

cardSize

);

final

Suit

suit

;

...

}

The code in the

KlondikeGame

class that generates the foundations will have to be adjusted accordingly in order to pass the suit index to e

Now, the rendering code for the foundation pile will look like this:

@override

void

render

(

Canvas

canvas

)

{

canvas

.

drawRRect

(

KlondikeGame

.

cardRRect

,

_borderPaint

);

suit

.

sprite

.

render

(

canvas

,

position:

size

/

2

,

anchor:

Anchor

.

center

,

size:

Vector2

.

all

(

KlondikeGame

.

cardWidth

*

0.6

),

overridePaint:

_suitPaint

,

);

}

Here we need to have two paint objects, one for the border and one for the suits:

final

_borderPaint

=

Paint

()

..

style

=

PaintingStyle

.

stroke

..

strokeWidth

=

10

..

color

=

const

Color

(

0x50ffffff

);

late

final

_suitPaint

=

Paint

()

..

color

=

suit

.

isRed

?

const

Color

(

0x3a000000

)

:

const

Color

(

0x64000000

)

..

blendMode

=

BlendMode

.

luminosity

;

The suit paint uses

BlendMode.luminosity

in order to convert the regular yellow/blue colors of the suit sprites into grayscale. The ?color? of the paint i

Tableau Piles

¶

The last piece of the game to be implemented is the

TableauPile

component. There are seven of these piles in total, and they are where the majority of the game play is hap

The

TableauPile

also needs a visual representation, in order to indicate that it?s a place where a King can be placed when i

class

TableauPile

extends

PositionComponent

{

TableauPile

({

super

.

position

})

:

super

(

size:

KlondikeGame

.

cardSize

);

final

_borderPaint

=

Paint

()

..

style

=

PaintingStyle

.

stroke

..

strokeWidth

=

10

..

color

=

const

Color

(

0x50ffffff

);

@override

void

render

(

Canvas

canvas

)

{

canvas

.

drawRRect

(

KlondikeGame

.

cardRRect

,

_borderPaint

);

}

}

Oh, and the class will need to be able hold the cards too, obviously. Here, some of the cards will be face d

WastePile

component:

/// Which cards are currently placed onto this pile.

final

List

<

Card

>

_cards

=

[];

final

Vector2

_fanOffset

=

Vector2

(

0

,

KlondikeGame

.

cardHeight

*

0.05

);

void

acquireCard

(

Card

card

)

{

if

(

_cards

.

isEmpty

)

{

card

.

position

=

position

;

}

else

{

card

.

position

=

_cards

.

last

.

position

+

_fanOffset

;

}

card

.

priority

=

_cards

.

length

;

_cards

.

add

(

card

);

}

All that remains now is to head over to the

KlondikeGame

and make sure that the cards are dealt into the

TableauPile

s at the beginning of the game. Modify the code at the end of the

onLoad()

method so that it looks like this:

@override

Future

<

void

>

onLoad

()

async

{

...

final

cards

=

[

for

(

var

rank

=

1

;

rank

<=

13

;

rank

++

)

for

(

var

suit

=

0

;

suit

<

4

;

suit

++

)

Card

(

rank

,

suit

)

];

cards

.

shuffle

();

world

.

addAll

(

cards

);

int

cardToDeal

=

cards

.

length

-

1

;

for

(

var

i

=

0

;

i

<

7

;

i

++

)

{

```
for
(
var
j
=
i
;
j
<
7
;
j
++
)
{
piles
[
j
].
acquireCard
(
cards
[
cardToDeal
--
]);
```

}

piles

[

i

].

flipTopCard

();

}

for

(

int

n

=

0

;

n

<=

cardToDeal

;

n

++

)

{

stock

.

acquireCard

(

cards

[

n

]);

}

}

Note how we deal the cards from the deck and place them into

TableauPile

s one by one, and only after that we put the remaining cards into the stock.

Recall that we decided earlier that all the cards would be owned by the

KlondikeGame

itself. So they are put into a generated List structure called

cards

, shuffled and added to the

world

. This List should always have 52 cards in it, so a descending index

cardToDeal

is used to deal 28 cards one by one from the top of the deck into piles that acquire references to the cards

Card

objects in the

cards

list. In the card piles we used

removeList()

to retrieve a card from a pile, but not here because it would remove cards from

KlondikeGame

?s ownership.

The

flipTopCard

method in the

TableauPile

class is as trivial as it sounds:

void

flipTopCard

()

{

assert

(

_cards

.

last

.

isFaceDown

);

_cards

.

last

.

flip

();

}

If you run the game at this point, it would be nicely set up and look as if it was ready to play. Except that we

Moving the cards

¶

Moving the cards is a somewhat more complicated topic than what we have had so far. We will split it into s

Simple movement: grab a card and move it around.

Ensure that the user can only move the cards that they are allowed to.

Check that the cards are dropped at proper destinations.

Drag a run of cards.

1. Simple movement

¶

So, we want to be able to drag the cards on the screen. This is even simpler than making the

StockPile

tappable: just head over into the

Card

class and add the

DragCallbacks

mixin:

```
class
Card
extends
PositionComponent
with
DragCallbacks
{
}
```

The next step is to implement the actual drag event callbacks:

onDragStart

,

onDragUpdate

, and

onDragEnd

.

When the drag gesture is initiated, the first thing that we need to do is to raise the priority of the card, so tha

@override

void

onDragStart

(

DragStartEvent

event

)

{

priority

=

100

;

}

During the drag, the

onDragUpdate

event will be called continuously. Using this callback we will be updating the position of the card so that it fo

event

object passed to this callback contains the most recent coordinate of the point of touch, and also the

localDelta

property ? which is the displacement vector since the previous call of

onDragUpdate

, considering the camera zoom.

@override

void

onDragUpdate

(

DragUpdateEvent

event

)

{

position

+=

event

.

delta

;

}

So far this allows you to grab any card and drag it anywhere around the table. What we want, however, is t

2. Move only allowed cards

¶

The first restriction that we impose is that the user should only be able to drag the cards that we allow, whic

Thus, in order to determine whether a card can be moved or not, we need to know which pile it currently be

So, let?s start by defining the abstract interface

Pile

that all our existing piles will be implementing:

abstract

class

Pile

{

bool

canMoveCard

(

Card

card

);

}

We will expand this class further later, but for now let?s make sure that each of the classes

StockPile

,

WastePile

,

FoundationPile

, and

TableauPile

are marked as implementing this interface:

class

StockPile

extends

PositionComponent

with

TapCallbacks

implements

```
Pile
{
...
@override
bool
canMoveCard
(
Card
card
)
=>
false
;
}
class
WastePile
extends
PositionComponent
implements
Pile
{
...
@override
bool
canMoveCard
(
```

Card

card

)

=>

_cards

.

isNotEmpty

&&

card

==

_cards

.

last

;

}

class

FoundationPile

extends

PositionComponent

implements

Pile

{

...

@override

bool

canMoveCard

(

Card

card

)

=>

_cards

.

isNotEmpty

&&

card

==

_cards

.

last

;

}

class

TableauPile

extends

PositionComponent

implements

Pile

{

...

@override

bool

canMoveCard

(

Card

card

)

=>

_cards

.

isNotEmpty

&&

card

==

_cards

.

last

;

}

We also wanted to let every

Card

know which pile it is currently in. For this, add the field

Pile?

pile

into the

Card

class, and make sure to set it in each pile?s

acquireCard()

method, like so:

void

acquireCard

(

Card

card

)

{

...

card

.

pile

=

this

;

}

Now we can put this new functionality to use: go into the

Card.onDragStart()

method and modify it so that it would check whether the card is allowed to be moved before starting the dra

void

onDragStart

(

DragStartEvent

event

)

{

```
if
(
pile
?
.
canMoveCard
(
this
)
??
false
)
{
super
.
onDragStart
(
event
);
priority
=
100
;
}
}
```

We have also added a call to

super.onDragStart()

which sets an

_isDragged

variable to

true

in the

DragCallbacks

mixin, we need to check this flag via the public

isDragged

getter in the

onDragUpdate()

method and use

super.onDragEnd()

in

onDragEnd()

so the flag is set back to

false

:

@override

void

onDragUpdate

(

DragUpdateEvent

event

)

{

if

(

!

isDragged

)

{

return

;

}

position

+=

event

.

delta

;

}

@override

void

onDragEnd

(

DragEndEvent

event

)

{

super

.

onDragEnd

(

event

);

}

Now only the proper cards can be dragged, but they still drop at random positions on the table, so let?s wo

3. Dropping the cards at proper locations

¶

At this point what we want to do is to figure out where the dragged card is being dropped. More specifically

pile

it is being dropped. This can be achieved by using the

componentsAtPoint()

API, which allows you to query which components are located at a given position on the screen.

Thus, my first attempt at revising the

onDragEnd

callback looks like this:

@override

void

onDragEnd

(

DragEndEvent

event

)

{

if

(

!

isDragged

)

{

return

;

}

super

.

onDragEnd

(

event

);

final

dropPiles

=

parent

!

.

componentsAtPoint

(

position

+

size

/

2

```
)
.
whereType
<
Pile
>
()
.
toList
();
if
(
dropPiles
.
isNotEmpty
)
{
// if (card is allowed to be dropped into this pile) {
//   remove the card from the current pile
//   add the card into the new pile
// }
}
// return the card to where it was originally
}
```

This still contains several placeholders for the functionality that still needs to be implemented, so let?s get t

First piece of the puzzle is the ?is card allowed to be dropped here?? check. To implement this, first head o

Pile

class and add the

canAcceptCard()

abstract method:

abstract

class

Pile

{

...

bool

canAcceptCard

(

Card

card

);

}

Obviously this now needs to be implemented for every

Pile

subclass, so let?s get to it:

class

FoundationPile

...

implements

Pile

{

...

@override

bool

canAcceptCard

(

Card

card

)

{

final

topCardRank

=

_cards

.

isEmpty

?

