STRANGE INVERSION OF CONTROL

Hamburg Unity Meetup, 2017-01-17

Jay Allan Jethwa
Team Lead Software Development
InnoGames Düsseldorf

Joined InnoGames in 2011
Before: Ubisoft Blue Byte

Currently working on
Unannounced Project

Die Siedler® online

FORGE OF EMPIRES

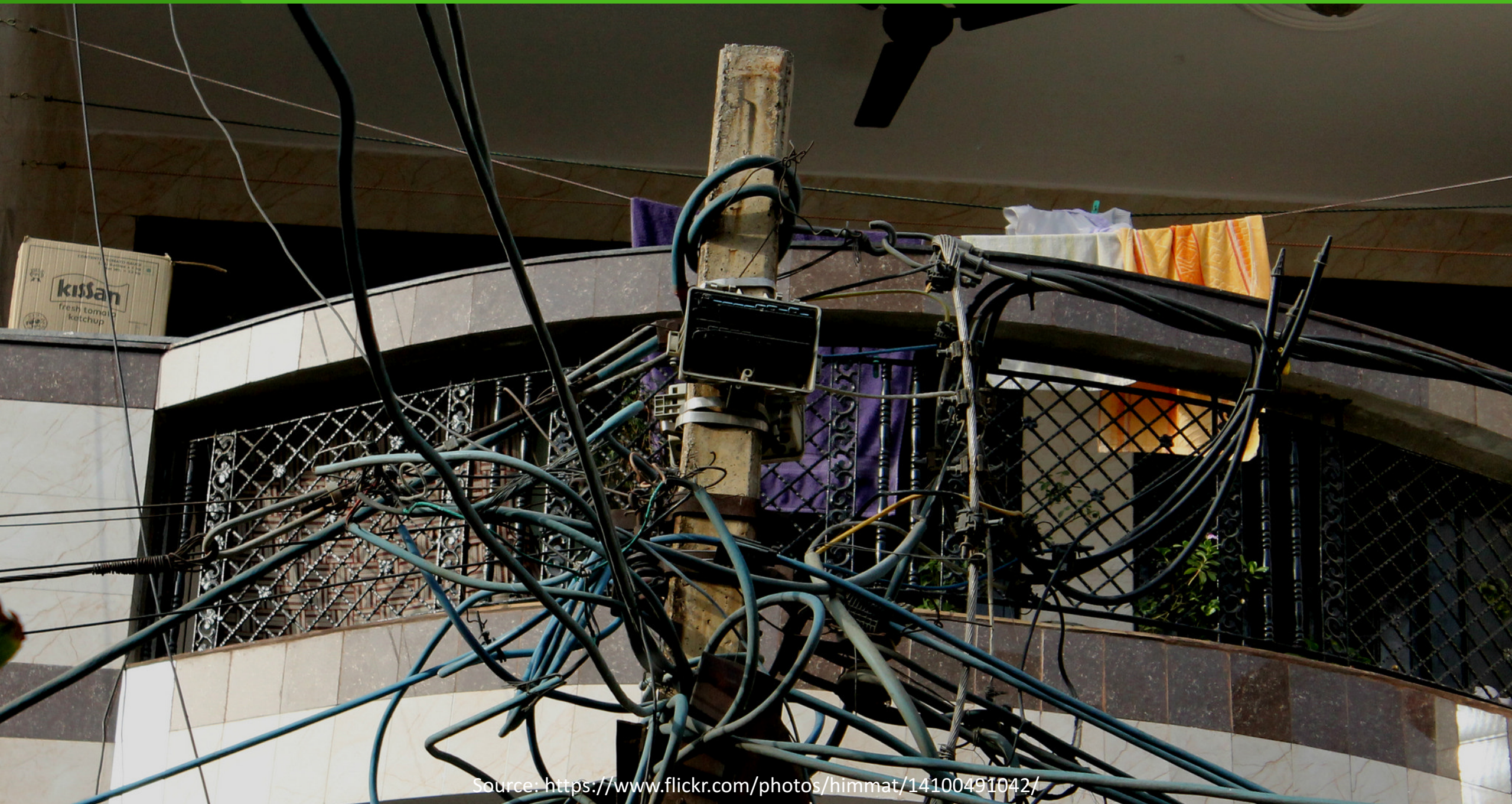
the Settlers My City

1. Introduction
2. Inversion of Control
3. Unity IoC Frameworks
4. Let‘s Become Strange
5. Conclusion

INTRODUCTION

The Struggle With Evolving Software Projects

Source: https://www.flickr.com/photos/4nitsirk/3203863340/



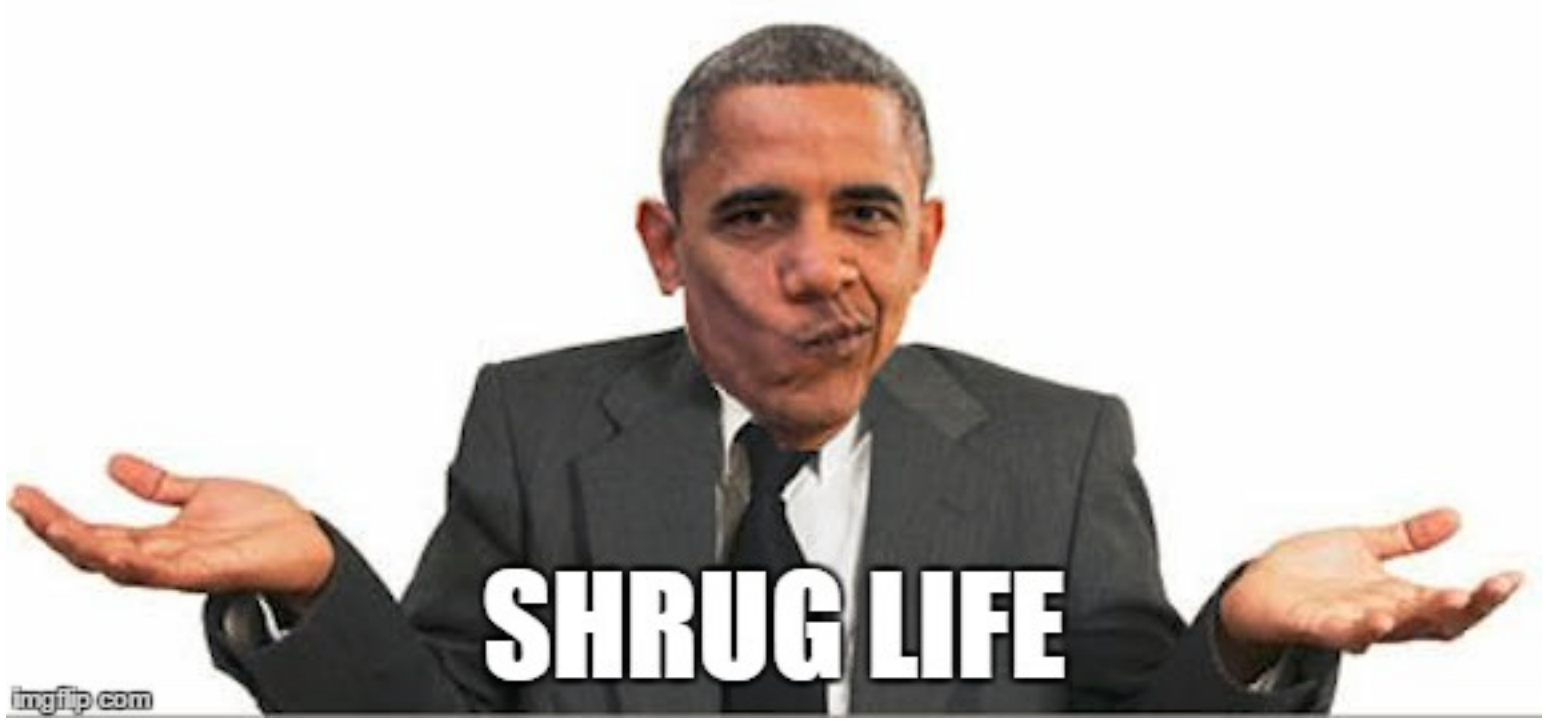
Source: https://www.flickr.com/photos/himmat/14100491042/

Source: https://www.flickr.com/photos/127969409@N03/15273796026/

MATCALFE'S LAW

"As the number of interacting components grows, the wiring and number of connections between them grows exponentially"

Source: https://www.flickr.com/photos/threar/15238870400/


SHRUG LIFE
imgflip.com

INVERSION OF CONTROL (IOC)

…To The Rescue!

…FROM THE PERSPECTIVE OF A FIVE-YEAR OLD

Source:
https://www.flickr.com/photos/nicocavallotto/3673367666/

When you go and get things out of the refrigerator for yourself, you can cause problems.

You might leave the door open, you might get something Mommy or Daddy doesn't want you to have.

