CO1301: Games Concepts
Lab 5 - Game States and Platformers

1. Introduction

1. In this lab you will practice working with states using state variables and enums.
2. You will be creating a simple endless platformer as part of the lab's activites.

2. Removing a Model from the Game Scene

1. Whenever we want to create a model, we use the 'CreateModel' method of an IMesh* variable.

   ```
   IMesh* sphereMesh = myEngine->CreateModel("Sphere.x"); //declare variable and load mesh
   IModel* sphere = sphereMesh->CreateModel( ); //declare variable and create model
   ```

   It is also possible to remove a model using the 'RemoveModel' method of its mesh. For example, the code below will remove the 'sphere' created in the code above:

   ```
   sphereMesh->RemoveModel( sphere ); //remove model
   ```

   Note that the 'RemoveModel' method is passed the IModel* variable of the model to be removed as a paramter.

2. The removal of a model means that it is completely destroyed. It is no longer possible to use the model since it no longer exists and, indeed, if you tried (for example) to move the model after removal you would generate a real time error.
3. Although the model has been destroyed the IModel* variable still exists. This is because the IModel* variable is a particular kind of variable called a pointer, which is declared using an asterisk (*). The pointer still exists, but the removal of the model means it no longer has anything to point at. However, the IModel variable can be re-used to point to a newly created model. For example, the code below serves no real purpose but it does show this process of removing a model and reusing its IModel* variable:

   ```
   IModel* sphere = sphereMesh->CreateModel( ); // declare variable and create model
   sphereMesh->RemoveModel( sphere ); // remove model
   sphere = sphereMesh->CreateModel( ); // create new model
   ```

4. In the next section, we will create a code that removes or creates a sphere when the 'Retun Key' is hit, based on the state of the game.

3. Using Variables to Store Game State

1. Create a new TL-Engine project named Lab5_SphereStates_Project.
2. Use Floor.x to create a floor model at the origin, then create a camera at location (0, 70, -100). You don't need to move the camera at all today so set camera type to 'kManual'.
3. Rotate the camera 20 units along the x axis so that it is facing downwards.
4. Use the sphere.x mesh to create a sphere model at (0, 10, 0) so that it is positioned on the floor.
5. Declare an integer variable called sphereState and set its value to 1. The variable should be declared outside the game loop.
6. The logic of the program we want to code is as follows:

- If the Return key is hit and the sphereState is 1, this means that sphere exists, therefore the sphere model is removed and the sphereState is set to 0.
- If the Return key is hit and the sphereState is 0, this means that sphere does not exist, therefore the sphere model is created and the sphereState is set to 1.

7. The code to implement this looks as follows (it should be added to the game loop):

```
if( myEngine->KeyHit( Key_Return ) )
{
    if( sphereState == 1 )
    {
        sphereMesh->RemoveModel( sphere );
        sphereState = 0;
    }
    else
    {
        sphere = sphereMesh->CreateModel( 0, 10, 0 );
        sphereState = 1;
    }
}
```

In this fashion you can track the state of your program. In this particular example you are keeping track of whether or not the sphere model exists. Furthermore, you are using the state of the sphere to determine what action the program needs to take: whether to create the sphere or to remove it. States are used extensively within game programming and throughout software engineering in general.

4. Setting Up A Platformer

1. In the sections to follow you will use what you have learnt so far to create an endless platformer game about a gravity defying cube.

2. Create a new TL-Engine project named **Lab5_GravityCube_Project**.

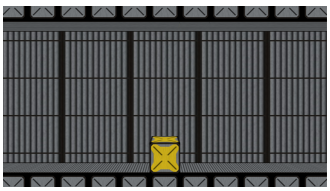
3. Download this project's associated models and textures from Blackboard, save them in a folder inside your project folder and use the AddMediaFolder() method to tell TL-Engine to get resources from the folder. The meshes included and their diemsions are:

    - playerCube.x - 20 x 20 x 20
    - background.x - 260 x 140
    - platform.x - 260 x 20 x 20
    - enemyCube.x - 20 x 20 x 20

4. Create a kManual camera at (0, 0, -130), and use the background.x mesh to create a background model named "background1" at position (0, 0, 10).

5. Next, use the the platform.x mesh to create 2 platform models: platform1 at (0, 60, 0) and platform2 at (0, -60, 0); the platforms should be positioned in front of background1 at the top and bottom respectively.

