Escaping State-Machine Hell with a Pseudo-Synchronous DSL

Sydney Rust Meetup 2022-10-10

by Stephen Sherratt (@gridbugs)

stephen@sherra.tt

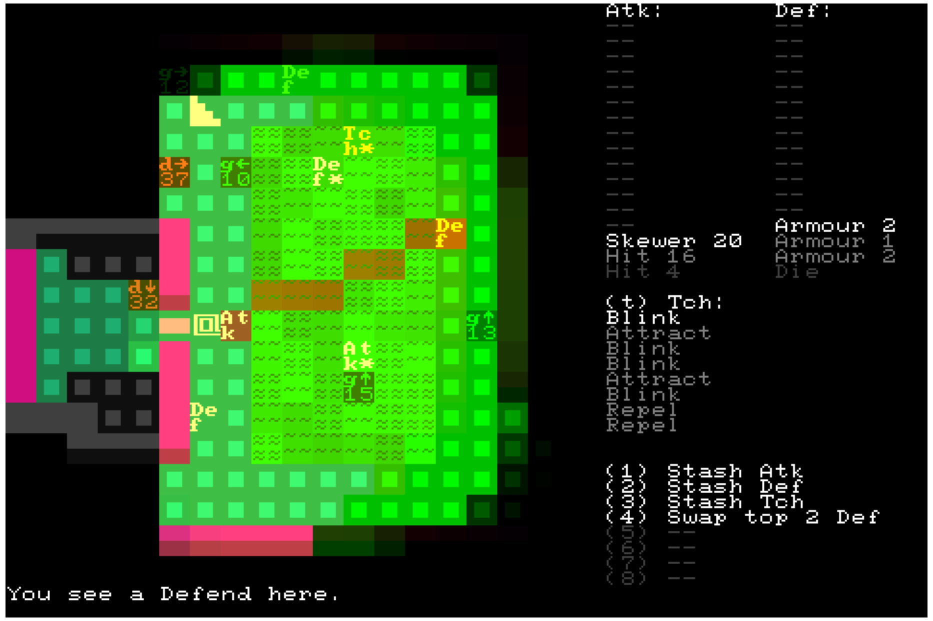
github.com/gridbugs

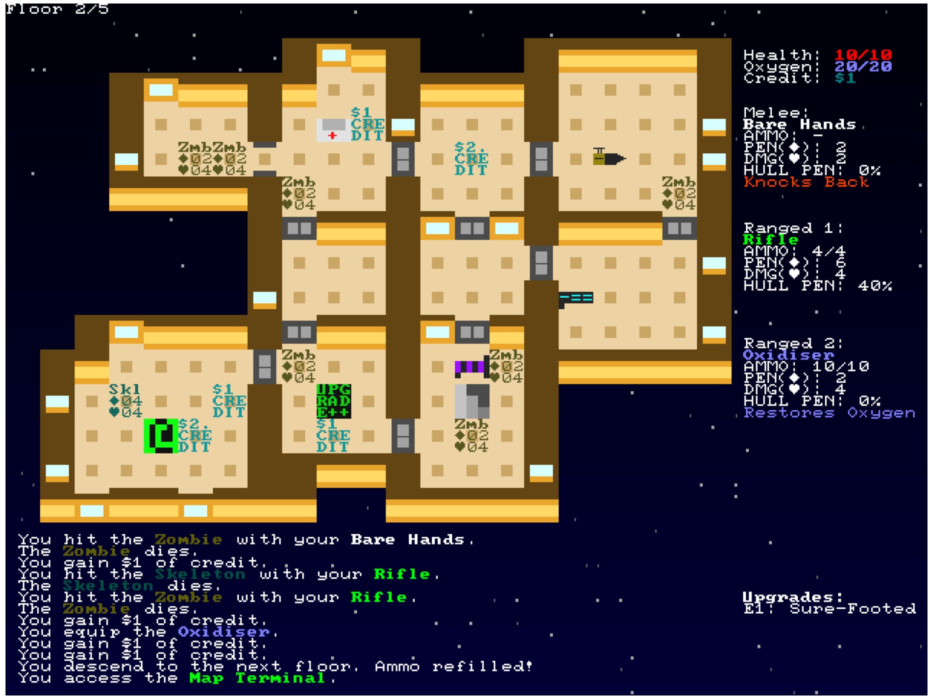
twitch.tv/gridbugs

gridbugs.itch.io

gridbugs.org

twitter.com/gridbugstv





```
Floor 2/5

Health: 10/10
Oxygen: 20/20
Credit: $1

Melee:
Bare Hands
AMMO: -
PEN(♦): 2
DMG(♥): 2
HULL PEN: 0%
Knocks Back

Ranged 1:
Rifle
AMMO: 4/4
PEN(♦): 6
DMG(♥): 4
HULL PEN: 40%

Ranged 2:
Oxidiser
AMMO: 10/10
PEN(♦): 2
DMG(♥): 4
HULL PEN: 0%
Restores Oxygen

You hit the Zombie with your Bare Hands.
The Zombie dies.
You gain $1 of credit.
You hit the Skeleton with your Rifle.