0

:

_cards

.

last

.

rank

.

value

;

return

card

.

suit

==

suit

&&

card

.

rank

.

value

==

topCardRank

+

1

;

}

}

class

TableauPile

...

implements

Pile

{

...

@override

```
bool
canAcceptCard
(
Card
card
)
{
if
(
_cards
.
isEmpty
)
{
return
card
.
rank
.
value
==
13
;
}
else
{
```

final

topCard

=

_cards

.

last

;

return

card

.

suit

.

isRed

==

!

topCard

.

suit

.

isRed

&&

card

.

rank

.

value

==

topCard

.

rank

.

value

-

1

;

}

}

}

(for the

StockPile

and the

WastePile

the method should just return false, since no cards should be dropped there).

Alright, next part is the ?remove the card from its current pile?. Once again, let?s head over to the

Pile

class and add the

removeCard()

abstract method:

abstract

class

Pile

{

...

void

removeCard

(

Card

card

);

}

Then we need to re-visit all four pile subclasses and implement this method:

class

StockPile

...

implements

Pile

{

...

@override

void

removeCard

(

Card

card

)

=>

throw

StateError

(

'cannot remove cards from here'

);

}

class

WastePile

...

implements

Pile

{

...

@override

void

removeCard

(

Card

card

)

{

assert

(

canMoveCard

(

card

));

_cards

.

removeLast

();

_fanOutTopCards

();

}

}

class

FoundationPile

...

implements

Pile

{

...

@override

void

removeCard

(

Card

card

)

{

assert

(

canMoveCard

(

card

));

_cards

.

removeLast

();

}

}

class

TableauPile

...

implements

Pile

{

...

@override

void

removeCard

(

Card

card

)

{

assert

(

_cards

.

contains

(

card

)

&&

card

.

isFaceUp

);

final

index

=

_cards

.

indexOf

(

card

);

_cards

.

removeRange

(

index

,

_cards

.

length

);

if

(

_cards

.

isNotEmpty

&&

_cards

.

last

.

isFaceDown

)

{

flipTopCard

();

}

}

}

The next action in our pseudo-code is to ?add the card to the new pile?. But this one we have already impl

acquireCard()

method. So all we need is to declare it in the

Pile

interface:

abstract

class

Pile

{

...

void

acquireCard

(

Card

card

);

}

The last piece that?s missing is ?return the card to where it was?. You can probably guess how we are goi

returnCard()

method into the

Pile

interface, and then implement this method in all four pile subclasses:

class

StockPile

...

implements

Pile

{

...

@override

void

returnCard

(

Card

card

)

=>

throw

StateError

(

'cannot remove cards from here'

);

}

class

WastePile

...

implements

Pile

{

...

@override

void

returnCard

(

Card

card

)

{

card

.

priority

=

_cards

.

indexOf

(

card

);

_fanOutTopCards

();

}

}

class

FoundationPile

...

implements

Pile

{

...

@override

void

returnCard

(

Card

card

)

{

card

.

position

=

position

;

card

.

priority

=

_cards

.

indexOf

(

card

);

}

}

class

TableauPile

...

implements

```
Pile
{
...
@override
void
returnCard
(
Card
card
)
{
final
index
=
_cards
.
indexOf
(
card
);
card
.
position
=
index
==
```

0

?

position

:

_cards

[

index

-

1

].

position

+

_fanOffset

;

card

.

priority

=

index

;

}

}

Now, putting this all together, the

Card

?s

onDragEnd

method will look like this:

```
@override
void
onDragEnd
(
DragEndEvent
event
)
{
if
(
!
isDragged
)
{
return
;
}
super
.
onDragEnd
(
event
);
final
dropPiles
```

=

parent

!

.

componentsAtPoint

(

position

+

size

/

2

)

.

whereType

<

Pile

>

()

.

toList

();

if

(

dropPiles

.

isNotEmpty

)

{

if

(

dropPiles

.

first

.

canAcceptCard

(

this

))

{

pile

!

.

removeCard

(

this

);

dropPiles

.

first

.

acquireCard

(

this

);

return

;

}

}

pile

!

.

returnCard

(

this

);

}

Ok, that was quite a lot of work ? but if you run the game now, you?d be able to move the cards properly fr

4. Moving a run of cards

¶

In this section we will be implementing the necessary changes to allow us to move small stacks of cards be

You have probably noticed when running the game in the previous section that the cards in the tableau pile

So, let?s head over into the

TableauPile

class and create a new method

layOutCards()

, whose job would be to ensure that all cards currently in the pile have the right positions:

final

Vector2

_fanOffset1

=

Vector2

(

0

,

KlondikeGame

.

cardHeight

*

0.05

);

final

Vector2

_fanOffset2

=

Vector2

(

0

,

KlondikeGame

.

cardHeight

*

0.20

);

```
void
layOutCards
()
{
if
(
_cards
.
isEmpty
)
{
return
;
}
_cards
[
0
].
position
.
setFrom
(
position
);
for
(
```

var

i

=

1

;

i

<

_cards

.

length

;

i

++

)

{

_cards

[

i

].

position

..

setFrom

(

_cards

[

i

-

1

].

position

)

..

add

(

_cards

[

i

-

1

].

isFaceDown

?

_fanOffset1

:

_fanOffset2

);

}

}

Make sure to call this method at the end of

removeCard()

,

returnCard()

, and

acquireCard()

? replacing any current logic that handles card positioning.

Another problem that you may have noticed is that for taller card stacks it becomes hard to place a card the

TableauPile

components ? but those components have only the size of a single card! To fix this inconsistency, all we ne

layOutCards()

method:

height

=

KlondikeGame

.

cardHeight

*

1.5

+

_cards

.

last

.

y

-

_cards

.

first

.

y

;

The factor

1.5

here adds a little bit extra space at the bottom of each pile. The card to be dropped should be overlapping t

Ok, let?s get to our main topic: how to move a stack of cards at once.

First thing that we?re going to add is the list of

attachedCards

for every card. This list will be non-empty only when the card is being dragged while having other cards on

Card

class:

final

List

<

Card

>

attachedCards

=

[];

Now, in order to create this list in

onDragStart

, we need to query the

TableauPile

for the list of cards that are on top of the given card. Let?s add such a method into the

TableauPile

class:

List

<

Card

>

cardsOnTop

(

Card

card

)

{

assert

(

card

.

isFaceUp

&&

_cards

.

contains

(

card

));

final

index

=

_cards

.

indexOf

(

card

);

return

_cards

.

getRange

(

index

+

1

,

_cards

.

length

).

toList

();

}

While we are in the

TableauPile

class, let?s also update the

canMoveCard()

method to allow dragging cards that are not necessarily on top:

@override

bool

canMoveCard

(

Card

card

)

=>

card

.

isFaceUp

;

Heading back into the

Card

class, we can use this method in order to populate the list of

attachedCards

when the card starts to move:

@override

void

onDragStart

(

DragStartEvent

event

)

{

if

(

pile

?

.

canMoveCard

(

this

)

??

false

)

{

super

.

onDragStart

();

priority

=

100

;

if

(

pile

is

TableauPile

)

{

attachedCards

.

clear

();

final

extraCards

=

(

pile

!

as

TableauPile

).

cardsOnTop

(

this

);

for

(

final

card

in

extraCards

)

{

card

.

priority

=

attachedCards

.

length

+

101

;

attachedCards

.

add

(

card

);

}

}

}

}

Now all we need to do is to make sure that the attached cards are also moved with the main card in the

onDragUpdate

method:

@override

void

onDragUpdate

(

DragUpdateEvent

event

)

{

if

(

!

isDragged

)

{

return

;

}

final

delta

=

event

.

delta

;

position

.

add

(

delta

);

attachedCards

.

forEach

((

card

)

=>

card

.

position

.

add

(

delta

));

}

This does the trick, almost. All that remains is to fix any loose ends. For example, we don?t want to let the

FoundationPile

class and modify the

canAcceptCard()

method accordingly:

@override

bool

canAcceptCard

(

Card

card

)

{

final

topCardRank

=

_cards

.

isEmpty

?

0

:

_cards

.

last

.

rank

.

value

;

return

card

.

suit

==

suit

&&

card

.

rank

.

value

==

topCardRank

+

1

&&

card

.

attachedCards

.

isEmpty

;

}

Secondly, we need to properly take care of the stack of card as it is being dropped into a tableau pile. So, g

Card

class and update its

onDragEnd()

method to also move the attached cards into the pile, and the same when it comes to returning the cards in

@override

void

onDragEnd

(

DragEndEvent

event

)

{

if

(

!

isDragged

)

{

return

;

}

super

.

onDragEnd

(

event

);

final

dropPiles

=

parent

!

.

componentsAtPoint

(

position

+

size

/

2

)

.

whereType

<

Pile

>

()

.

toList

();

if

(

dropPiles

.

isNotEmpty

)

{

if

(

dropPiles

.

first

.

canAcceptCard

(

this

))

{

pile

!

.

removeCard

(

this

);

dropPiles

.

first

.

acquireCard

(

this

);

if

```
(
attachedCards
.
isNotEmpty
)
{
attachedCards
.
forEach
((
card
)
=>
dropPiles
.
first
.
acquireCard
(
card
));
attachedCards
.
clear
();
}
```

return

;

}

}

pile

!

.

returnCard

(

this

);

if

(

attachedCards

.

isNotEmpty

)

{

attachedCards

.

forEach

((

card

)

=>

pile

!

