Ders 5

Oyun Programlama

Tasarım Örüntüleri

Gözlemci (Observer) Örüntüsü

Nedir?

- En sık kullanılan örüntülerden biri
- Model-View-Controller mimarisinin altyapısı
- Java ve .Net gibi dillerde direk gömülü geliyor

Başarım Elde Edildi

- Başarım tabanlı bir oyun yaptığımızı düşünelim
    - 100 Maymun Öldür -> İnsanlar Gezegeni Başarımı
    - Bir Köprüden Düş -> İntihar Başarımı
- Bu başarımların hemen hemen kodun birçok yerinde kontrol edilmesi gerekir
- Başarımla ilgili kod, diğer kodların her yerine sızar
- Tek bir olay bir çok başarımı tetikleyebilir
- Bu kirlilik yaratır

Örnek

```
void Physics::updateEntity(Entity& entity)
{
  bool wasOnSurface = entity.isOnSurface();
  entity.accelerate(GRAVITY);
  entity.update();
  if (wasOnSurface && !entity.isOnSurface())
  {
    notify(entity, EVENT_START_FALL);
  }
}
```

- `notify` metodu birşeyin olduğunu haber verir
- Kime?
  - Kim bilmek istiyorsa ona

Nasıl Çalışıyor?

Gözlemci

```
class Observer
{
public:
  virtual ~Observer() {}
  virtual void onNotify(const Entity& entity,
        Event event) = 0;
};
```

- `Observer` sınıfını implement eden nesneler gözlemci haline gelirler.
- Artık onlar gerçekleşen tüm olaylardan haberdar edilirler

```cpp
class Achievements : public Observer {
public:
  virtual void onNotify(const Entity& entity,
        Event event)
  {
    switch (event) {
    case EVENT_ENTITY_FELL:
      if (entity.isHero() && heroIsOnBridge_)
        unlock(ACHIEVEMENT_FELL_OFF_BRIDGE);
      break;
      // Handle other events, update heroIsOnBridge_...
    }
  }

private:
  void unlock(Achievement achievement) {
    // Unlock if not already unlocked...
  }
  bool heroIsOnBridge_;
};
```

Nesne

- Gözlemlenen objeye "nesne" (subject) denir.

```
class Subject
{
private:
  Observer* observers_[MAX_OBSERVERS];
  int numObservers_;
};
```

- Nesne'nin görevi, kendisini gözlemleyenlerin bir listesini tutmaktır

Nesne (devam)

- Nesne'nin tuttuğu bu liste dışarıdan kontrol edilebilir

```
class Subject {
public:
  void addObserver(Observer* observer)  {
    // Add to array...
  }

  void removeObserver(Observer* observer)  {
    // Remove from array...
  }

  // Other stuff...
};
```

Nesne (devam)

- Nesne'nin son görevi, bir olay olduğunda bunu gözlemcilerine iletmektir

```
class Subject
{
protected:
  void notify(const Entity& entity, Event event)
  {
    for (int i = 0; i < numObservers_; i++)
    {
      observers_[i]->onNotify(entity, event);
    }
  }

  // Other stuff...
};
```

Physics tarafı

```
class Physics : public Subject
{
public:
  void updateEntity(Entity& entity);
};
```

```
void Physics::updateEntity(Entity& entity)
{
  bool wasOnSurface = entity.isOnSurface();
  entity.accelerate(GRAVITY);
  entity.update();
  if (wasOnSurface && !entity.isOnSurface())
  {
    notify(entity, EVENT_START_FALL);
  }
}
```

