Introduction

Introduction

Céu-SDL supports the development of SDL applications in the programming language Céu.

Modes of Operation

Modes of Operation

A mode of operation specifies how Céu-SDL captures events from the environment (e.g., key presses) and redirects them to the Céu application.

Céu-SDL supports *polling*, *waiting*, and *frame-based* modes of operation.

Each mode of operation uses a different compilation flag:

```
$ make CEU_ARGS="-DCEU_SDL_MODE_POLLING"  CEU_SRC=<path-to-ceu-application>
$ make CEU_ARGS="-DCEU_SDL_MODE_WAITING"  CEU_SRC=<path-to-ceu-application>
$ make CEU_ARGS="-DCEU_SDL_MODE_FRAME=33" CEU_SRC=<path-to-ceu-application>
```

The frame-based mode also requires the desired period of the frame (in milliseconds).

The frame-based mode is the default mode of operation and is preset to 33 milliseconds.

The modes of operation are implemented in C and are part of Céu-SDL. Each mode is described in pseudo-code as follows.

Polling

The *polling mode* of Céu-SDL continually checks for SDL input events in an infinite loop:

```
int main (void) {
    ceu_start();
    while (<program-is-running>) {
        if (SDL_PollEvent(<...>)) {
            ceu_input(CEU_INPUT__*, <...>);     /* input SDL_QUIT, SDL_KEY*, SDL_MOUSE*, et
        }
        ceu_input(CEU_INPUT__NONE, <time>);     /* input async and timer */
        ceu_input(CEU_INPUT_SDL_REDRAW, <...>); /* input SDL_REDRAW after every input */
    }
    ceu_stop();
```

```
    return <program-escape-value>;
}
```

The inputs are polled on each loop iteration and changes are notified to the Céu application through `ceu_input` calls.

The polling mode uses *100%* of the CPU time.

Waiting

The *waiting mode* of Céu-SDL continually waits for SDL input events in an infinite loop:

```
int main (void) {
    ceu_start();
    while (<program-is-running>) {
        int timeout = (<program-has-pending-async>) ? 0 : <next-timer-in-ms>;
        if (SDL_WaitEventTimeout(<...>, timeout);
            ceu_input(CEU_INPUT__*, <...>);     /* input SDL_QUIT, SDL_KEY*, SDL_MOUSE*, etc
        }
        ceu_input(CEU_INPUT__NONE, <time>);     /* input async and timer */
        ceu_input(CEU_INPUT_SDL_REDRAW, <...>); /* input SDL_REDRAW after every input */
    }
    ceu_stop();
    return <program-escape-value>;
}
```

The inputs are polled on each loop iteration and changes are notified to the Céu application through `ceu_input` calls.

Frame

The *frame-based mode* of Céu-SDL continually waits for SDL input events in an infinite loop. The waiting period is limited to the next frame period:

```
int main (void) {
    ceu_start();
    while (<program-is-running>) {
        int timeout = (<program-has-pending-async>) ?
                        0 : MIN(<next-timer-in-ms>,<next-frame-in-ms>);
        if (<next-frame-expired>) {
            ceu_input(CEU_INPUT_SDL_DT, CEU_SDL_MODE_FRAME);
        }
        if (SDL_WaitEventTimeout(<...>, timeout);
            ceu_input(CEU_INPUT__*, <...>);     /* input SDL_QUIT, SDL_KEY*, SDL_MOUSE*, etc
        }
        ceu_input(CEU_INPUT__NONE, <time>);     /* input async and timer */
```

```
        ceu_input(CEU_INPUT_SDL_REDRAW, <...>); /* input SDL_REDRAW after every input */
    }
    ceu_stop();
    return <program-escape-value>;
}
```

The inputs are polled on each loop iteration and changes are notified to the Céu application through `ceu_input` calls.

The frame-based mode also provides the `CEU_SDL_DT` input, which is notified after each frame period expires.

Input Events

Input Events

Céu-SDL provides mappings to all SDL events:

https://wiki.libsdl.org/SDL_Event

`TODO: not all inputs have been mapped`

In addition, it provides the `SDL_REDRAW` and `SDL_DT` input events.

General

SDL_REDRAW

```
input _SDL_Renderer&&&& SDL_REDRAW;
```

Occurs in the end of every loop iteration in all modes of operation.

`TODO: payload`

SDL_DT

```
input int SDL_DT;
```

Occurs in the beginning of every loop iteration in the frame-based mode of operation.

The input value of type `int` is always equal to `CEU_SDL_MODE_FRAME`.

