DSL Case Study in Game Development

External DSL for Character Abilities in Ember Conflict

Internal DSL for Game GUI with F# Computation Expression

and Others

Yingjun Piao

yjpark@gmail.com

2019-01

DSL - Domain Specific Language

- A computer programming language of *limited expressiveness* focused on a *particular domain*. [1]
- **Benefits:**
  - easier to modify, improves productivity
  - allow domain experts to work on it directly
  - different mindset, more specific tools

---

[1] https://martinfowler.com/dsl.html

External DSL

- is a completely *separate language* that is parsed into data that the host language can understand
- **Examples:**
  - Xaml, for GUI element, layout, and event binding
  - CSS, HTML, many template languages
  - Ant, Makefile, Ini, package.json

External DSL Benefits

- very domain specific, easier for domain experts
- more separated approach, data-driven approach
- not limited by host language, can have own syntax
- can support multi languages, and multiple tools

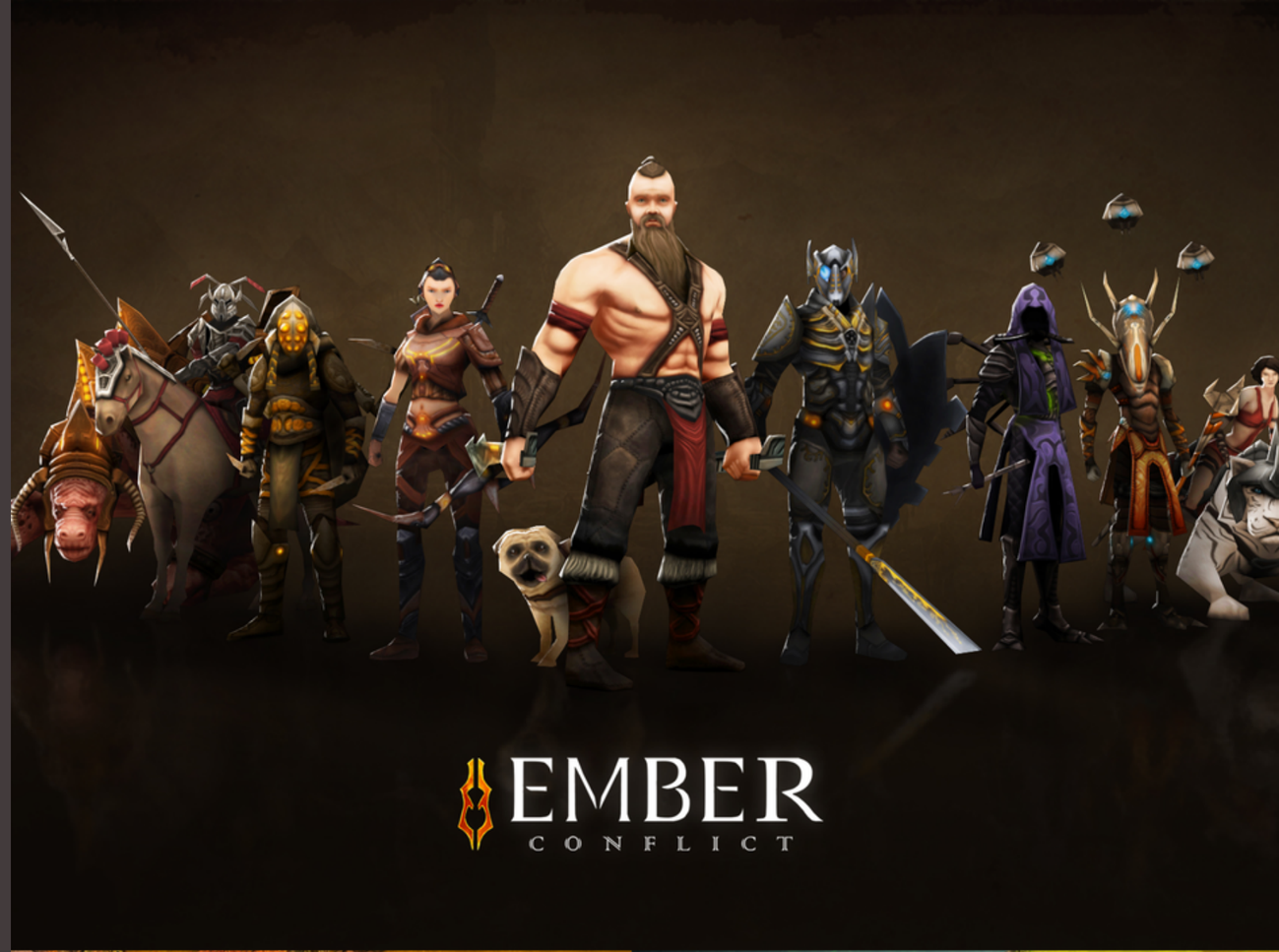
External DSL Drawbacks

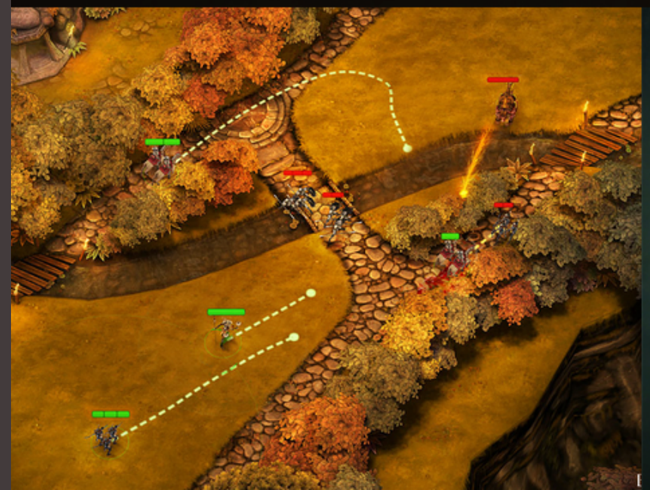
- more works, hard to implement or hard to use
- often too limited, hard to do things not designed
- syntax and semantic separated
- no (free) editor support for semantic
- no (free) correctness check from compiler or linter

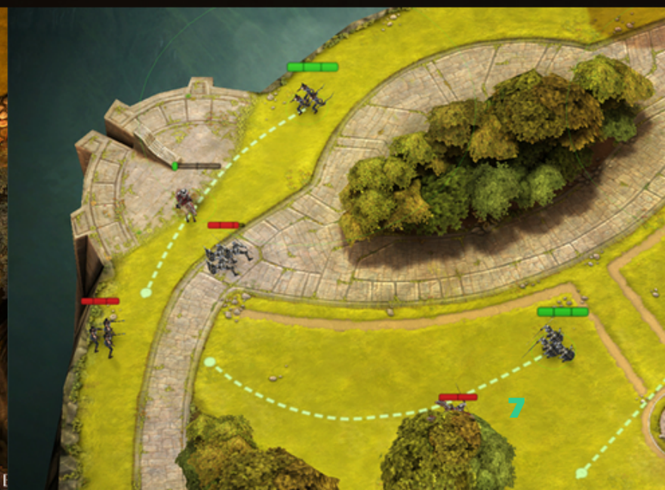
External DSL
for Character Abilities

Ember Conflict

- Real-Time Strategy
- Multi Player, Team Combat
- Full-Stack Unity 3D
- Flexible Abilities
- 23 Characters, 80 Gears
- A Lot More in Plan
- Released on iOS at 2015
- Featured by Apple in US, China







External DSL: [Ember Conflict] `Metadata/Abilities/base/doctor.json` 1/2

```
{
    "abilities": [{
        "skills": [{
            "type": "repeat_trigger",
            "interval": 2.0,
            "kind": "doctor_heal"

        }],
        "define": {
            "doctor_heal_amount": 60,
