Game Programming using Functional Reactive Programming

Andreas Granström

andreas.granstrom@gmail.com
`https://github.com/andreasg/FRPCopter/tree/tutorial`

July 21, 2015

Functional Reactive Programming

- Explicitly model time.
- Declarative style. Focus the programmer on *what* we want the computer to do instead of *how* to do it.
- The reactive parts are superficially similar to the Observer Pattern.

Model continuous behaviors and discrete events. Behaviors react to events.

Netwire

- Author: Ertugrul Söylemez
- `https://hackage.haskell.org/package/netwire`
- Model behaviors as `Wire`.
- Model events as `Event`.

`Wire`

`Wire` is Netwire's model for behavior.

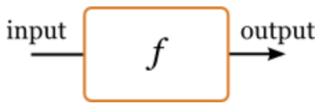
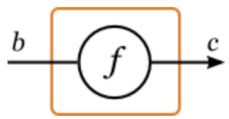
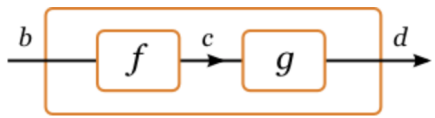
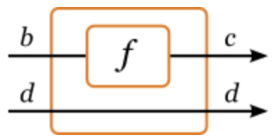
`Wire` instantiates

- Category
- Arrow
- Functor
- Applicative
- and more...

Produce/Inhibit

Wires are either *producing* or *inhibiting*.

Arrows[1]

Basic model	
Wrap normal function	
Arrow composition	
Partial application	

[1] https://www.haskell.org/arrows/

Arrow funs

```
1 arr :: Arrow a => (b -> c) -> a b c
2 (>>>) :: Arrow a => a b c -> a c d -> a b d
3 first :: Arrow a => a b c -> a (b, d) (c, d)
4
5 (&&&) :: Arrow a => a b c -> a b d -> a b (c, d)
```

Behaviors (continuous)

```
1 time :: Wire a t
2 for :: t -> Wire a a
3 arr (*2) :: Num a => Wire a a
```

Events (discrete)

```
1 hold :: Wire (Event a) a
2 periodic :: t -> Wire a (Event a)
3
4 stdNoiseR :: Random b => t -> (b, b) -> Int
5                       -> Wire a (Event b)
```

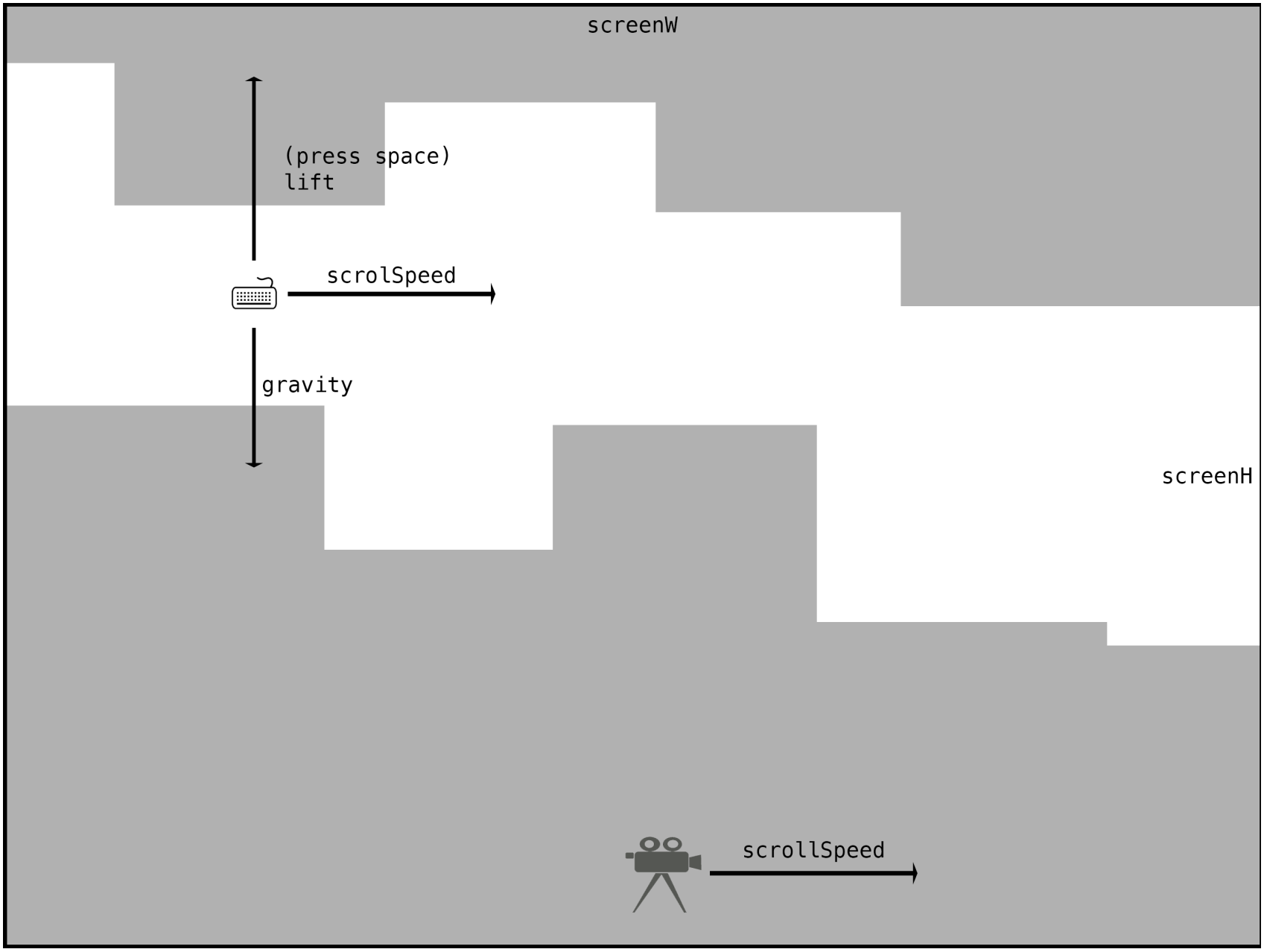
Switching

```haskell
(-->) :: Wire a b -> Wire a b -> Wire a b

a, b :: Wire a

a --> b -- a until inhibit, then b
```

Goal


screenW
(press space)
lift
scrolSpeed
gravity
screenH
scrollSpeed