Gamedev

Dagens agenda

“The Art of Juice” - kunsten å legge til “juice”, eller “game feel”

Simpel 2D Platform Shooter - Mega Man, men mye dårligere

Lære noen av de vanligste patterns og teknikker innen gamedev

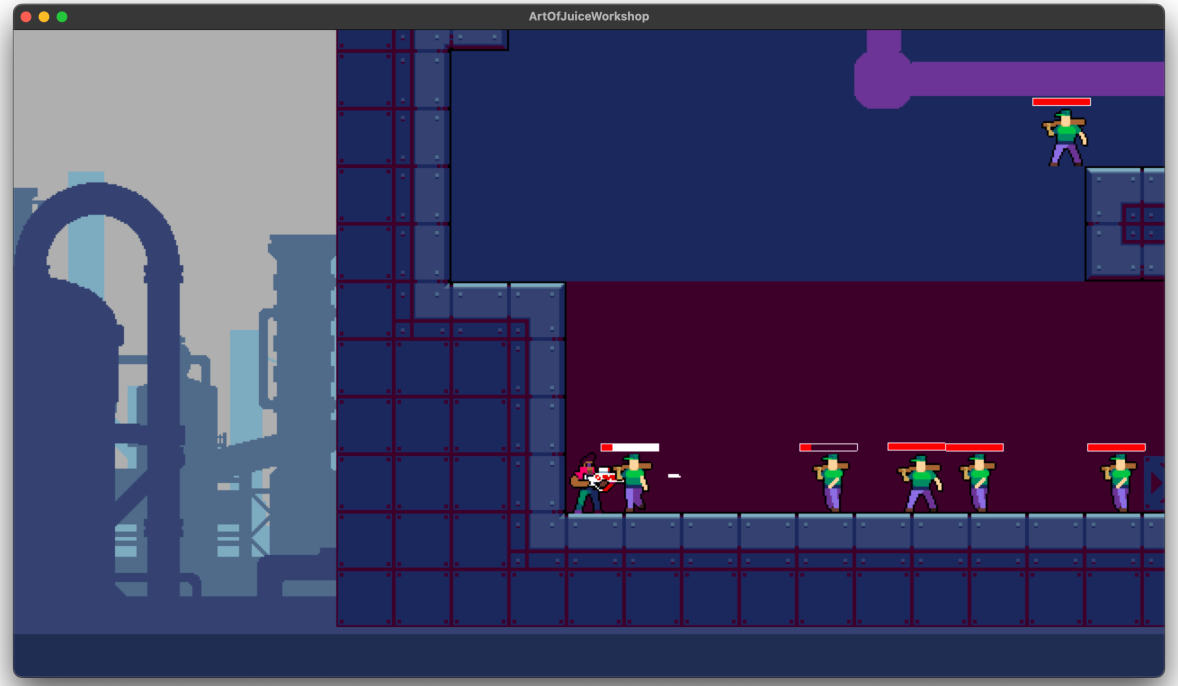
Lære litt om hva som får et spill til å føles bra å spille

Ha et repo man kan tukle videre med!

Agenda:

Jeg yapper litt

Dere får repoet, konkrete oppgaver med fasit



Ressurser

"The art of screenshake" Jan Willem Nijman ("Vlambeer")

https://www.youtube.com/watch?v=AJdEqssNZ-U

"Juice it or lose it" Martin Jonasson ("grapefrukt")

https://www.youtube.com/watch?v=Fy0aCDmgnxg

"Game Programming Patterns" (bok) av Robert Nystrom.

Hele boka er gratis på
https://gameprogrammingpatterns.com/contents.html

Alle disse er lenket til i README.md!