            "doctor_heal_radius": 4
        },
        "triggers": [{
            "type": "custom",
            "kind": "doctor_heal"
        }],
        ...
```

External DSL: [Ember Conflict] Metadata/Abilities/base/doctor.json 2/2

```
        ...
        "targets": [{
            "type": "dynamic_sensor",
            "radius": "doctor_heal_radius",
            "side": "team"
        },{
            "type": "self"
        },{
            "type": "wounded"
        }],
            "spells": [{
                "type": "heal",
                "key": "doctor_heal",
                "visual_key": "heal_aura",
                "amount": "doctor_heal_amount"
        }]
    }]
}
```

External DSL: [Ember Conflict] Metadata/Abilities/gears/doctor_gear1.json

```
{
    "abilities": [{
        "triggers": [{
            "type": "builtin",
            "kind": "alive"
        }],
        "targets": [{
            "type" :"self"
        }],
        "spells": [{
            "type": "add_buff",
            "key": "add_heal_radius",
            "buff_key": "add_heal_radius_doctor",
            "attrib_key": "doctor_heal_radius",
            "factor": 1,
            "delta": 1,
            "comment": "Flat 1 heal radius buff to Doctor"
        },
        ...
        }]
    }]
}
```

External DSL: [Ember Conflict] `Metadata/Effects/effects.json` Group Selection

```
"group_selection" : {
    "add" : {
        "_": [
            "sprite/destroy?key=selection",
            "sprite/do?key=selection&prefab=squads.effect_sprite",
                "#color=1,1,1,0&sprite=flash&zoom=1.0&play.flash&done.flash=destroy",
            "sprite/do?key=selection_highlight&prefab=squads.effect_sprite",
                "#color=1,1,1,1&save_alpha&sprite=SelectionHighlight&zoom=0.3",
            "animation/cast?speed=1"
        ]
    },
    "remove" : {
        "_": [
            "@sprite/destroy?key=selection_highlight"
        ]
    }
},
```