Dikkat edilmesi gerekenler

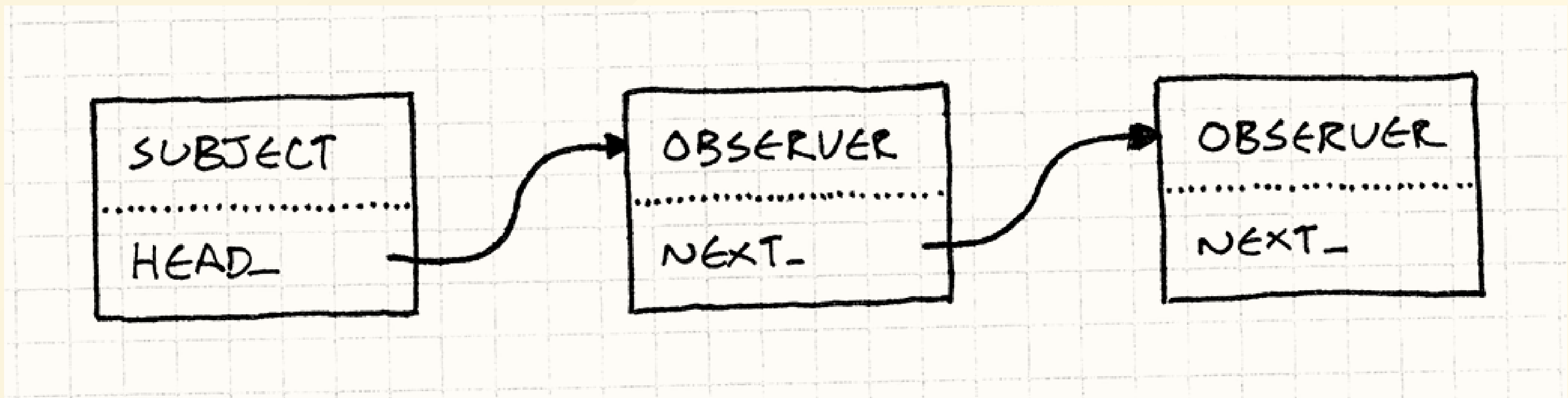
- Gözlemci örüntüsü çok mu yavaş?
  - Hayır! Genelllikle event ve messaging kavramlarıyla karıştırılıyor
  - Bunlarda Queueing olabilir, Gözlemci'de yoktur

Dikkat edilmesi gerekenler

- Eşzamanlı çağrım
  - Gözlemciler, nesneyi bloklayabilir!
  - Stay off the UI Thread!

Dikkat edilmesi gerekenler

- Çok fazla dinamik tahsis
  - Oyunun başında nesne-gözlemci linkleri kurulup sonrasında değiştirilmezse çok sorun olmaz
  - Bağlı gözlemciler modeli:



Bağlı gözlemciler

```
class Subject {
  Subject() : head_(NULL) {}

private:
  Observer* head_;
};
```

```
class Observer {
  friend class Subject;

public:
  Observer() : next_(NULL) {}

private:
  Observer* next_;
};
```

Bağlı gözlemciler

```
void Subject::addObserver(Observer* observer)
{
  observer->next_ = head_;
  head_ = observer;
}
```

Bağlı gözlemcilerde gözlemci çıkarma

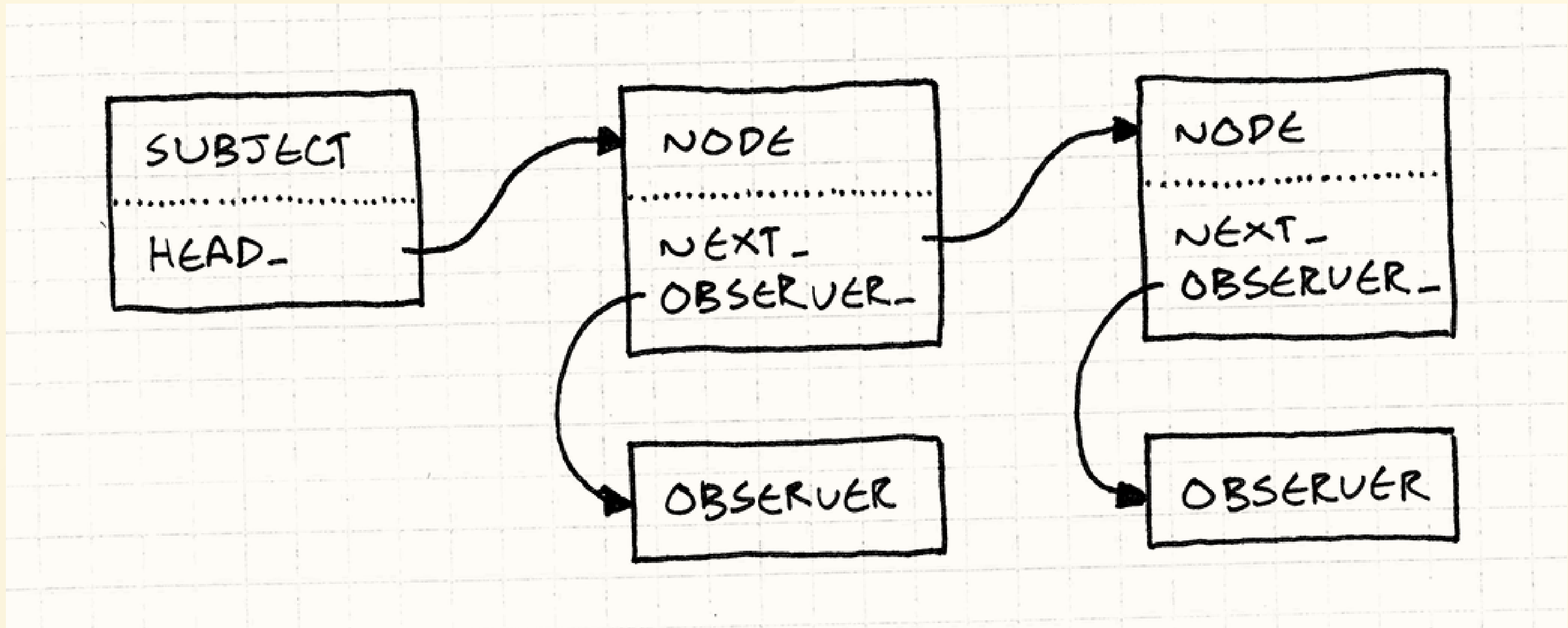
```
void Subject::removeObserver(Observer* observer){
  if (head_ == observer)  {
    head_ = observer->next_;
    observer->next_ = NULL;
    return;
  }

  Observer* current = head_;
  while (current != NULL)  {
    if (current->next_ == observer)    {
      current->next_ = observer->next_;
      observer->next_ = NULL;
      return;
    }

    current = current->next_;
```

