```
//
// ********************************************************************
// * License and Disclaimer                                           *
// *                                                                  *
// * The  Geant4 software  is  copyright of the Copyright Holders  of *
// * the Geant4 Collaboration.  It is provided  under  the terms  and *
// * conditions of the Geant4 Software License,  included in the file *
// * LICENSE and available at  http://cern.ch/geant4/license .  These *
// * include a list of copyright holders.                             *
// *                                                                  *
// * Neither the authors of this software system, nor their employing *
// * institutes,nor the agencies providing financial support for this *
// * work  make  any representation or  warranty, express or implied, *
// * regarding  this  software system or assume any liability for its *
// * use.  Please see the license in the file  LICENSE  and URL above *
// * for the full disclaimer and the limitation of liability.         *
// *                                                                  *
// * This  code  implementation is the result of  the  scientific and *
// * technical work of the GEANT4 collaboration.                      *
// * By using,  copying,  modifying or  distributing the software (or *
// * any work based  on the software)  you  agree  to acknowledge its *
// * use  in  resulting  scientific  publications,  and indicate your *
// * acceptance of all terms of the Geant4 Software license.          *
// ********************************************************************
//
/// \file electromagnetic/TestEm3/src/HEPDSWPrimaryGeneratorAction.cc
/// \brief Implementation of the HEPDSWPrimaryGeneratorAction class
//
// $Id$
//
//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo......
//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo......

#include "HEPDSWPrimaryGeneratorAction.hh"

#include "HEPDSWPrimaryGeneratorMessenger.hh"
#include "HEPDSWDetectorConstruction.hh"

#include "G4Event.hh"
#include "G4ParticleGun.hh"
#include "G4ParticleTable.hh"
#include "G4ParticleDefinition.hh"
#include "G4SystemOfUnits.hh"
#include "Randomize.hh"

//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo......

HEPDSWPrimaryGeneratorAction::HEPDSWPrimaryGeneratorAction(HEPDSWDetectorConstru
ction* det)
  :position(0),direction(0),fDetector(det)
{
  G4int n_particle = 1;
  fParticleGun  = new G4ParticleGun(n_particle);
  SetDefaultKinematic();
  random=false;
  dummy=false;
  centerpointing=false;
  powerlaw = false;
  //create a messenger for this class
  fGunMessenger = new HEPDSWPrimaryGeneratorMessenger(this);
}

//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo......

HEPDSWPrimaryGeneratorAction::~HEPDSWPrimaryGeneratorAction()
{
  delete fParticleGun;
  delete fGunMessenger;
}
```



```
//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo......

void HEPDSWPrimaryGeneratorAction::SetDefaultKinematic()
{
  G4ParticleTable* particleTable = G4ParticleTable::GetParticleTable();
  G4String particleName;
  G4ParticleDefinition* particle = particleTable->FindParticle(particleName="e-"
);
  fParticleGun->SetParticleDefinition(particle);
  fParticleGun->SetParticleMomentumDirection(G4ThreeVector(0.,0.,-1.));
  fParticleGun->SetParticleEnergy(1.*GeV);
  G4double position = 0.5*(fDetector->GetWorldSizeZ());
  fParticleGun->SetParticlePosition(G4ThreeVector(0.*cm,0.*cm,position));
}

void HEPDSWPrimaryGeneratorAction::SetEnergy(G4double ene)
{
  fParticleGun->SetParticleEnergy(ene);
}

void HEPDSWPrimaryGeneratorAction::SetParticle(G4String part)
{
  G4ParticleTable* particleTable = G4ParticleTable::GetParticleTable();
  G4String particleName = part;
  G4ParticleDefinition* particle = particleTable->FindParticle(particleName);
  fParticleGun->SetParticleDefinition(particle);
}

void HEPDSWPrimaryGeneratorAction::SetDummy(G4double Xpos,G4double Ypos,G4double
 theta){
  dummy=true;
  position = G4ThreeVector(Xpos,Ypos,0.5*(fDetector->GetWorldSizeZ()));
  G4double phi = 0;
  direction = G4ThreeVector(cos(phi)*sin(theta),sin(phi)*sin(theta),-cos(theta))
;

}

void HEPDSWPrimaryGeneratorAction::SetPowerLaw(G4double aEmin,G4double aEmax,G4d
ouble aGamma){
  powerlaw = true;
  eminPL = aEmin;
  emaxPL = aEmax;
  gammaPL = aGamma;
}


//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo......

void HEPDSWPrimaryGeneratorAction::GeneratePrimaries(G4Event* anEvent)
{
  //this function is called at the begining of event
  //
  //randomize the beam, if requested.
  if(random){
    G4double phi = 2*pi*G4RandFlat::shoot();
    G4double theta = std::sqrt(G4RandFlat::shoot());
    theta = std::acos(theta);
    G4double Xmax = 0.5*(fDetector->GetWorldSizeX());
    G4double Ymax = 0.5*(fDetector->GetWorldSizeY());
    G4double Zmax = 0.5*(fDetector->GetWorldSizeZ());
    position = G4ThreeVector(-Xmax+2*Xmax*G4RandFlat::shoot(),-Ymax+2*Ymax*G4Ran
dFlat::shoot(),Zmax);
    if(powerlaw)
      fParticleGun->SetParticleEnergy(SpectrumPowerLaw(eminPL,emaxPL,gammaPL));

    if(centerpointing)
      direction = -1*position;
```

```
    else
      direction = G4ThreeVector(cos(phi)*sin(theta),sin(phi)*sin(theta),-cos(the
ta));
    fParticleGun->SetParticlePosition(position);
    fParticleGun->SetParticleMomentumDirection(direction.unit());
    fParticleGun->GeneratePrimaryVertex(anEvent);
  }else if(dummy){
    fParticleGun->SetParticlePosition(position);
    fParticleGun->SetParticleMomentumDirection(direction.unit());
    fParticleGun->GeneratePrimaryVertex(anEvent);
  }else{
    fParticleGun->GeneratePrimaryVertex(anEvent);
  }
}

G4double HEPDSWPrimaryGeneratorAction::SpectrumPowerLaw(G4double Emin,G4double E
max, G4double gamma){
  G4double energy;
  if(gamma == 0.){
    energy = CLHEP::RandFlat::shoot(Emin, Emax);
    return energy;
  }
  G4double alpha = 1. + gamma; //integral spectral index
  if (alpha == 0.) {
    energy = exp(log(Emin) + CLHEP::RandFlat::shoot(0., 1.) * (log(Emax) - log(E
min)));
  }
  else {
    if (Emin == 0.)
      Emin = 1.E-10;
    energy = pow((CLHEP::RandFlat::shoot(0., 1.) * (pow(Emax, alpha) - pow(Emin,
 alpha)) + pow(Emin, alpha)),1./alpha);
  }
  return energy;

}
//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo......
```