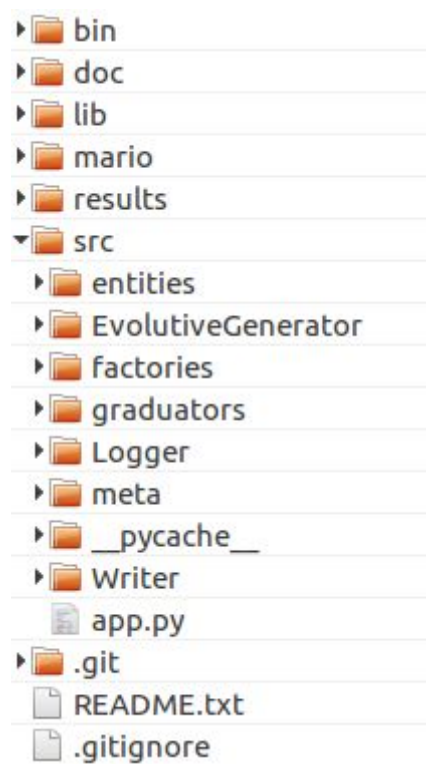


Annexe

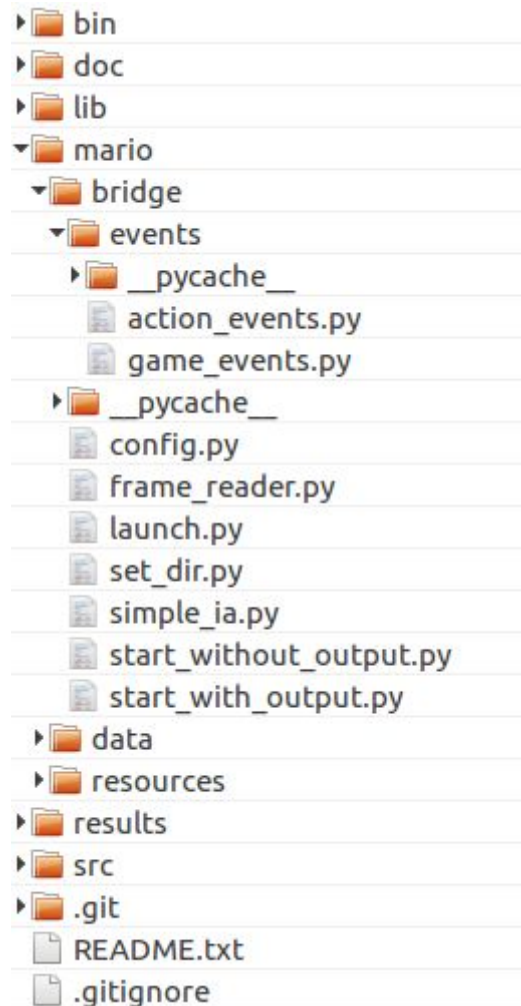
Arborescence globale du projet



Les codes sources présents dans cette annexe ont été rassemblés par thèmes.

Étape 1 : Adapter le jeu

Dossier /mario/bridge



- Modifications du jeu non incluses ici (5000 lignes de codes dans 29 fichiers).
- *FrameReader* : composant chargé de lire ce qui arrive en jeu et de le traduire en évènements compréhensibles par les intelligences artificielles.