Bağlı gözlemcilerde `notify` metodu

```
void Subject::notify(const Entity& entity, Event event)
{
  Observer* observer = head_;
  while (observer != NULL)
  {
    observer->onNotify(entity, event);
    observer = observer->next_;
  }
}
```

Dikkat edilmesi gerekenler

- Bu şekilde gözlemci yalnız bir nesneyi gözlemler
- Gerçekte bu doğru olmayabilir
- Bunun için wrapper node kullanılabilir



Diğer problemler

- Bir nesne veya gözlemciyi silerseniz ne olur?
  - Karşılıklı haberleşerek silinmeleri gerekir

Singleton (tek kişilik) örüntüsü

Nedir?

> “ Bir sınıfın bir örneği vardır, ve bu örneğe heryerden ulaşılabilir. ”

Tek örnek

- Bir dosya sistemi (sınıfı) implementasyonu düşünün
- Bu implementasyon üzerinden dosya yaratıp, silebiliyorsunuz
- Aynı anda iki farklı örnek yaratıp, birine X dosyasını yarattırırken, diğerine X dosyasını sildirirseniz ne olur?
- Tek örnek bu tür problemleri çözmeye yarar

Heryerden ulaşılabilirlik

- Eğer tek bir örnek olacaksa, bu örneği kullanacak herkesin bu örneğe direk erişiminin olması gerekir
- Dosya sistemi örneği
    - Loglama
    - İçerik yükleme
    - Kayıt etme
    - vs.

