THE DEVELOPER'S CONFERENCE

Design Patterns in Game Programming

Bruno Cicanci

Senior Software Engineer @ Aquiris Game Studio

Agenda

THE DEVELOPER'S CONFERENCE

- Who am I?
- Design Patterns
- Other Patterns
- Software Architecture

Who am I?

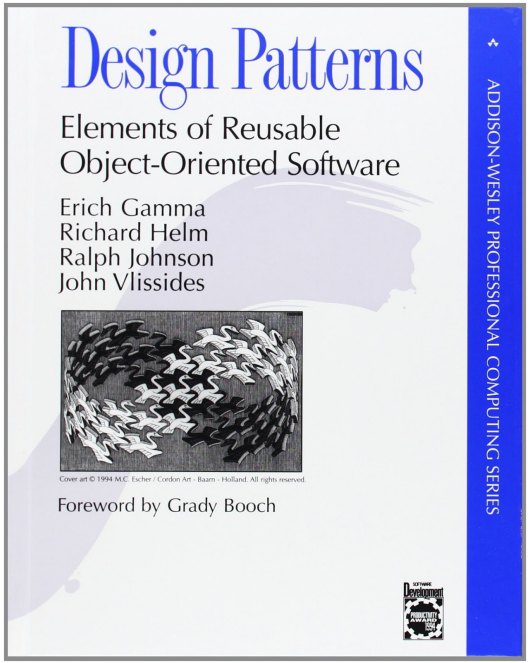
THE DEVELOPER'S CONFERENCE

- Bruno Cicanci (@cicanci)
- Senior Software Engineer at Aquiris Game Studio
- 14+ years as a professional programmer
- 9+ years working with mobile games
- 20+ games developed
- 10+ years blogging: gamedeveloper.com.br

Design Patterns

THE DEVELOPER'S CONFERENCE

- Command
- Flyweight
- Observer
- Prototype
- Singleton
- State



Command: The problem

THE DEVELOPER'S CONFERENCE

```
void InputHandler::handleInput()
{
  if (isPressed(BUTTON_X)) jump();
  else if (isPressed(BUTTON_Y)) fireGun();
  else if (isPressed(BUTTON_A)) swapWeapon();
  else if (isPressed(BUTTON_B)) reloadWeapon();
}
```

https://gameprogrammingpatterns.com/command.html

Command: The solution

THE DEVELOPER'S CONFERENCE

```
class Command
{
public:
  virtual ~Command() {}
  virtual void execute() = 0;
};
class JumpCommand : public Command
{
public:
  virtual void execute() { jump(); }
};
```

```
void InputHandler::handleInput()
{
  if (isPressed(BUTTON_X))
    buttonX->execute();
  else if (isPressed(BUTTON_Y))
    buttonY->execute();
  else if (isPressed(BUTTON_A))
    buttonA->execute();
  else if (isPressed(BUTTON_B))
    buttonB->execute();
}
```

https://gameprogrammingpatterns.com/command.html

Command: The usage

- Input
- Undo
- Redo

https://gameprogrammingpatterns.com/command.html

Flyweight: The problem

THE DEVELOPER'S CONFERENCE

```
class Tree
{
private:
  Mesh mesh;
  Texture bark;
  Texture leaves;
  Vector position;
  double height;
  double thickness;
  Color barkTint;
  Color leafTint;
};
```

https://gameprogrammingpatterns.com/flyweight.html

Flyweight: The solution

THE DEVELOPER'S CONFERENCE

```
class TreeModel
{
private:
  Mesh mesh;
  Texture bark;
  Texture leaves;
};
```

```
class Tree
{
private:
  TreeModel* model;

  Vector position;
  double height;
  double thickness;
  Color barkTint;
  Color leafTint;
};
```

https://gameprogrammingpatterns.com/flyweight.html

Flyweight: The usage

THE DEVELOPER'S CONFERENCE

- Multiple instances
- Tile maps

https://gameprogrammingpatterns.com/flyweight.html

Observer: The problem

THE DEVELOPER'S CONFERENCE

```
void Physics::updateEntity(Entity& entity)
{
  bool wasOnSurface = entity.isOnSurface();
  entity.accelerate(GRAVITY);
  entity.update();

  if (wasOnSurface && !entity.isOnSurface() && entity.isHero())
  {
    unlockFallOffBridge();
  }
}
```

https://gameprogrammingpatterns.com/observer.html

Observer: The solution (1)