You might even be looking for something we don't even have or which has expired.

What you should be doing is stating a need,
"I need something to drink with lunch," and then we will make sure you have something when you sit down to eat.

John Munsch, 28 October 2009.
Source: http://en.wikipedia.org/wiki/Dependency_injection

```
public class Player : MonoBehavior
{
	private void Update()
	{
		if (Input.GetKey(KeyCode.UpArrow))
		{
			MoveUp();
		}
	}

	…
}
```

```
public class InputController
{
	public bool IsUpKeyPressed
	{
		get
		{
			return Input.GetKey(KeyCode.UpArrow);
		}
	}
}
```

```
public class Player : MonoBehavior
{
	private InputController controller = new InputController();

	private void Update()
	{
		if (controller.IsUpKeyPressed)
		{
			MoveUp();
		}
	}
}
```

```
public interface IInputController
{
        bool IsUpKeyPressed { get; }
}
```

```
public class InputController : IInputController
{
	public bool IsUpKeyPressed
	{
		get
		{
			return Input.GetKey(KeyCode.UpArrow);
		}
	}
}
```

```
public class Player : MonoBehavior
{
	private IInputController controller;

	public void SetController(IInputController input)
	{
		controller = input;
	}

	private void Update()
	{
		if (controller.IsUpKeyPressed)
		{
			MoveUp();
		}
	}
}
```

DEPENDENCY INJECTION

Source: https://www.flickr.com/photos/tombullock/15713223772

```
public class Player : MonoBehavior
{
	[Inject]
	public IInputController InputController { private get; set; }

	private void Update()
	{
		if (controller.IsUpKeyPressed)
		{
			MoveUp();
		}
	}
}
```

INVERSION OF CONTROL

”Moving the decision of which concrete class to use away from the part of the system which uses it”

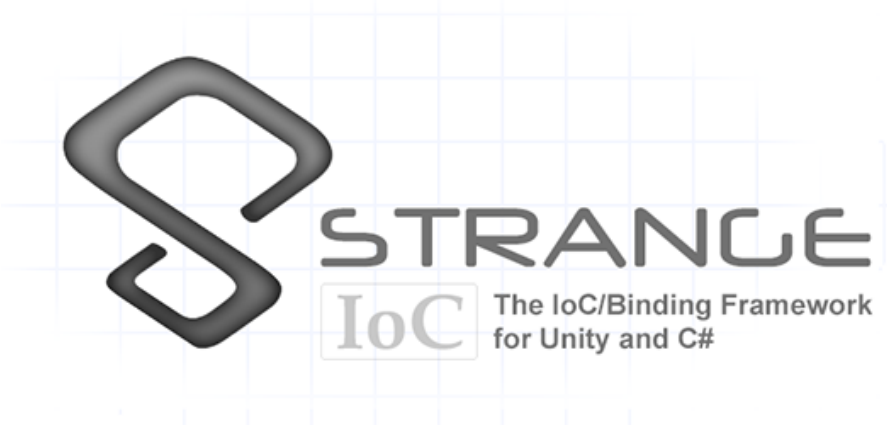
3

UNITY INVERSION OF CONTROL FRAMEWORKS

I want IoC in my project! But how can I get it?


STRANGE
IoC
The IoC/Binding Framework
for Unity and C#


Zenject



- Dependency Injection
- Signals
- Model-View-Controller
- Promises

- Powerful Dependency Injection
- Signals
- Commands
- Object Graph Validation
- Auto-Mocking

Syrinj Svelto ECS

Source: https://www.flickr.com/photos/29638108@N06/8571234513/

LET'S BECOME STRANGE!

A Dependency Injection Framework for Unity

CONTEXTVIEW

“Entry point for a Strange-driven application or module“

- MonoBehavior which instantiates the Context
- Needs to be attached to a root object in Unity‘s object hierarchy

```
public class Main : ContextView
{
	private void Awake()
	{
		context = new MainContext(this, true);
		context.Start();
	}
}
```

CONTEXT

“Glue which connects the otherwise disconnected parts of the application“

- There may be multiple Contexts within an application, further separating it into separated modules

```
public class MainContext : MVCSContext
{
	override public void mapBindings()
	{
		…
	}
}
```

BINDING

“Connection of one or more of anything to one or more of anything else“

- Enforces loose coupling of components throughout the application.

```
Bind<IInputController>().To<KeyboardInputController>();

Bind<PlayerView>().To<PlayerMediator>();

Bind<GameModel>().ToSingleton();
```