.

returnCard

(

card

));

attachedCards

.

clear

();

}

}

Well, this is it! The game is now fully playable. Press the button below to see what the resulting code looks

Run

components/card.dart

1

import

'dart:math'

;

2

import

'dart:ui'

;

3

4

import

'package:flame/components.dart'

;

5

import

'package:flame/events.dart'

;

6

import

'../klondike_game.dart'

;

7

import

'../pile.dart'

;

8

import

'../rank.dart'

;

9

import

'../suit.dart'

;

10

import

'tableau_pile.dart'

;

11

12

class

Card

extends

PositionComponent

with

DragCallbacks

{

13

Card

(

int

intRank

,

int

intSuit

)

14

:

rank

=

Rank

.

fromInt

(

intRank

),

15

suit

=

Suit

.

fromInt

(

intSuit

),

16

super

(

size:

KlondikeGame

.

cardSize

);

17

18

final

Rank

rank

;

19

final

Suit

suit

;

20

Pile

?

pile

;

21

bool

_faceUp

=

false

;

22

bool

_isDragging

=

false

;

23

final

List

<

Card

>

attachedCards

=

[];

24

25

bool

get

isFaceUp

=>

_faceUp

;

26

bool

get

isFaceDown

=>

!

_faceUp

;

27

void

flip

()

=>

_faceUp

=

!

_faceUp

;

28

29

@override

30

String

toString

()

=>

rank

.

label

+

suit

.

label

;

// e.g. "Q?" or "10?"

31

32

//#region Rendering

33

34

```
@override
35
void
render
(
Canvas
canvas
)
{
36
if
(
_faceUp
)
{
37
_renderFront
(
canvas
);
38
}
else
{
39
_renderBack
```

(

canvas

);

40

}

41

}

42

43

static

final

Paint

backBackgroundPaint

=

Paint

()

44

..

color

=

const

Color

(

0xff380c02

);

45

static

final

Paint

backBorderPaint1

=

Paint

()

46

..

color

=

const

Color

(

0xffdbaf58

)

47

..

style

=

PaintingStyle

.

stroke

48

..

strokeWidth

=

10

;

49

static

final

Paint

backBorderPaint2

=

Paint

()

50

..

color

=

const

Color

(

0x5CEF971B

)

51

..

style

=

PaintingStyle

.

stroke

52

..

strokeWidth

=

35

;

53

static

final

RRect

cardRRect

=

RRect

.

fromRectAndRadius

(

54

KlondikeGame

.

cardSize

.

toRect

(),

55

const

Radius

.

circular

(

KlondikeGame

.

cardRadius

),

56

);

57

static

final

RRect

backRRectInner

=

cardRRect

.

deflate

(

40

);

58

static

final

Sprite

flameSprite

=

klondikeSprite

(

1367

,

6

,

357

,

501

);

59

60

void

_renderBack

(

Canvas

canvas

)

{

61

canvas

.

drawRRect

(

cardRRect

,

backBackgroundPaint

);

62

canvas

.

drawRRect

(

cardRRect

,

backBorderPaint1

);

63

canvas

.

drawRRect

(

backRRectInner

,

backBorderPaint2

);

64

flameSprite

.

render

(

canvas

,

position:

size

/

2

,

anchor:

Anchor

.

center

);

65

}

66

67

static

final

Paint

frontBackgroundPaint

=

Paint

()

68

..

color

=

const

Color

(

0xff000000

);

69

static

final

Paint

redBorderPaint

=

Paint

()

70

..

color

=

const

Color

(

0xffece8a3

)

71

..

style

=

PaintingStyle

.

stroke

72

..

strokeWidth

=

10

;

73

static

final

Paint

blackBorderPaint

=

Paint

()

74

..

color

=

const

Color

(

0xff7ab2e8

)

75

..

style

=

PaintingStyle

.

stroke

76

..

strokeWidth

=

10

;

77

static

final

blueFilter

=

Paint

()

78

..

colorFilter

=

```
const
ColorFilter
.
mode
(
79
Color
(
0x880d8bff
),
80
BlendMode
.
srcATop
,
81
);
82
static
final
Sprite
redJack
=
klondikeSprite
(
81
```

,

565

,

562

,

488

);

83

static

final

Sprite

redQueen

=

klondikeSprite

(

717

,

541

,

486

,

515

);

84

static

final

Sprite

redKing

=

klondikeSprite

(

1305

,

532

,

407

,

549

);

85

static

final

Sprite

blackJack

=

klondikeSprite

(

81

,

565

,

562

,

488

)

86

..

paint

=

blueFilter

;

87

static

final

Sprite

blackQueen

=

klondikeSprite

(

717

,

541

,

486

,

515

)

88

..

paint

=

blueFilter

;

89

static

final

Sprite

blackKing

=

klondikeSprite

(

1305

,

532

,

407

,

549

)

90

..

paint

=

blueFilter

;

91

92

void

_renderFront

(

Canvas

canvas

)

{

93

canvas

.

drawRRect

(

cardRRect

,

frontBackgroundPaint

);

94

canvas

.

drawRRect

(

95

cardRRect

,

96

suit

.

isRed

?

redBorderPaint

:

blackBorderPaint

,

97

);

98

99

final

rankSprite

=

suit

.

isBlack

?

rank

.

blackSprite

:

rank

.

redSprite

;

100

final

suitSprite

=

suit

.

sprite

;

101

_drawSprite

(

canvas

,

rankSprite

,

0.1

,

0.08

);

102

_drawSprite

(

canvas

,

suitSprite

,

0.1

,

0.18

,

scale:

0.5

);

103

_drawSprite

(

canvas

,

rankSprite

,

0.1

,

0.08

,

rotate:

true

);

104

_drawSprite

(

canvas

,

suitSprite

,

0.1

,

0.18

,

scale:

0.5

,

rotate:

true

);

105

switch

(

rank

.

value

)

{

106

case

1

:

107

_drawSprite

(

canvas

,

suitSprite

,

0.5

,

0.5

,

scale:

2.5

);

108

break

;

109

case

2

:

110

_drawSprite

(

canvas

,

suitSprite

,

0.5

,

0.25

);

111

_drawSprite

(

canvas

,

suitSprite

,

0.5

,

0.25

,

rotate:

true

);

112

break

;

113

case

3

:

114

_drawSprite

(

canvas

,

suitSprite

,

0.5

,

0.2

);

115

_drawSprite

(

canvas

,

suitSprite

,

0.5

,

0.5

);

116

_drawSprite

(

canvas

,

suitSprite

,

0.5

,

0.2

,

rotate:

true

);

117

break

;

118

case

4

:

119

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.25

);

120

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.25

);

121

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.25

,

rotate:

true

);

122

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.25

,

rotate:

true

);

123

break

;

124

case

5

:

125

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.25

);

126

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.25

);

127

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.25

,

rotate:

true

);

128

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.25

,

rotate:

true

);

129

_drawSprite

(

canvas

,

suitSprite

,

0.5

,

0.5

);

130

break

;

131

case

6

:

132

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.25

);

133

```
_drawSprite
(
canvas
,
suitSprite
,
0.7
,
0.25
);
134
_drawSprite
(
canvas
,
suitSprite
,
0.3
,
0.5
);
135
_drawSprite
(
canvas
,
```

suitSprite

,

0.7

,

0.5

);

136

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.25

,

rotate:

true

);

137

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.25

,

rotate:

true

);

138

break

;

139

case

7

:

140

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.2

);

```
141
_drawSprite
(
canvas
,
suitSprite
,
0.7
,
0.2
);
142
_drawSprite
(
canvas
,
suitSprite
,
0.5
,
0.35
);
143
_drawSprite
(
canvas
```

,

suitSprite

,

0.3

,

0.5

);

144

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.5

);

145

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.2

,

rotate:

true

);

146

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.2

,

rotate:

true

);

147

break

;

148

case

8

:

149

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.2

);

150

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.2

);

151

_drawSprite

(

canvas

,

suitSprite

,

0.5

,

0.35

);

152

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.5

);

153

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.5

);

154

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.2

,

rotate:

true

);

155

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.2

,

rotate:

true

);

156

_drawSprite

(

canvas

,

suitSprite

,

0.5

,

0.35

,

rotate:

true

);

157

break

;

158

case

9

:

159

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.2

);

160

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.2

);

161

_drawSprite

(

canvas

,

suitSprite

,

0.5

,

0.3

);

162

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.4

);

163

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.4

);

164

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.2

,

rotate:

true

);

165

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.2

,

rotate:

true

);

166

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.4

,

rotate:

true

);

167

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.4

,

rotate:

true

);

168

break

;

169

case

10

:

170

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.2

);

```
171
_drawSprite
(
canvas
,
suitSprite
,
0.7
,
0.2
);
172
_drawSprite
(
canvas
,
suitSprite
,
0.5
,
0.3
);
173
_drawSprite
(
canvas
```

,

suitSprite

,

0.3

,

0.4

);

174

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.4

);

175

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.2

,

rotate:

true

);

176

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.2

,

rotate:

true

);

177

_drawSprite

(

canvas

,

suitSprite

,

0.5

,

0.3

,

rotate:

true

);

178

_drawSprite

(

canvas

,

suitSprite

,

0.3

,

0.4

,

rotate:

true

);

179

_drawSprite

(

canvas

,

suitSprite

,

0.7

,

0.4

,

rotate:

true

);

180

break

;

181

case

11

:

182

_drawSprite

(

canvas

,

suit

.

isRed

?

redJack

:

blackJack

,

0.5

,

0.5

);

183

break

;

184

case

12

:

185

_drawSprite

(

canvas

,

suit

.

isRed

?

redQueen

:

blackQueen

,

0.5

,

0.5

);

186

break

;

187

case

13

:

188

_drawSprite

(

canvas

,

suit

.

isRed

?

redKing

:

blackKing

,

0.5

,

0.5

);

189

break

;

190

}

191

}

192

193

void

_drawSprite

(

194

Canvas

canvas

,

195

Sprite

sprite

,

196

double

relativeX

,

197

double

relativeY

,

{

198

double

scale

=

1

,

199

bool

rotate

=

false

,

200

})

{

201

if

(

rotate

)

{

202

canvas

.

save

();

203

canvas

.

translate

(

size

.

x

/

2

,

size

.

y

/

2

);

204

canvas

.

rotate

(

pi

);

205

canvas

.

translate

(

-

size

.

x

/

2

,

-

size

.

y

/

2

);

206

}

207

sprite

.

render

(

208

canvas

,

209

position:

Vector2

(

relativeX

*

size

.

x

,

relativeY

*

size

.

y

),

210

anchor:

Anchor

.

center

,

211

size:

sprite

.

srcSize

.

scaled

(

scale

),

212

);

213

if

(

rotate

)

{

214

canvas

.

restore

();

215

}

216

}

217

218

//#endregion

219

220

//#region Dragging

221

222

@override

223

void

onDragStart

(

DragStartEvent

event

)

{

224

super

.

onDragStart

(

event

);

225

if

(

pile

?