The Skeleton dies.
You hit the Zombie with your Rifle.            Upgrades:
The Zombie dies.                               E1: Sure-Footed
You gain $1 of credit.
You equip the Oxidiser.
You gain $1 of credit.
You gain $1 of credit.
You descend to the next floor. Ammo refilled!
You access the Map Terminal.
```

Chargrid

Crates for building cross-platform text-UIs

```
use chargrid;
```

Supports

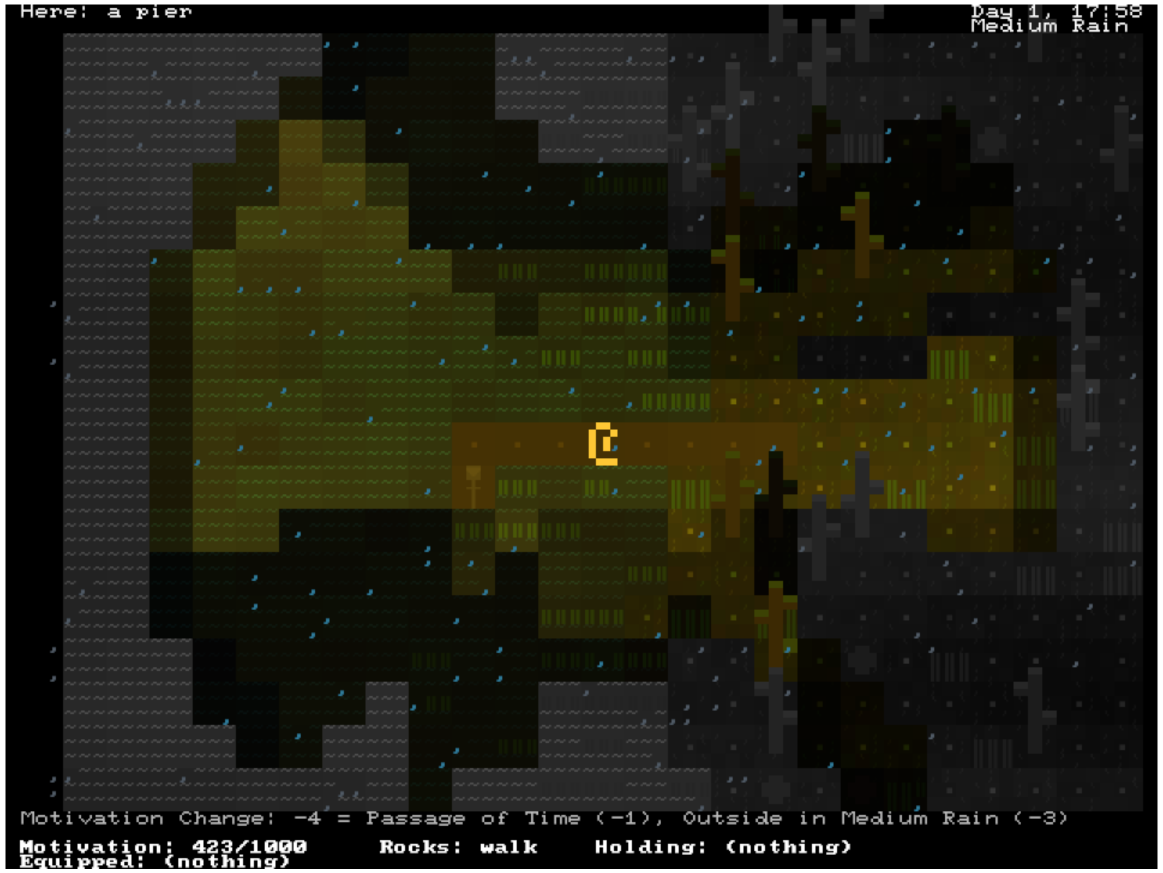
- Graphical Windows
- ANSI Terminals
- Web Browsers

Lowest common denominator control flow:

- **Tick-based applications only!**

Point of this talk

It's tedious to program in a tick-based environment. Here's an EDSL that can help!

