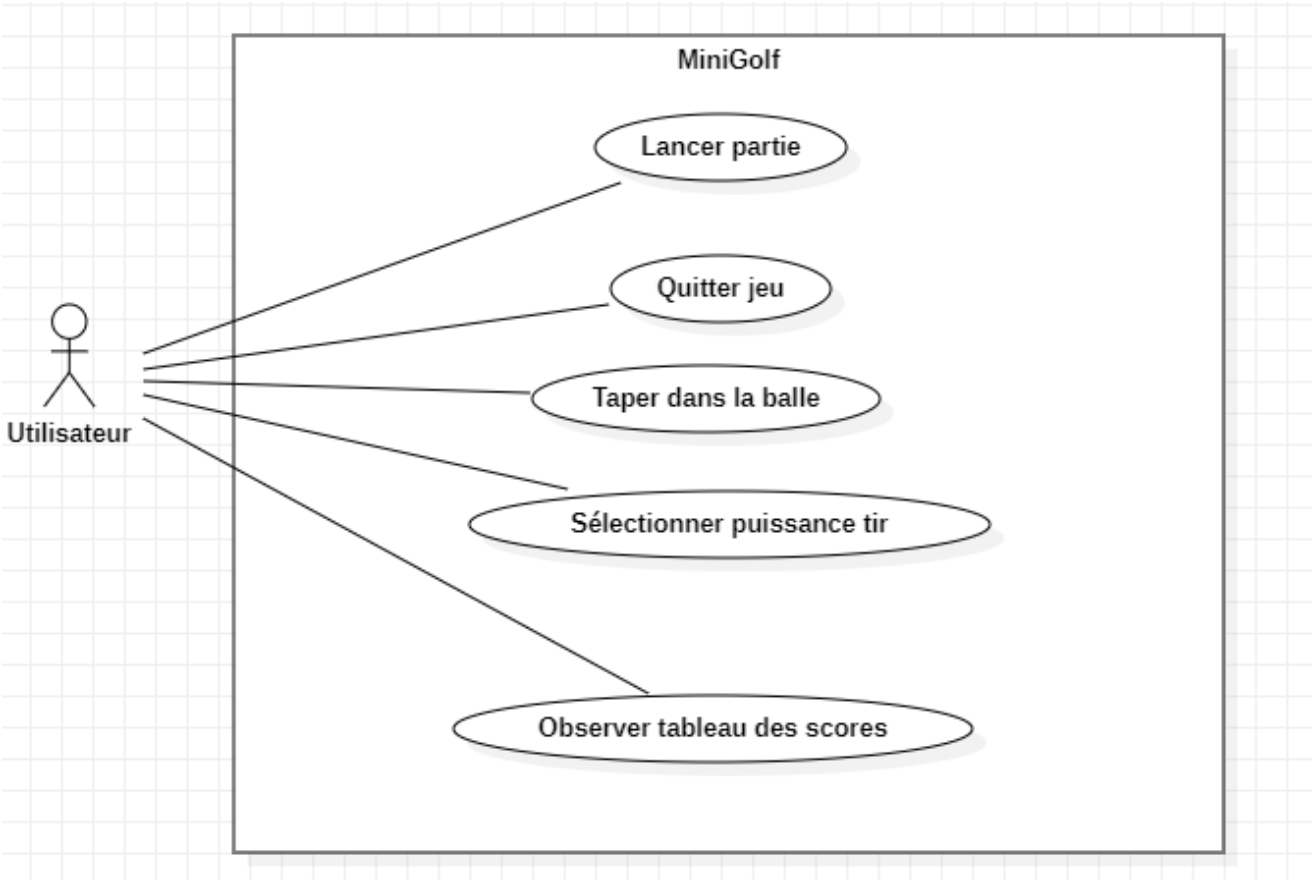
Documentation du projet MiniGolf

Diagramme de cas d'utilisation



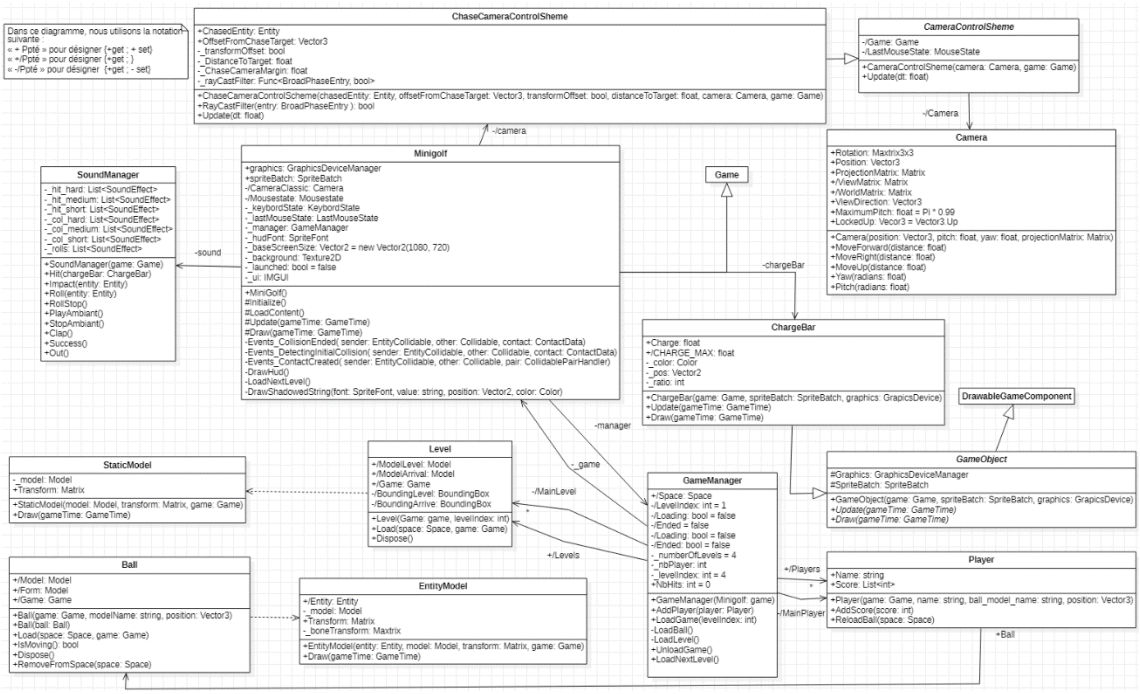


Diagramme de classe

L'objectif est de pourvoir simuler une partie de mini-golf en 3D. Le joueur doit envoyer la balle dans le bogey (le trou). Il peut utiliser autant de coup qu'il veut mais l'objectif est d'en faire le moins possible. Dès que la balle arrive dans le bogey le joueur change de niveau. Le jeu se termine lorsque tous les niveaux sont accomplis.

Nous avons utilisé le Framework Monogame mais nous avons aussi utilisé Bepuphysics afin de gérer la physique du jeu plus simplement. Les collisions et la gravité sont gérer par celui-ci.

La classe ***Minigolf*** est la classe principale du projet. On y retrouve la méthode « Update » qui sert de boucle du jeu. Et de la méthode « Draw » qui permet d'afficher les différentes interfaces utilisateurs.

La classe ***Camera**** permet de s'orienter dans l'espace 3D qui va être créé. Cette classe est utilisé par la classe abstraite CameraControlSheme qui déclare une méthode « Update » permettant de changer l'orientation de la caméra en fonction du déplacement de la souris.

La classe ***ChaseCameraControlSheme**** hérite de *CameraControlSheme* et redéfinis la méthode « Update » afin d'orienter la cible de la camera sur une entité donnée en paramètre de constructeur. Cette classe est utilisé par la classe principale Minigolf et la camera est mise à jour dans la méthode « Update ».

La classe ***StaticModel***** est une classe fournie par BepuPhysics et permet de fournir le modèle d'un objet qui sera immobile dans l'environnement du jeu. Il n'aura donc pas de masse et ne sera pas soumis à la gravité. Cette classe sera donc appelé pour générer le modèle d'un terrain dans la classe Level.