6. Now that we have a background and platforms to move on, it is time to add our player. Use the playerCube.x mesh to create a model called "playerCube". The X and Z positions of playerCube should be 0, but the Y position playerCube should be such that it appears to sit on platform2, the bottom platform.



The Y value that you use here should be the lowest Y position that playerCube can be positioned at. if you have set your scene correctly, then it should be a negative number

and its positive equivalent should be the highest position that playerCube can be positioned at. Make a note of these two values.

5. Using Enums to Store Game State

1. The game we are creating should be played as follows:
    - When the game starts, playerCube should be positioned so that it appears to sit on the bottom platform.
    - When the Space Key is hit playerCube should move up until it is directly beneath the top platform. It should appear as if playerCube moved up and stopped after hitting the top platform.
    - Pressing the Space Key again should make the cube move down until it is positioned on the bottom platform.
    - Basically, hitting the Space Key should make playerCube move to the opposite platform.
2. To implement this logic, start by creating a Enumeration named EPositionState outside of the game loop for storing the two possible positions of playerCube (Top and Bottom). Then create an enum variable of type EPositionState named playerPosition and initialise it to Bottom as you want playerCube to be positioned on the bottom platform when the game starts.

```
enum EPositionState {Bottom, Top};
EPositionState playerPosition = Bottom;
```

3. Next write code to check if the Space key is hit, if it is then check the playerPosition and change it change it.

```
if (myEngine->KeyHit(Key_Space)) {
    if (playerPosition == Bottom) {
        playerPosition = Top;
    }
    else {
        playerPosition = Bottom;
    }
}
```

4. Next, define a constant float named kGravity and give it a value of 0.5f. Then in the game loop, write code to check the positionState and:
    - If the positionState is Bottom, then playerCube should be moved down using `playerCube->MoveY(-kGravity)` until its Y position is such that it appears to be sitting on the bottom platform (Hint: playerCube's Y position should not be less than the lowest Y value you noted in Task 4.6)
    - If the positionState is Top, then playerCube should be moved up using `playerCube->MoveY(kGravity)` unitl its Y position is such that it is just underneath the top platform (Hint: playerCube's Y position should not be greater than the highest Y value you noted in Task 4.6).

    The code should be structured as follows:

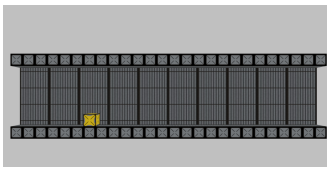
```
if (playerPosition == Bottom) {
    //Add another if statement here
    //if playerCube's Y is greater that the lowest Y position, move cube down some more
}
else if (playerPosition == Top){
    //Add another if statement here
    // if playerCube's Y is less that the highest Y position, move cube up some more
}
```

6. Endless Platfroms

1. playerCube can now move vertically between the top and bottom platforms, we will now add horizontal movement on the platforms.

2. Rather than moving playerCube, it will be more efficient to move the platforms and the background to create the illusion of playerCube's movement. Another advantage of this is that once the platforms and the background exit the camera's view from one side they can be recycled into view from the other side to create an endless platformer. But for this approach to work we need addional models (two platforms and a background).

   Therefore use platform.x to create models for platform3 at (260, 60, 0) and platform4 at (260, -60, 0); then use background.x to create background2 model at (260, 0, 10).



3. To have a better view of how these models and what will be created using them, change your camera type to kFPS so that you can zoom out and move around to have a better look at all the models in the scene.

4. To improve efficieny, we are going to use parenting to create two sets of models each containg two platforms and a background. Therefore use the AttachToParent( ) method to attach platform1 and platform2 to background1; and attach platform3 and platform4 to background2.

5. When you run your program, you should notice that the position of your platforms has changed. This is beacuse when a model is attached as a child to another model, the orign of the child model is no longer the world origin but the origin of the parent model. Therefore you will have to reposition your platforms relative to their parents using SetLocalPosition( ).

```
platform1->SetLocalPosition(0, 60, -10); // at the top and in front of background1
platform2->SetLocalPosition(0, -60, -10); //at the bottom and in front of background1

platform3->SetLocalPosition(0, 60, -10); //at the top and in front of background2
platform4->SetLocalPosition(0, -60, -10); //at the bottom and in front of background2
```

6. Next, define a constant float named kGameSpeed and initialise it to -0.05f. Then in the game loop add code to move background1 and background2 along the x axis using kGameSpeed. When you run your program, each background and its children (platforms) should be moving to the left thus creating the illusion that the cube is moving to the right.