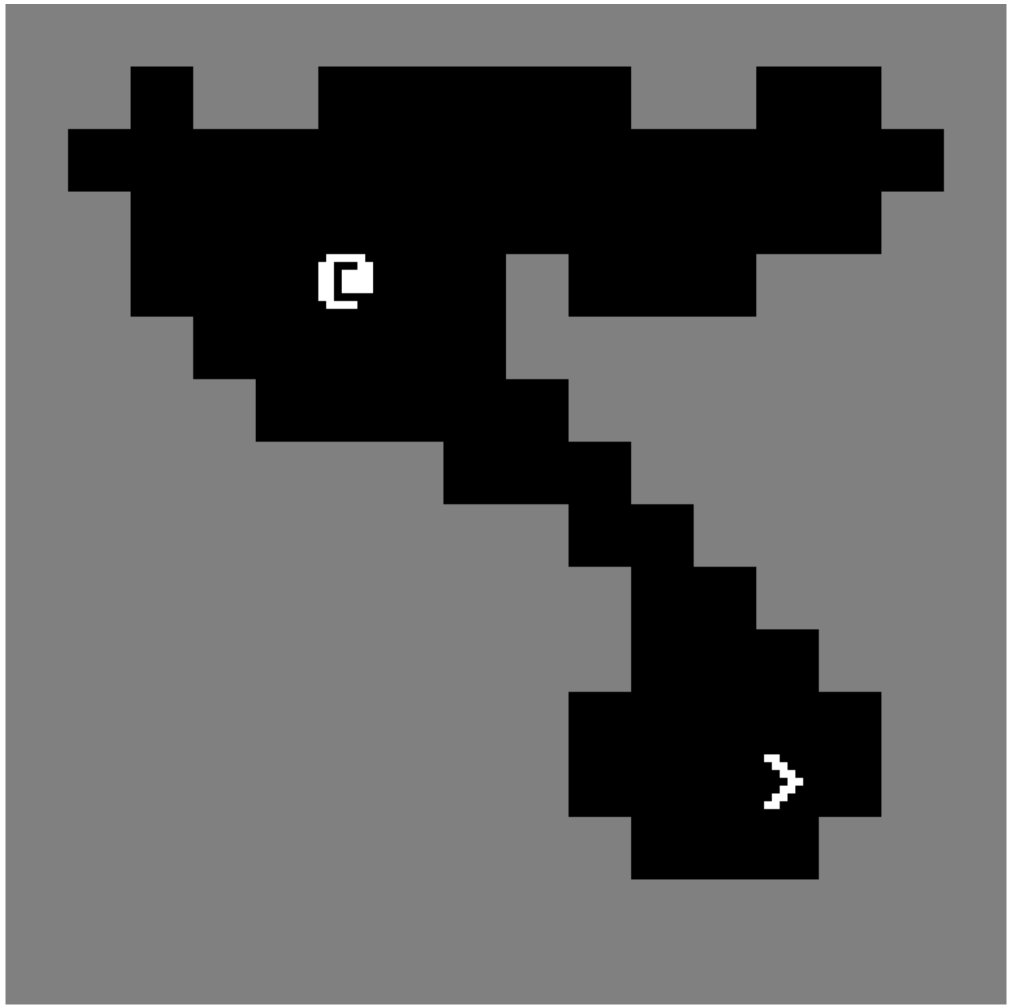


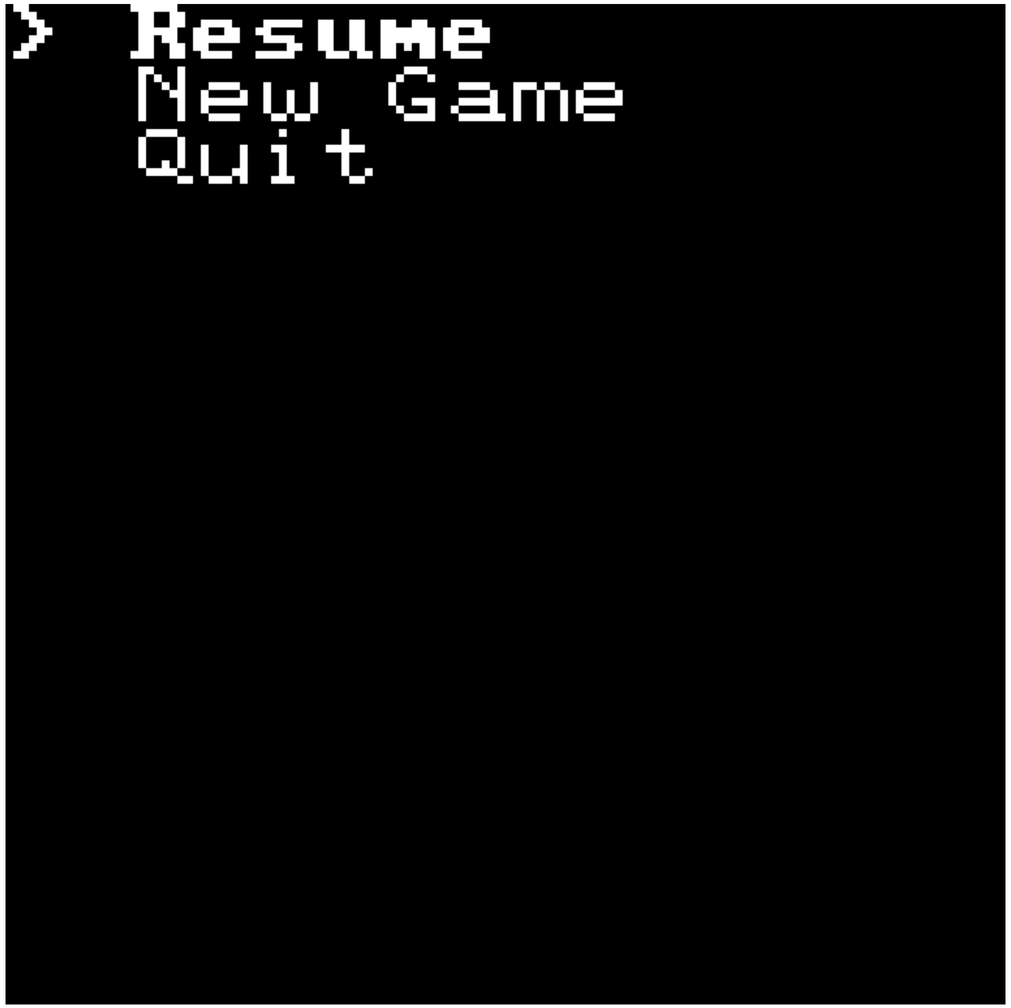
```
Here: a pier                                                  Day 1, 17:58
                                                              Medium Rain

Motivation Change: -4 = Passage of Time (-1), Outside in Medium Rain (-3)
Motivation: 423/1000      Rocks: walk     Holding: (nothing)
Equipped: (nothing)
```