Contrairement à la classe ***EntityModel***** (aussi fournis par BepuPhysics) qui sera utilisé par un objet qui à vocation à évoluer au file du jeu, bouger, être supprimé, créé… C'est pour cela qu'il est quant à lui soumis aux lois de la gravité et qu'on lui fournit une masse. Cette classe sera donc appelé pour générer le modèle d'une balle dans la classe Ball.

La classe ***Player*** représente un joueur dans la partie. Il possède un nom, une balle et un score. Il peut changer son score et changer de balle.

La classe ***Ball*** représente une balle dans la partie. Elle possède un modèle qui sera affiché à l'écran, d'une entité sphérique qui sera sa forme dans l'espace dans lequel elle sera ajouté. Une balle peut être ajouté ou retirer d'un espace. La méthode « IsMoving » envoie un booléen pour savoir si la balle roule.

La classe ***Level*** représente un niveau dans une partie. Elle possède deux modèles :

-« ModelLevel » représente le modèle entier du niveau

-« ModelArrivale » représente seulement le modèle du bogey (l'endroit où la balle doit atterrir afin de compléter le niveau).

```
public void Load(Space space, Game game)
{
    //Using the ModelDataExtractor in order to extract the bounding of the model and activate the collisions
    ModelDataExtractor.GetVerticesAndIndicesFromModel(ModelLevel, out Vector3[] vertices, out int[] indices);
    var meshLevel = new StaticMesh(vertices, indices, new AffineTransform(new Vector3(0, -40, 0)));
    BoundingLevel = meshLevel.BoundingBox;
    space.Add(meshLevel);
    game.Components.Add(new StaticModel(ModelLevel, meshLevel.WorldTransform.Matrix, game));

    ModelDataExtractor.GetVerticesAndIndicesFromModel(ModelArrival, out vertices, out indices);
    var meshArrival = new StaticMesh(vertices, indices, new AffineTransform(new Vector3(0, -40, 0)));
    BoundingArrive = meshArrival.BoundingBox;
}
```

