﻿using System;

using System.Collections.Generic;

using PacMan.Ghosts;

namespace PacMan {

public class Map {

public static Map Instance;

public Tile[,] Tiles;

public Vector2 MapSize;

public int Level;

private readonly string[] \_levels = new string[] {

@"##############################

##############################

## ## ##

## #### ##### ## ##### #### ##

## #### ##### ## ##### #### ##

## #### ##### ## ##### #### ##

## ##

## #### ## ######## ## #### ##

## #### ## ######## ## #### ##

## ## ## ## ##

####### ##### ## ##### #######

####### ##### ## ##### #######

####### ## ## #######

####### ## ###--### ## #######

####### ## # # ## #######

#T # # T#

####### ## # # ## #######

####### ## ######## ## #######

####### ## ## #######

####### ## ######## ## #######

####### ## ######## ## #######

## ## ##

## #### ##### ## ##### #### ##

## #### ##### ## ##### #### ##

## ## ## ##

#### ## ## ######## ## ## ####

#### ## ## ######## ## ## ####

## ## ## ## ##

## ########## ## ########## ##

## ##

##############################

##############################"};

private readonly List<Teleport> \_teleports = new List<Teleport>();

public Map() {

Instance = this;

Level = 0;

GenerateMap();

GetNeighbors();

LinkTeleports();

}

private void LinkTeleports() {

if (\_teleports.Count < 2) return;

\_teleports[0].TeleportTo = \_teleports[1];

\_teleports[1].TeleportTo = \_teleports[0];

}

private void GenerateMap() {

string s = \_levels[Level];

string[] lines = s.Split(new[] { "\r\n", "\n" }, StringSplitOptions.None);

MapSize = new Vector2(lines[0].Length, lines.Length);

Tiles = new Tile[MapSize.X, MapSize.Y];

for (int x = 0; x < MapSize.X; x++) {

for (int y = 0; y < MapSize.Y; y++) {

CreateTile(new Vector2(x, y), new Chixel(' '), TileType.Wall);

}

}

for (int x = 0; x < MapSize.X; x++) {

for (int y = 0; y < MapSize.Y; y++) {

char c = lines[y][x];

Tile currentTile = Tiles[x, y];

switch (c) {

case '#':

currentTile.Chixel.BackgroundColor = ConsoleColor.Blue;

currentTile.Type = TileType.Wall;

break;

case '-':

currentTile.Type = TileType.Door;

break;

case 'T':

currentTile.Type = TileType.Teleport;

currentTile.Chixel.BackgroundColor = ConsoleColor.Red;

\_teleports.Add(new Teleport(currentTile));

break;

}

}

}

}

private void GetNeighbors() {

foreach (Tile tile in Tiles) {

tile.Neighbors = new Tile[4];

int neighbors = 0;

Vector2 topNeighborPos = tile.Position + Vector2.Up;

Tile topNeighbor = GetTile(topNeighborPos);

if (topNeighborPos.Y > 0 && topNeighbor.Type == TileType.Space) {

tile.Neighbors[0] = topNeighbor;

neighbors++;

}

Vector2 rightNeighborPos = tile.Position + Vector2.Right;

Tile rightNeighbor = GetTile(rightNeighborPos);

if (rightNeighborPos.X < MapSize.X) {

tile.Neighbors[1] = rightNeighbor;

neighbors++;

}

Vector2 downNeighborPos = tile.Position + Vector2.Down;

Tile downNeighbor = GetTile(downNeighborPos);

if (topNeighborPos.Y < MapSize.Y) {

tile.Neighbors[2] = downNeighbor;

neighbors++;

}

}

}

private void CreateTile(Vector2 pos, Chixel ch, TileType type) {

Tile tile = new Tile(ch, new Vector2(pos.X, pos.Y), type);

FrameBuffer.Instance.SetChixel(tile.Position, tile.Chixel, FrameBuffer.BufferLayers.Obstacles);

Game.Instance.Tiles.Add(tile);

Tiles[pos.X, pos.Y] = tile;

}

public Tile GetTile(Vector2 pos) {

return Tiles[pos.X, pos.Y];

}

}

}