SIGNAL

“Alternative Event Dispatching system“

- Callbacks instead of allocating events
- Support for 0 – 4 strongly typed parameters
- Type-safe and will break at compile-time if callback signature does not match

```
public class PlayerMovedSignal : Signal<float>
{
}
```

VIEW

“Controls visible and audible output and input for the user“

- MonoBehavior attached to corresponding GameObject
- Views are “dumb“:
  - Views do not contain any logic
  - Views do not possess any knowledge of the rest of the application

```
public class Player : View
{
        [Inject]
        public IInputController InputController { private get; set; }

        public InputSignal Input { get; private set; }

        protected void Awake()
        {
                base.Awake();
                Input = new InputSignal();
        }

        public void MoveTo(float y)
        {
                transform.position = new Vector3(0f, y);
        }

        private void Update()
        {
                if (InputController.IsUpKeyPressed)
                        Input.Dispatch(InputDirection.UP);
        }
}
```

MEDIATOR

“Connects the View to the rest of the application“

- Automatically created and attached to corresponding View
- Listens to events from the View
- Sends/receives Signals to/from the Application

```
public class PlayerMediator : Mediator
{
	[Inject]
	public PlayerView View { private get; set; }
	[Inject]
	public InputSignal InputSignal { private get; set; }

	override public void OnRegister()
	{
		View.Input.AddListener(OnInputReceived);
	}

	private void OnInputReceived(InputDirection direction)
	{
		InputSignal.Dispatch(direction);
	}
}
```

COMMAND

“Controls the flow of the Application“

- Commands are the Controllers in the classic Model-View-Controller pattern
- Strange automatically listens to Signals that are bound to Commands
- A Signal can be bound to one or more Commands

BINDING SIGNALS TO COMMANDS

```
Bind<StartupSignal>().To<StartupCommand>().Once();

Bind<InputSignal>().To<PlayerInputCommand>().Pooled();
```

```
public class InputSignal : Signal<InputDirection>
{
}
```

```
public class PlayerInputCommand : Command
{
	[Inject]
	public PlayerModel PlayerModel { private get; set; }
	[Inject]
	public InputDirection InputDirection { private get; set; }

	override public void Execute()
	{
		PlayerModel.MovePlayer(InputDirection);
	}
}
```

MODEL

“Stores the data of the application“

- Models do not listen to Signals
- Should not have any dependencies on other Models
- Are not aware of anything outside of their scope

```
public class PlayerModel
{
	[Inject]
	public PlayerMovedSignal PlayerMovedSignal { private get; set; }

	public float PositionY { get; private set; }

	public void MovePlayer(InputDirection direction)
	{
		float newPos = CalculateNewPosition(direction);

		if (IsValidPosition(newPos))
		{
			Position = newPos;
			PlayerMovedSignal.Dispatch(Position);
		}
	}
}
```

```
public class PlayerMediator : Mediator
{
        …
        [Inject]
        public PlayerMovedSignal PlayerMovedSignal { private get; set; }

        override public void OnRegister()
        {
                …
                PlayerMovedSignal.AddListener(OnPlayerMoved);
        }

        private void OnPlayerMoved(float position)
        {
                View.MoveTo(position);
        }

        …
}
```

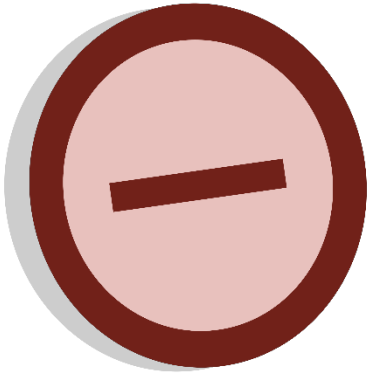
5

CONCLUSION

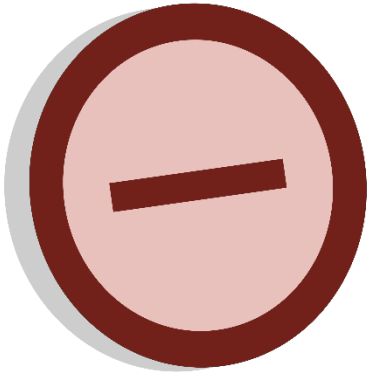
Are You Feeling Strange?

STRUCTURED CODE

FRAMEWORK



PERFORMANCE



LEARNING CURVE

THANK YOU FOR YOUR ATTENTION!

 @Sarky_de

 /jayallanjethwa

 /jayallanjethwa

 jay.allan-jethwa@innogames.com