Correction du Tutoriel

1 Modification du Player

1.1 Constructeur

Todo 1 (Ajout d'une animation)

```
// Animation "Falling"
part = atlas->GetPart("Falling");
AssertNew(part);
RE_TexAnim *fallingAnim = new RE_TexAnim(
  m_animator, "Falling", part
);
fallingAnim->SetCycleCount(-1);
fallingAnim->SetCycleTime(0.2f);
```

1.2 Start()

Todo 2 (Taille de collider cohérente)

```
// Création du collider
PE_ColliderDef colliderDef;
PE_CapsuleShape capsule(
  PE_Vec2(0.0f, 0.35f), PE_Vec2(0.0f, 0.85f), 0.35f
);
colliderDef.friction = 1.0f;
colliderDef.filter.categoryBits = CATEGORY_PLAYER;
colliderDef.shape = &capsule;
PE_Collider *collider = body->CreateCollider(colliderDef);
```

1.3 Update()

Todo 3 (Récupération de l'information de saut)

```
if (controls.jumpPressed)
{
  m_jump = true;
}
```

1.4 FixedUpdate()

Todo 4 (Récupération de la vitesse du joueur)

```
PE_Body *body = GetBody();
PE_Vec2 velocity = body->GetLocalVelocity();
PE_Vec2 position = body->GetPosition();
```

Todo 5 (Application du saut)

```
// Saut
if (m_jump)
{
  m_jump = false;
  velocity.y = 15.0f;
}
// Rebond sur les ennemis
if (m_bounce)
{
    m_bounce = false;
    velocity.y = 15.0f;
}
```

Todo 6 (Mise à jour de la vitesse)

```
// Définit la nouvelle vitesse du corps
body->SetVelocity(velocity);
```

Todo 7 (Gestion de la vitesse)

```
// Application des forces
// Définit la force d'accélération horizontale du joueur
PE_Vec2 direction = PE_Vec2::right;
PE_Vec2 force = (15.0f * m_hDirection) * direction;
body->ApplyForce(force);

// Limite la vitesse horizontale
float maxHSpeed = 9.0f;
velocity.x = PE_Clamp(velocity.x, -maxHSpeed, maxHSpeed);
```

1.5 Render()

Todo 8 (Taille correcte du player)

```
float scale = camera->GetWorldToViewScale();
SDL_RendererFlip flip = SDL_FLIP_NONE;
SDL_FRect rect = { 0 };
rect.h = 1.375f * scale; // Le sprite fait 1.375 tuile de haut
rect.w = 1.000f * scale; // Le sprite fait 1 tuile de large
camera->WorldToView(GetPosition(), rect.x, rect.y);

// Dessine l'animateur du joueur
m_animator.RenderCopyExF(
  &rect, RE_Anchor::SOUTH, 0.0f, Vec2(0.5f, 0.5f), flip
);
```

1.6 FixedUpdate(), partie 2

Todo 9 (Animation du joueur)

Cf ci-dessous.

Todo 10 (Animation correcte du joueur)

```
// Détermine l'état du joueur et change l'animation si nécessaire
if (m_onGround)
{
  if (m_state != State::IDLE)
  {
    m_state = State::IDLE;
    m_animator.PlayAnimation("Idle");
  }
}
else
{
  if (m_state != State::FALLING)
  {
    m_state = State::FALLING;
    m_animator.PlayAnimation("Falling");
  }
}
```

2 Modification de la noisette

2.1 Start()

Todo 11 (Filtres de collision)

```
// Crée le collider
PE_CircleShape circle(PE_Vec2(0.0f, 0.45f), 0.45f);
PE_ColliderDef colliderDef;
colliderDef.friction = 0.005f;
colliderDef.filter.categoryBits = CATEGORY_ENEMY;
colliderDef.filter.maskBits = CATEGORY_TERRAIN | CATEGORY_PLAYER
  | CATEGORY_ENEMY;
colliderDef.shape = &circle;
PE_Collider *collider = body->CreateCollider(colliderDef);
```

2.2 OnCollisionStay()

Todo 12 (Collision Player - Nut)

CF ci-dessous.