.

canMoveCard

(

this

)

??

false

)

{

226

_isDragging

=

true

;

227

priority

=

100

;

228

if

(

pile

is

TableauPile

)

{

229

attachedCards

.

clear

();

230

final

extraCards

=

(

pile

!

as

TableauPile

).

cardsOnTop

(

this

);

231

for

(

final

card

in

extraCards

)

{

232

card

.

priority

=

attachedCards

.

length

+

101

;

233

attachedCards

.

add

(

card

);

234

}

235

}

236

}

237

}

238

239

@override

240

void

onDragUpdate

(

DragUpdateEvent

event

)

{

241

if

(

!

```
_isDragging
)
{
242
return
;
243
}
244
final
delta
=
event
.
localDelta
;
245
position
.
add
(
delta
);
246
attachedCards
.
```

```
forEach
((
card
)
=>
card
.
position
.
add
(
delta
));
247
}
248
249
@override
250
void
onDragEnd
(
DragEndEvent
event
)
{
```

```
251
super
.
onDragEnd
(
event
);
252
if
(
!
_isDragging
)
{
253
return
;
254
}
255
_isDragging
=
false
;
256
final
```

dropPiles

=

parent

!

257

.

componentsAtPoint

(

position

+

size

/

2

)

258

.

whereType

<

Pile

>

()

259

.

toList

();

260

if

(

dropPiles

.

isNotEmpty

)

{

261

if

(

dropPiles

.

first

.

canAcceptCard

(

this

))

{

262

pile

!

.

removeCard

(

this

);

263

dropPiles

.

first

.

acquireCard

(

this

);

264

if

(

attachedCards

.

isNotEmpty

)

{

265

attachedCards

.

forEach

((

card

)

=>

dropPiles

.

first

.

acquireCard

(

card

));

266

attachedCards

.

clear

();

267

}

268

return

;

269

}

270

}

271

pile

!

.

returnCard

(

this

);

272

if

(

attachedCards

.

isNotEmpty

)

{

273

attachedCards

.

forEach

((

card

)

=>

pile

!

.

returnCard

(

card

));

274

attachedCards

.

clear

();

275

}

276

}

277

278

//#endregion

279

}

components/foundation_pile.dart

1

import

'dart:ui'

;

2

3

import

'package:flame/components.dart'

;

4

5

import

'../klondike_game.dart'

;

6

import

'../pile.dart'

;

7

import

'../suit.dart'

;

8

import

'card.dart'

;

9

10

class

FoundationPile

extends

PositionComponent

implements

Pile

{

11

FoundationPile

(

int

intSuit

,

{

super

.

position

})

12

:

suit

=

Suit

.

fromInt

(

intSuit

),

13

super

(

size:

KlondikeGame

.

cardSize

);

14

15

final

Suit

suit

;

16

final

List

<

Card

>

_cards

=

[];

17

18

//#region Pile API

19

20

@override

21

bool

canMoveCard

(

Card

card

)

{

22

return

_cards

.

isNotEmpty

&&

card

==

_cards

.

last

;

23

}

24

25

@override

26

bool

canAcceptCard

(

Card

card

)

{

27

final

topCardRank

=

_cards

.

isEmpty

?

0

:

_cards

.

last

.

rank

.

value

;

28

return

card

.

suit

==

suit

&&

29

card

.

rank

.

value

==

topCardRank

+

1

&&

30

card

.

attachedCards

.

isEmpty

;

31

}

32

33

```
@override
34
void
removeCard
(
Card
card
)
{
35
assert
(
canMoveCard
(
card
));
36
_cards
.
removeLast
();
37
}
38
39
@override
```

```
40
void
returnCard
(
Card
card
)
{
41
card
.
position
=
position
;
42
card
.
priority
=
_cards
.
indexOf
(
card
);
```

```
43
}
44
45
@override
46
void
acquireCard
(
Card
card
)
{
47
assert
(
card
.
isFaceUp
);
48
card
.
position
=
position
```

;

49

card

.

priority

=

_cards

.

length

;

50

card

.

pile

=

this

;

51

_cards

.

add

(

card

);

52

}

53

54

//#endregion

55

56

//#region Rendering

57

58

final

_borderPaint

=

Paint

()

59

..

style

=

PaintingStyle

.

stroke

60

..

strokeWidth

=

10

61

..

color

=

const

Color

(

0x50ffffff

);

62

late

final

_suitPaint

=

Paint

()

63

..

color

=

suit

.

isRed

?

const

Color

(

0x3a000000

)

:

const

Color

(

0x64000000

)

64

..

blendMode

=

BlendMode

.

luminosity

;

65

66

@override

67

void

render

(

Canvas

canvas

)

{

68

canvas

.

drawRRect

(

KlondikeGame

.

cardRRect

,

_borderPaint

);

69

suit

.

sprite

.

render

(

70

canvas

,

71

position:

size

/

2

,

72

anchor:

Anchor

.

center

,

73

size:

Vector2

.

all

(

KlondikeGame

.

cardWidth

*

0.6

),

74

overridePaint:

_suitPaint

,

75

);

76

}

77

78

//#endregion

79

}

components/stock_pile.dart

1

import

'dart:ui'

;

2

3

import

'package:flame/components.dart'

;

4

import

'package:flame/events.dart'

;

5

6

import

'../klondike_game.dart'

;

7

import

'../pile.dart'

;

8

import

'card.dart'

;

9

import

'waste_pile.dart'

;

10

11

class

StockPile

extends

PositionComponent

with

TapCallbacks

implements

Pile

{

12

StockPile

({

super

.

position

})

:

super

(

size:

KlondikeGame

.

cardSize

);

13

14

/// Which cards are currently placed onto this pile. The first card in the

15

/// list is at the bottom, the last card is on top.

16

final

List

<

Card

>

_cards

=

[];

17

18

//#region Pile API

19

20

@override

21

bool

canMoveCard

(

Card

card

)

=>

false

;

22

23

@override

24

bool

canAcceptCard

(

Card

card

)

=>

false

;

25

26

@override

27

void

removeCard

(

Card

card

)

=>

throw

StateError

(

'cannot remove cards'

);

28

29

@override

30

void

returnCard

(

Card

card

)

=>

throw

StateError

(

'cannot remove cards'

);

31

32

@override

33

void

acquireCard

(

Card

card

)

{

34

assert

(

card

.

isFaceDown

);

35

card

.

pile

=

this

;

36

card

.

position

=

position

;

37

card

.

priority

=

_cards

.

length

;

38

_cards

.

add

(

card

);

39

}

40

41

//#endregion

42

43

@override

44

void

onTapUp

(

TapUpEvent

event

)

{

45

final

wastePile

=

parent

!

.

firstChild

<

WastePile

>

()

!

;

46

if

(

_cards

.

isEmpty

)

{

47

wastePile

.

removeAllCards

().

reversed

.

forEach

((

card

)

{

48

card

.

flip

();

49

acquireCard

(

card

);

50

});

51

}

else

{

52

for

(

var

i

=

0

;

i

<

3

;

i

++

)

{

53

if

(

_cards

.

isNotEmpty

)

{

54

final

card

=

_cards

.

removeLast

();

55

card

.

flip

();

56

wastePile

.

acquireCard

(

card

);

57

}

58

}

59

}

60

}

61

62

//#region Rendering

63

64

final

_borderPaint

=

Paint

()

65

..

style

=

PaintingStyle

.

stroke

66

..

strokeWidth

=

10

67

..

color

=

const

Color

(

0xFF3F5B5D

);

68

final

_circlePaint

=

Paint

()

69

..

style

=

PaintingStyle

.

stroke

70

..

strokeWidth

=

100

71

..

color

=

const

Color

(

0x883F5B5D

);

72

```
73
@override
74
void
render
(
Canvas
canvas
)
{
75
canvas
.
drawRRect
(
KlondikeGame
.
cardRRect
,
_borderPaint
);
76
canvas
.
drawCircle
(
```

77

Offset

(

width

/

2

,

height

/

2

),

78

KlondikeGame

.

cardWidth

*

0.3

,

79

_circlePaint

,

80

);

81

}

82

83

//#endregion

84

}

components/tableau_pile.dart

1

import

'dart:ui'

;

2

3

import

'package:flame/components.dart'

;

4

5

import

'../klondike_game.dart'

;

6

import

'../pile.dart'

;

7

import

'card.dart'

;

8

9

class

TableauPile

extends

PositionComponent

implements

Pile

{

10

TableauPile

({

super

.

position

})

:

super

(

size:

KlondikeGame

.

cardSize

);

11

12

/// Which cards are currently placed onto this pile.

