Spellie - Programmable Spells for Unity3D

Developed by Patryk Pastuszka

What is spellie?

Spellie is an C# library that allows to create spells programatically or parse **Embedded Spell Language** into Spell Objects. This allows for either procedural or player-created spells, so this makes your life easier.

Currently supported Spell Elements

Sub-lists contain information about ESL names and its usage.

Abstract

- SpellieElement - base for all spellie elements, contains base methods and logic
- SpawningSpellieElement - base for all spellie elements that spawns objects on scene, contains base methods and logic for spawning objects
- [P] at - defines location to spawn object (caster, source, enemy)
- [P] use_global - defines if global positions should be used instead of local ones.
- [P] movement - defines movement of spawned object. Movement is processed by IMovementType interface inheritors.
- [E] onEnd - event that is executed when the spawned object is being destroyed eg. by switching off the switchable element or by ending channeling

Entity

- Damage `damage` - damages entity by specified amount of HP, can be affined
- [P] amount - amount of damage to deal
- [P] affinity - name of affinity this damage is eg. fire
- Effect `effect` - adds effect to entity eg. potion, poison, bleeding
- [P] name - name of the effect to apply
- [P] duration - duration of effect (in seconds)
- [E] onEffectRemoved - executed if effect is removed
- [E] onEffectWearOff - executed if effect duration ends

- [E] onEffectApplied - executed if effect is applied
- Heal `heal` - heals entity by specified amount of HP
- [P] amount - amount of HP to heal
- Remove effect `remove_effect` - removes effect from entity
- [P] name - name of effect to remove

Other

- Destroy source `destroy_source` - element that destroys its source - eg. when used on projectile then it destroys projectile, if on caster, then destroys caster etc.

Primal

- Channeling element `channeling` - allows to channel spells eg. series of projectile, zone that increases size in time and explodes on key up etc.
- [P] frequency - frequency of channel events (events per second)
- [E] onChannel - executed on channel (*frequency* times per second)
- [E] onChannelStarted - executed when channeling begins
- [E] onChannelWearOff - executed when channeling ends
- Switching element `switching` - allows to switch spells, useful for creating auras
- [E] onBegin - executed when switched on
- [E] onEnd - executed when switched off
- [E] onSwitch - executed when switched either on or off
- Delay element `delay` - delays it's sub-spells
- [P] **time** - defines delay time
- [E] **onDelayPassed** - event executed after delay passed

Projectiles

- Projectile `projectile` - shots single projectile in specified direction
- [P] **object** - name of prefab to spawn (Spellie uses own Prefab storage)
- [P] **forward** - defines direction of projectile (forward, backward)
- [P] **distance** - defines distance of projectile from source
- [P] **target** - selects target of the projectile
- [P] duration - duration of object lifetime
- [P] attached - defines parent object (source, target, caster)
- [P] ghost - defines if projectile travels through obstacles like non-entity walls

- [P] piercing - defines amount of targets the projectile pierces
- [E] **onHit** - executed when projectile hits target
- [E] onSpawned - executed when projectile is spawned
- Projectile circle : Projectile `projectile_circle` - shots circle of projectiles around target
- [S] - set of properties from projectile
- [P] amount - amount of projectiles to spawn
- Projectile cone : Projectile `projectile_cone` - shots a cone of projectiles (partial circle) in specified direction
- [S] - set of properties from projectile
- [P] amount - amount of projectiles in cone
- [P] centered - defines if cone is centered around forward direction
- [P] startAngle - offset of cone (in degrees)
- [P] angle - angle of cone (spread)
- Random circle : Projectile `random_projectile_circle` - shots projectiles randomly around circle
- [S] - set of properties from projectile
- [P] amount - amount of projectiles to spawn
- [P] delay - delay between projectile spawns
- Random sphere : Projectile `random_projectile_sphere` - shots projectiles randomly around sphere
- [S] - set of properties from projectile
- [P] amount - amount of projectiles to spawn
- [P] delay - delay between projectile spawns

Zone

- Zone element `zone` - creates zone, eg. burning ground etc.
- [P] object - name of prefab to spawn
- [P] duration - lifetime of zone
- [P] attached - name of parent which the zone is attached to (source, target, caster)
- [E] onEnter - executed when entity enters zone
- [E] onExit - executed when entity exits zone
- [E] onStay - executed each frame entity stays in zone
- [E] onSpawned - executed when zone is spawned

Inside `those fields` you have ESL names specified. ESL is described later.