Örnek kod

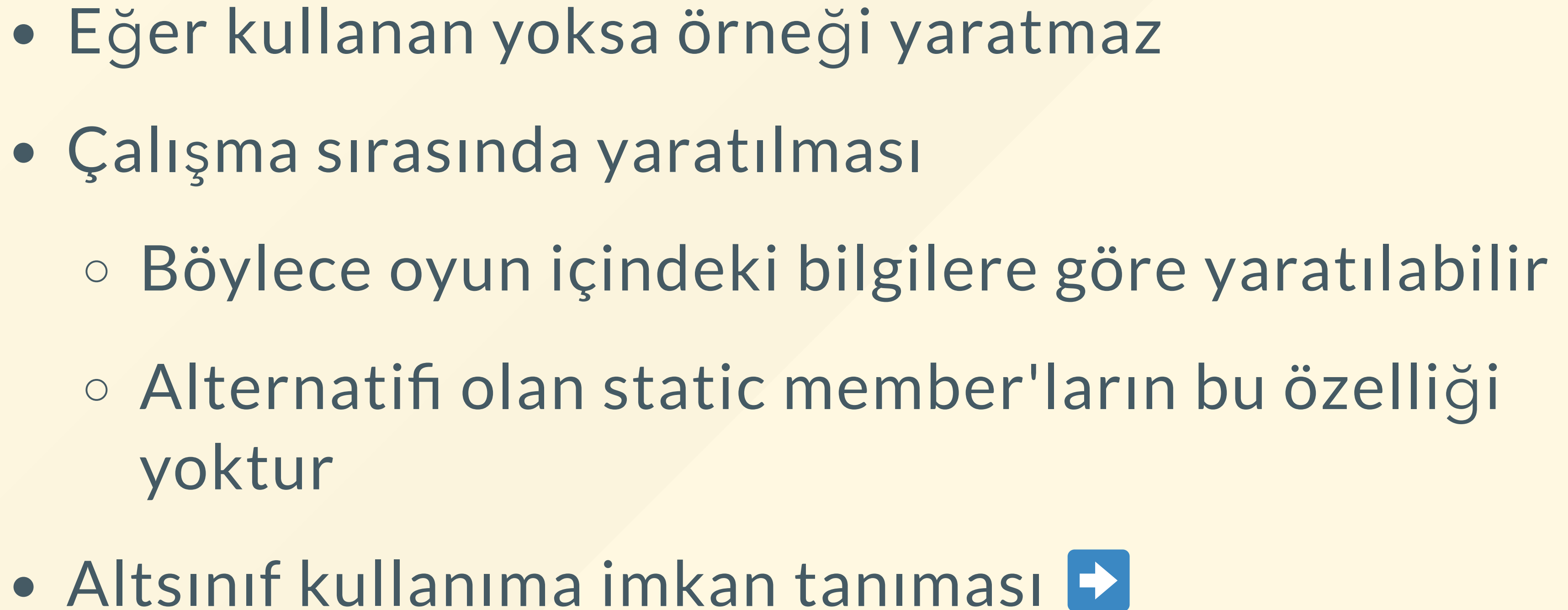
```
class FileSystem
{
public:
  static FileSystem& instance()
  {
    // Lazy initialize.
    if (instance_ == NULL) instance_ = new FileSystem();
    return *instance_;
  }

private:
  FileSystem() {}

  static FileSystem* instance_;
};
```

Modern yaklaşım

```
class FileSystem
{
public:
  static FileSystem& instance()
  {
    static FileSystem *instance = new FileSystem();
    return *instance;
  }

private:
  FileSystem() {}
};
```

- Bu kod modern c++ compilerlarında thread-safe çalışmaktadır

Avantajları



Altsınıf kullanımı

```
class FileSystem
{
public:
  static FileSystem& instance();

  virtual ~FileSystem() {}
  virtual char* readFile(char* path) = 0;
  virtual void  writeFile(char* path, char* contents) = 0;

protected:
  FileSystem() {}
};
```

Altsınıf kullanımı - PS3

```
class PS3FileSystem : public FileSystem
{
public:
  virtual char* readFile(char* path)
  {
    // Use Sony file IO API...
  }

  virtual void writeFile(char* path, char* contents)
  {
    // Use sony file IO API...
  }
};
```

Altsınıf kullanımı - Wii

```
class WiiFileSystem : public FileSystem
{
public:
  virtual char* readFile(char* path)
  {
    // Use Nintendo file IO API...
  }

  virtual void writeFile(char* path, char* contents)
  {
    // Use Nintendo file IO API...
  }
};
```

Altsınıf kullanımı - `instance()`

```
FileSystem& FileSystem::instance()
{
  #if PLATFORM == PLAYSTATION3
    static FileSystem *instance = new PS3FileSystem();
  #elif PLATFORM == WII
    static FileSystem *instance = new WiiFileSystem();
  #endif

  return *instance;
}
```

Sorunlar

- Globaller kötüdür
  - Global çağrılar kodun anlaşılmasını ve izlenmesini zorlaştırır
  - Gözlemci örüntüsü ile çatışabilir. İnsanlar heryerden erişilebilir bir örneği direk olarak kullanmak isteyebilirler. Mesela yere düşen bir taşın ses çıkarması.
  - Singleton örüntüsü parallelizme çok uygun değildir.

Sorunlar

- Bir probleminiz varken iki problem çözer
  - Ya sadece tek bir örneğe ihtiyacınız varsa?
  - Ya sadece heryerden erişime ihtiyacınız varsa?