13

final

List

<

Card

>

_cards

=

[];

14

final

Vector2

_fanOffset1

=

Vector2

(

0

,

KlondikeGame

.

cardHeight

*

0.05

);

15

final

Vector2

_fanOffset2

=

Vector2

(

0

,

KlondikeGame

.

cardHeight

*

0.20

);

16

17

//#region Pile API

18

19

@override

20

bool

canMoveCard

(

Card

card

)

=>

card

.

isFaceUp

;

21

22

@override

23

bool

canAcceptCard

(

Card

card

)

{

24

if

(

_cards

.

isEmpty

)

{

```
25
return
card
.
rank
.
value
==
13
;
26
}
else
{
27
final
topCard
=
_cards
.
last
;
28
return
card
.
```

suit

.

isRed

==

!

topCard

.

suit

.

isRed

&&

29

card

.

rank

.

value

==

topCard

.

rank

.

value

-

1

;

30

}

31

}

32

33

@override

34

void

removeCard

(

Card

card

)

{

35

assert

(

_cards

.

contains

(

card

)

&&

card

.

isFaceUp

);

36

final

index

=

_cards

.

indexOf

(

card

);

37

_cards

.

removeRange

(

index

,

_cards

.

length

);

38

if

(

_cards

.

isNotEmpty

&&

_cards

.

last

.

isFaceDown

)

{

39

flipTopCard

();

40

}

41

layOutCards

();

42

}

43

44

@override

45

```
void
returnCard
(
Card
card
)
{
46
card
.
priority
=
_cards
.
indexOf
(
card
);
47
layOutCards
();
48
}
49
50
@override
```

51

void

acquireCard

(

Card

card

)

{

52

card

.

pile

=

this

;

53

card

.

priority

=

_cards

.

length

;

54

_cards

.

add

(

card

);

55

layOutCards

();

56

}

57

58

//#endregion

59

60

void

flipTopCard

()

{

61

assert

(

_cards

.

last

.

isFaceDown

);

62

_cards

.

last

.

flip

();

63

}

64

65

void

layOutCards

()

{

66

if

(

_cards

.

isEmpty

)

{

67

return

;

68

}

69

_cards

[

0

].

position

.

setFrom

(

position

);

70

for

(

var

i

=

1

;

i

<

_cards

.

length

;

i

++

)

{

71

_cards

[

i

].

position

72

..

setFrom

(

_cards

[

i

-

1

].

position

)

73

..

add

(

_cards

[

i

-

1

].

isFaceDown

?

_fanOffset1

:

_fanOffset2

);

74

}

75

height

=

KlondikeGame

.

cardHeight

*

1.5

+

_cards

.

last

.

y

-

_cards

.

first

.

y

;

76

}

77

78

List

<

Card

>

cardsOnTop

(

Card

card

)

{

```
79
assert
(
card
.
isFaceUp
&&
_cards
.
contains
(
card
));
80
final
index
=
_cards
.
indexOf
(
card
);
81
return
_cards
```

.

getRange

(

index

+

1

,

_cards

.

length

).

toList

();

82

}

83

84

//#region Rendering

85

86

final

_borderPaint

=

Paint

()

87

```
..
style
=
PaintingStyle
.
stroke
88
..
strokeWidth
=
10
89
..
color
=
const
Color
(
0x50ffffff
);
90
91
@override
92
void
render
```

(

Canvas

canvas

)

{

93

canvas

.

drawRRect

(

KlondikeGame

.

cardRRect

,

_borderPaint

);

94

}

95

96

//#endregion

97

}

components/waste_pile.dart

1

import

'package:flame/components.dart'

;

2

3

import

'../klondike_game.dart'

;

4

import

'../pile.dart'

;

5

import

'card.dart'

;

6

7

class

WastePile

extends

PositionComponent

implements

Pile

{

8

WastePile

({

super

.

position

})

:

super

(

size:

KlondikeGame

.

cardSize

);

9

10

final

List

<

Card

>

_cards

=

[];

11

final

Vector2

_fanOffset

=

Vector2

(

KlondikeGame

.

cardWidth

*

0.2

,

0

);

12

13

//#region Pile API

14

15

@override

16

bool

canMoveCard

(

Card

card

)

=>

_cards

.

isNotEmpty

&&

card

==

_cards

.

last

;

17

18

@override

19

bool

canAcceptCard

(

Card

card

)

=>

false

;

20

21

@override

```
22
void
removeCard
(
Card
card
)
{
23
assert
(
canMoveCard
(
card
));
24
_cards
.
removeLast
();
25
_fanOutTopCards
();
26
}
27
```

```
28
@override
29
void
returnCard
(
Card
card
)
{
30
card
.
priority
=
_cards
.
indexOf
(
card
);
31
_fanOutTopCards
();
32
}
```

```
33
34
@override
35
void
acquireCard
(
Card
card
)
{
36
assert
(
card
.
isFaceUp
);
37
card
.
pile
=
this
;
38
```

card

.

position

=

position

;

39

card

.

priority

=

_cards

.

length

;

40

_cards

.

add

(

card

);

41

_fanOutTopCards

();

42

}

43

44

//#endregion

45

46

List

<

Card

>

removeAllCards

()

{

47

final

cards

=

_cards

.

toList

();

48

_cards

.

clear

();

49

return

cards

;

50

}

51

52

void

_fanOutTopCards

()

{

53

final

n

=

_cards

.

length

;

54

for

(

var

i

=

0

;

i

<

n

;

i

++

)

{

55

_cards

[

i

].

position

=

position

;

56

}

57

if

(

n

==

2

)

{

58

_cards

[

1

].

position

.

add

(

_fanOffset

);

59

}

else

if

(

n

>=

3

)

{

60

_cards

[

n

-

2

].

position

.

add

(

_fanOffset

);

61

_cards

[

n

-

1

].

position

.

addScaled

(

_fanOffset

,

2

);

62

}

63

}

64

}

klondike_game.dart

1

import

'dart:ui'

;

2

3

import

'package:flame/components.dart'

;

4

import

'package:flame/flame.dart'

;

5

import

'package:flame/game.dart'

;

6

7

```
import
'components/card.dart'
;
8
import
'components/foundation_pile.dart'
;
9
import
'components/stock_pile.dart'
;
10
import
'components/tableau_pile.dart'
;
11
import
'components/waste_pile.dart'
;
12
13
class
KlondikeGame
extends
FlameGame
{
```

14

static

const

double

cardGap

=

175.0

;

15

static

const

double

cardWidth

=

1000.0

;

16

static

const

double

cardHeight

=

1400.0

;

17

static

const

double

cardRadius

=

100.0

;

18

static

final

Vector2

cardSize

=

Vector2

(

cardWidth

,

cardHeight

);

19

static

final

cardRRect

=

RRect

.

fromRectAndRadius

(

20

const

Rect

.

fromLTWH

(

0

,

0

,

cardWidth

,

cardHeight

),

21

const

Radius

.

circular

(

cardRadius

),

22

);

23

24

@override

25

Future

<

void

>

onLoad

()

async

{

26

await

Flame

.

images

.

load

(

'klondike-sprites.png'

);

27

28

final

stock

=

StockPile

(

position:

Vector2

(

cardGap

,

cardGap

));

29

final

waste

=

30

WastePile

(

position:

Vector2

(

cardWidth

+

2

*

cardGap

,

cardGap

));

31

final

foundations

=

List

.

generate

(

32

4

,

33

(

i

)

=>

FoundationPile

(

34

i

,

35

position:

Vector2

((

i

+

3

)

*

(

cardWidth

+

cardGap

)

+

cardGap

,

cardGap

),

36

),

37

);

38

final

piles

=

List

.

generate

(

39

7

,

40

(

i

)

=>

TableauPile

(

41

position:

Vector2

(

42

cardGap

+

i

*

(

cardWidth

+

cardGap

),

43

cardHeight

+

2

*

cardGap

,

44

),

45

),

46

);

47

48

world

.

add

(

stock

);

49

world

.

add

(

waste

);

50

world

.

addAll

(

foundations

);

51

world

.

addAll

(

piles

);

52

53

camera

.

viewfinder

.

visibleGameSize

=

54

Vector2

(

cardWidth

*

7

+

cardGap

*

8

,

4

*

cardHeight

+

3

*

cardGap

);

55

camera

.

viewfinder

.

position

=

Vector2

(

cardWidth

*

3.5

+

cardGap

*

4

,

0

);

56

camera

.

viewfinder

.

anchor

=

Anchor

.

topCenter

;

57

58

final

cards

=

[

59

for

(

var

rank

=

1

;

rank

<=

13

;

rank

++

)

60

for

(

var

suit

=

0

;

suit

<

4

;

suit

++

)

Card

(

rank

,

suit

),

61

];

62

cards

.

shuffle

();

63

world

.

addAll

(

cards

);

64

65

var

cardToDeal

=

cards

.

length

-

1

;

66

for

(

var

i

=

0

;

i

<

7

;

i

++

)

{

67

for

(

var

j

=

i

;

j

<

7

;

j

++

)

{

68

piles

[

j

].

acquireCard

(

cards

[

cardToDeal

--

]);

69

}

70

piles

[

i

].

flipTopCard

();

71

}

72

for

(

var

n

=

0

;

n

<=

cardToDeal

;

n

++

)

{

73

stock

.

acquireCard

(

cards

[

n

]);

74

}

75

}

76

}

77

78

Sprite

klondikeSprite

(

double

x

,

double

y

,

double

width

,

double

height

)

{

79

return

Sprite

(

80

Flame

.

images

.

fromCache

(

'klondike-sprites.png'

),

81

srcPosition:

Vector2

(

x

,

y

),

82

srcSize:

Vector2

(

width

,

height

),

83

);

84

}

main.dart

1

import

'package:flame/game.dart'

;

2

import

'package:flutter/widgets.dart'

;

3

```
4
import
'klondike_game.dart'
;
5
6
void
main
()
{
7
final
game
=
KlondikeGame
();
8
runApp
(
GameWidget
(
game:
game
));
9
}
```

pile.dart

1

import

'components/card.dart'

;

2

3

abstract

class

Pile

{

4

/// Returns true if the [card] can be taken away from this pile and moved

5

/// somewhere else.

6

bool

canMoveCard

(

Card

card

);

7

8

/// Returns true if the [card] can be placed on top of this pile. The [card]

9

/// may have other cards "attached" to it.

10

bool

canAcceptCard

(

Card

card

);

11

12

/// Removes [card] from this pile; this method will only be called for a card

13

/// that both belong to this pile, and for which [canMoveCard] returns true.

14

void

removeCard

(

Card

card

);

15

16

/// Places a single [card] on top of this pile. This method will only be

17

/// called for a card for which [canAcceptCard] returns true.