THE DEVELOPER'S CONFERENCE

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

```
class Achievements : public Observer
{
public:
  virtual void onNotify(const Entity& entity, Event event)
  {
    switch (event)
    {
    case EVENT_ENTITY_FELL:
      if (entity.isHero())
      {
        unlock(ACHIEVEMENT_FELL_OFF_BRIDGE);
      }
      break;
    }
  }
private:
  void unlock(Achievement achievement) { }
};
```

https://gameprogrammingpatterns.com/observer.html

Observer: The solution (2)

THE DEVELOPER'S CONFERENCE

```
class Subject
{
private:
  Observer* observers[MAX_OBSERVERS];
  int numObservers;
protected:
  void notify(const Entity& entity, Event event)
  {
    for (int i = 0; i < numObservers; i++)
    {
      observers[i]->onNotify(entity, event);
    }
  }
public:
  void addObserver(Observer* observer) { }
  void removeObserver(Observer* observer) { }
};
```

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

https://gameprogrammingpatterns.com/observer.html

Observer: The usage

THE DEVELOPER'S CONFERENCE

- Achievements
- VFX/SFX
- Events
- Messages
- Callbacks async

https://gameprogrammingpatterns.com/observer.html

Prototype: The problem

THE DEVELOPER'S CONFERENCE

```
class Monster { };

class Ghost : public Monster {};
class Demon : public Monster {};
class Sorcerer : public Monster {};
```

```
class Spawner
{
public:
  virtual ~Spawner() {}
  virtual Monster* spawnMonster() = 0;
};

class GhostSpawner : public Spawner
{
public:
  virtual Monster* spawnMonster()
  {
    return new Ghost();
  }
};

...
```

https://gameprogrammingpatterns.com/prototype.html

Prototype: The solution (1)

THE DEVELOPER'S CONFERENCE

```
class Monster
{
public:
  virtual ~Monster() {}
  virtual Monster* clone() = 0;
};
```

```
class Ghost : public Monster {
public:
  Ghost(int h, int s)
  : health(h), speed(s)
  {}
  virtual Monster* clone()
  {
    return new Ghost(health, speed);
  }
private:
  int health;
  int speed;
}
```

https://gameprogrammingpatterns.com/prototype.html

Prototype: The solution (2)

THE DEVELOPER'S CONFERENCE

```
class Spawner
{
public:
  Spawner(Monster* p)
  : prototype(p)
  {}
  Monster* spawnMonster()
  {
    return prototype->clone();
  }
private:
  Monster* prototype;
};
```

https://gameprogrammingpatterns.com/prototype.html

Prototype: The usage

- Spawner
- Player Items
- Shoots
- Loot

https://gameprogrammingpatterns.com/prototype.html

States: The problem

THE DEVELOPER'S CONFERENCE

```
void Heroine::handleInput(Input input)
{
  if (input == PRESS_B)
  {
    if (!isJumping && !isDucking)
    {
      // Jump...
    }
  }
  else if (input == PRESS_DOWN)
  {
    if (!isJumping)
    {
      isDucking = true;
    }
  }
  else if (input == RELEASE_DOWN)
  {
    if (isDucking)
    {
      isDucking = false;
    }
  }
}
```

https://gameprogrammingpatterns.com/state.html

States: The solution

THE DEVELOPER'S CONFERENCE

```
void Heroine::handleInput(Input input)
{
  switch (state)
  {
    case STATE_STANDING:
      if (input == PRESS_B)
      {
        state = STATE_JUMPING;
      }
      else if (input == PRESS_DOWN)
      {
        state = STATE_DUCKING;
      }
      break;
    case STATE_JUMPING:
      if (input == PRESS_DOWN)
      {
        state = STATE_DIVING;
      }
      break;
    case STATE_DUCKING:
      if (input == RELEASE_DOWN)
      {
        state = STATE_STANDING;
      }
      break;
  }
}
```

https://gameprogrammingpatterns.com/state.html

States: The usage

THE DEVELOPER'S CONFERENCE

- Animations
- Screen states
- Character actions

https://gameprogrammingpatterns.com/state.html

Singleton: The problem

THE DEVELOPER'S CONFERENCE

Ensure that only one instance of the singleton class ever exists;

and *provide global access to that instance.*

Gang of Four, Design Patterns

https://gameprogrammingpatterns.com/singleton.html

Singleton: The solution

THE DEVELOPER'S CONFERENCE

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

https://gameprogrammingpatterns.com/singleton.html

Singleton: The usage

THE DEVELOPER'S CONFERENCE

"Friends don't let friends create singletons."