Sorunlar

- Lazy-initialization kontrolü sizden alır
- Oyun başladıktan sonra ses modülünün yüklenmesini istemezsiniz

```
class FileSystem {
public:
  static FileSystem& instance() { return instance_; }
private:
  FileSystem() {}
  static FileSystem instance_;
};
```

- Bu çözüm de polymorphism vs. gibi lazy initialization'ın getirdiği özellikleri yok eder

Gerçekten ihtiyacınız var mı?

```cpp
class Bullet
{
public:
  int getX() const { return x_; }
  int getY() const { return y_; }

  void setX(int x) { x_ = x; }
  void setY(int y) { y_ = y; }

private:
  int x_, y_;
};
```

```cpp
class BulletManager {
public:
  Bullet* create(int x, int y)  {
    Bullet* bullet = new Bullet();
    bullet->setX(x);
    bullet->setY(y);
    return bullet;
  }

  bool isOnScreen(Bullet& bullet)  {
    return bullet.getX() >= 0 &&
           bullet.getX() < SCREEN_WIDTH &&
           bullet.getY() >= 0 &&
           bullet.getY() < SCREEN_HEIGHT;
  }

  void move(Bullet& bullet)  {
    bullet.setX(bullet.getX() + 5);
  }
};
```

Halbuki...

```
class Bullet
{
public:
  Bullet(int x, int y) : x_(x), y_(y) {}

  bool isOnScreen()
  {
    return x_ >= 0 && x_ < SCREEN_WIDTH &&
           y_ >= 0 && y_ < SCREEN_HEIGHT;
  }

  void move() { x_ += 5; }

private:
  int x_, y_;
};
```

Yalnızca tek örnek yaratmak

```
class FileSystem
{
public:
  FileSystem()
  {
    assert(!instantiated_);
    instantiated_ = true;
  }

  ~FileSystem() { instantiated_ = false; }

private:
  static bool instantiated_;
};

bool FileSystem::instantiated_ = false;
```

Heryerden erişim

- Parametre olarak geçirin
- Ata sınıfa gömün
- Tek bir global sınıf üzerinden çalışın
- Servis Sağlayıcı örüntüsünü kullanın

Durum Örüntüsü

Hata nerede?

```
void Heroine::handleInput(Input input)
{
  if (input == PRESS_B)
  {
    yVelocity_ = JUMP_VELOCITY;
    setGraphics(IMAGE_JUMP);
  }
}
```

Havada zıplamayı engelleyin

```
void Heroine::handleInput(Input input)
{
  if (input == PRESS_B)
  {
    if (!isJumping_)
    {
      isJumping_ = true;
      // Jump...
    }
  }
}
```

Çökme ekleyelim

```
void Heroine::handleInput(Input input){
  if (input == PRESS_B)  {
    // Jump if not jumping...
  }
  else if (input == PRESS_DOWN)
  {
    if (!isJumping_)    {
      setGraphics(IMAGE_DUCK);
    }
  }
  else if (input == RELEASE_DOWN)  {
    setGraphics(IMAGE_STAND);
  }
}
```

Hatayı farkettiniz mi?

```
void Heroine::handleInput(Input input){
  if (input == PRESS_B)  {
    if (!isJumping_ && !isDucking_)    {
      // Jump...
    }
  }
  else if (input == PRESS_DOWN)  {
    if (!isJumping_)    {
      isDucking_ = true;
      setGraphics(IMAGE_DUCK);
    }
  }
  else if (input == RELEASE_DOWN)  {
    if (isDucking_)    {
      isDucking_ = false;
      setGraphics(IMAGE_STAND);
    }
  }
}
```

