Software Engineering Principles

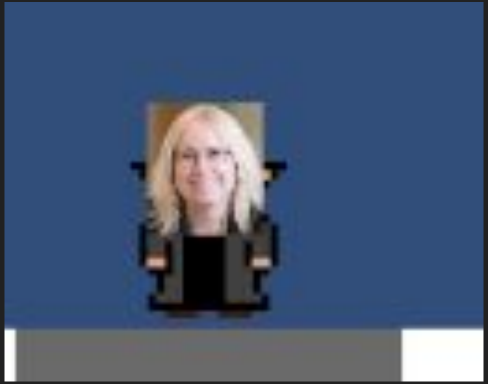
1. PUBLIC - Software engineers shall act consistently with the public interest.
2. CLIENT AND EMPLOYER - Software engineers shall act in a manner that is in the best interests of their client and employer consistent with the public interest.
3. PRODUCT - Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.
4. JUDGMENT - Software engineers shall maintain integrity and independence in their professional judgment.
5. MANAGEMENT - Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.
6. PROFESSION - Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.
7. COLLEAGUES - Software engineers shall be fair to and supportive of their colleagues.
8. SELF - Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

Copyright

Utilized asset: BC headshot for BC mode in game.

The four factors judges consider in fair use are:

1. Purpose and character of your use (nonprofit/educational vs. commercial) (transformative)
   a. My use is transformative: adding new context to the image rather than substituting the original.
2. Nature of the copyrighted work (factual vs. creative)
   a. This image has been published by the photographer and U of I.
   b. Dr. BC's face is factual.
3. Amount and substantiality of the portion taken.
   a. This image is not the main component of the game.
4. Effect of the use upon the potential market.
   a. This use does not compromise the market of photography.
   b. Video games and headshots are a different market completely.



Adaptive Color Player Health Bar (Prefab)

Features:

- Unity UI and Sprite-based 2D Health Bar.
- Easy to use and customize.
- Health Bar color and fill changes smoothly depending on health percent.

How to use:

- Add HealthBar.cs and insert HealthBarCanvas Prefab to your project and resize as needed.
- Add Health Bar component to any player and simply call the HealthBar's decreaseHealth(float damage) and increaseHealth(float damage) where the player is attacked to decrease health and where they are healed to increase health, respectively.

Quick Start sample in Player control script:

```
[SerializeField] private HealthBar healthBar;
private float maxHealth, curHealth;
void Start(){
    maxHealth=100;
    curHealth=maxHealth;
}
void Update(){
    // for testing health bar
    if(Input.GetKeyDown(KeyCode.Return))
    {
        Debug.Log("Taking damage");
        curHealth-=20;
        healthBar.UpdateHealthBar(maxHealth,curHealth);
        if(curHealth<=0) Die();
    }
    if(Input.GetKeyDown(KeyCode.Tab))
    {
        Debug.Log("Healing");
        if((curHealth+modifier)<=maxHealth) curHealth+=modifier;
        else if ( (curHealth<maxHealth) && (curHealth+modifier>maxHealth) ) curHealth=maxHealth;
        healthBar.UpdateHealthBar(maxHealth,curHealth);
    }
}
```

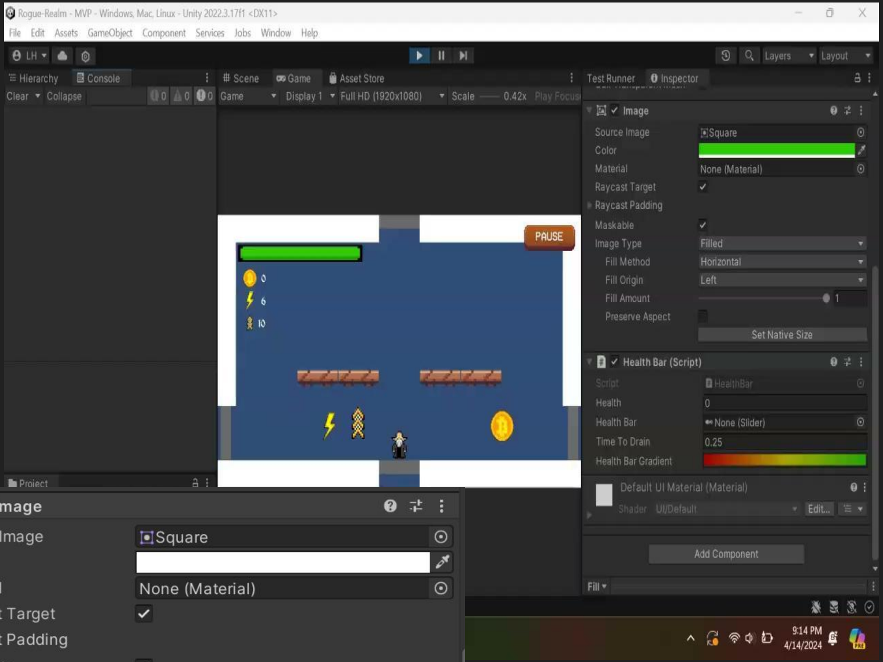

Rogue-Realm - MVP - Windows, Mac, Linux - Unity 2022.3.17f1 <DX11>
File Edit Assets GameObject Component Services Jobs Window Help
Hierarchy Console Scene Game Asset Store Test Runner Inspector
PAUSE
Add Component

Image
Source Image: Square
Color
Material: None (Material)
Raycast Target: ✓
Raycast Padding
Maskable: ✓
Image Type: Filled
Fill Method: Horizontal
Fill Origin: Left
Fill Amount: 1
Preserve Aspect
Set Native Size

Health Bar (Script)
Script: HealthBar
Health: 0
Health Bar: None (Slider)
Time To Drain: 0.25
Health Bar Gradient

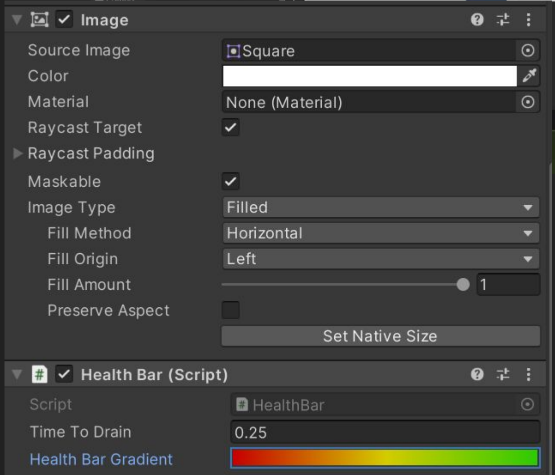


Image
Source Image: Square
Color
Material: None (Material)
Raycast Target: ✓
Raycast Padding
Maskable: ✓
Image Type: Filled
Fill Method: Horizontal
Fill Origin: Left
Fill Amount: 1
Preserve Aspect
Set Native Size

Health Bar (Script)
Script: HealthBar
Time To Drain: 0.25
Health Bar Gradient

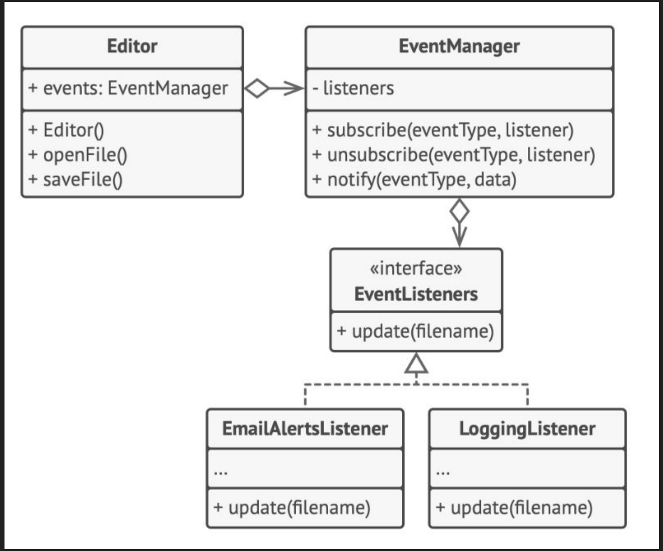
Observer Pattern

How it works:

- Observers (in my case UI elements) subscribe to a subject (Player)
- Upon specified conditions (pick ups), the subject notifies all observers at once and Observer.cs defines handlers for observer behavior

My Implementation:

Sample Overview:

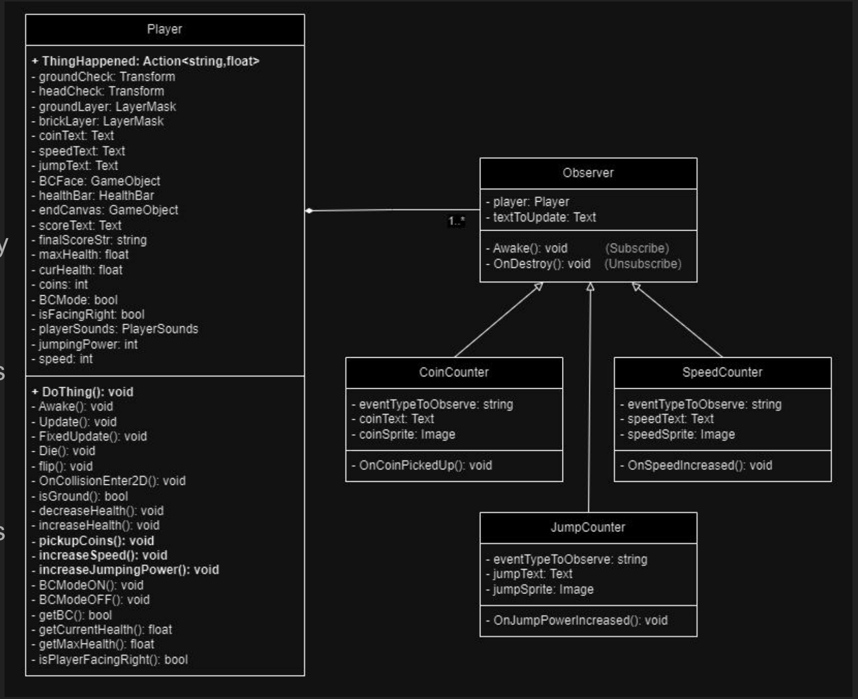


Pros:

- New observers are easy to implement (scalable)
- Decoupling of Subject and Observers
- Reusability of observers
- Real-time updates

Cons:

- Notifies observers in random order
- Overhead (all observers are notified every call)
- With decoupling comes complexity


Player
+ ThingHappened: Action<string,float>
- groundCheck: Transform
- headCheck: Transform
- groundLayer: LayerMask
- brickLayer: LayerMask
- coinText: Text
- speedText: Text
- jumpText: Text
- BCFace: GameObject
- healthBar: HealthBar
- endCanvas: GameObject
- scoreText: Text
- finalScoreStr: string
- maxHealth: float
- curHealth: float
- coins: int
- BCMode: bool
- isFacingRight: bool
- playerSounds: PlayerSounds
- jumpingPower: int
- speed: int
+ DoThing(): void
- Awake(): void
- Update(): void
- FixedUpdate(): void
- Die(): void
- flip(): void
- OnCollisionEnter2D(): void
- isGround(): bool
- decreaseHealth(): void
- increaseHealth(): void
- pickupCoins(): void
- increaseSpeed(): void
- increaseJumpingPower(): void
- BCModeON(): void
- BCModeOFF(): void
- getBC(): bool
- getCurrentHealth(): float
- getMaxHealth(): float
- isPlayerFacingRight(): bool
1..*
Observer
- player: Player
- textToUpdate: Text
- Awake(): void (Subscribe)
- OnDestroy(): void (Unsubscribe)
CoinCounter
- eventTypeToObserve: string
- coinText: Text
- coinSprite: Image
- OnCoinPickedUp(): void
SpeedCounter
- eventTypeToObserve: string
- speedText: Text
- speedSprite: Image
- OnSpeedIncreased(): void
JumpCounter
- eventTypeToObserve: string
- jumpText: Text
- jumpSprite: Image
- OnJumpPowerIncreased(): void