Document technique pour :

Distorsion

Un jeu d'aventure révolutionnaire

Écrit par Francis Desbiens, Loïc Durand, Patrice Pedneault, Jérôme Scipion

Version 1.0

avril 2014

Plateforme

Pour la conception du jeu Distorsion la plate-forme de création utilisée serait Unity. Notre choix c'est arrêté sur cette "Engine" du à sa simplicité de création et d' implémentation de mécanique de jeu. Contrairement à Unreal Engine qui demande beaucoup plus d'effort pour la production d'un jeu autre qu'un "First Person Shooter".

Langage

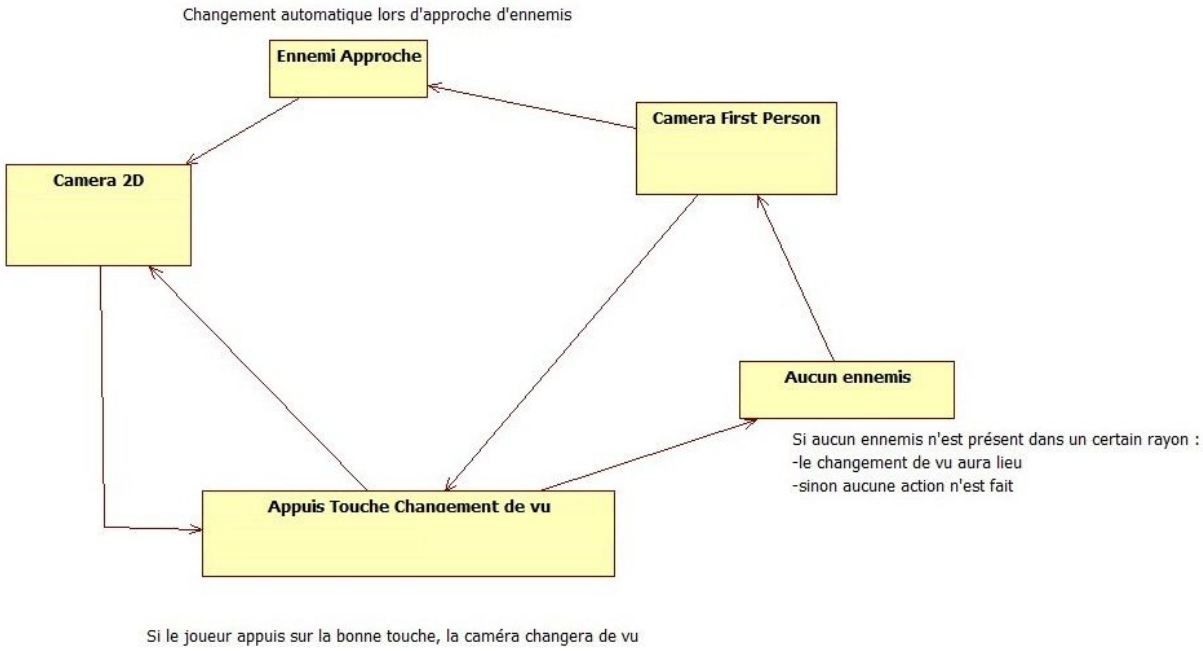
L'utilisation de "l'engine" Unity nous pousse à choisir entre trois langages de script supportés, Boo, C# et Javascript. Du à nos compétences nous avons choisi l'utilisation de C# comme langage de programmation.

Défi technique

On peut ressortir 3 défis techniques de programmation qui devrait demander plus de temps.

Caméra:

Le premier des trois sera l'implémentation de la caméra. Dû aux conditions que celle-ci demandera pour la création et de son importance dans le gameplay, la caméra subira une attention particulière au courant du développement. L'image qui suis montre un exemple de diagramme qui montre la gestion de la caméra dans le jeu.