SDL_QUIT

```
input void SDL_QUIT;
```

SDL Reference: https://wiki.libsdl.org/SDL_QuitEvent

Keyboard

SDL_KEYDOWN

`input _SDL_KeyboardEvent&& SDL_KEYDOWN;`

SDL Reference: https://wiki.libsdl.org/SDL_KeyboardEvent

SDL_KEYUP

`input _SDL_KeyboardEvent&& SDL_KEYUP;`

SDL Reference: https://wiki.libsdl.org/SDL_KeyboardEvent

Mouse

SDL_MOUSEBUTTONDOWN

`input _SDL_MouseButtonEvent&& SDL_MOUSEBUTTONDOWN;`

SDL Reference: https://wiki.libsdl.org/SDL_MouseButtonEvent

SDL_MOUSEBUTTONUP

`input _SDL_MouseButtonEvent&& SDL_MOUSEBUTTONUP;`

SDL Reference: https://wiki.libsdl.org/SDL_MouseButtonEvent

SDL_MOUSEMOTION

`input _SDL_MouseMotionEvent&& SDL_MOUSEMOTION;`

SDL Reference: https://wiki.libsdl.org/SDL_MouseMotionEvent

Data Abstractions

Data Abstractions

SDL_Color

```
data SDL_Color with
    var u8 r;
    var u8 g;
    var u8 b;
    var u8 a;
```

```
end
```

SDL Reference: https://wiki.libsdl.org/SDL_Color

SDL_Point

```
data SDL_Point with
    var int x;
    var int y;
end
```

SDL Reference: https://wiki.libsdl.org/SDL_Point

SDL_Rect

```
data SDL_Rect with
    var int x;
    var int y;
    var int w;
    var int h;
end
```

SDL Reference: https://wiki.libsdl.org/SDL_Rect

SDL_Texture

```
data SDL_Texture with
    var& _SDL_Texture tex;
    var  int          width;
    var  int          height;
end
```

SDL Reference: https://wiki.libsdl.org/SDL_Texture

Code/Await Abstractions

Code/Await Abstractions

SDL_Init

Initializes Céu-SDL, creates a SDL window, and provides a renderer to the application.

`SDL_Init` terminates once `SDL_QUIT` occurs.

```
code/await SDL_Init (var _char&& title, var int width, var int height, var SDL_Color? bg)
                        -> (var& _SDL_Renderer ren)
                            -> void
```

- Parameters
  - `title`: title of the window
  - `width`: width of the window in pixels
  - `height`: height of the window in pixels
  - `bg`: background color of the window (optional)
- Initialization
  - `ren`: created renderer
- Return
  - terminates on `SDL_QUIT` and returns no value

`SDL_Init` performs a number of initializations on Céu-SDL:

- initializes SDL
- creates a window
- creates a renderer
- initializes the text and audio subsystems

If a `bg` is provided, `SDL_Init` fills the window background with the provided color on every `SDL_REDRAW`.

Example:

```
##include "sdl/sdl.ceu"

var& _SDL_Renderer ren; ;
watching SDL_Init("Rectangle", 300,300, SDL_Color(0xFF,0xFF,0x00,0xFF)) -> (&ren) do
    var SDL_Rect rect = val SDL_Rect(100,100 , 100,100);
    every SDL_REDRAW do
        _SDL_SetRenderDrawColor(&&ren, 0xFF,0x00,0x00,0xFF);
        _SDL_RenderFillRect(&&ren, (&&rect as _SDL_Rect&&));
    end
end

escape 0;
```

Draws a *100x100* red rectangle centered in a *300x300* yellow window.

SDL References: `SDL_Init`, `SDL_Quit`, `SDL_CreateWindow`, `SDL_DestroyWindow`, `SDL_CreateRenderer`, `SDL_DestroyRenderer`, `TTF_Init`, `TTF_Quit`, `Mix_OpenAudio`, `Mix_CloseAudio`, `SDL_SetRenderDrawColor`, `SDL_RenderClear`.

Note: all allocated SDL resources are automatically released on termination.

SDL_Open_Image

Opens an image file into a new texture.

```
code/await SDL_Open_Image (var& _SDL_Renderer ren, var _char&& path)
                            -> (var& SDL_Texture tex)
                                -> FOREVER
```

- Parameters
  - `ren`: rendering context
  - `path`: path to the image
- Initialization
  - `tex`: created texture
- Return
  - never terminates naturally

Example:

```
##include "sdl/sdl.ceu"

var& _SDL_Renderer ren; ;
watching SDL_Init("Image", 68,68, SDL_Color(0xFF,0xFF,0x00,0xFF)) -> (&ren) do
    var& SDL_Texture img;
    spawn SDL_Open_Image(&ren, "img.png") -> (&img);
    var SDL_Rect rect = val SDL_Rect(10,10 , img.width,img.height);
    every SDL_REDRAW do
        _SDL_RenderCopy(&&ren, &&img.tex, null, &&rect as _SDL_Rect&&);
    end
end

escape 0;
```

Draws `img.png` in a *68x68* yellow window.