18

```
void
acquireCard
(
Card
card
);
19
20
/// Returns the [card] (which already belongs to this pile) in its proper
21
/// place.
22
void
returnCard
(
Card
card
);
23
}
```

rank.dart

```
1
import
'package:flame/components.dart'
;
2
```

import

'package:flutter/foundation.dart'

;

3

import

'klondike_game.dart'

;

4

5

@immutable

6

class

Rank

{

7

factory

Rank

.

fromInt

(

int

value

)

{

8

assert

(

9

value

>=

1

&&

value

<=

13

,

10

'value is outside of the bounds of what a rank can be'

,

11

);

12

return

_singletons

[

value

-

1

];

13

}

14

15

Rank

.

_

(

16

this

.

value

,

17

this

.

label

,

18

double

x1

,

19

double

y1

,

20

double

x2

,

21

double

y2

,

22

double

w

,

23

double

h

,

24

)

:

redSprite

=

klondikeSprite

(

x1

,

y1

,

w

,

h

),

25

blackSprite

=

klondikeSprite

(

x2

,

y2

,

w

,

h

);

26

27

final

int

value

;

28

final

String

label

;

29

final

Sprite

redSprite

;

30

final

Sprite

blackSprite

;

31

32

static

final

List

<

Rank

>

_singletons

=

[

33

Rank

.

_

(

1

,

'A'

,

335

,

164

,

789

,

161

,

120

,

129

),

34

Rank

.

–

(

2

,

'2'

,

20

,

19

,

15

,

322

,

83

,

125

),

35

Rank

.

–

(

3

,

'3'

,

122

,

19

,

117

,

322

,

80

,

127

),

36

Rank

.

_

(

4

,

'4'

,

213

,

12

,

208

,

315

,

93

,

132

),

37

Rank

.

_

(

5

,

'5'

,

314

,

21

,

309

,

324

,

85

,

125

),

38

Rank

.

_

(

6

,

'6'

,

419

,

17

,

414

,

320

,

84

,

129

),

39

Rank

.

–

(

7

,

'7'

,

509

,

21

,

505

,

324

,

92

,

128

),

40

Rank

.

–

(

8

,

'8'

,

612

,

19

,

607

,

322

,

78

,

127

),

41

Rank

.

–

(

9

,

'9'

,

709

,

19

,

704

,

322

,

84

,

130

),

42

Rank

.

_

(

10

,

'10'

,

810

,

20

,

805

,

322

,

137

,

127

),

43

Rank

.

_

(

11

,

'J'

,

15

,

170

,

469

,

167

,

56

,

126

),

44

Rank

.

_

(

12

,

'Q'

,

92

,

168

,

547

,

165

,

132

,

128

),

45

Rank

.

–

(

13

,

'K'

,

243

,

170

,

696

,

167

,

92

,

123

),

46

];

47

}

suit.dart

1

import

'package:flame/sprite.dart'

;

2

import

'package:flutter/foundation.dart'

;

3

import

'klondike_game.dart'

;

4

5

@immutable

6

class

Suit

{

7

factory

Suit

.

fromInt

(

int

index

)

{

8

assert

(

9

index

>=

0

&&

index

<=

3

,

10

'index is outside of the bounds of what a suit can be'

,

11

);

12

return

_singletons

[

index

];

13

}

14

15

Suit

.

_

(

this

.

value

,

this

.

label

,

double

x

,

double

y

,

double

w

,

double

h

)

16

:

sprite

=

klondikeSprite

(

x

,

y

,

w

,

h

);

17

18

final

int

value

;

19

final

String

label

;

20

final

Sprite

sprite

;

21

22

static

final

List

<

Suit

>

_singletons

=

[

23

Suit

.

_

(

0

,

'?'

,

1176

,

17

,

172

,

183

),

24

Suit

.

_

(

1

,

'?'

,

973

,

14

,

177

,

182

),

25

Suit

.

–

(

2

,

'?'

,

974

,

226

,

184

,

172

),

26

Suit

.

_

(

3

,

'?'

,

1178

,

220

,

176

,

182

),

27

];

28

29

/// Hearts and Diamonds are red, while Clubs and Spades are black.

30

bool

get

isRed

=>

value

<=

1

;

31

bool

get

isBlack

=>

value

>=

2

;

32

}

Code

Tiled

¶

Tiled

is a great tool to design levels and maps.  From

Tiled

?s documentation:

Tiled is a 2D level editor that helps you develop the content of your game. Its primary feature is to edit tile r

In terms of tile maps, it supports straight rectangular tile layers, but also projected isometric, staggered isor

Flame provides a package (

flame_tiled

) that bundles a

dart

package which allows you to parse TMX (XML) files and access the tiles, objects, and everything in there.

The

dart

package provides a simple

Tiled

class and

flame_tiled

provides a component wrapper

TiledComponent

, for the map rendering, which renders the tiles on the screen and supports rotations and flips.

Tiled Editor

¶

You can choose to download the

Tiled

map editor and create interactive maps that can be loaded into your game.  At its core, the Tiled

map editor creates a TMX file that can be parsed and used within your game.

Writing tests

¶

All new functionality must be tested, if at all possible. When fixing a bug, tests must be added to ensure tha

Run

melos

run

coverage

to execute all tests in the ?coverage? mode. The results will be saved in the

coverage/index.html

file, which can be opened in a browser. Try to achieve 100% coverage for any new functionality added.

Every source file should have a corresponding test file, with the

_test

suffix. For example, if you?re making a

SpookyEffect

and the source file is

src/effects/spooky_effect.dart

, then the test file should be

test/effects/spooky_effect_test.dart

mirroring the source directory.

The test file should contain a

main()

function with a single

group()

whose name matches the name of the class being tested. If the source file contains multiple public classes

void

main

()

{

group

(

'SpookyEffect'

,

()

{

// tests here

});

}

For a larger class, multiple groups can be created inside the top-level group, allowing to navigate the test s

The names of the individual tests should normally start with a lowercase.

Often, you would need to define multiple helper classes to run the tests. Such classes should be private (st

main()

function at the top of the file makes this process much easier.

Types of tests

¶

Simple tests

¶

test

(

'the name of the test'

,

()

{

expect

(...);

});

This is the simplest kind of test available, and also the fastest. Use these tests for checking some classes/r

FlameGame tests

¶

It is very common to want to have a

FlameGame

instance inside a test, so that you can add some components to it and verify various behaviors. The followi

testWithFlameGame

(

'the name of the test'

,

(

game

)

async

{

game

.

add

(...);

await

game

.

ready

();

expect

(...);

});

Here the

game

instance that is passed to the test body is a fully initialized game that behaves as if it was mounted to a

GameWidget

. The

game.ready()

method waits until all the scheduled components are loaded and mounted to the component tree.

The time within the

game

can be advanced with

game.update(dt)

.

If you need to have a custom game inside this test (say, a game with some mixin), then use

testWithGame

<

_MyGame

>

(

'the name of the test'

,

_MyGame

.

new

,

(

game

)

async

{

// test body...

},

);

Widget tests

¶

Sometimes having a ?naked?

FlameGame

is insufficient, and you want to have access to the Flutter infrastructure as well. That is, to have a game mo

GameWidget

embedded into an actual Flutter framework. In such cases, use

testWidgets

(

'test name'

,

(

tester

)

async

{

```
final
game
=
_MyGame
();
await
tester
.
pumpWidget
(
GameWidget
(
game:
game
));
await
tester
.
pump
();
await
tester
.
pump
();
// At this point the game is fully initialized, and you can run your checks
```

// against it.

expect

(...);

// Equivalent to game.update(0)

await

tester

.

pump

();

// Advances in-game time by 20 milliseconds

await

tester

.

pump

(

const

Duration

(

milliseconds:

20

));

});

There are some additional methods available on the

tester

controller, for example in order to simulate taps, or drags, or key presses.

Golden tests

¶

These tests verify that things render as intended. The process of creating a golden test is simple:

Write the test, using the following template:

```
testGolden
(
'the name of the test'
,
(
game
)
async
{
// Set up the game by adding the necessary components
// You can add `expect()` checks here too, if you want to
},
size:
Vector2
(
300
,
200
),
goldenFile:
'.../_goldens/my_test_file.png'
,
);
```

Here the

size

parameter determines the size of the game canvas and of the output image. The

goldenFile

parameter is the name of the file where you want to store the ?golden? results. This should be a relative pa

test/_goldens

directory, starting from your test file.

Run

flutter

test

--update-goldens

this would create the golden file for the first time. Open the file to verify that it renders exactly as you intenc

Subsequent runs of

flutter

test

will check whether the output of the golden test matches the saved golden file. If not, Flutter will save the in

failures/

directory where your test is located.

Note

Avoid using text in your golden tests ? it does not render reliably across different platforms, due to font disc

Random tests

¶

These are the tests that use a random number generator in order to construct a randomized input and then

testRandom

(

'test name'

,

(

Random

random

)

{

// Use [random] to generate random input

});

You can add

repeatCount:

1000

parameter to run this test the specified number of times, each one with a different seed. It is useful to run a

repeatCount

when developing the test, to ensure that it doesn?t break. However, when submitting the test to the main re

If the test breaks at some particular seed, then that seed will be shown in the test output. Add it as the

seed:

NNN

parameter to your test, and you?ll be able to run it for the same seed as long as you need until the test is fi

seed:

parameter when submitting your code, as it defeats the purpose of having the test randomized.

<<declare>>

¶

The

<<declare>>

command creates a new global variable and assigns an initial value to it. After this command is encountere

Unlike most other commands, the

<<declare>>

command is executed at compile time, i.e. when the yarn scripts are parsed. When the dialogue runs, it ha

<<declare>>

commands must be placed outside of nodes, at the root level of the script, making it clear that these comm

For example:

<<

declare

$monicker

=

"

boy

"

>>

---

title

:

Greeting

---

Teacher

:

Welcome to the class,

{

$monicker

}

!

===

Here the

<<declare>>

command introduces a new variable called

$monicker

, of type

String

, and assigns it an initial value of

"boy"

. Later on, this variable is used inside the ?Greeting? node. By that time, the value of the variable can be a

<<declare>>

statement, however, is necessary to tell Jenny that this is a valid variable name, and what type it has.

