Flappy Bird Open GL / C++

Ömer Ertekin

Muhammed Erdoğan

Nous avons décidé de réaliser le jeu Flappy Bird, qui était autrefois très populaire, en raison de notre intérêt pour le développement de jeux. Voici les étapes que nous devons suivre pour ce projet:

1- Dessiner et faire bouger le joueur

2 -Dessiner les tuyaux et augmenter le score à chaque fois que le joueur passe à travers

3- Détecter les collisions avec les tuyaux

4- Créer une boucle de niveau infinie en ajoutant de nouveaux tuyaux et en accélérant le jeu au fil du temps

5- Ajouter des éléments d'interface utilisateur simples pour afficher le score de début à fin.

1- Nous voulions ajouter le joueur en tant que modèle, mais nous avons rencontré des problèmes pour ajouter une bibliothèque et d'autres scripts, ce qui a causé le manque de temps. C'est pourquoi nous avons affiché le joueur avec un modèle de sphère. Pour rendre son mouvement plus réaliste, nous avons augmenté la vitesse de chute du joueur à mesure que la durée de son saut augmentait. Nous avons également ajouté des contrôles pour que le joueur reste à l'intérieur des limites de l'écran

```
void DecreasePlayerYValue(int value)
{
    if(didCollide) return;
    if(isGameStarted)
    {
        timeSinceJump += gravityWithTime * playerSize;
        calculatedY = playerY - (timeSinceJump + gravity) * playerSize;
        playerY = std::max(-4.5,calculatedY);
    }

    glutPostRedisplay();

    didCollide = CollisionControl();

    glutTimerFunc(25, DecreasePlayerYValue, 0);
}
```

2- Nous avons utilisé la fonction glut cylinder pour les tuyaux. En déterminant le point d'espace entre les tuyaux, nous avons dessiné deux tuyaux en haut et en bas pour former un bloc. Lorsque le joueur passe par le milieu de ces cylindres, il est considéré comme ayant gagné des

points.

```
void DrawSinglePipe(double pipeX,double pipeSpaceY)
{
    glColor3f(0,1,0);
    glPushMatrix();
        glTranslated(pipeX,pipeSpaceY + defaultPipeSpacing/2 + defaultPipeHeight,-10);
        glRotated(90,1,0,0);
        glutSolidCylinder(defaultPipeWidth/2,defaultPipeHeight,16,16);
    glPopMatrix();
    glPushMatrix();
        glTranslated(pipeX,pipeSpaceY - defaultPipeSpacing/2 ,-10);
        glRotated(90,1,0,0);
        glutSolidCylinder(defaultPipeWidth/2,defaultPipeHeight,16,16);
    glPopMatrix();
}
```

3- Pour détecter les collisions avec les tuyaux, nous avons récupéré les tuyaux proches de nous que nous pourrions frapper et vérifié si les coordonnées du joueur entraient en collision avec un point à l'intérieur de l'un de ces tuyaux. Comme des contrôles séparés étaient nécessaires pour les tuyaux en haut et en bas, nous avons séparé le jeu de tuyaux proches en haut et en bas et avons contrôlé chacun séparément.

```
bool CollisionControl()
{
    for(int i = 0;i<pipeCount;i++)
    {
        if(fabs(pipeXPositions[i] - playerX) > playerSize) continue;

        topPipeVertices[0] = pipeXPositions[i] - defaultPipeWidth / 2;
        topPipeVertices[1] = pipeYPositions[i] + defaultPipeSpacing / 2;

        topPipeVertices[2] = topPipeVertices[0] + defaultPipeWidth;
        topPipeVertices[3] = topPipeVertices[1];

        topPipeVertices[4] = topPipeVertices[0];
        topPipeVertices[5] = topPipeVertices[1] + defaultPipeHeight;

        topPipeVertices[6] = topPipeVertices[1];
        topPipeVertices[7] = topPipeVertices[5];
        //-------------------------------------------------------------//
        botPipeVertices[0] = pipeXPositions[i] - defaultPipeWidth / 2;
        botPipeVertices[1] = pipeYPositions[i] - defaultPipeSpacing / 2;

        botPipeVertices[2] = botPipeVertices[0] + defaultPipeWidth;
        botPipeVertices[3] = botPipeVertices[1];

        botPipeVertices[4] = botPipeVertices[0];
        botPipeVertices[5] = botPipeVertices[1] - defaultPipeHeight;

        botPipeVertices[6] = botPipeVertices[1];
        botPipeVertices[7] = botPipeVertices[5];


        if(playerY + playerSize >= topPipeVertices[1] && playerY - playerSize <= topPipeVertices[5])
        {
            if(playerX + playerSize > topPipeVertices[0] && playerX - playerSize < topPipeVertices[2])
            {
                return true;
            }
        }

        if(playerY - playerSize <= botPipeVertices[1] && playerY + playerSize >= botPipeVertices[5])
        {
            if(playerX + playerSize > botPipeVertices[0] && playerX - playerSize < botPipeVertices[2])
            {
                return true;
            }
        }
    }
```