Affinities explained

Affinity is type of damage eg. fire, water, earth, air. Water deals more damage to fire, when fire deals no damage against fire. It's a system that allows to set damage multipliers against damage types.

Events

Every events consist of `target` property, which allows to be used as limit of event executions. Acceptable targets are: `evil`, `friendly`, `player`, `npc`, `world`, `caster`

Attached objects

Spawned objects may be attached using `attached` property. Accepted attachments are: `none`, `caster`, `source`, `target`. Default is `none` which means that object is world-space based.

Direction

Some moving object have `direction` property which defines direction of movement - `forward` or `reverse`, where reverse just simply makes object fly toward source instead of from source.

Extending SpellieElement

You can create your own Spellie Elements. To do so it's recommended to copy existing element and clean code.

```
public class DamageElement<TSelf> : SpellieElement<DamageElement<TSelf>>
        where TSelf:DamageElement<TSelf>
```

As you can see DamageElement extends from SpellieElement that is generic of DamageElement. This allows for Spellie to have C# fluent builders. You just need to replace DamageElement with name of your element.

```
[SpellieElement("damage")]
   public abstract class DamageElement
```

Also above you need an abstract that will be used to create your element. Unforunately generic types cannot be easily instantiated, so that abstract contains builder to make your generic type available for construction.

Also see SpellieElement attribute, it's used to define ESL name for this element. We'll mention ESL later...

To add an property to your SpellieElement you need to use SpellieProperty attribute

```
[SpellieProperty("affinity")] public string affinity;
```

SpelliePropertyAttribute has 2 parameters - string that is name of property in ESL and boolean that is false - which causes spell to throw error if the property is missing. Used commonly in ESL compilation.

To add an event into your elemeny, you need SpellieEventAttribute

```
    [SpellieEvent("onEffectRemoved")]
       public SpellieEvent onEffectRemoved = new SpellieEvent();
```

it also has 2 parameters as SpellieProperty with same usage, however the name is for event instead of property, it matters when it comes to ESL.

ESL - Embedded Spell Language

ESL is a simple way to create spells. It can be done quickly and makes it easily readable. Example is below

```
projectile_circle
 piercing 0
 amount 8
 direction forward
 object projectile
 distance 1.5
 movement linear
  speed (0,0,2)
 onHit
  target evil
  damage
   amount 50
```

You need to start ESL with an Spellie Element - this will be a primary element executed on caster. Then every Spellie Element may have properties or events. Event cannot be used on other event or on movement. Spellie Movement and Events also can have properties, however the property must be implemented inside that Element/Event/Movement, otherwise Parser will throw error. Same happens if you miss required property. Every deeper element needs one space longer indent. As you can see above properties and events on projectile_circle has one indent, however properties and elements at onHit event or linear movement has 2 space indent. Property of damage element has 3 space indent. Thanks to this parser creates spell dynamically and does not need to create whole spell when it finds error.

Movement of SpellieObjects

As you probably have seen Spellie has possibility to define movement of the objects. It's quite good, because it's processed on separate thread, which allows you to move large amount of objects with quite good performance.

Every movement type needs to inherit from IMovementType and have SpellieMovement attribute which defines name of the movement used in ESL.

It can also inherit from IIgnorePosition, IIgnoreRotation or IIgnoreScale which makes the movement to ignore specified movement information. Or you just can create movement that is completely ignored and create physics-engine-based one.

See currently existing movements for reference, because nobody would understand the description of movement creation...