From the project organization standpoint, the recommended approach is to put all the

<<declare>>

statements into a separate file, and then make sure that this yarn file is parsed first. This will ensure that al

If your game supports save-games, then you would probably want to store the values of yarn global variabl

after

all yarn scripts are parsed (otherwise the engine will think that a variable is declared twice).

Syntax

¶

There are several forms of the

<<declare>>

statement. The most common one is the following:

<<

declare

$VARIABLE

=

EXPRESSION

>>

Here

$VARIABLE

is the name of the variable being declared (all variables in Yarn start with a

$

sign), and

EXPRESSION

is either a literal or a more complicated

expression

that will be evaluated at compile time in order to provide the initial value for the variable. The type of the va

EXPRESSION

.

Another possible syntax for the

<<declare>>

command is this:

<<

declare

$VARIABLE

as

TYPE

>>

where

TYPE

is one of

Bool

,

Number

, or

String

. This will create a variable of the given type, and initialize it with values

false

,

0

, or

""

respectively.

Finally, it is possible to combine these two syntaxes:

<<

declare

$VARIABLE

=

EXPRESSION

as

TYPE

>>

This can be useful when the type of the

EXPRESSION

is not immediately obvious, and you want to make the declaration more explicit. The compiler will check tha

EXPRESSION

is the same as

TYPE

, and will throw a compile-time error otherwise.

Examples

¶

<<

declare

$prefix

=

"

Mr.

"

>>

<<

declare

$gold

=

100

>>

<<

declare

$been_to_hell

=

false

>>

<<

declare

$name

as

String

>>

<<

declare

$distanceTraveled

as

Number

>>

<<

declare

$birthDay

=

randomRange

(

1

,

365

)

as

Number

>>

<<

declare

$vulgarity

=

GetObscenitySetting

()

as

Bool

>>

Note

It is a good idea to accompany each

<<declare>>

with a doc-comment explaining the purpose of the variable, similarly to how you would document public me

Functions

¶

A

function

in YarnSpinner is the same notion as in any other programming language, or in math: it takes a certain nu

<<

set

$roll_2d6

=

dice

(

6

)

+

dice

(

6

)>>

<<

set

$random

=

random

()>>

There are around 20 built-in functions in Jenny, listed below; and it is also possible to add user-defined fu

Built-in functions

¶

Random functions

dice(n)

random()

random_range(a,

b)

Numeric functions

ceil(x)

dec(x)

decimal(x)

floor(x)

inc(x)

int(x)

round(x)

round_places(x,

n)

Type conversion functions

bool(x)

number(x)

string(x)

Other functions

if(condition,

then,

else)

plural(x,

...)

visit_count(node)

visited(node)

User-defined functions

¶

In addition to the built-in functions, you can also define any number of

user-defined functions

which can later be used in your yarn scripts. The syntax for these functions is exactly the same as for the b

Each user-defined function has a fixed signature, declared at the time when the function is added to the

YarnProject

. A function must have a fixed number of arguments of specific types, and a fixed return type.

All user-defined functions must be added to the

YarnProject

before they can be used. A compile error will be raised if the parser encounters an unknown function, or if t

User-defined functions can be used for a variety of purposes, such as:

implement functionality that is currently missing in Jenny;

interface with the game engine;

provide access to ?variables? stored outside of Jenny;

etc.

title

:

Blacksmith

---

// This example showcases several hypothetical user-defined functions:

// - broken(slot): checks whether the item in the given slot is broken;

// - name(slot): gives the name for an item in a slot, e.g. "sword" or "bow";

// - money(): returns the current amount of money that the player has.

// At the same time, functions `round()` and `plural()` are built-in.

<<

if

broken

(

"

main_hand

"

)>>

<<

local

$repair_cost

=

round

(

value

(

"

main_hand

"

)

/

5

)>>

Blacksmith

:

Your

{

name

(

"

main_hand

"

)}

seems to be completely broken!

Blacksmith

:

I can fix it for just

{

plural

(

$repair_cost

,

"

% coin

"

)}

->

Ok, do it

<<

if

money

()

>=

$repair_cost

>>

->

I'll be fine...

<<

endif

>>

===

See also

FunctionStorage

? document describing how to add user-defined functions to a

YarnProject

.

## Enemies and Bullets collision

¶

Right, we are really close to a playable game, we have enemies and we have the ability to shoot bullets at

Flame provides a collision detection system out of the box, which we will use to implement our logic that wi

First we need to let our

FlameGame

know that we want collisions between components to be checked. In order to do so, simply add the

HasCollisionDetection

mixin to the declaration of the game class:

class

SpaceShooterGame

extends

FlameGame

with

PanDetector

,

HasCollisionDetection

{

// ...

}

With that, Flame now will start to check if components has collided with each other. Next we need to identif

In our case those are the

Bullet

and

Enemy

components and we need to add hitboxes to them.

A hitbox is nothing more than a defined part of the component?s area that can hit other objects. Flame offe

RectangleHitbox

, which like the name implies will make a rectangular area as the component?s hitbox.

Hitboxes are also components, so in order to add them to our components we can simply add them to the

Enemy

class:

add

(

RectangleHitbox

());

For the bullet we will do the same, but with a slight difference:

add

(

RectangleHitbox

(

collisionType:

CollisionType

.

passive

,

),

);

The

collisionType

s are very important to understand, since they can directly impact the game performance!

There are three types of collisions in Flame:

active

collides with other

Hitbox

es of type active or passive

passive

collides with other

Hitbox

es of type active

inactive

will not collide with any other

Hitbox

es

Usually it is smart to mark hitboxes from components that will have a higher number of instances as passiv

And since in this game we anticipate that there will be more bullets than enemies, we choose the bullets to

From this point on, Flame will take care of checking the collision between those two components, we now n

We can start that by receiving the collision events in one of the classes. Since

Bullet

s have a passive collision type, we will also add the collision checking logic to the

Enemy

class.

To listen to collision events we need to add the

CollisionCallbacks

mixin to the component. By doing so we will be able to override some methods like

onCollisionStart

and

onCollisionEnd

.

So let?s do that and make a few changes to the

Enemy

class:

class

Enemy

extends

SpriteAnimationComponent

with

HasGameReference

<

SpaceShooterGame

>

,

CollisionCallbacks

{

// Other methods omitted

@override

void

onCollisionStart

(

Set

<

Vector2

>

intersectionPoints

,

PositionComponent

other

,

)

{

super

.

onCollisionStart

(

intersectionPoints

,

other

);

if

(

other

is

Bullet

)

{

removeFromParent

();

other

.

removeFromParent

();

}

}

}

As you can see, we added the mixin to the class, overrode the

onCollisionStart

method, where we check whether the component that collided with us was a

Bullet

and if it was, then we remove both the current

Enemy

instance and the

Bullet

.

If you run the game now you will finally be able to defeat the enemies crawling down the screen!

To add some final touches, let?s add some explosion animations and add more action to the game!

First, let?s create the explosion class:

class

Explosion

extends

SpriteAnimationComponent

with

HasGameReference

<

SpaceShooterGame

>

{

Explosion

({

super

.

position

,

})

:

super

(

size:

Vector2

.

all

(

150

),

anchor:

Anchor

.

center

,

removeOnFinish:

true

,

);

```
@override
Future
<
void
>
onLoad
()
async
{
await
super
.
onLoad
();
animation
=
await
game
.
loadSpriteAnimation
(
'explosion.png'
,
SpriteAnimationData
.
sequenced
```

(

amount:

6

,

stepTime:

.

1

,

textureSize:

Vector2

.

all

(

32

),

loop:

false

,

),

);

}

}

There is not much new in it, the biggest difference compared to the other animation components is that we

loop:

false

in the

SpriteAnimationData.sequenced

constructor and that we are setting

removeOnFinish:

true;

. We do that so that when the animation is finished, it will automatically be removed from the game!

And finally, we make a small change in the

onCollisionStart

method from the

Enemy

class in order to add the explosion to the game:

@override

void

onCollisionStart

(

Set

<

Vector2

>

intersectionPoints

,

PositionComponent

other

,

)

{

super

.

onCollisionStart

(

intersectionPoints

,

other

);

if

(

other

is

Bullet

)

{

removeFromParent

();

other

.

removeFromParent

();

game

.

add

(

Explosion

(

position:

position

));

}

}

And that is it! We finally have a game which provides all the minimum necessary elements for a space sho

Good hunting pilot, and happy coding!

flame_riverpod

¶

Riverpod

¶

Riverpod

is a reactive caching and data-binding framework for Dart & Flutter.

In

flutter_riverpod

, widgets can be configured to rebuild when the state of a provider changes.

When using Flame, we are interacting with components, which are

not

Widgets.

flame_riverpod

provides the

RiverpodAwareGameWidget

,

RiverpodGameMixin

, and

RiverpodComponentMixin

to facilitate managing state from

Provider

s in your Flame Game.

Usage

¶

You should use the

RiverpodAwareGameWidget

as your Flame

GameWidget

, the

RiverpodGameMixin

mixin on your game that extends

FlameGame

, and the

RiverpodComponentMixin

on any components interacting with Riverpod providers.

Subscriptions to a provider are managed in accordance with the lifecycle of a Flame Component: initializati

By default, the

RiverpodAwareGameWidget

is rebuilt when Riverpod-aware (i.e. using the

RiverpodComponentMixin

) components are mounted and when they are removed.

/// An excerpt from the Example. Check it out!

class

RefExampleGame

extends

FlameGame

with

RiverpodGameMixin

{

@override

Future

<

```
void
>
onLoad
()
async
{
await
super
.
onLoad
();
add
(
TextComponent
(
text:
'Flame'
));
add
(
RiverpodAwareTextComponent
());
}
}
class
RiverpodAwareTextComponent
```

extends

PositionComponent

with

RiverpodComponentMixin

{

late

TextComponent

textComponent

;

int

currentValue

=

0

;

@override

void

onMount

()

{

addToGameWidgetBuild

(()

{

ref

.

listen

(

countingStreamProvider

,

(

p0

,

p1

)

{

if

(

p1

.

hasValue

)

{

currentValue

=

p1

.

value

!