Exemple de code pour la caméra:

```
using UnityEngine;
using System.Collections;

[AddComponentMenu("Camera-Control/Mouse")]
public class camera_Follow : MonoBehaviour {

    // Mouse buttons in the same order as Unity
    public enum MouseButton { Left = 0, Right = 1, Middle = 2, None = 3 }

    public GameObject character;
    public bool isIn1stPerson = false;

    [System.Serializable]
    // Handles left modifiers keys (Alt, Ctrl, Shift)
    public class Modifiers
    {
        public bool leftAlt;
        public bool leftControl;
        public bool leftShift;

        public bool checkModifiers()
        {
            return (!leftAlt ^ Input.GetKey(KeyCode.LeftAlt)) &&
                (!leftControl ^ Input.GetKey(KeyCode.LeftControl)) &&
                    (!leftShift ^ Input.GetKey(KeyCode.LeftShift));
        }
    }

    [System.Serializable]
    // Handles common parameters for translations and rotations
    public class MouseControlConfiguration
    {

        public bool activate;
        public MouseButton mouseButton;
        public Modifiers modifiers;
        public float sensitivity;

        public bool isActivated()
        {
            return activate && Input.GetMouseButton((int)mouseButton) && modifiers.checkModifiers();
        }
    }

    [System.Serializable]
    // Handles scroll parameters
    public class MouseScrollConfiguration
    {

        public bool activate;
        public Modifiers modifiers;
```

```
        public float sensitivity;

        public bool isActivated()
        {
            return activate && modifiers.checkModifiers();
        }
    }

    // Yaw default configuration
    public MouseControlConfiguration yaw = new MouseControlConfiguration { mouseButton = MouseButton.Right,
    sensitivity = 10F };

    // Xaw default configuration
    public MouseControlConfiguration xaw = new MouseControlConfiguration { mouseButton = MouseButton.Right,
    sensitivity = 10F };

    // Pitch default configuration
    public MouseControlConfiguration pitch = new MouseControlConfiguration { mouseButton = MouseButton.Right,
    modifiers = new Modifiers{ leftControl = true }, sensitivity = 10F };

    // Roll default configuration
    public MouseControlConfiguration roll = new MouseControlConfiguration();

    // Vertical translation default configuration
    public MouseControlConfiguration verticalTranslation = new MouseControlConfiguration { mouseButton =
    MouseButton.Middle, sensitivity = 2F };

    // Horizontal translation default configuration
    public MouseControlConfiguration horizontalTranslation = new MouseControlConfiguration { mouseButton =
    MouseButton.Middle, sensitivity = 2F };

    // Depth (forward/backward) translation default configuration
    public MouseControlConfiguration depthTranslation = new MouseControlConfiguration { mouseButton =
    MouseButton.Left, sensitivity = 2F };

    // Scroll default configuration
    public MouseScrollConfiguration scroll = new MouseScrollConfiguration { sensitivity = 2F };

    // Default unity names for mouse axes
    public string mouseHorizontalAxisName = "Mouse X";
    public string mouseVerticalAxisName = "Mouse Y";
    public string scrollAxisName = "Mouse ScrollWheel";

    void LateUpdate ()
    {

        if (Input.GetKey (KeyCode.Space))
        {
            if(!isIn1stPerson)
            {
```

```
            transform.position = new Vector3 (character.transform.position.x,
            character.transform.position.y+1, character.transform.position.z);
            transform.localRotation = new Quaternion(0, 90, 0, 90);
            isIn1stPerson = true;
        }
        if (yaw.isActivated ()) {
                float rotationX = Input.GetAxis (mouseHorizontalAxisName) * yaw.sensitivity;
                transform.Rotate (0, rotationX, 0);
        }
        if (xaw.isActivated ()) {
                float rotationY = Input.GetAxis (mouseVerticalAxisName) * xaw.sensitivity;
                transform.Rotate (-rotationY, 0, 0);
        }
        if (pitch.isActivated ()) {
                float rotationY = Input.GetAxis (mouseVerticalAxisName) * pitch.sensitivity;
                transform.Rotate (-rotationY, 0, 0);
        }
        if (roll.isActivated ()) {
                float rotationZ = Input.GetAxis (mouseHorizontalAxisName) * roll.sensitivity;
                transform.Rotate (0, 0, rotationZ);
        }

        if (verticalTranslation.isActivated ()) {
                float translateY = Input.GetAxis (mouseVerticalAxisName) * verticalTranslation.sensitivity;
                transform.Translate (0, translateY, 0);
        }

        if (horizontalTranslation.isActivated ()) {
                float translateX = Input.GetAxis (mouseHorizontalAxisName) * horizontalTranslation.sensitivity;
                transform.Translate (translateX, 0, 0);
        }

        if (depthTranslation.isActivated ()) {
                float translateZ = Input.GetAxis (mouseVerticalAxisName) * depthTranslation.sensitivity;
                transform.Translate (0, 0, translateZ);
        }

        if (scroll.isActivated ()) {
                float translateZ = Input.GetAxis (scrollAxisName) * scroll.sensitivity;

                transform.Translate (0, 0, translateZ);
        }

    }
    else
    {
        transform.position = new Vector3 (character.transform.position.x, character.transform.position.y+2,
        character.transform.position.z-10);
        transform.localRotation = new Quaternion(0, 0, 0, 0);
        isIn1stPerson = false;
    }
```

Déplacement d'objet :

Le deuxième des trois défis sera l'implantation du système de déplacement des objets qui est possible à l'aide du bras distortionel du héros. L'utilisation d'une tellement mécanique de jeu demandera beaucoup d'effort. La grosse partie sera pour que tout semble beau et logique surtout au niveau de la physique qui parfois peut être un fardeau.