“Game loop” pattern

https://gameprogrammingpatterns.com/game-loop.html

A **game loop** runs continuously during gameplay. Each turn of the loop, it **processes user input** without blocking, **updates the game state**, and **renders the game**. It tracks the passage of time to **control the rate of gameplay**.

```
double lastTime = getCurrentTime();
while (true)
{
  double current = getCurrentTime();
  double elapsed = current - lastTime;
  processInput();
  update(elapsed);
  render();
  lastTime = current;
}
```

```
override fun update(delta: Float) {
    enemy.position.x += enemy.speed * delta
}
```

“Update method” pattern

https://gameprogrammingpatterns.com/update-method.html

The **game world** maintains a **collection of objects**. Each object implements an **update method** that **simulates one frame** of the object’s behavior. Each frame, the game updates every object in the collection.

“Component” pattern

https://gameprogrammingpatterns.com/component.html

A **single entity spans multiple domains**. To keep the domains isolated, the code for each is placed in its own **component class**. The entity is reduced to a simple **container of components**.

Basic component

```
class MyComponent : Component() {

}
```

Update

```
class MyComponent : Component() {
    override fun update(delta: Float) {
        entity.position.x += entity.velocity.x * delta
    }
}
```

Update og render

```
class MyComponent : Component() {
    override fun update(delta: Float) {
        entity.position.x += entity.velocity.x * delta
    }

    override fun render(batch: SpriteBatch, shape: ShapeRenderer) {
        shape.use(ShapeRenderer.ShapeType.Filled) {
            it.circle(entity.position.x, entity.position.y, 10f)
        }
    }
}
```

Konstruktør, state og metoder

```
class HealthComponent(val maxHealth: Float) : Component() {
    var health = maxHealth
        private set

    fun damage(amount: Float) {
        health -= amount
        if (health <= 0) {
            health = 0
        }
    }

    fun heal(amount: Float) {
        health += amount
        if (health > maxHealth) {
            health = maxHealth
        }
    }
}
```

Interagere med andre komponenter

```
class ConstantDamageComponent(val damagePerSecond: Float) : Component() {
    override fun update(delta: Float) {
        val healthComponent = entity.getComponent<HealthComponent>()
        healthComponent.damage(damagePerSecond * delta)
    }
}
```

Interagere med andre komponenter

```
class ConstantDamageComponent(val damagePerSecond: Float) : Component() {
    override fun update(delta: Float) {
        val healthComponent = getComponent<HealthComponent>()
        healthComponent.damage(damagePerSecond * delta)
    }
}
```

Interagere med andre komponenter

```
class ConstantDamageComponent(val damagePerSecond: Float) : Component() {
    private val healthComponent: HealthComponent by getComponentLazy()

    override fun update(delta: Float) {
        healthComponent.damage(damagePerSecond * delta)
    }
}
```

Callbacks

```
class HealthComponent(maxHealth: Float) : Component() {
    var health: Float = maxHealth
        private set
    val onDeath = Event()
    val onDamage = Event1<Float>()

    fun damage(amount: Float) {
        onDamage.invoke(amount)
        health -= amount
        if (health <= 0f) {
            health = 0f
            onDeath.invoke()
        }
    }
}
```

Callbacks

```
class JumpWhenHurtComponent : Component() {
    override fun lateInit() {
        val healthComponent = getComponent<HealthComponent>()
        healthComponent.onDamage += ::onDamage
        healthComponent.onDeath += { entity.velocity.setZero() }
    }

    private fun onDamage(float: Float) {
        entity.velocity.y += float * 10f
    }
}
```

Lage nye entities

```
spawnEntity(Vector2(0f, 0f)) {
    +HealthComponent(120f)
    +ConstantDamageComponent(10f)
    +JumpWhenHurtComponent()
}
```

Systemer

```
class EnemySoundsComponent : Component() {
    private val soundSystem: SoundSystem by getSystemLazy()

    override fun lateInit() {
        val healthComponent = getComponent<HealthComponent>()
        healthComponent.onDeath += { soundSystem.playSound("enemy_death") }
    }
}
```

Spørsmål?

https://github.com/sondremb/art-of-the-juice

Slides ligger i repo 😘

Kodetid!!!

* Klon repo
* Åpne og begynn å lese README.md
* Spill spillet bittelitt
* Juster på noen parametere
* Begynn på oppgavene