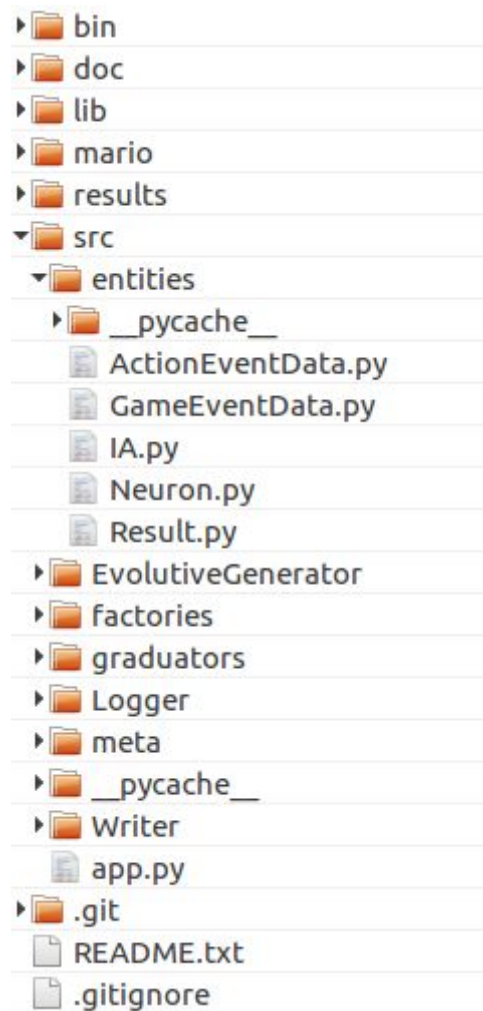
```

1  import pygame as pg
2  from lib.inject_arguments import inject_arguments
3
4  from mario.bridge.events.game_events import *
5
6
7  class FrameReader:
8      """Read the Frame event and make other game events """
9
10     @inject_arguments
11     def __init__(self, event_dispatcher):
12         self.event_dispatcher.listen('game.frame', self.handle_frame)
13         self.frame = None
14
15     @inject_arguments
16     def handle_frame(self, frame):
17         """Handle Frame event """
18
19         self.build_events('game.block', Block,
20                          ['brick_group', 'coin_box_group', 'ground_group', 'pipe_group', 'step_group'], frame)
21         self.build_events('game.enemy', Enemy, ['enemy_group'], frame)
22         self.build_events('game.powerup', Powerup, ['powerup_group'], frame)
23         self.build_events('game.coin', Coin, ['coin_group'], frame)
24
25
26     def build_events(self, event_name, event_class, groups, frame):
27         """Build DetectedComponent game event for each displayed sprite of the groups """
28
29         sprites = []
30         for group in groups:
31             sprites.extend(frame.sprite_groups[group].sprites())
32         viewport_sprite = ViewportSprite(frame.viewport)
33         displayed_sprites = pg.sprite.spritecollide(viewport_sprite, sprites, False)
34
35         # Make the events and dispatch
36         for block in displayed_sprites:
37             self.event_dispatcher.dispatch(event_name, event_class(block.rect, frame.mario.rect, frame.current_frame))
38             # print('game.block', Block(block.rect, frame.mario.rect, frame.current_frame))
39
40         #debug
41         pg.display.set_caption("Displayed blocks = " + str(len(displayed_sprites)))
42
43
44     class ViewportSprite(pg.sprite.Sprite):
45         """A false sprite containing viewport """
46
47         def __init__(self, viewport_rect):
48             pg.sprite.Sprite.__init__(self)
49             self.rect = viewport_rect

```

Étape 2 : Modéliser les intelligences avec des *GeneticElement*

Dossier /src/entities



- *GeneticElement*
- *IA*
- *Neuron*
- *GameEvent*
- *ActionEvent*

06/14/17 05:28:38 /home/zz/Documents/TIPE/src/EvolutiveGenerator/GeneticElement.py

```
1  #!/usr/bin/python3.4
2  # -*-coding:Utf-8 -*
3
4  from abc import ABCMeta
5
6
7  class GeneticElement(metaclass=ABCMeta):
8      """Carry the genetic information
9
10     This is an abstract class to inherit.
11     A genetic element carries one or several genetic informations or contains
12     other genetic elements.
13
14     Evolution logic is handled by an external GeneticElementFactory.
15     """
16
17     pass
```