7. Now we want to make sure that once a background and its attached models are out of view from the left, they are recycled so that they appear again from the right. Therefore, add code in the game loop that checks whether a background and its children are out of view, if they are then they should be repositioned to the right of the camera view so that they can move left into view.

```
if (background1->GetX() <= -260) {
    background1->SetPosition(260, 0, 10);
}
else if (background2->GetX() <= -260) {
    background2->SetPosition(260, 0, 10);
}
```

8. Run your program and move your camera back so you can view how the process works. Then update your code to change your camera back to kManual positioned at (0, 0, -130).

9. When you run your program now you should see a cube that can jump from the bottom platform to the top and vice versa, moving on endless platforms.

7.Exercises - Improving the Game

1. By default, playerCube and the platform models all use "yellow.jpg" as their skin, write code that changes the skin of these models when the Return Key is hit.

    - Define an integer variable named gameSkin to keep track of the skin used in the game. gameSkin should be initialised as 0 when the game starts and that shows the skin is "yellow.jpg".
    - If Return Key is hit when gameSkin is equal to 0, then gameSkin should be changed to 1 and the skin of the models should be set to "red.jpg".
    - If the return Key is hit and gameSkin is equal to 1, then gameSKin should be changed to 2 and the skin of the models should be changed ot "green.jpg,
    - If the return key is hit again and gameSkin is equal to 2, then gameSkin should be changed to 0 and the skin of the modesl should be set to "yellow.jpg".

2. At the moment, playerCube can switch between position states at any time meaning you can change cubePlayer's direction of movement while it is mid air, lets change that.

    - Add two states to your program, MovingUp and MovingDown.
    - If playerCube is in the Bottom state an the Space key is hit, playerCube should transition to the MovingUp state and then move up until it reaches the highest Y position then transition to the Top state
    - If playerCube is in the Top state an the Space key is hit, playerCube should transition to the MovingDown state and then move down until it reaches the lowest Y position then transition to the Bottom state
    - playerCube should not respond to Space key hits while in the MovingUp or MovingDown states.

3. Make playerCube complete a 90 units rotation along its local z axis while in the MovingUp or MovingDown states, so that playerCube appears to do a "half-flip" everytime it moves to the top or the bottom.

8.Exercises - Advanced

1. Use the enemyCube.x model to create two models named enemy1 and enemy2.

    Attach enemy1 to background1 and enemy2 to background2. Then use SetLocalPosition( ) to position enemy1 in front of background1 and just beneath platform1. Position enemy2 in front of background2 so that it appears to sit on platform4. Use 0 as the X position when positioning both enemies.

    When you run your program you should have top and bottom enemies that move with the backgrounds and platforms.

2. To make things more exciting copy the code below and add it to the game loop, it checks for collisions between the player and the enemies and stops the game everytime the player hits an enemy.

```
//the code below implements sphere to sphere collision detection
//it uses spheres of radius 20 as bounding volumes for playerCube and the enemies
//it calculates the distance between each enemy's and playerCube's bounding spheres
//if  any of the distances is less than 20 then there is a collision
//20 is the sum of the raidus of the enemy's and playerCUbe's bounding spheres

float x1, y1, z1, x2, y2, z2;

x1 = playerCube->GetX() - enemy1->GetX();
y1 = playerCube->GetY() - enemy1->GetY();
z1 = playerCube->GetZ() - enemy1->GetZ();

x2 = playerCube->GetX() - enemy2->GetX();
y2 = playerCube->GetY() - enemy2->GetY();
z2 = playerCube->GetZ() - enemy2->GetZ();

float collisionDist1 = sqrt(x1 * x1 + y1 * y1 + z1 * z1);
```

```
float collisionDist2 = sqrt(x2 * x2 + y2 * y2 + z2 * z2);

if (collisionDist2 < 20 || collisionDist1 < 20)
{
    myEngine->Stop();
}
```

3. Everytime a background and its attached enemy are out of view, reposition the enemy to a random position either beneath the background's attached top platform or on the background's attached bottom platform. (Hint: you may want to use rand(), the C++ function for generating random numbers)

4. You can go further with this if you are having fun. You can add functionalities such as:

    - Restarting the game when playerCube collides with an enemy.
    - Adding more enemies.
    - Adding enemies with different behaviours.
    - Keeping score