Dalış saldırısı ekleyelim

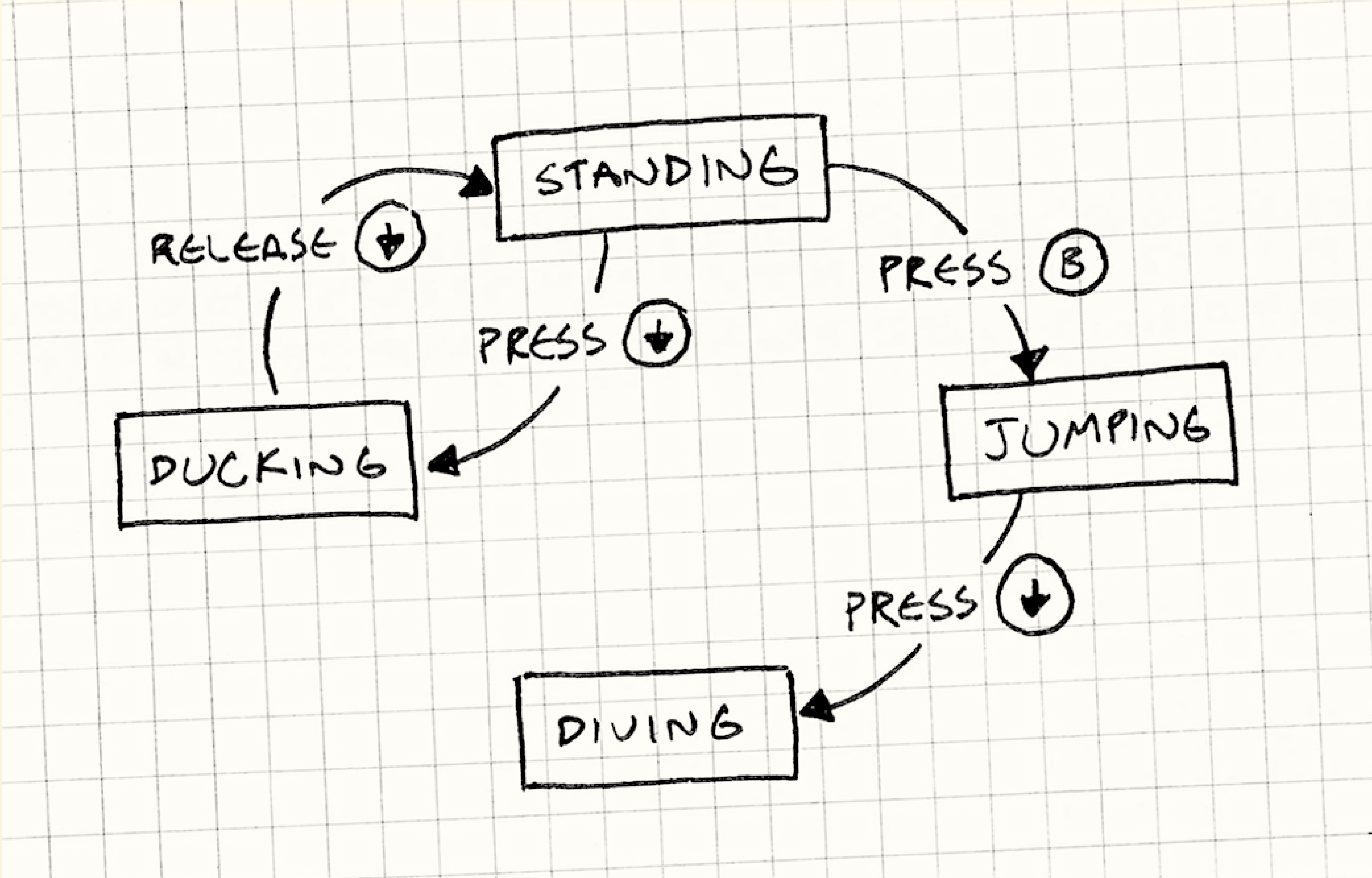
```
void Heroine::handleInput(Input input){
  if (input == PRESS_B)  {
    if (!isJumping_ && !isDucking_) { /* Jump... */ }
  }
  else if (input == PRESS_DOWN)  {
    if (!isJumping_) {
      isDucking_ = true;
      setGraphics(IMAGE_DUCK);
    }
    else    {
      isJumping_ = false;
      setGraphics(IMAGE_DIVE);
    }
  }
  else if (input == RELEASE_DOWN)  {
    if (isDucking_) { /* Stand... */ }
  }
}
```

Hatayı farkettiniz mi?

Sorun

- Her aktiviteyi eklediğimizde kodu bozuyoruz
- Tekrardan tüm aktiviteleri dizayn etmek ve kodlamak gerekiyor
- 5 aktivite sonrasını hayal edin!

Sonlu Durum Makineleri



Nedir?

- SDM (FSM)
  - Makinenin girebileceği sabit ve sonlu sayıda durum vardır
  - Makine bir anda yalnızca bir durumda olabilir
  - Makineye bir seri *girdi* veya *olay* gönderilir
  - Her durumun bir *geçiş kümesi* vardır ve gönderilen girdi ve olaylara karşılık gelir

Hadi yapalım!

```
enum State
{
  STATE_STANDING,
  STATE_JUMPING,
  STATE_DUCKING,
  STATE_DIVING
};
```

```
void Heroine::handleInput(Input input){
  switch (state_)  {
    case STATE_STANDING:
      if (input == PRESS_B)      {
        state_ = STATE_JUMPING;
        yVelocity_ = JUMP_VELOCITY;
        setGraphics(IMAGE_JUMP);
      }
      else if (input == PRESS_DOWN)      {
        state_ = STATE_DUCKING;
        setGraphics(IMAGE_DUCK);
      }
      break;

    case STATE_JUMPING:
      if (input == PRESS_DOWN)      {
        state_ = STATE_DIVING;
        setGraphics(IMAGE_DIVE);
      }
      break;

    case STATE_DUCKING:
      if (input == RELEASE_DOWN)      {
        state_ = STATE_STANDING;
        setGraphics(IMAGE_STAND);
      }
      break;
  }
}
```