06/14/17 05:29:02 /home/zz/Documents/TIPE/src/entities/IA.py

```
1  #!/usr/bin/python3.4
2  # -*-coding:Utf-8 -*
3
4  from lib.inject_arguments import inject_arguments
5  from lib.XMLRepr import XMLRepr
6
7  from src.EvulativeGenerator.GeneticElement import GeneticElement
8
9
10 class IA(GeneticElement, XMLRepr):
11     """An IA"""
12
13     @inject_arguments
14     def __init__(self, ia, neurons=list()):
15         """Init the IA
16
17         Expects:
18             id to be a integer, unique among IA's of a processus
19             neurons to be a list of Neuron
20         """
21
22         if not type(neurons) is list:
23             raise ValueError('Neurons should be a list. ')
24
25
26     def reprJSON(self):
27         return self.__dict__
28
29     def __repr__(self):
30         return super().__repr__(displaySequencesNames=False)
```

```

1  #!/usr/bin/python3.4
2  # -*-coding:Utf-8 -*
3
4  from lib.inject_arguments import inject_arguments
5  from lib.XMLRepr import XMLRepr
6
7  from src.EvolutiveGenerator.GeneticElement import GeneticElement
8  from src.entities.GameEventData import GameEventData
9  from src.entities.ActionEventData import ActionEventData
10
11
12  class Neuron(GeneticElement, XMLRepr):
13      """A link between an game event and an action event """
14
15      @inject_arguments
16      def __init__(self, game_event_data, action_event_data):
17          """Init the neuron
18
19          Expects:
20              game_event_data to be a GameEventData or a tuple (event_name, coord)
21              action_event_data to be a ActionEventData or a tuple (action_class, duration)
22          """
23
24          if type(game_event_data) is tuple:
25              self.game_event_data = GameEventData(*game_event_data)
26          if type(action_event_data) is tuple:
27              self.action_event_data = ActionEventData(*action_event_data)
28
29
30      def event_dispatcher():
31          doc = "The event_dispatcher property. "
32
33          def fget(self):
34              return self._event_dispatcher
35
36          def fset(self, event_dispatcher):
37              """Register the neuron to the event_dispatcher """
38              if hasattr(self, 'listener_id'):
39                  del self.event_dispatcher
40
41              self._event_dispatcher = event_dispatcher
42              self.listener_id = self._event_dispatcher.listen(self.game_event_data.event_name, self.onEvent)
43
44          def fdel(self):
45              """Detach the listener """
46              if hasattr(self, 'listener_id'):
47                  self._event_dispatcher.detach(self.listener_id)
48                  del self.listener_id
49
50              del self._event_dispatcher
51          return locals()
52      event_dispatcher = property(**event_dispatcher())
53
54      def __del__(self):
55          if hasattr(self, 'event_dispatcher'):
56              del self.event_dispatcher
57
58
59      def onEvent(self, event):
60          if self.game_event_data.checkCoord(event):
61              self.event_dispatcher.dispatch('action', self.action_event_data.buildAction(event))
62
63
64      def reprJSON(self):
65          return {
66              'game_event_data': self.game_event_data,
67              'action_event_data': self.action_event_data
68          }
69
70      def __repr__(self):
71          return super().__repr__(['game_event_data', 'action_event_data'])

```

06/14/17 05:29:59 /home/zz/Documents/TIPE/src/entities/GameEventData.py

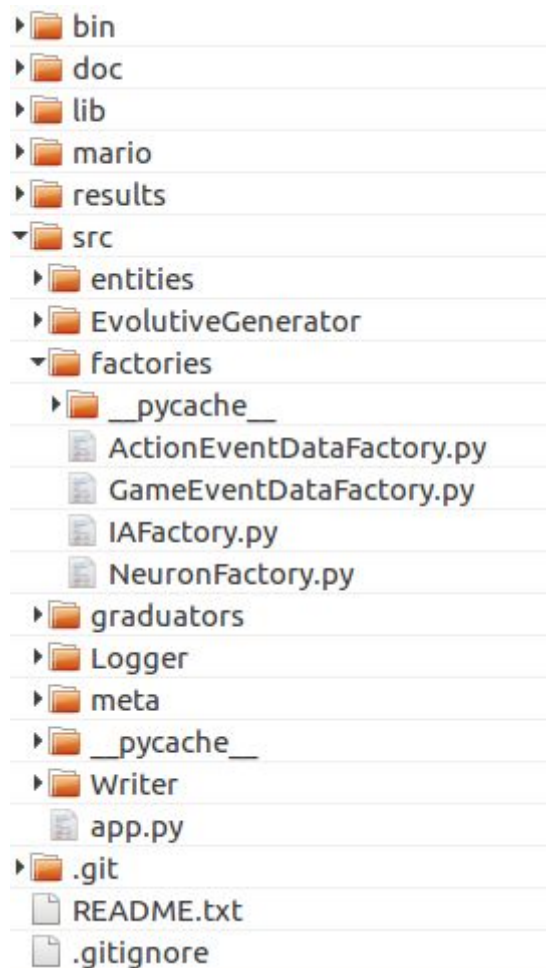
```
1  #!/usr/bin/python3.4
2  # -*-coding:Utf-8 -*
3
4  from lib.inject_arguments import inject_arguments
5  from lib.XMLRepr import XMLRepr
6
7  from src.EvolutiveGenerator.GeneticElement import GeneticElement
8
9
10 class GameEventData(GeneticElement, XMLRepr):
11     """An game event data, part of a neuron """
12
13     @inject_arguments
14     def __init__(self, event_name, coord):
15         pass
16
17
18     def checkCoord(self, event):
19         return self.coord['x'] >= event.left and self.coord['x'] <= event.right \
20             and self.coord['y'] >= event.top and self.coord['y'] <= event.bottom
21
22
23     def reprJSON(self):
24         return self.__dict__
25
26     def __repr__(self):
27         return super().__repr__(
28             attributes=['event_name', 'coord'],
29             __dict__={'event_name': self.event_name, 'coord': (self.coord['x'], self.coord['y'])})
30
```