SDL References: `IMG_LoadTexture`, `SDL_DestroyTexture`, `SDL_QueryTexture`.

Note: all allocated SDL resources are automatically released on termination.

SDL_Open_Font

Opens a `TTF` font file.

```
code/await SDL_Open_Font (var _char&& path, var int size)
                            -> (var& _TTF_Font font)
                                -> FOREVER
```

- Parameters
  - `path`: path to the font
  - `size`: size of the font

- Initialization
  - `font`: created font
- Return
  - never terminates naturally

Example:

```
##include "sdl/sdl.ceu"

var& _SDL_Renderer ren; ;
watching SDL_Init("Font 1", 220,60, SDL_Color(0xFF,0xFF,0x00,0xFF)) -> (&ren) do
    var& _TTF_Font font;
    spawn SDL_Open_Font("samples/Deutsch.ttf", 40) -> (&font);

    var& SDL_Texture txt;
    spawn SDL_New_Text(&ren, &font, "Hello World!", SDL_Color(0x00,0x00,0x00,0xFF))
            -> (&txt);
    var SDL_Rect rect = val SDL_Rect(10, 10, txt.width, txt.height);

    every SDL_REDRAW do
        _SDL_RenderCopy(&&ren, &&txt.tex, null, &&rect as _SDL_Rect&&);
    end
end

escape 0;
```

Draws a black `Hello World!` text in yellow window.

SDL References: `_TTF_OpenFont`, `_TTF_WasInit`, `_TTF_CloseFont`.

Note: all allocated SDL resources are automatically released on termination.

SDL_New_Text

Writes a text into a new texture.

```
code/await SDL_New_Text (var& _SDL_Renderer ren, var& _TTF_Font font, var _char&& text, var
                            -> (var& SDL_Texture tex)
                                -> FOREVER
```

- Parameters
  - `ren`: rendering context
  - `font`: text font
  - `text`: text to write
  - `color`: text color
- Initialization
  - `tex`: created texture
- Return

- never terminates naturally

Example:

```
##include "sdl/sdl.ceu"

var& _SDL_Renderer ren; ;
watching SDL_Init("Font 1", 220,60, SDL_Color(0xFF,0xFF,0x00,0xFF)) -> (&ren) do
    var& _TTF_Font font;
    spawn SDL_Open_Font("font.ttf", 40) -> (&font);

    var& SDL_Texture txt;
    spawn SDL_New_Text(&ren, &font, "Hello World!", SDL_Color(0x00,0x00,0x00,0xFF))
            -> (&txt);
    var SDL_Rect rect = val SDL_Rect(10, 10, txt.width, txt.height);

    every SDL_REDRAW do
        _SDL_RenderCopy(&&ren, &&txt.tex, null, &&rect as _SDL_Rect&&);
    end
end

escape 0;
```

Draws a black `Hello World!` text in yellow window.

SDL References: [`_TTF_RenderText_Blended`], [`_SDL_FreeSurface`], [`_SDL_CreateTextureFromSurface`], [`_SDL_DestroyTexture`], [`_SDL_QueryTexture`].

Note: all allocated SDL resources are automatically released on termination.

SDL_Open_Sound

Opens a sound file.

```
code/await SDL_Open_Sound (var _char&& path)
                          -> (var& _Mix_Chunk sound)
                              -> FOREVER
```

- Parameters
  - `path`: path to the sound
- Initialization
  - `sound`: created sound
- Return
  - never terminates naturally

Example:

```
##include "sdl/sdl.ceu"
```

```
var SDL_Color bg = val SDL_Color(0x00,0x00,0x00,0xFF);

var& _SDL_Renderer ren;
watching SDL_Init("Sound 1", 10,10, bg) -> (&ren)
do
    var& _Mix_Chunk sound;
    spawn SDL_Open_Sound("sound.wav") -> (&sound);
    every 1s do
        _Mix_PlayChannel(-1, &&sound, 0);
    end
end

escape 0;
```

Plays `sound.wav` every second.

SDL References: `_Mix_LoadWAV`, `_Mix_FreeChunk`.

Note: all allocated SDL resources are automatically released on termination.

License

License

Céu-Arduino is distributed under the MIT license reproduced below: