```python
#!/usr/bin/python3.4
# -*-coding:Utf-8 -*

from argparse import ArgumentParser
from math import inf
from time import time

from lib.eventdispatcher import EventDispatcher
from mario.bridge.frame_reader import FrameReader
from mario.bridge.config import Config
from .EvolutiveGenerator.Generator import Generator
from .factories.IAFactory import IAFactory
from .graduators.IAGraduator import IAGraduator
from .graduators.GameOptimizer import GameOptimizer
from .Writer.Writer import Writer
from .Logger.FileLogger import FileLogger
from .Logger.ConsoleLogger import ConsoleLogger
from .Writer.PathManager import PathManager
from .Writer.Reader import Reader


def instanciateGenerator(show):
    event_dispatcher = EventDispatcher()
    FrameReader(event_dispatcher)
    GameOptimizer(event_dispatcher)
    return Generator(IAFactory, IAGraduator(event_dispatcher, show), [Writer(), FileLogger(), ConsoleLogger()],
        lambda state: True in [score.percent >= 1. for score, individual in state.grading]
    )

def checkProcessusExists(processus_id):
    if not Reader.processusExists(processus_id):
        raise ValueError("Processus with id={} doesn't exist.".format(processus_id))

def new(args):
    """New processus"""
    population = instanciateGenerator(args.show).process(
        PathManager.newProcessusId(), args.generations, args.pop_length, args.proportion, args.chance
    )

def resume(args):
    """Resume a processus"""
    checkProcessusExists(args.processus_id)
    population = instanciateGenerator(args.show).resume(Reader.getProcessusState(args.processus_id))

def play(args):
    """Play the best individual of a processus' last generation """
    checkProcessusExists(args.processus_id)
    # Get IA
    if args.ia_id is None:
        ia, generation_id = Reader.getBestIa(args.processus_id, args.generation_id)
        print('The best AI is {}.'.format(ia.id), flush=True)
    else:
        ia, generation_id = Reader.getIa(args.processus_id, args.ia_id)
    # Play IA
    event_dispatcher = EventDispatcher()
    FrameReader(event_dispatcher)
    graduator = IAGraduator(event_dispatcher, show=True)
    if args.as_grading:
        print(
            "Attention : Malgré que le visionnage présenté soit le plus proche possible des conditions d'évaluation, des aléas 
subsistent. "
            "Si vous cherchez à visionner une performance difficile à reproduire, n'hésitez pas à rééssayer plusieurs fois.   "
        , flush=True)
        GameOptimizer(event_dispatcher)
        graduator.grade(ia, generation_id)
    else:
        graduator.gradeIAWithConfig(ia, Config(True, event_dispatcher))

def print_data(args):
    checkProcessusExists(args.processus_id)

    data = Reader.getData(args.processus_id)
    txt1 = 'Générations,Scores des intelligences'
    for generation_id, grading in data:
        txt1 += '\n' + str(generation_id)
        for result, ia_id in grading:
            txt1 += ',' + str(result['score'])
    txt2 = 'Générations,Scores des intelligences'
    for generation_id, grading in data:
        txt2 += '\n' + str(generation_id)
        for result, ia_id in grading:
            txt2 += ',' + str(result['max_x'])

    path1 = PathManager.getPath(args.processus_id, read_only=True).parent / 'data' / (str(time()) + '.score.csv')
    path2 = PathManager.getPath(args.processus_id, read_only=True).parent / 'data' / (str(time()) + '.distance.csv')
    PathManager.makeDir(path1.parent)
    path1.write_text(txt1)
    path2.write_text(txt2)


# Build parser
parser = ArgumentParser()
subparsers = parser.add_subparsers()

new_parser = subparsers.add_parser('new')
new_parser.add_argument('pop_length', type=int)
new_parser.add_argument('--generations', default=inf, type=int)
new_parser.add_argument('--proportion', default=0.5, type=float)
new_parser.add_argument('--chance', default=0, type=float)
new_parser.add_argument('--show', dest='show', action='store_true')
new_parser.set_defaults(command=new, show=False)

resume_parser = subparsers.add_parser('resume')
resume_parser.add_argument('processus_id', type=int)
resume_parser.add_argument('--show', dest='show', action='store_true')
resume_parser.set_defaults(command=resume, show=False)

play_parser = subparsers.add_parser('play')
play_parser.add_argument('processus_id', type=int)
play_parser.add_argument('--generation_id', type=int)
play_parser.add_argument('--ia_id', type=int)
play_parser.add_argument('--as_grading', dest='as_grading', action='store_true')
play_parser.set_defaults(command=play, as_grading=False)

print_parser = subparsers.add_parser('print')
print_parser.add_argument('processus_id', type=int)
print_parser.set_defaults(command=print_data)

# Parse arguments
args = parser.parse_args()
if hasattr(args, 'command'):
    args.command(args)
else:
    print('No command given, use --help')
```