06/14/17 05:30:14 /home/zz/Documents/TIPE/src/entities/ActionEventData.py

```
1  #!/usr/bin/python3.4
2  # -*-coding:Utf-8 -*
3
4  from lib.inject_arguments import inject_arguments
5  from lib.XMLRepr import XMLRepr
6
7  from src.EvolutiveGenerator.GeneticElement import GeneticElement
8
9
10 class ActionEventData (GeneticElement, XMLRepr):
11     """An action event data, part of a neuron """
12
13     @inject_arguments
14     def __init__(self, action_class, duration):
15         pass
16
17
18     def buildAction(self, event):
19         return self.action_class(self.duration, event.current_frame)
20
21
22     def reprJSON(self):
23         return {
24             'action_class': self.action_class.__name__,
25             'duration': self.duration
26         }
27
28     def __repr__(self):
29         return super().__repr__(__dict__=self.reprJSON())
```

Étape 3 : Manipuler les *GeneticElement* avec les *GeneticElementFactory*

Dossier /src/factories



- *GeneticElementFactory*
- *IAFactory*
- *NeuronFactory*
- *GameEventFactory*
- *ActionEventFactory*

```

1  #!/usr/bin/python3.4
2  # -*-coding:Utf-8 -*
3
4  from abc import ABCMeta, abstractmethod
5
6
7  class GeneticElementFactory (metaclass=ABCMeta):
8      """Handle the evolution logic of a GeneticElement
9
10     This is an abstract class to inherit.
11     This is a static class.
12     It brings the evolution logic of the GeneticElement through the following
13     class methods:
14         +create() -> GeneticElement
15         +mutate(GeneticElement)
16         +combine(GeneticElement, GeneticElement) -> GeneticElement
17         +breed(GeneticElement, GeneticElement) -> GeneticElement
18     Evolution logic may typically use recursive process over children of elements.
19     """
20
21     @property
22     @abstractmethod
23     def genetic_element_class (self):
24         """The GeneticElement based class """
25
26         raise NotImplementedError
27
28
29     @staticmethod
30     @abstractmethod
31     def create(parent = None, children = [], cascade = True):
32         """Create a GeneticElement from void
33
34         An essential element of the generation process.
35         This is a static method which has to be implemented.
36
37         return GeneticElement
38         """
39
40         raise NotImplementedError
41
42
43     @staticmethod
44     @abstractmethod
45     def mutate(element):
46         """Operates a genetic mutation
47
48         This is a static method which has to be implemented.
49
50         This is rather designed for internal use, see generate() instead.
51         """
52
53         raise NotImplementedError
54
55
56     @staticmethod
57     def combine(element1, element2):
58         """Form a new GeneticElement, combination of two ones
59
60         Combine two GeneticElement to form an offspring.
61         This is a static method which has to be implemented.
62
63         This is rather designed for internal use, see generate() instead.
64
65         Expects:
66             element1, element2 to be GeneticElement's
67
68         return GeneticElement
69         """
70
71         raise NotImplementedError
72
73
74     @classmethod
75     def breed(cls, element1, element2):
76         """Generate a new GeneticElement, final offspring of two ones
77
78         Call combine() then mutate().
79         This is a class method.
80
81         Expects:
82             element1, element2 to be a GeneticElement's
83
84         return GeneticElement
85         """
86
87         new_element = cls.combine(element1, element2)
88         cls.mutate(new_element)
89         return new_element