Charge time

- Çökme sırasında enerji depolayıp saldırmak
- Ne kadar süre çökersek, o kadar enerji depolayabiliriz
- Bunu tutmak için bir sayaca ihtiyacımız var

update()

```
void Heroine::update()
{
  if (state_ == STATE_DUCKING)
  {
    chargeTime_++;
    if (chargeTime_ > MAX_CHARGE)
    {
      superBomb();
    }
  }
}
```

handleInput()

```
void Heroine::handleInput(Input input)
{
  switch (state_)
  {
    case STATE_STANDING:
      if (input == PRESS_DOWN)
      {
        state_ = STATE_DUCKING;
        chargeTime_ = 0;
        setGraphics(IMAGE_DUCK);
      }
      // Handle other inputs...
      break;

      // Other states...
  }
}
```

Daha iyisini yapabilir miyiz?

Durum arayüzü

```
class HeroineState
{
public:
  virtual ~HeroineState() {}
  virtual void handleInput(Heroine& heroine,
                           Input input) {}
  virtual void update(Heroine& heroine) {}
};
```

Her durum için bir sınıf

```
class DuckingState : public HeroineState {
public:
  DuckingState() : chargeTime_(0) {}
  virtual void handleInput(Heroine& heroine, Input input){
    if (input == RELEASE_DOWN)    {
      // Change to standing state...
      heroine.setGraphics(IMAGE_STAND);
    }
  }
  virtual void update(Heroine& heroine) {
    chargeTime_++;
    if (chargeTime_ > MAX_CHARGE)    {
      heroine.superBomb();
    }
  }
private:
  int chargeTime_;
};
```

Sadece durum sınıfına bir işaretçi

```
class Heroine
{
public:
  virtual void handleInput(Input input)
  {
    state_->handleInput(*this, input);
  }

  virtual void update()
  {
    state_->update(*this);
  }

  // Other methods...
private:
  HeroineState* state_;
};
```

Durum objelerini yaratalım

```
class HeroineState
{
public:
  static StandingState standing;
  static DuckingState ducking;
  static JumpingState jumping;
  static DivingState diving;

  // Other code...
};
```

- Böylece tek durum örneğini ihtiyacı olan tüm Heroine'ler kullanabilir!

Durum'u Heroine'de tutmak

```
void Heroine::handleInput(Input input) {
  HeroineState* state = state_->handleInput(*this, input);
  if (state != NULL)  {
    delete state_;
    state_ = state;
  }
}
```

```
HeroineState* StandingState::handleInput(Heroine& heroine,
                                         Input input){
  if (input == PRESS_DOWN)  {
    // Other code...
    return new DuckingState();
  }
  // Stay in this state.
  return NULL;
}
```

Giriş ve Çıkış

- Mevcut kodda oyuncunun Sprite'ı çıkış yaptığı durum tarafından değiştiriliyor
- Bu son derece yanlış bir yaklaşım
- Bunun yerine duruma giriş ve durumdan çıkış metodlarını kullanbiliriz

```cpp
class StandingState : public HeroineState {
public:
  virtual void enter(Heroine& heroine) {
    heroine.setGraphics(IMAGE_STAND);
  }

  // Other code...
};
```

```cpp
void Heroine::handleInput(Input input) {
  HeroineState* state = state_->handleInput(*this, input);
  if (state != NULL) {
    delete state_;
    state_ = state;

    // Call the enter action on the new state.
    state_->enter(*this);
  }
}
```

Çökme kodu da basitleşti

```
HeroineState* DuckingState::handleInput(Heroine& heroine,
                                        Input input)
{
  if (input == RELEASE_DOWN)
  {
    return new StandingState();
  }

  // Other code...
}
```

İşler karmaşıklaşıyor

- Diyelim ki karakterimiz tüm bu hareketleri yaparken aynı zamanda ateş de edebilsin
- Bu durumda durum sayısı ikiye katlanacak!
  - Ateş ederken zıplama
  - Ateş etmeden zıplama
  - Ateş ederek çöküş
  - Ateş etmeden çöküş
  - ...

İki farklı durum makinesi

```
class Heroine
{
  // Other code...

private:
  HeroineState* state_;
  HeroineState* equipment_;
};
```

```
void Heroine::handleInput(Input input)
{
  state_->handleInput(*this, input);
  equipment_->handleInput(*this, input);
}
```

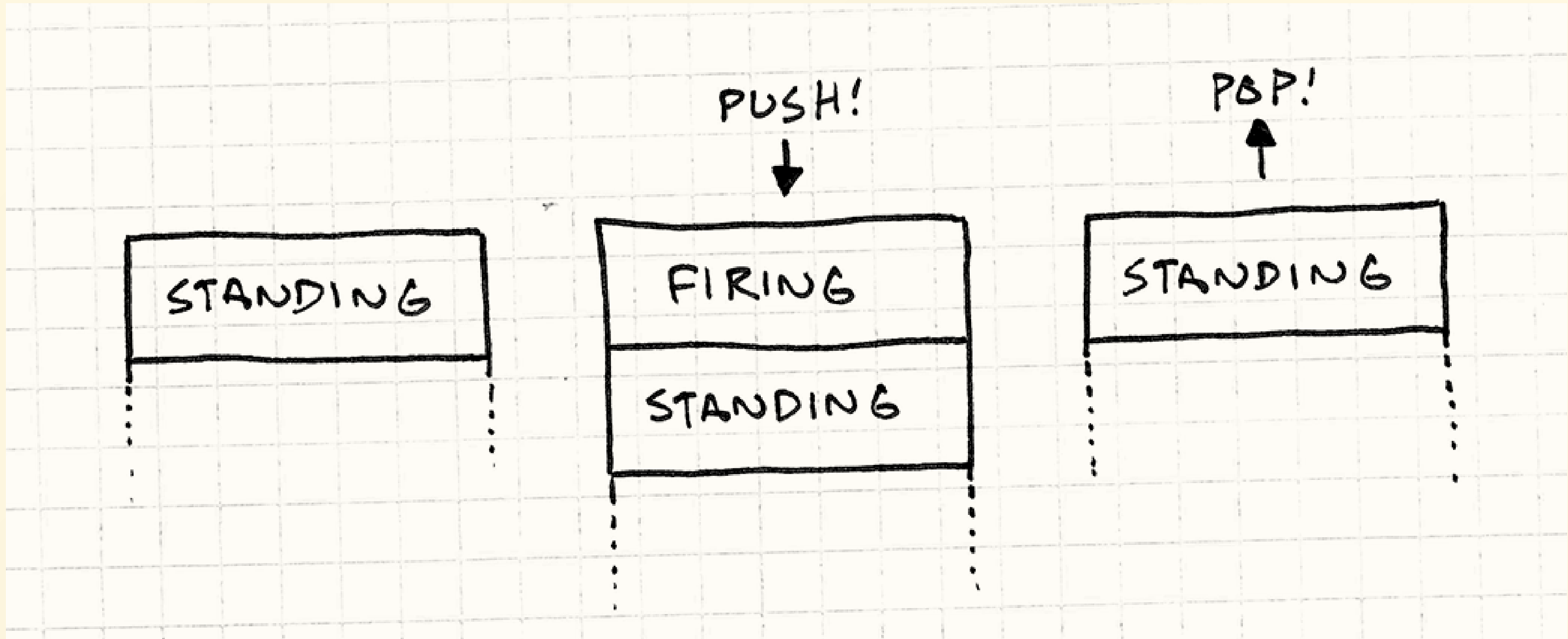
Hiyerarşik durum makinesi

```
class OnGroundState : public HeroineState
{
public:
  virtual void handleInput(Heroine& heroine, Input input)
  {
    if (input == PRESS_B)
    {
      // Jump...
    }
    else if (input == PRESS_DOWN)
    {
      // Duck...
    }
  }
};
```

Hiyerarşik Durum Makinesi

```
class DuckingState : public OnGroundState
{
public:
  virtual void handleInput(Heroine& heroine, Input input)
  {
    if (input == RELEASE_DOWN)
    {
      // Stand up...
    }
    else
    {
      // Didn't handle input, so walk up hierarchy.
      OnGroundState::handleInput(heroine, input);
    }
  }
};
```

Pushdown Automata



Reflection

Aşağıdaki soruları cevaplayın

- Gözlemci örüntüsünde, gözlemci ve nesneyi tanımlayın
- Singleton örüntüsünde elde edilmek istenen iki özellik nedir?
- Sonlu Durum Makinelerinin bir oyunda kullanılmasının zor olmasının sebepleri sizce nelerdir?