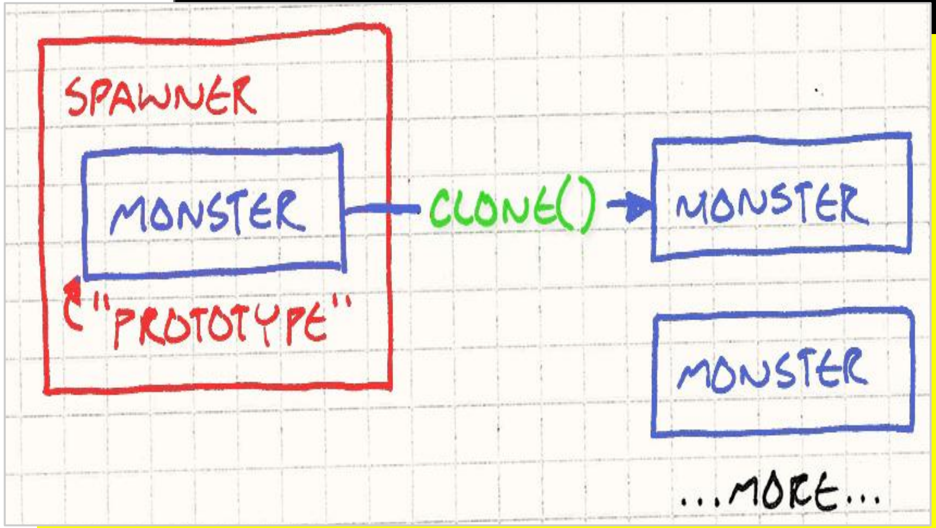
Game Programming Patterns

Update Method

Game Programming Patterns

- Game Loop
- **Update Method**
- Component
- Command
- State
- Prototype
- Observer
- ...



x2

Game Programming Patterns - Update Method

Motivation

- Imagine a **dungeon like** game **level**
- We want a re-animated **skeleton warrior pattroling back and forth** on the front door
- The *simplest code* would be:

```
while (true)
{
  // Patrol right.
  for (double x = 0; x < 100; x++)
  {
    skeleton.setX(x);
  }

  // Patrol left.
  for (double x = 100; x > 0; x--)
  {
    skeleton.setX(x);
  }
}
```

The problem?

- The skeleton moves back and forth, but the player never sees it.
- The program is locked in an infinite loop, which is not exactly a fun gameplay experience.

x3

Game Programming Patterns - Update Method

Let's make an update... We want the skeleton to move **one step *each frame***

```
Entity skeleton;
bool patrollingLeft = false;
double x = 0;

// Main game loop:
while (true) {
  if (patrollingLeft) {
    x--;
    if (x == 0) patrollingLeft = false;
  }
  else {
    x++;
    if (x == 100) patrollingLeft = true;
  }
  skeleton.setX(x);

  // Handle user input and render game...
}
```

It works ...

- We removed those loops and the logic now relies on the outer game loop for iteration
- The game keeps responding to the user inputs and rendering
- We added more complexity, but this *more or less* works, so we keep going...

x4

Game Programming Patterns - Update Method

```
// Skeleton variables...
Entity leftStatue;
Entity rightStatue;
int leftStatueFrames = 0;
int rightStatueFrames = 0;

// Main game loop:
while (true) {
  // Skeleton code...

  if (++leftStatueFrames == 90) {
    leftStatueFrames = 0;
    leftStatue.shootLightning();
  }

  if (++rightStatueFrames == 80) {
    rightStatueFrames = 0;
    rightStatue.shootLightning();
  }

  // Handle user input and render game...
}
```

Starting to become hard to maintain...

- We’ve got an increasingly large pile of variables and imperative code all stuffed in the game loop, each handling one specific entity in the game
- The Flying Spaghetti-Code Monster is arising!

x5

The Update Method

Game Update Pattern

- The **game world** maintains a **collection of objects**.
- **Each frame**, the game **updates** every object in the **collection**.
- **Each entity** in the game should **encapsulate** its **own behavior**
    - Each implements an update method that simulates one frame of the object's behavior.
    - This will keep the game loop uncluttered and make it easy to add and remove entities.

Update Pattern Objective

"Simulate a collection of independent objects by telling each to process one frame of behavior at a time."

Robert Nystrom

The Update Method

When to use it

- Update method works well when:
  - Your game has a **number of objects or systems** that need to **run simultaneously**.
  - Each **object's behavior** is mostly **independent of** the **others**.
  - The **objects** need to be **simulated over time**.

And when not...

For example, in a **game like chess**, you **don't need** to **simulate** all of the **pieces** concurrently, and probably **don't need** to **tell** the **pawns** to **update themselves** every frame.