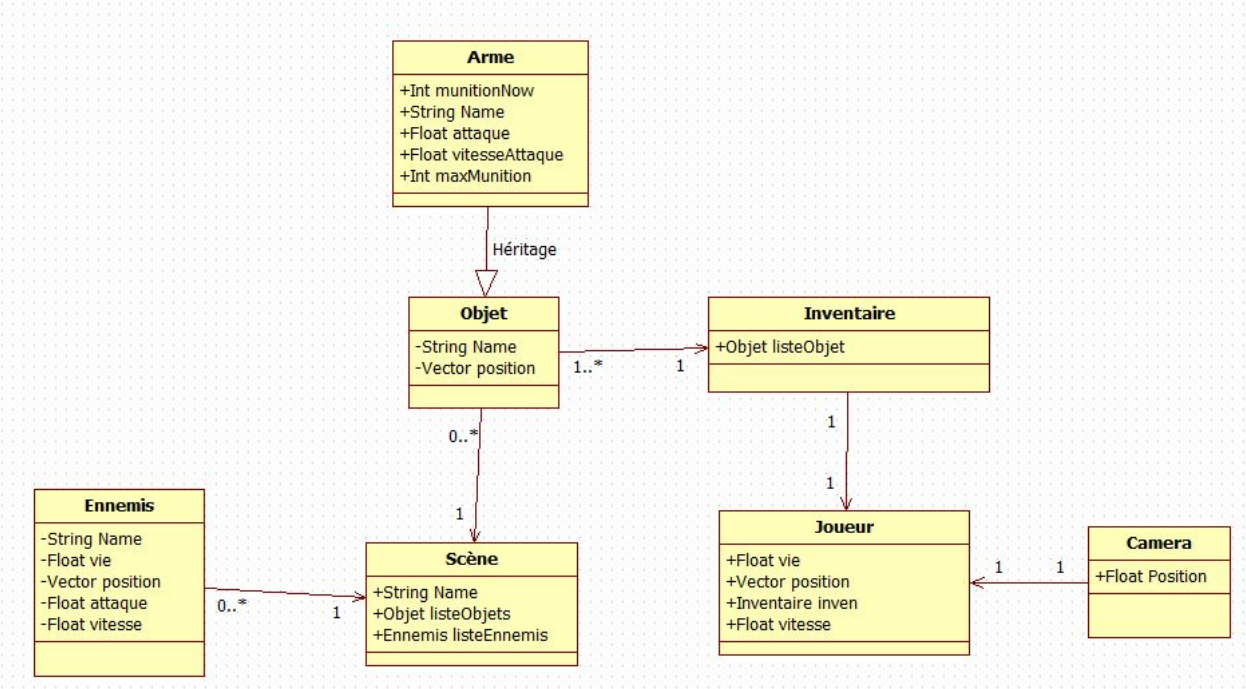
*Si durant le développement le déplacement des objets de façon libre semble trop compliqué, l'utilisation d'animation pourrait être une alternative .

Force “Push”:

La force “Push” qui permettra de propulser des objets/ennemis, sera tout comme le déplacement d’objet une mécanique qui devra être pris au sérieux dès le départ du développement, on peut déjà prévoir que l’implémentation d’une physique correct et le peaufinage de la “feature” demandera beaucoup de temps.

Diagramme de classe

Le diagramme qui suis, montre notre idée générale de ce que pourrait contenir en terme d’objet le jeu, plusieurs autres feront leurs apparitions durant l’implémentation.



Conclusion

Après avoir concrétisé nos idées et conçu le jeu, on peut dire qu’avec nos compétences actuels en matière de programmation tout ce qui a été décrit nous semble réalisable et logique. Par contre, il faudra faire attention pour mettre les ressources appropriées au tout début du projet dans les mécanismes principaux et nécessaire pour le “gameplay” du jeu.