4- Pour générer de nouveaux tuyaux au fur et à mesure que nous avançons, nous avons utilisé la méthode "Object Pooling" souvent utilisée en jeu pour l'optimisation. Comme seuls 4 tuyaux peuvent être visibles à l'écran, nous avons attendu que le tuyau le plus à gauche disparaisse de l'écran pour l'ajouter à la droite, ce qui nous a permis de simuler un nombre infini de tuyaux avec seulement 4 tuyaux. Cela a amélioré les performances en évitant de gaspiller de l'espace inutilement dans la mémoire et le rendu pour les tuyaux et leurs valeurs. Nous avons

également déplacé ces 4 tuyaux selon l'axe X vers la gauche à la vitesse du jeu plutôt que de déplacer le joueur et la caméra. En d'autres termes, le joueur n'a bougé que sur l'axe Y, tandis que la caméra est restée fixe sur les axes X et Y et que les tuyaux ont seulement bougé sur l'axe X. Dans l'étape suivante, nous avons évité de rendre le jeu redondant en ajoutant aléatoirement une différence dans la position Y du tuyau précédent à une distance où le joueur peut sauter.

```
void MoveThePipes(int value)
{
    if(!isGameStarted || didCollide) return;
    for(int i = 0;i<pipeCount;i++)
    {
        pipeXPositions[i] -= speedFactor;
        if(pipeXPositions[i] < -(pipeCount / 2) * distanceBetweenPipes)
        {
            pipeXPositions[i] += pipeCount * distanceBetweenPipes;
            previousIndex = (i + pipeCount - 1) % pipeCount;
            double nextY = pipeYPositions[previousIndex];
            srand(GetTickCount());
            double randomDifference = (3.0f * rand()) / RAND_MAX - 1.5;
            calculatedPipeY += randomDifference;
            calculatedPipeY = std::max(-3.0,calculatedPipeY);
            calculatedPipeY = std::min(3.0,calculatedPipeY);
            pipeYPositions[i] = calculatedPipeY;
        }

        if(pipeXPositions[i] - playerX > 0 && pipeXPositions[i] - playerX < distanceBetweenPipes)
        {
            if(lastIndex != nextPipeIndex)
            {
                lastIndex = nextPipeIndex;
                currentScore++;
            }
            nextPipeIndex = i;
        }
    }

    glutTimerFunc(10, MoveThePipes, 0);
}
```

5- Pendant que le joueur joue, nous avons progressivement approché la vitesse maximale du jeu pouvant être joué à des intervalles de temps spécifiques

```
void IncreaseGameSpeed(int value)
{
    if(speedFactor >= maxSpeedFactor || !isGameStarted || didCollide) return;

    speedFactor += 0.001;
    glutTimerFunc(500,IncreaseGameSpeed,0);
}
```

6- Nous avons utilisé glutBitmapCharacter pour imprimer du texte à l'écran. Nous avons affiché sur l'écran ce que l'utilisateur doit faire pour démarrer ou renaître après avoir perdu, et nous avons effectué les actions souhaitées lorsque les inputs souhaités sont reçus.

```
void drawText(char *string)
{
    char *c;
    glRasterPos3f(-2.0,0.0f, -9.0f);
    for (c=string; *c != '\0'; c++)
    {
        glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, *c);
    }
}
```

Bien que nous voulions ajouter du son, des textures d'arrière-plan et des modèles au jeu, nous n'avons pas pu tout ajouter en raison des problèmes rencontrés avec CodeBlocks et de la charge de travail accrue de nos projets récents. Cependant, nous croyons que le jeu fonctionne de manière fluide et performante.