To Keep in Mind

1. **Splitting code** into single frame slices **makes it more complex**
2. **You have to store state** to resume where you left off each frame
   - We needed to create the ***patrollingLeft*** variable
   - State Pattern can be helpful...
3. Objects all **simulate each frame** but are **not truly concurrent**
   - If **A comes before B** in the list of objects, then **when A updates**, it will **see B's previous state**
4. Be **careful modifying** the object **list** while **updating**

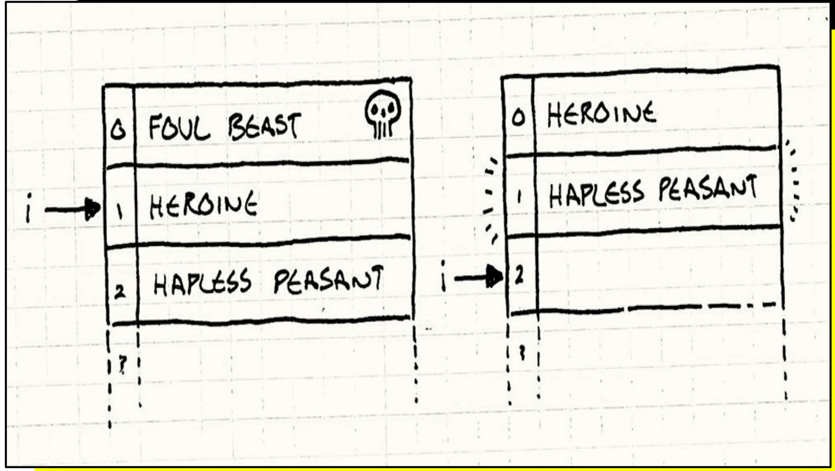
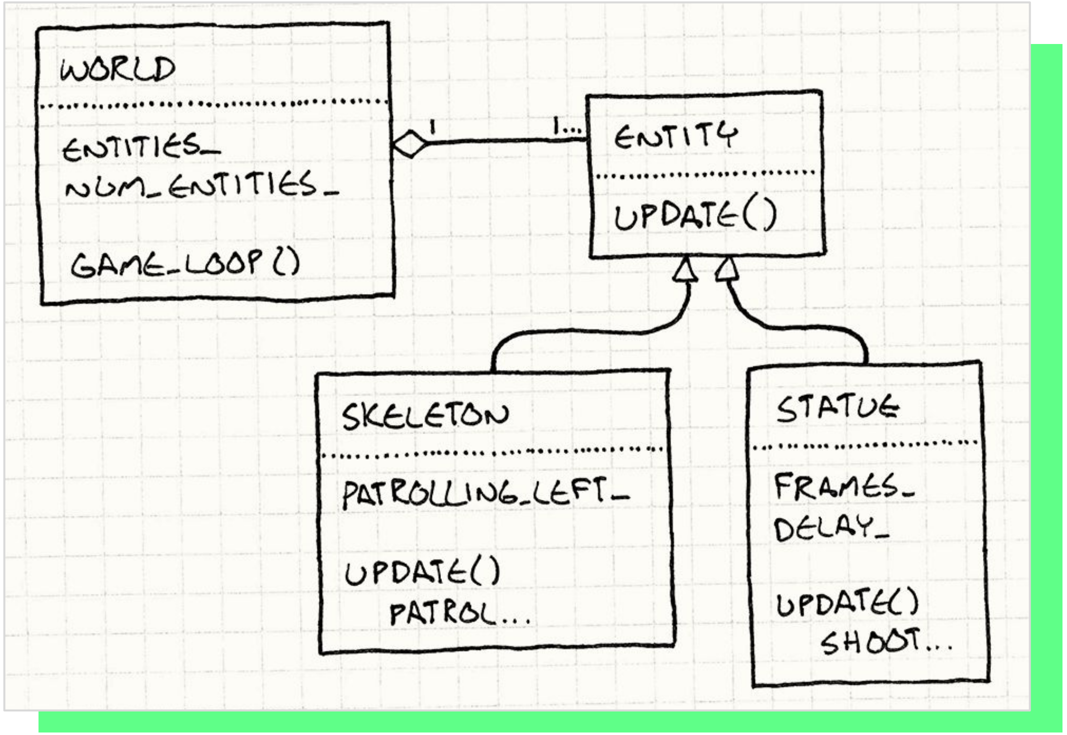


Illustration of a possible problem arising from the Point 4.

From: Robert Nystrom; "Game Programming Patterns"

Update Method



Update Method - Sample Code (1/2)

```
class Entity
{
public:
  Entity()
  : x_(0), y_(0)
  {}

  virtual ~Entity() {}
  virtual void update() = 0;

  double x() const { return x_; }
  double y() const { return y_; }

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

private:
  double x_;
  double y_;
};
```

```
class World
{
public:
  World()
  : numEntities_(0)
  {}

  void gameLoop();

private:
  Entity*
entities_[MAX_ENTITIES];
  int numEntities_;
};
```

```
void World::gameLoop()
{
  while (true)
  {
    // Handle user input...

    // Update each entity.
    for (int i = 0; i <
numEntities_; i++)
    {
      entities_[i]->update();
    }

    // Physics and rendering...
  }
}
```

x10

Update Method - Sample Code (2/2)

```
class Skeleton : public Entity {
public:
  Skeleton()
  : patrollingLeft_(false) {}

  virtual void update() {
    if (patrollingLeft_) {
      setX(x() - 1);
      if (x() == 0) patrollingLeft_ = false;
    }
    else {
      setX(x() + 1);
      if (x() == 100) patrollingLeft_ = true;
    }
  }

private:
  bool patrollingLeft_;
};
```

```
class Statue : public Entity {
public:
  Statue(int delay)
  : frames_(0),
    delay_(delay) {}

  virtual void update() {
    if (++frames_ == delay_) {
      shootLightning();
      // Reset the timer.
      frames_ = 0;
    }
  }

private:
  int frames_;
  int delay_;

  void shootLightning() {
    // Shoot the lightning...
  }
};
```

Game Programming Patterns

Update Method