Ces deux modèles vont permettre d'extraire leurs boites de collisions (« BoundingBox ») grâce à la méthode « Load ». Ceci sera utile pour tester si la balle touche l'arrivée.

```
//Managing the hole and the finish of the course
if (mainEntity.CollisionInformation.BoundingBox.Intersects(_manager.MainLevel.BoundingArrive) && !_manager.Loading)
{
    _sound.Success();
    BoundingBox box = new BoundingBox(new Vector3(-1, -21, -1), new Vector3(1, -19, 1));
    mainEntity.CollisionInformation.BoundingBox = box;
    mainEntity.Position = Vector3.Zero;
    mainEntity.LinearVelocity = Vector3.Zero;
    _manager.LoadNextLevel();
}
```

Ce test est effectué dans la méthode « Update » de la classe Minigolf.

La classe ***GameManager*** correspond à la façade du jeu car elle regroupe plusieurs classes et permet de les utiliser plus facilement entre elle. Cette classe permet d'ajouter des joueurs, de charger le terrain et les balles dans l'espace ainsi que de changer de niveau. Elle possède une instance de type « Space » dans laquelle on va rajouter les entités voulus (terrain, balle)

dès que l'on va charger le jeu. De plus on y retrouve une liste chainée de « Player » car à la base on comptait faire un jeu multijoueur.

```
0 références
protected override void LoadContent()
{
    SpriteBatch = new SpriteBatch(GraphicsDevice);

    //Create the different managers
    _chargeBar = new ChargeBar(this, SpriteBatch, Graphics, new Vector2((1900 * Window.ClientBounds.Width) / 1980, (1100 * Window.ClientBounds.Height) / 1080));

    _sound = new SoundManager(this);

    _manager = new GameManager(this);
    _manager.AddPlayer(new Player(this, "jojo", "ball_red", new Vector3(0, -20, 0)));
    _manager.LoadGame(1);
```

D'ailleurs Il suffit juste d'appeler la méthode « AddPlayer » pour rajouter un nouveau joueur avec sa balle dans l'espace souhaité.

La classe ***SoundManager*** charge tous les bruitages du jeu et possède plusieurs méthodes afin d'émettre un son dans le jeu.

```
private void Events_DetectingInitialCollision(EntityCollidable sender, Collidable other, CollidablePairHandler pair)
{
    _sound.Impact(sender.Entity);
}
```

Ses méthodes sont utilisées par la classe principale Minigolf dans la méthode « Update » ou lorsqu'un évènement est déclencher (ex : lorsque la balle initie une nouvelle collision il y a un bruit d'impact).

La classe abstraite ***GameObject*** hérite de la classe *DrawableGameComponent*. Elle représente un objet 2D qui peut être dessiner à l'écran (ex : ChargeBar). Elle requiert que ses classes filles redéfinissent la méthode « Update » et « Draw ».

La classe ***ChargeBar*** hérite de GameObject et affiche une barre de chargement qui augmente en fonction de la puissance de lancé de la balle. Elle possède un attribut « charge » représentant la charge à appliquer.

```
//Managing the loading of the shot
if (!_manager.MainPlayer.Ball.IsMoving())
{
    if (MouseState.LeftButton == ButtonState.Pressed)
    {
        if (_chargeBar.Charge <= _chargeBar.ChargeMax)
        {
            _chargeBar.Charge += 0.1f * (float)gameTime.ElapsedGameTime.TotalMilliseconds;
        }
        if (_chargeBar.Charge >= _chargeBar.ChargeMax)
        {
            _chargeBar.Charge = _chargeBar.ChargeMax;
        }
    }
    if(_lastMouseState.LeftButton == ButtonState.Pressed)
    {
        //Hitting the ball
        if (MouseState.LeftButton == ButtonState.Released)
        {
            mainEntity.LinearVelocity += Camera.Camera.ViewDirection * _chargeBar.Charge;
            _sound.Hit(_chargeBar);
            _chargeBar.Charge = 0;
            _manager.NbHits++;
        }
    }
}
else
{
    _chargeBar.Charge = 0;
}
_lastMouseState = MouseState;
```

La charge est modifiée dans la méthode « Update » de la classe Minigolf lorsque l'utilisateur fais un clic gauche. La barre est dessinée à l'écran à l'aide de la méthode « Draw » et sa méthode « Update » met à jour la couleur de la barre.

*: Classe fournis par la démo de Bepuphysics mais nous avons adapter la méthode «Update».

**: Classe fournis par la démo de Bepuphysics.