Entity Component System

[0] ECS merupakan sebuah teknik yang biasanya digunakan dalam pembuatan game yang bertujuan untuk memisahkan data dengan prilaku, ECS merupakan singkatan dari 3 kata, yaitu Entity Component System, ECS hanya digunakan untuk, dalam contoh kali ini saya menggunakan implementasi ECS yang bernama Apecs yang ada di bahasa Haskell.

Entity

Entity dalam ECS yaitu adalah sebuah id yang nantinya akan diberikan sebuah component, berikut cara membuat id dalam Apecs

```
newEntity_ ( Ball
           , Color (RTC.Color 72 207 255 255)
           , Radius 20
           , Position (V2 (fromIntegral windowWidth/2)
                          (fromIntegral windowHeight/2))
           , Velocity (V2 300 200)
           )
```

dalam contoh di atas kita membuat sebuah entity dengan component Ball, Color, Radius, Position dan Velocity, dalam ECS component lah yang terpenting, karena di ECS kita melakukan iterasi pada component karena setiap entity yang berbeda memiliki component yang berbeda.

Component

pada component ini lah hal yang terpenting, karena untuk mengubah suatu component dalam entity kita bisa melakukan iterasi pada component dan mengubahnya, berikut cara membuat component dan melakukan perubahan pada Apecs

```
data Ball = Ball deriving Show
instance Component Ball where
  type Storage Ball = Map Ball

newtype Color = Color RL.Color
instance Component Color where
  type Storage Color = Map Color

newtype Radius = Radius Float
instance Component Radius where
  type Storage Radius = Map Radius

newtype Position = Position (V2 Float)
instance Component Position where
  type Storage Position = Map Position

newtype Velocity = Velocity (V2 Float)
instance Component Velocity where
  type Storage Velocity = Map Velocity
```

selaint component kita diharuskan juga membuat sebuah world untuk penempatan component, jadi component hanya berada di word yang kita tentukan

```
makeWorld "World" [ ''Ball
                  , ''Color
                  , ''Radius
                  , ''Position
                  , ''Velocity
                  ]
```

System

untuk memodifikasi component kita menggunakan cmap atau cmapM_, perbedaanya adalah penggunaan monad, disinilah kita merubah sebuah prilaku pada entity, disini juga kita bisa menggunakan rendering enggine seperti [1] Raylib & [2] SDL atau kita juga bisa menggunakan graphic API seperti [3] OpenGL, [4] DirecX, [5] Metal atau [6] Vulkan

```
cmap $ \(Ball, Position x, Velocity y, Radius r) ->
  wallCollide (Position x, Velocity y, Radius r) (V2 sw sh)

wallCollide :: (Position, Velocity, Radius) -> V2 Float -> Velocity
wallCollide ( Position (V2 x y)
            , Velocity (V2 vx vy)
            , Radius r
            ) (V2 sw sh) =
  Velocity (V2 (if x-r-5 < 0 || x > sw-r-10 then -vx else vx)
               (if y-r-5 < 0 || y > sh-r-10 then -vy else vy))
```

dalam system ini juga kita melakukan rendering, disinilah kita bisa menggunakan varian monad dari cmap, yaitu cmapM_

```
render :: System World ()
render =
  do
    liftIO $ RC.beginDrawing
          >> RC.clearBackground (RTC.Color 240 173 194 255)

    cmapM_ $ \(Ball, Position x, Radius r, Color c) ->
      liftIO $ RCS.drawCircleV (v2cast x) (realToFrac r) c

    liftIO RC.endDrawing
```

disini saya menggunakan Raylib sebagai rendering system, kamu bisa melikat contohnya di aplikasi yang saya buat bernama [7] eternal-pong.

Referensi

[0] *Plan 9 from Bell Labs Overview* https://github.com/SanderMertens/ecs-faq

[1] *Raylib* https://www.raylib.com

[2] *SDL* https://libsdl.org

[3] *OpenGL* https://www.opengl.org

[4] *DirecX* https://learn.microsoft.com/en-us/windows/win32/directx

[5] *Metal* https://developer.apple.com/metal

[6] *Vulkan* https://www.vulkan.org

[7] *eternal-pong* https://codeberg.org/aerphanas/eternal-pong