extern crate piston\_window ;

extern crate lib\_2048 ;

extern crate gfx\_device\_gl;

extern crate find\_folder;

extern crate gfx\_graphics;

extern crate gfx;

use piston\_window::\* ;

use lib\_2048::{ Seed, Grid, Evolution, Cell } ;

trait HasPow {

fn get\_pow(&self) -> u32;

fn get\_val(&self) -> usize;

}

impl HasPow for Cell{

fn get\_pow(& self) -> u32 {self.pow()}

fn get\_val(& self) -> usize { 2usize.pow(self.get\_pow()) }

}

fn handle\_input(

grid: & mut Grid, key: keyboard::Key

) -> Option<Evolution> {

use piston\_window::keyboard::Key::\* ;

match key {

Up | W => {

// println!("up") ;

Some( grid.up() )

},

Down | S => {

// println!("down") ;

Some( grid.down() )

},

Left | A => {

// println!("left") ;

Some( grid.left() )

},

Right | D => {

// println!("right") ;

Some( grid.right() )

},

\_ => None,

}

}

//displays grid in console

fn display\_grid(grid: & Grid) {

for row in grid.grid() {

for cell\_opt in row {

let val = match \* cell\_opt {

None => 0,

Some(ref c) => c.val(),

} ;

print!("| {: ^4}", val)

} ;

println!(" |")

}

}

//matches a value with a RGBA color.

fn color\_of<C:HasPow>(cell:&Option<C>)->[f32;4]{

match \*cell{

None =>[1.0,1.0,1.0,1.0],

Some(ref cell)=>

match cell.get\_val(){

2=>[1.0,0.0,1.0,1.0],

4 => [0.0,1.0,1.0,1.0],

8 => [1.0,0.0,0.0,1.0],

16 => [0.0,1.0,0.0,1.0],

32 => [0.0,0.0,1.0,1.0],

64 => [0.5,0.5,1.0,1.0],

128 => [1.0,1.0,0.5,1.0

256 => [0.40,1.0,0.25,1.0],

512 => [1.0,0.25,1.0,1.0],

1024 => [1.0,0.2,0.5,1.0],

2048 => [1.0,0.5,0.25,1.0],

\_=>[1.0,1.0,1.0,1.0],

},

}

}

//Returns the value of a cell

fn get\_val<C:HasPow>(cell:&Option<C>)->usize{

match \*cell{

None => 0,

Some(ref cell)=>cell.get\_val(),

}

}

fn main() {

let seed = Seed::mk() ;

let mut grid = Grid::mk(seed) ;

grid.spawn() ;

let mut window: PistonWindow = WindowSettings::new(

"2048", [512; 2]

).exit\_on\_esc(true).build().unwrap();

//variable to store the best move

let mut gameover = 0;

let mut pr = 0;

let mut increase = 0;

let assets = find\_folder::Search::ParentsThenKids(3, 3).for\_folder("assets").unwrap();

println!("{:?}", assets);

let ref font = assets.join("FiraSans-Regular.ttf");

let factory = window.factory.clone();

let mut glyphs = Glyphs::new(font, factory).unwrap();

while let Some(e) = window.next() {

let s = grid.score();

match e {

Event::Render(\_) =>

window.draw\_2d(&e, |c, g| {

clear([0.0, 0.0, 0.0, 1.0], g);

for x in 0..4{

for y in 0..4{

let col=y as f64;

let row=x as f64;

let color= color\_of(&grid.grid()[y][x]);

let number = if get\_val(&grid.grid()[y][x])!=0{

format!("{}",get\_val(&grid.grid()[y][x]))

}else{

format!("")

};

//draws cells

rectangle(color,

[10.0+row\*80.0+10.0\*row,10.0+col\*80.0+10.0\*col, 80.0,80.0], // rectangle (1,1)

c.transform, g);

//draws value of cell

let transform = c.transform.trans(40.0+row\*80.0+10.0\*row,60.0+col\*80.0+10.0\*col);

text::Text::new\_color([0.0, 0.0, 0.0, 1.0], 32).draw(

&number,

&mut glyphs,

&c.draw\_state,

transform, g

);

// Draws game title

text::Text::new\_color([1.0, 1.0, 1.0, 1.0], 50).draw(

"2048",

&mut glyphs,

&c.draw\_state,

c.transform.trans(80.0\*5.0,80.0\*6.0), g

);

if gameover == 1 {

//println!("game is over");

text::Text::new\_color([0.0, 0.0, 0.0, 1.0], 35).draw(

&format!("Game Over!!!"),

&mut glyphs,

&c.draw\_state,

c.transform.trans(80.0\*2.0,80.0\*6.0-90.0), g

);

}

text::Text::new\_color([1.0, 1.0, 1.0, 1.0], 20).draw(

&format!("Best Move: {}", pr),

&mut glyphs,

&c.draw\_state,

c.transform.trans(80.0\*2.0,80.0\*6.0-30.0), g

);

text::Text::new\_color([1.0, 1.0, 1.0, 1.0], 20).draw(

&format!("Score: {} ", grid.score()),

&mut glyphs,

&c.draw\_state,

c.transform.trans(80.0\*2.0,80.0\*6.0), g

);

text::Text::new\_color([1.0, 1.0, 1.0, 1.0], 20).draw(

&format!("Last Move: {}", increase),

&mut glyphs,

&c.draw\_state,

c.transform.trans(80.0\*2.0,80.0\*6.0-60.0), g

);

}

}

}),

Event::Input(

Input::Press(

Button::Keyboard(key)

)

) => {

match handle\_input(& mut grid, key) {

None =>(),

Some( evol ) => if evol.changed() {

grid.spawn() ;

display\_grid(& grid) ;

//let

increase = grid.score()- s;

if increase > pr {

pr = increase;

}

println!("+ {}", increase) ;

println!("Best: {}", pr) ;

println!("Score: {} ", grid.score());

println!("") ;

}

//detects if game is over

else{

let mut a = grid.clone();

if (a.up()).changed() == false {

//println!("cant go up");

if (a.down()).changed() == false {

// println!("cant go down");

if (a.left()).changed() == false{

// println!("cant go left");

if (a.right()).changed() == false {

gameover = 1;

println!("Game Over!")

// println!("cant go right, game over");

}

}

}

}

},

}

},

\_ => (),

}

}

}