External DSL: [Ember Conflict] `Metadata/Global/ui.json` Dev GUI Panels

```
"dev": {
    "home": {
        "show": [
            "topleft/do?key=lobby_servers&prefab=ui.dev.launch.lobby_servers#anchor",
            "topright/do?key=battle_servers&prefab=ui.dev.launch.battle_servers#anchor",
            "bottomleft/do?key=ftue_button&prefab=ui.dev.launch.ftue_button"
        ],
        "hide": [
            "topleft/destroy?key=lobby_servers",
            "topright/destroy?key=battle_servers",
            "bottomleft/destroy?key=ftue_button",
            "@try_hide:dev/home/battle_connect"
        ],
    },
    ...
}
```

Internal DSL

- is just a particular idiom of writing code *in the host language*
- **Examples:**
  - LINQ in .NET
  - Rake, Fake

Internal DSL Benefits

- easy to implement, can focus on actual logic
- flexibility, can adjust code structure as code (is real code)
- same syntax, easier to maintain
- compile time verification
- better editor / IDE support

Internal DSL Drawbacks

- restricted by host language, limited syntax
- code might be hard to read, if too much (or bad) abstraction
- can be hard to learn, especially if implemented with macro
- might confuse with mindsets, if the idiom is not distinguished enough from host language
- no (free) hot reload for compiled language, and no (free) correctness check for scripting language

F# Computation Expression

- Computation expressions in F# provide a convenient syntax for writing computations that can be sequenced and combined using control flow constructs and bindings.
- Pretty powerful and complex [2]
- Mostly use `Custom Operations` [3]

---

[2] https://fsharpforfunandprofit.com/series/computation-expressions.html

[3] https://docs.microsoft.com/en-us/dotnet/fsharp/language-reference/computation-expressions#custom-operations

Internal DSL: [F# Game Tutorial] Game.Gui/Builder/Base.fs

```
type WidgetBuilder<'widget when 'widget :> Widget> () =
    member this.Yield (_ : 'a) = this.Zero ()
    abstract Zero : unit -> 'widget

    [<CustomOperation("pos")>]
    member __.Pos (widget : 'widget, l : int, t : int) =
        widget.Left <- l
        widget.Top <- t
        widget
    ...
```

Internal DSL: [F# Game Tutorial] Game.Gui/Builder/Button.fs

```
type TextButtonBuilder () =
    inherit WidgetBuilder<TextButton> ()
    override __.Zero () = new TextButton ()

    [<CustomOperation("text")>]
    member __.Text (widget : TextButton, v : string) =
        widget.Text <- v
        widget
    [<CustomOperation("onUp")>]
    member __.OnUp (widget : TextButton, onUp : EventArgs -> unit) =
        widget.Up.Add onUp
        widget
```

Internal DSL: [F# Game Tutorial] Tank.Sandbox/GuiHelper.fs

```
let boxButton (x : int) (y : int) (t : string) upAction =
    button {
        text t
        pos x y
        size 64 64
        onUp upAction
    }
```

Internal DSL: [F# Game Tutorial] `Tank.Sandbox/TankGui.fs` 1/2

```
let init (x : int) (y : int) (gui : IGui<Panel>) =
    let tank = gui.Game.GetAddon<TankAddon> ()
    let body = tank.Body
    gui.AddChildrenWithOffset (x, y,
        label {
            text "Tank"
            pos 0 0
        },
        boxButton 68 32 "W" <| move body 5.0f,
        boxButton 68 100 "S" <| move body -2.0f,
        boxButton 0 100 "A" <| turn body -90.0f,
        boxButton 136 100 "D" <| turn body 90.0f
    )
```

Internal DSL: [F# Game Tutorial] `Tank.Sandbox/TankGui.fs` 2/2

```
let private move (entity : IEntity) (speed : float32) =
    fun _args ->
        let r = entity.Transform.AbsoluteRotation
        let dir = Vector2 (-MathF.Sin (r), MathF.Cos (r))
        let offset = dir * speed
        entity.Transform.Position <- entity.Transform.Position + offset

let private turn (entity : IEntity) (a : float32) =
    fun _args ->
        entity.Transform.Angle <- entity.Transform.Angle + a
```

Internal 'DSL': [Ember Conflict] Scripts/Battle/Server/Squads/Ai/SquadMind.cs

```
...
root.AddChild(
    //If there is a path, will move along it
    new Sequencer().AddChild(
        new InvertDecorator(new StateCondition<SquadCharactor, SquadData.PathState>(
            SquadData.OWNER_STATE_PATH, SquadData.PathState.None))
    ).AddChild(
        GetMoveAlongPathAction()
    )
);
...
```

Summary

- DSL is powerful and useful in game development
- need to understand the benefits and drawbacks
- and your language regarding to DSL implementation
- choose wisely with DSL design
- and be careful with the implementation

Thank You