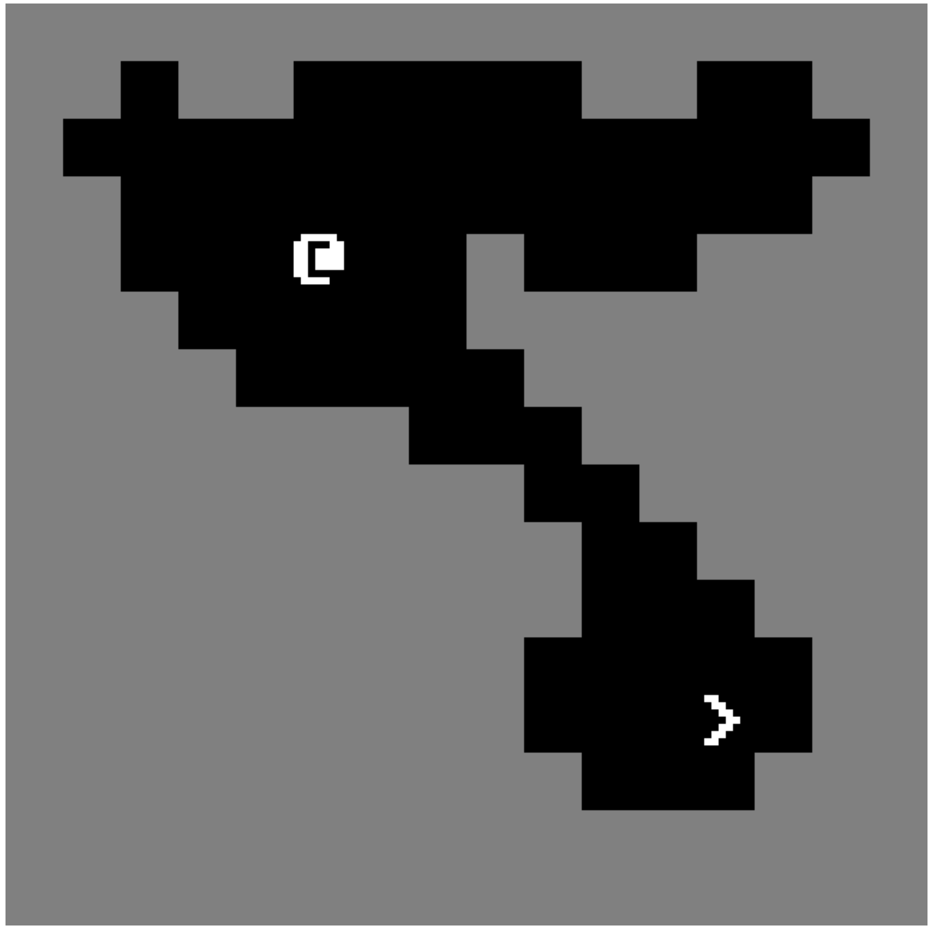
Chargrid Components

```rust
pub trait Component {

    /// type of yielded values
    type Output;

    /// render to frame buffer
    fn render(&self, fb: &mut FrameBuffer);

    /// process events and yield values
    fn update(&mut self, event: Event) -> Self::Output;
}
```