```

06/14/17 05:31:13 /home/zz/Documents/TIPE/src/factories/IAFactory.py

```
1  #!/usr/bin/python3.4
2  # -*-coding:Utf-8 -*
3
4  from random import randint, random
5  from math import ceil
6  from copy import deepcopy
7
8  from lib.inherit_docstring import inherit_docstring
9  from src.meta.ABCInheritableDocstringsMeta import ABCInheritableDocstringsMeta
10 from src.EvolutiveGenerator.GeneticElementFactory import GeneticElementFactory
11 from src.entities.IA import IA
12 from src.factories.NeuronFactory import NeuronFactory
13
14
15 randindex = lambda it: randint(0, len(it)-1)
16 def randindex_safe(it):
17     if len(it) < 3:
18         raise ValueError("Iterable should have a least 3 elements. ")
19     return randint(1, len(it)-2)
20
21
22 class IAFactory(GeneticElementFactory, metaclass=ABCInheritableDocstringsMeta):
23     """IA factory"""
24
25     @property
26     @inherit_docstring
27     def genetic_element_class(self):
28         return IA
29
30     last_ia_id = -1
31
32     @classmethod
33     def onProcessusStart(cls, event):
34         cls.last_ia_id = -1
35
36     @classmethod
37     def newIaId(cls):
38         cls.last_ia_id += 1
39         return cls.last_ia_id
40
41     @classmethod
42     def updateIaId(cls, ia_id):
43         cls.last_ia_id = max(cls.last_ia_id, ia_id)
44
45
46     @classmethod
47     @inherit_docstring
48     def create(cls):
49         neurons = list()
50         for i in range(3 + randint(0, 3)):
51             neurons.append(NeuronFactory.create())
52         return IA(cls.newIaId(), neurons)
53
54
55     @staticmethod
56     @inherit_docstring
57     def mutate(element):
58         if random() < .2:
59             element.neurons.insert(randindex(element.neurons), NeuronFactory.create())
60         if random() < .1 and len(element.neurons) > 3:
61             element.neurons.pop(randindex(element.neurons))
62         for neuron in element.neurons:
63             if random() < .2:
64                 NeuronFactory.mutate(neuron)
65
66
67     @classmethod
68     @inherit_docstring
69     def combine(cls, element1, element2):
70         neurons = element1.neurons[:randindex_safe(element1.neurons)] + element2.neurons[randindex_safe(element2.neurons):]
71
72         # Ensure you have a least 3 neurons
73         if len(neurons) < 3:
74             return cls.combine(element1, element2)
75
76         # Duplicate neurons instead of reuse ones
77         neurons = [deepcopy(neuron) for neuron in neurons]
78         return IA(cls.newIaId(), neurons)
79
80
81     @classmethod
82     def hydrate(cls, data):
83         cls.updateIaId(data['id'])
84
85         return IA(data['id'], [ NeuronFactory.hydrate(neuron_data) for neuron_data in data['neurons'] ])
```

