IN628 Programming 4
Semester 1, 2020

IN628 2020 13 – Terrain Collision

Introduction

In this practical you will extend your classes to control where the **Sprites** may walk on the tile map. Initially, simply identify each **Tile** as walkable or not walkable by adding a Boolean property to the **Tile** class. Modify the game engine so that **Sprites** will only move onto **Tiles** whose walkable property is true (algorithm details below). To implement more complex terrain collision behavior's, you can later introduce additional properties or states to your **Tile** class and modify the **Sprite** code as required.

Implementation

The basic algorithm is as follows:

When the sprite moves:

1. Determine what tile the sprite will move onto
2. Determine if that tile is walkable
3. If so, move the sprite. If not, don't move the sprite.

Determining what tile the sprite will move onto

We store the world location of our **Sprite** in its xPos and yPos properties. These are pixel locations, and we will need to convert them to **TileMap** column and row values to find out what tile the **Sprite** is about to step onto. We could perform this computation in the **Sprite's** code as:

```
xTile = xPos / TILE_SIDE
yTile = yPos / TILE_SIDE
```

We could then query the **TileMap** to find out if the tile at column and row (xTile, yTile) is walkable.

Unfortunately, this will not give us very accurate collision detection. Recall that (xPos, yPos) are the pixel coordinates of the upper-left corner of the sprite, and this may not be the position you want to look at to determine if the sprite can move. For example, if your sprite is moving **SOUTH**, you would rather check the lower left corner; if your sprite is moving **EAST**, you would rather check the lower right corner (assuming that your sprite has feet of some kind). Thus, to get accurate collision detection we need to implement this logic:

```
switch (SpriteDirection)
{
        case (NORTH):
        Determine what tile the upper left corner will be in if we move
        case(EAST):
        Determine what tile the lower right corner will be in if we move
        case(SOUTH):
        case(WEST):
        Determine what tile the lower left corner will be in if we move
}
```

Start by determining the projected location of the upper-left corner as usual:

```
int newX = xPos + (xVel * velocityDirections[spriteDirection].X);
int newY = yPos + (yVel * velocityDirections[spriteDirection].Y);
```

Then compute the location of the corner of interest. For example, if spriteDirection is **EAST**, you want the lower right corner, so you might say:

```
int directionCornerX = newX + frameWidth;         // directionCornerX is the right-hand edge
int directionCornerY = newY + frameHeight;        // directionCornerY is the bottom edge
```

You can then divide directionCornerX and directionCornerY by TILE_SIDE to find out which tile in the **TileMap** your **sprite's** lower right corner is about to move onto.

Determining if the new tile is walkable

How can the **Sprite** class find out the value of a particular **Tile's** isWalkable property, given that it knows the Tile's column and row? The **TileMap** class knows which tile index belongs at a given column and row, but it does not have direct access to the associated Tile object – **TileList** has that. The **TileList** knows which Tile is associated with each tile index, but it does not have direct access to the isWalkable property of that **Tile**, because it is private data belonging to the **Tile** class. To get the value of isWalkable (which is what the Sprite really needs to know), the **TileList** must call the **Tile's** get method (or access its IsWalkable property). The **TileList** can then expose a method to pass that value to the TileMap. The **TileMap** can expose a method to pass this value to the **Sprite**. This is the same pattern we follow to give the **TileMap** access to the Bitmap of individual tiles. This time, the chain extends from the **Tile** to the **TileList** to the **TileMap** and finally to the **Sprite**. This "passing information up the chain" technique which is very common in OO programming.
You will need to add new methods to the involved classes. All of the methods are very simple, and just work to pass the isWalkable value up the chain from the Tile to the Sprite. The required methods and message passing structure are summarised in the following table:

Object	Job	Possible Signature	Who Will Call It?
Tile	Must provide a method that exposes its isWalkable value, or make IsWalkable a class property	bool getIsWalkable() or declare IsWalkable as a class property	TileList
TileList	Must provide a method that queries the isWalkable of a specific Tile from its array of Tiles and returns it.	bool isTileWalkable(int tileIndex)	TileMap
TileMap	Must provide a method that says whether the tile at a particular column and row of the tilemap is walkable.	bool isTileWalkable(int col, int row)	Sprite, in its move method.

If the tile the **Sprite** will move on to is found to be walkable, copy your temporary X and Y coordinates (newX and newY in the pseudocode above) into the **Sprite's** xPos and yPos. If not, do nothing, because the **Sprite** should not change location.

Deliverable

Modify your **Tile**, **TileList** and **TileMap** classes as described above, to provide access to the walkability of each Tile. Modify your **Sprite** class to us **a priori** collision detection to avoid stepping onto non-walkable tiles. Build a program that combines a tile map background and a player character sprite to demonstrate this new functionality.