Little Example Game

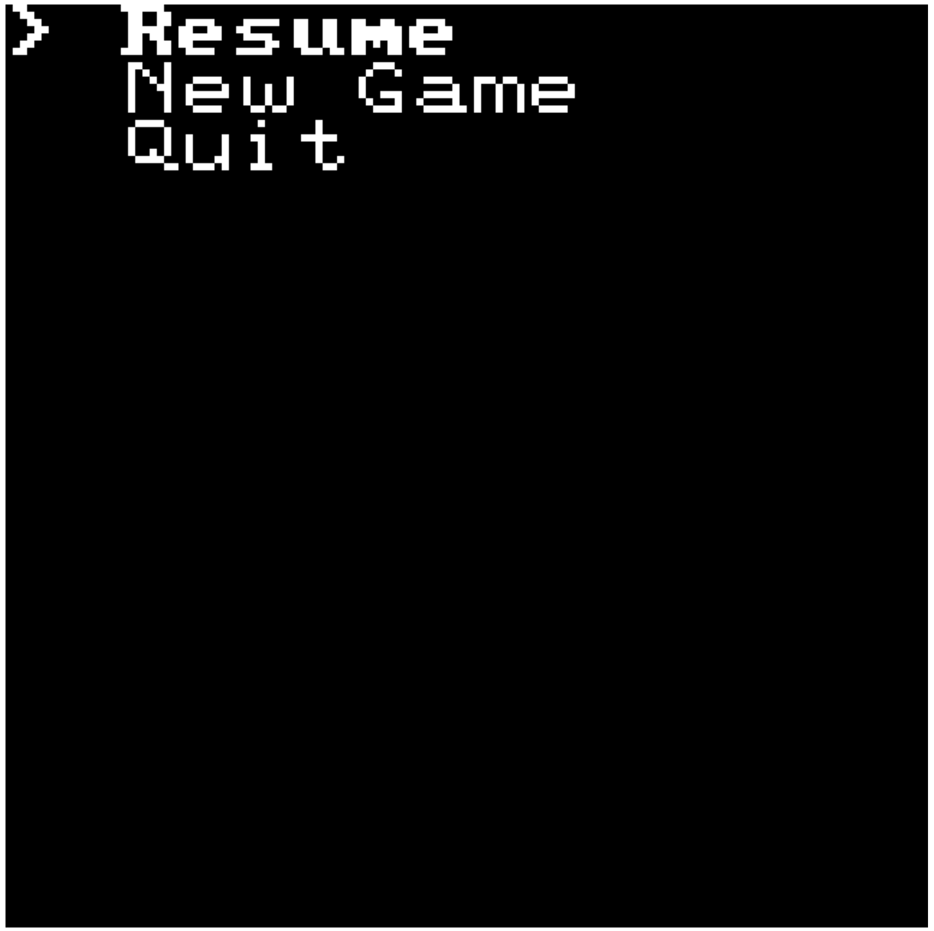






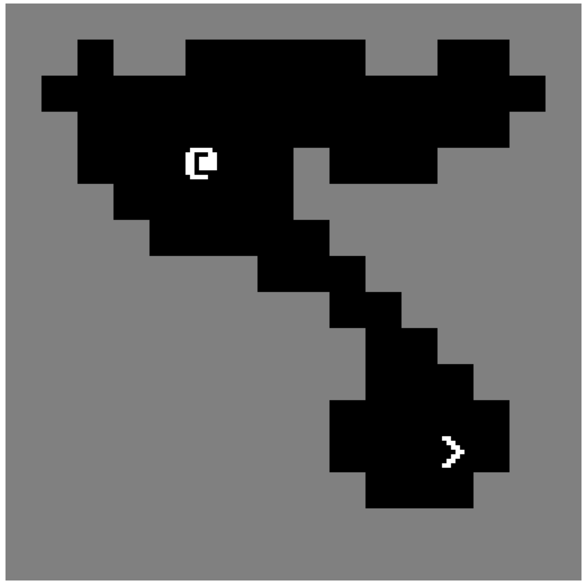
Game Component

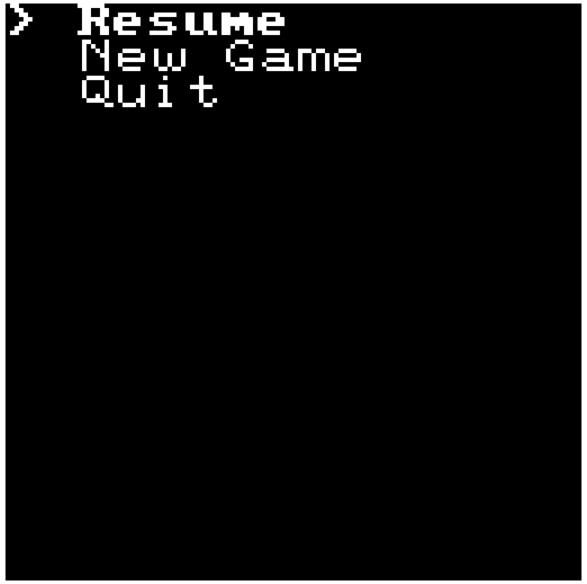
```
enum GameOutput {
    GameOver,
    EscapeWasPressed,
}

impl Component for Game {

    type Output = Option<GameOutput>;

    fn render(&self, fb: FrameBuffer) {
        // render the current frame
    }

    fn update(&mut self, event: Event) -> Self::Output {
        // pass the event to the game engine
    }
}
```



Menu Component

```rust
enum MenuSelection {
    Resume,
    NewGame,
    Quit,
}

impl Component for Menu {

    type Output = Option<MenuSelection>;

    fn render(&self, fb: FrameBuffer) {
        // render the menu
    }

    fn update(&mut self, event: Event) -> Self::Output {
        // update the menu state, returning `Some(...)`
        // if a selection was finalized
    }
}
```





App Component

```rust
enum App {
    Game(Game),
    Menu(Menu),
}

impl Component for App {
    type Output = ();

    fn render(&self, fb: FrameBuffer) {
        match self {
            Self::Game(game) => game.render(fb),
            Self::Menu(menu) => menu.render(fb),
        }
    }

    fn update(&mut self, event: Event) -> Self::Output {
        match self {
            Self::Game(game) => game.update(event),
            Self::Menu(menu) => menu.update(event),
        };
    }
}
```