06/14/17 05:31:33 /home/zz/Documents/TIPE/src/factories/NeuronFactory.py

```
1  #!/usr/bin/python3.4
2  # -*-coding:Utf-8 -*
3
4  from lib.inherit_docstring import inherit_docstring
5  from random import randint
6
7  from src.meta.ABCInheritableDocstringsMeta import ABCInheritableDocstringsMeta
8  from src.EvolutiveGenerator.GeneticElementFactory import GeneticElementFactory
9  from src.entities.Neuron import Neuron
10 from src.factories.GameEventDataFactory import GameEventDataFactory
11 from src.factories.ActionEventDataFactory import ActionEventDataFactory
12
13
14 class NeuronFactory(GeneticElementFactory, metaclass=ABCInheritableDocstringsMeta):
15     """Neuron factory"""
16
17     @property
18     @inherit_docstring
19     def genetic_element_class(self):
20         return Neuron
21
22
23     @staticmethod
24     @inherit_docstring
25     def create():
26         return Neuron(GameEventDataFactory.create(), ActionEventDataFactory.create())
27
28
29     @staticmethod
30     @inherit_docstring
31     def mutate(element):
32         if randint(0, 1):
33             GameEventDataFactory.mutate(element.game_event_data)
34         else:
35             ActionEventDataFactory.mutate(element.action_event_data)
36
37
38     @staticmethod
39     def hydrate(data):
40         return Neuron(
41             GameEventDataFactory.hydrate(data['game_event_data']),
42             ActionEventDataFactory.hydrate(data['action_event_data'])
43         )
```

```

1  #!/usr/bin/python3.4
2  # -*-coding:Utf-8 -*
3
4  from lib.inherit_docstring import inherit_docstring
5  from random import choice, randint
6
7  from src.meta.ABCInheritableDocstringsMeta import ABCInheritableDocstringsMeta
8  from mario.data.constants import SCREEN_HEIGHT, SCREEN_WIDTH, GROUND_HEIGHT
9  from src.EvolutiveGenerator.GeneticElementFactory import GeneticElementFactory
10 from src.entities.GameEventData import GameEventData
11
12
13 class GameEventDataFactory (GeneticElementFactory, metaclass=ABCInheritableDocstringsMeta):
14     """GameEventData factory"""
15
16     @property
17     @inherit_docstring
18     def genetic_element_class (self):
19         return GameEventData
20
21     GAME_EVENT_NAMES = ('game.block', 'game.enemy', 'game.powerup', 'game.coin')
22
23     MIN_X = -int(SCREEN_WIDTH / 2) # max left
24     MAX_X = SCREEN_WIDTH # max right
25     MIN_Y = -GROUND_HEIGHT # max top
26     MAX_Y = SCREEN_HEIGHT # max bottom
27
28
29     @classmethod
30     @inherit_docstring
31     def create(cls):
32         return GameEventData (cls.createEventName (), cls.createCoor ())
33
34     @classmethod
35     @inherit_docstring
36     def mutate(cls, element):
37         if randint(0, 1):
38             element.event_name = cls.createEventName ()
39         else:
40             element.coor = cls.mutateCoor (element.coor)
41
42
43     @staticmethod
44     def hydrate(data):
45         return GameEventData (**data)
46
47
48     @classmethod
49     def createEventName (cls):
50         if randint(0, 9):
51             return cls.GAME_EVENT_NAMES [0]
52         return choice (cls.GAME_EVENT_NAMES [1:])
53
54     @classmethod
55     def createCoor (cls):
56         return {
57             'x': randint (cls.MIN_X, cls.MAX_X),
58             'y': randint (cls.MIN_Y, cls.MAX_Y)
59         }
60
61     @classmethod
62     def mutateCoor (cls, coor):
63         coor['x'] += randint (-100, 100)
64         coor['y'] += randint (-100, 100)
65
66         if coor['x'] < cls.MIN_X:
67             coor['x'] = cls.MIN_X
68         if coor['x'] > cls.MAX_X:
69             coor['x'] = cls.MAX_X
70         if coor['y'] < cls.MIN_Y:
71             coor['y'] = cls.MIN_Y
72         if coor['y'] > cls.MAX_Y:
73             coor['y'] = cls.MAX_Y
74         return coor

```

```

1  #!/usr/bin/python3.4
2  # -*-coding:Utf-8 -*
3
4  from lib.inherit_docstring import inherit_docstring
5  from lib.choices import choices
6  from lib.gauss_int import gauss_int
7  from random import randint
8
9  from src.meta.ABCInheritableDocstringsMeta import ABCInheritableDocstringsMeta
10 from mario.bridge.events.action_events import Jump, Left, Right, Down, Fire
11 from src.EvolutiveGenerator.GeneticElementFactory import GeneticElementFactory
12 from src.entities.ActionEventData import ActionEventData
13
14
15 class ActionEventDataFactory (GeneticElementFactory, metaclass=ABCInheritableDocstringsMeta):
16     """ActionEventData factory"""
17
18     @property
19     @inherit_docstring
20     def genetic_element_class (self):
21         return ActionEventData
22
23     ACTION_CLASSES = (Jump, Left, Right, Down, Fire)
24
25
26     @classmethod
27     @inherit_docstring
28     def create(cls):
29         action_class = cls.createActionClass ()
30         return ActionEventData (action_class, cls.createDuration (action_class))
31
32     @classmethod
33     @inherit_docstring
34     def mutate(cls, element):
35         if randint(0, 1):
36             element.action_class = cls.createActionClass ()
37         else:
38             element.duration = cls.createDuration (element.action_class)
39
40
41     @classmethod
42     def hydrate(cls, data):
43         for action_class in cls.ACTION_CLASSES:
44             if action_class.__name__ == data['action_class']:
45                 return ActionEventData (action_class, data['duration'])
46         return ValueError("Action class {} doesn't exist.".format(data['action_class']))
47
48
49     @classmethod
50     def createActionClass(cls):
51         return choices(cls.ACTION_CLASSES, weights=[35, 10, 35, 10, 10])[0]
52
53     @staticmethod
54     def createDuration (action_class):
55         if action_class == Jump:
56             return gauss_int (2, 38)
57         else:
58             return randint(0, 25)

```