Todo 13 (Éviter trop de collisions)

```
PE_Manifold &manifold = collision.manifold;
PE_Collider *otherCollider = collision.otherCollider;

if (m_state == State::DYING)
{
  collision.SetEnabled(false);
  return;
}

// Collision avec le joueur
if (otherCollider->CheckCategory(CATEGORY_PLAYER))
{
  Player *player = dynamic_cast<Player *>(collision.gameBody);
  if (player == nullptr)
  {
    assert(false);
    return;
  }
  float angle = PE_SignedAngleDeg(manifold.normal, PE_Vec2::down);
  if (fabsf(angle) > PLAYER_DAMAGE_ANGLE)
  {
    player->Damage();
  }
  return;
}
```

2.3 Damage()

Todo 14 (Rebond du joueur)

```
Player *player = dynamic_cast<Player *>(damager);
if (player)
{
  player->Bounce();
}
SetEnabled(false);
```

2.4 FixedUpdate(), fin de méthode

Todo 15 (Attaque de la noisette)

```
if (dist > 24.f)
{
    body->SetAwake(false);
    return;
}
else if (dist <= 5.0f && m_state == State::IDLE)
{
  // Le joueur est à moins de 5 tuiles de la noisette
  m_state = State::SPINNING;
  body->SetVelocity(PE_Vec2(-3.0f, 10.0f));
}
```

3 Création de la firefly

3.1 Constructeur

Todo (16 à 18)

```
m_name = "Firefly";

RE_Atlas *atlas =
 ↪  scene.GetAssetManager().GetAtlas(AtlasID::COLLECTABLE);
AssertNew(atlas);

// Animation "Idle"
RE_AtlasPart *part = atlas->GetPart("Firefly");
AssertNew(part);
RE_TexAnim *flyingAnim = new RE_TexAnim(m_animator, "Idle", part);
flyingAnim->SetCycleCount(-1);
flyingAnim->SetCycleTime(0.3f);

// Couleur des colliders en debug
m_debugColor.r = 255;
m_debugColor.g = 0;
m_debugColor.b = 255;
```

3.2 Start()

Todo (16 à 18)

```
// Joue l'animation par défaut
m_animator.PlayAnimation("Idle");

// Crée le corps
PE_World &world = m_scene.GetWorld();
PE_BodyDef bodyDef;
bodyDef.type = PE_BodyType::STATIC;
bodyDef.position = GetStartPosition() + PE_Vec2(0.5f, 0.5f);
bodyDef.name = "Firefly";
PE_Body *body = world.CreateBody(bodyDef);
SetBody(body);

// Crée le collider
PE_ColliderDef colliderDef;
PE_CircleShape circle(PE_Vec2::zero, 0.25f);
colliderDef.isTrigger = true;
colliderDef.filter.categoryBits = CATEGORY_COLLECTABLE;
colliderDef.shape = &circle;
PE_Collider *collider = body->CreateCollider(colliderDef);
```

3.3 Render()

Todo (16 et 17)

```
SDL_Renderer *renderer = m_scene.GetRenderer();
Camera *camera = m_scene.GetActiveCamera();

m_animator.Update(m_scene.GetTime());

float scale = camera->GetWorldToViewScale();
SDL_FRect rect = { 0 };
rect.h = 1.0f * scale;
rect.w = 1.0f * scale;
camera->WorldToView(GetPosition(), rect.x, rect.y);
m_animator.RenderCopyF(&rect, RE_Anchor::CENTER);
```

3.4 OnRespawn()

Todo (16 à 17)

```
SetBodyEnabled(true);
SetEnabled(true);
m_animator.StopAnimations();
m_animator.PlayAnimation("Idle");
```

3.5 Collect()

Todo (16 à 17)

```
Player *player = dynamic_cast<Player *>(collector);
if (player == nullptr)
  return;

SetToRespawn(true);
SetEnabled(false);
player->AddFirefly(1);
```

3.6 Assets/LevelParser.cpp

Todo (18)

```
// Pensez au #include "Firefly.h"
case 'o' :
{
    Firefly *firefly = new Firefly(scene);
    firefly->SetStartPosition(position);
    break;
}
```