The Dream

```rust
let mut game = Game::new();

loop {

    match game.run() {
        GameOver => break,
        EscapeWasPressed => (),
    }

    match Menu::new().run() {
        Resume => (),
        NewGame => game = Game::new(),
        Quit => break,
    }
}
```

With Chargrid `control_flow` API

```
loop_(Game::new(), |game| {
    cf(game).and_then(|game_output| match game_output {
        GameOver => LoopControl::Break(()),
        EcapeWasPressed => cf(Menu::new()).map(|selection| match selection {
            Resume => LoopControl::Continue(()),
            NewGame => {
                *game = Game::new();
                LoopControl::Continue(())
            }
            Quit => LoopControl::Break(()),
        })
    })
})
```

Peeking Inside

```rust
struct CF<C: Component>(C);

fn cf<C: Component>(c: C) -> CF<C> { CF(c) }

impl<C: Component> Component for CF<C> { ... }

impl<T, C: Component<Output = Option<T>>> {
    fn and_then<U, C2: Component<Output = U>, F: FnOnce(T) -> C2>(self, f: F)
    -> CF<AndThen<C, C2, F>
    {
        ...
    }

    ...
}

enum AndThen<C1, C2, F> { ... }
impl<C1, C2, F> Component for AndThen<C1, C2, F> { ... }
...
```

More Chargrid

My github: https://github.com/gridbugs

My gamedev blog: https://gridbugs.org

Occasional programming livestreams:

https://twitch.tv/gridbugs

Some games made with chargrid:

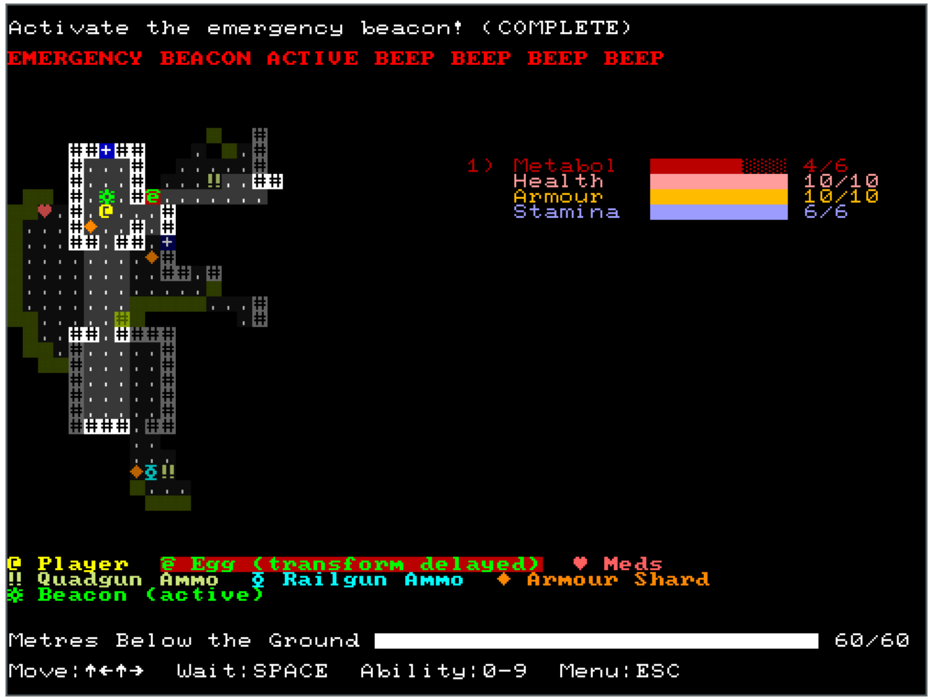
https://gridbugs.itch.io

Minimal re-implementation for demo purposes:

https://github.com/gridbugs/chargrid-talk

Chargrid roguelike tutorial:

https://gridbugs.org/roguelike-tutorial-2020/



```
Activate the emergency beacon! (COMPLETE)
EMERGENCY BEACON ACTIVE BEEP BEEP BEEP BEEP

                         1) Metabol    4/6
                            Health    10/10
                            Armour    10/10
                            Stamina    6/6

@ Player   @ Egg (transform delayed)   ♥ Meds
!! Quadgun Ammo   ☿ Railgun Ammo   ◆ Armour Shard
☼ Beacon (active)

Metres Below the Ground                        60/60
Move:↑←↑→   Wait:SPACE   Ability:0-9   Menu:ESC
```