Robert Nystrom, Game Programming Patterns

https://gameprogrammingpatterns.com/singleton.html

Other Patterns

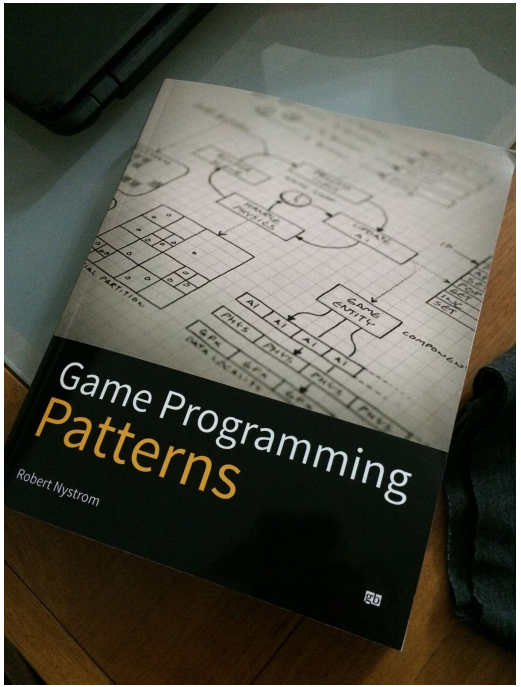
THE DEVELOPER'S CONFERENCE

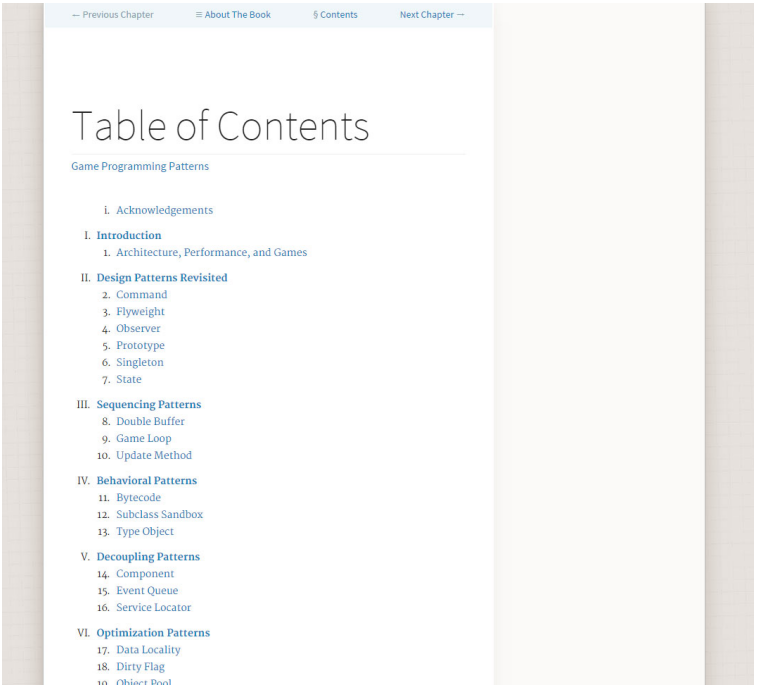
- Sequencing Patterns
  - Double Buffer
  - Game Loop
  - Update Method
- Behavioral Patterns
  - Bytecode
  - Subclass Sandbox
  - Type Object
- Decoupling Patterns
  - Component
  - Event Queue
  - Service Locator
- Optimization Patterns
  - Data Locality
  - Dirty Flag
  - Object Pool
  - Spatial Partition

https://gameprogrammingpatterns.com

Game Programming Patterns

THE DEVELOPER'S CONFERENCE





← Previous Chapter ≡ About The Book § Contents Next Chapter →

Table of Contents

gameprogrammingpatterns.com

Software Architecture

THE DEVELOPER'S CONFERENCE

- It is not about making diagrams
- It is about solving problems
  - … and communicating the solution!
- K.I.S.S.
- Occam's razor
- Less is more

SOLID Principles

THE DEVELOPER'S CONFERENCE

- Single Responsibility
- Open/Closed
- Liskov Substitution
- Interface Segregation
- Dependency Inversion

https://devopedia.org/solid-design-principles

gamedeveloper.com.br

ggdevcast.com.br

aquiris.com.br/work-with-us

THE DEVELOPER'S CONFERENCE

Obrigado!

bruno@gamedeveloper.com.br
Twitter/GitHub: @cicanci