;

textComponent

.

text

=

'

$

currentValue

'

;

}

});

});

super

.

onMount

();

add

(

textComponent

=

TextComponent

(

position:

position

+

Vector2

(

0

,

27

)));

}

}

The order of operations in

Component.onMount

is important. The

RiverpodComponentMixin

interacts with

RiverpodGameMixin

(inside of

RiverpodComponentMixin.onMount

) to co-ordinate adding and removing listeners as the corresponding component is mounted and removed,

Variables

¶

A

variable

is a place to store some piece of information ? it is the same notion as in any other programming language

name

,

value

,

type

, and a

scope

.

Name

¶

The

name

of a variable is how you refer to it in a

.yarn

script. The names of all variables start with a

$

sign, followed by a letter or an underscore, and then by any number of letters, digits, or underscores. Thus

$i $WARNING $_secret_ $door10 $climbed_over_wall_and_avoided_all_guard_patrols $DoorPassword

while the following are NOT valid names:

$2000_years $[main] @today victory

Type

¶

Each variable has a certain

type

associated with it. The type of a variable is determined when the variable is first declared, and it never cha

There are three types of variables in YarnSpinner:

string

,

number

, and

bool

.

bool

variables can store either

true

or

false

and nothing else;

number

variables may contain either integer or decimal numbers, such as

0

,

42

,

2.5

;

string

variables contain arbitrary text, for example

"the

most

random

number

is

4"

.

// Creates a variable $money of type number, and gives it initial value of 100

<<

declare

$money

=

100

>>

// Creates variable $name of type string, the initial value will be ""

<<

declare

$name

as

String

>>

Value

¶

Each variable stores a single

value

. This value can be replaced with another value at any time, but the type of the new value must be the sam

Each variable will have an initial value assigned to it when the variable is first created, and then new values

<<set>>

command.

<<

set

$money

+=

10

>>

// increases the value of $money by 10

Scope

¶

The

scope

of a variable is where exactly it can be accessed. In YarnSpinner, the variables can be either global or loca

The

global

variables are introduced via the

<<declare>>

command, and once created can be accessed anywhere. The names of all global variables are unique.

The

local

variables are created with the

<<local>>

command, and can only be used within the node where they were created. It is possible to have a local var

<<

declare

$global_variable

=

0

>>

title

:

MyNode

---

<<

local

$local_variable

=

1

>>

===

<<visit>>

¶

The

<<visit>>

command temporarily puts the current node on hold, executes the target node, and after it finishes, resume

The

<<visit>>

command can be useful for splitting a large dialogue into several smaller nodes, or for reusing some comm

title

:

RoamingTrader1

---

<<

if

$roaming_trader_introduced

>>

Hello again,

{

$player

}

!

<<

else

>>

<<

visit

RoamingTraderIntro

>>

<<

endif

>>

->

What do you think about the Calamity?

<<

if

$calamity_started

>>

<<

visit

RoamingTrader_Calamity

>>

->

Have you seen a weird-looking girl running by?

<<

if

$quest_little_girl

>>

<<

visit

RoamingTrader_LittleGirl

>>

->

What do you have for trade?

<<

OpenTrade

>>

Pleasure doing business with you!

#auto

===

The argument of this command is the id of the node to jump to. It can be given either as a plain node ID, or

<<

visit

{

"

RewardChoice_

"

+

string

(

$choice

)}>>

If the expression evaluates at runtime to an unknown name, then a

NameError

exception will be thrown.

## Widgets

¶

One advantage when developing games with Flutter is the ability to use Flutter?s extensive toolset for build

Here you can find all the available widgets provided by Flame.

You can also see all the widgets showcased inside a

Dashbook

sandbox

here

### Nine Tile Box

¶

A Nine Tile Box is a rectangle drawn using a grid sprite.

The grid sprite is a 3x3 grid and with 9 blocks, representing the 4 corners, the 4 sides and the middle.

The corners are drawn at the same size, the sides are stretched on the side direction and the middle is exp

The

NineTileBox

widget implements a

Container

using that standard. This pattern is also implemented as a component in the

NineTileBoxComponent

so that you can add this feature directly to your

FlameGame

. To get to know more about this, check the component docs

here

.

Here you can find an example of how to use it (without using the

NineTileBoxComponent

):

import

'package:flame/widgets/nine_tile_box.dart'

;

NineTileBox

.

asset

(

image:

image

,

// dart:ui image instance

tileSize:

16

,

// The width/height of the tile on your grid image

destTileSize:

50

,

// The dimensions to be used when drawing the tile on the canvas

child:

SomeWidget

(),

// Any Flutter widget

)

SpriteButton

¶

SpriteButton

is a simple widget that creates a button based on Flame sprites. This can be very useful when trying to cre

How to use it:

SpriteButton

.

asset

(

onPressed:

()

{

print

(

'Pressed'

);

},

label:

const

Text

(

'Sprite Button'

,

style:

const

TextStyle

(

color:

const

Color

(

0xFF5D275D

))),

sprite:

_spriteButton

,

pressedSprite:

_pressedSprite

,

)

SpriteWidget

¶

SpriteWidget

is a widget used to display a

Sprite

inside a widget tree.

This is how to use it:

SpriteWidget

.

asset

(

sprite:

yourSprite

,

anchor:

Anchor

.

center

,

)

SpriteAnimationWidget

¶

SpriteAnimationWidget

is a widget used to display

SpriteAnimations

inside a widget tree.

This is how to use it:

SpriteAnimationWidget

.

asset

(

animation:

_animation

,

playing:

true

,

anchor:

Anchor

.

center

,

)

# Tutorials

¶

The following are tutorials that are maintained with every release of Flame.

## Bare Flame game

? this tutorial focuses on setting up your environment for making a new Flame game. This ?initial state? is

## Klondike game tutorial

? in this tutorial, we will build the Klondike solitaire card game.

## Ember Quest Game Tutorial

? in this tutorial, we will build Ember Quest, a simple side-scrolling platformer.

## Space Shooter Game Tutorial

? in this tutorial, we will build Space Shooter, a classic top-down shooting game.

Flame SVG

¶

flame_svg provides a simple API for rendering SVG images in your game.

Installation

¶

Svg support is provided by the

flame_svg

bridge package, be sure to put it in your pubspec file to use it.

If you want to know more about the installation visit

flame_svg on pub.dev

.

How to use flame_svg

¶

To use it just import the

Svg

class from

'package:flame_svg/flame_svg.dart'

, and use the following snippet to render it on the canvas:

final

svgInstance

=

await

Svg

.

load

(

'android.svg'

);

final

position

=

Vector2

(

100

,

100

);

final

size

=

Vector2

(

300

,

300

);

svgInstance

.

renderPosition

(

canvas

,

position

,

size

);

or use the

SvgComponent

and add it to the component tree:

class

MyGame

extends

FlameGame

{

@override

Future

<

void

>

onLoad

()

async

{

final

svgInstance

=

await

Svg

.

load

(

'android.svg'

);

final

size

=

Vector2

.

all

(

100

);

final

position

=

Vector2

.

all

(

100

);

final

svgComponent

=

```
SvgComponent
(
size:
size
,
position:
position
,
svg:
svgInstance
,
);
add
(
svgComponent
);
}
}
```

flame_rive

¶

flame_rive

is a bridge library for using

rive

animations in your Flame game. Rive is a real-time interactive design and animation tool and you use it to

To use a file created by Rive in your game you need to add

flame_rive

to your pubspec.yaml, as can be seen in the

Flame Rive example

and in the pub.dev

installation instructions

.

How to use it

¶

First, start with adding the

animation.riv

file to the assets folder. Then load the artboard of the animation to the game using the

loadArtboard

method. After that, create the

StateMachineController

from the artboard and add a controller to it. Then you can create a

RiveComponent

using that artboard.

rive_example.dart

1

```
import
'dart:async'
;
2
3
import
'package:flame/events.dart'
;
4
import
'package:flame/game.dart'
;
5
import
'package:flame_rive/flame_rive.dart'
;
6
7
class
RiveExampleGame
extends
FlameGame
with
TapDetector
{
8
```

late

SMIInput

<

double

>?

levelInput

;

9

10

@override

11

Future

<

void

>

onLoad

()

async

{

12

final

skillsArtboard

=

13

await

loadArtboard

(

RiveFile

.

asset

(

'assets/skills.riv'

));

14

15

final

controller

=

StateMachineController

.

fromArtboard

(

16

skillsArtboard

,

17

"Designer's Test"

,

18

);

19

20

```
skillsArtboard
.
addController
(
controller
!
);
21
22
levelInput
=
controller
.
findInput
<
double
>
(
'Level'
);
23
24
add
(
RiveComponent
(
```

artboard:

skillsArtboard

,

size:

canvasSize

));

25

}

26

27

@override

28

void

onTap

()

{

29

super

.

onTap

();

30

if

(

levelInput

!=

null

)

{

31

levelInput

!

.

value

=

(

levelInput

!

.

value

+

1

)

%

3

;

32

}

33

}

34

}

Code

class

RiveExampleGame

extends

FlameGame

{

@override

Future

<

void

>

onLoad

()

async

{

final

skillsArtboard

=

await

loadArtboard

(

RiveFile

.

asset

(

'assets/skills.riv'

));

final

controller

=

StateMachineController

.

fromArtboard

(

skillsArtboard

,

"Designer's Test"

,

);

skillsArtboard

.

addController

(

controller

!

);

add

(

RiveComponent

(

artboard:

skillsArtboard

,

size:

Vector2

.

all

(

550

)));

}

}

You can use the controller to manage the state of animation. Check out the example for more information.

Full Example

¶

You can check an example

here

.