HW4: Platformer tips by Mohamed El Tahaway

Things to keep in mind:

• Think about whether you want to use a tilemap or manually alternate between static/dynamic entities in your code. Static entities are entities that do not move at all but your player "may" interact with them, such as the blocks Mario jumps over. Dynamic entities are entities that move within your game; examples are Mario, the coins he collects, or the enemies he fights.

Note: If you choose tilemap, we recommend you use Tiled (if you choose a different editor, just make sure to adjust your code accordingly).

• Tilemaps:

1. Open Tiled and create a new map. Specify your in-game world's height and width. This should be a fixed number that you will use later in your code.

2. Create a new tileset (i.e. spritesheet to use within Tiled). Tile width and tile height refer to the size of each sprite in the spritesheet you want the tileset to be created from (in arne\_sprites each one is 16x16). A tileset is just a mechanism of breaking up a uniform spritesheet for you so that you can place each sprite quickly and easily. Please note how many rows and columns your uniform spritesheet consists of; it will be used in your code later.

3. Click on the sprites from the tileset you just created and place them on your map! Remember that currently we are in the tile layer, so this is where you draw all your static entities.

4. Once you are done designing the static part of your map, create a new object layer and select it. This is where you will place all your dynamic entities. Select the "Insert Tile" tool as opposed to the "Stamp Brush" you were using. Then select a sprite from the tileset and place it on your map! Note that the X and Y position you place it in will be where it will appear in your world in-game. Make sure to give this object a type so that you know how to draw it in your game!

5. Check how to export your world as a flare map .txt file from the slides and do so. Please take note of your object layers' name in Tiled. The load function in the Flaremap class that parses the .txt file as a whole (the "while (get line...)" loop that calls readEntityData and readLayerData) assumes your object layer is called "ObjectsLayer". The main layer is automatically renamed to [layer] in the flare map file, so you do not need to change that one.

Now you have a flare map file which tells you the width and height of your map (which you already knew). It gives you a 2D array where [0][0] refers to position (0,0) in your tiled tilemap and the value contained in it is the sprite's index in the spritesheet + 1 (it is index plus one because in tiled spritesheet indices start from 1, not 0. You will see that 0 represents empty). It also contains all your objects' types and locations.

Now all you have to do is parse it in your code! Create an instance of the FlareMap class. It contains the variables mapHeight, mapWidth, a 2d array of pointers named mapData (the tiled tile layer tiles), and a vector of FlareMapEntities. First load the map data into the 2d array by using the load function and passing in the location/name of your flare map .txt file. This automatically loads the width and height of your map, the layer data 2d array and creates a vector of FlareMapEntity and places all your objects inside of it. It does so by using readEntityData and readLayerData.

Explanation:

The idea is once you see '[layer]' in your flare map file you want to begin parsing the data to create your map. You want to store every value in the flare map file's 2D array into your own 2D array of pointers in the code (mapData). That's what your readLayerData function does! Remember: each value is the index of the sprite in the sprite sheet you are using offset by 1.

To draw your tile map, refer to the March\_05th\_2018.pdf slides. Specifically slide 23.

This function draws every element in your 2D array based on its position in Tiled and its index in the spritesheet. Every value is multiplied by TILE\_SIZE. I will explain this soon.

Again, the idea is once you see '[ObjectLayer]' in your flare map file, you want to begin parsing your objects. That's what readEntityData does! Entity data creates a FlareMapEntity object with a type and position and pushes it to the FlareMapEntities vector. As Professor Ivan sent out in the e-mail, every FlareMapEntity in the FlareMapEntities vector are passed into placeEntity and their positions are multiplied by TILE\_SIZE. In other words, placeEntity is where you would want to initialize (set its position, sprite, acceleration, etc...) the object/entity in your code based on its type and position. Also note that the object's position is multiplied by the TILE\_SIZE; this is explained below.

Note: Positions of objects in Tiled are way too big. We know that placing an element at (16, 5) will not show on a world with an orthographic projection of (-3.55 to 3.55) on the X and (-2 to 2) on the Y. To balance this, we create a variable called TILE\_SIZE which is multiplied to the positions of objects and entities we get from Tiled to make them fit into our world more compactly. We multiply every tile's position with TILE\_SIZE when drawing the tileMap/mapData and every entity's position when storing its position in the code (readEntityData and placeEntity). This also helps us figure out the size of each tile is in our world. It is simply a conversion from Tiled coordinates to our world coordinates.

How then can we check for collisions if the position of an object is tiled position \* TILE\_SIZE and positions of static entities are based on tiled position alone (our 2d array of pointers)?

Good question! We use the function worldToTileCoordinates to switch between tiled position and entity positions in our game. We know TILE\_SIZE so this is easy! This function returns void and directly changes the variables passed into it, so make sure to use that accordingly.

How do we check for solid tiles and/or keep track of which tile is solid and which isn't?

One way is to keep an array/vector/unordered\_set of objects and store indices of sprites that are solid then create a function that takes in an index number and checks if it is a solid entity by looking in the array/vector/unordered\_set. Take your player in-game coordinates, switch it to the player's tiled position and check the 2D array you have (mapData) to see what block the player is on.

How to get a tile's top?

Multiply the tile's row index by the TILE\_SIZE.

That's it! Everything else you know how to do, from using box-to-box collision, to moving and rendering your entities using model matrices.

• If you are not using a map editor. You can use entities just like we have been using before. To do this for a platformer you need to give each entity a type (player/enemy/collectible/etc..), and a bool declaring whether it is static or dynamic. Apply gravity and/or movement to dynamic entities and keep static entities in the same position at all times. For every dynamic entity you need to check if it collided with every other dynamic and static entity you have created.

• Remember that the viewMatrix is just what part of your world you're focusing on at the moment. It is by default set to focus on the center of your screen at (0,0) with width 7.1 (coming from 3.55\*2) and height 4 (2\*2), if you are using the default orthographic projection values. This doesn't mean that other parts of your world do not exist, say for example an entity at (16, 24)! To focus on these parts, just move your viewMatrix. In order to center it on an object: translate the view matrix by the inverse of the object's position.

• To "collect" a coin or "kill" an enemy, you just have to move them off-screen to a random position on collision.

• Remember to use our updated version of box-to-box collision. We use it so that we can resolve collisions more accurately and also know exactly where the collision occurred on both objects colliding. The idea is simple, first move on the x-axis and check for x-axis collisions, if any then resolve them. Then move on the y-axis and check for y-axis collisions, if any resolve them. It is basically what we have been doing except broken down into two steps.

• Gravity is just a force downwards (negative y velocity). Just change every entity's y position using its gravity value and set the gravity value to 0 if you do not want any change.

• Note that your map will not be clamped to the left and bottom of your screen correctly. It will be drawn awkwardly near the middle. You can fix this by adjusting your viewMatrix. When you translate your viewMatrix on the x, either translate the x by -player.position.x OR -3.55 (depending on your orthoprojection), whichever is lower. Think about how you can do this for the y.

\*Extra Exploration (\_not required\_)\*:

Make your character/player/enemies jump!

If you decide to walk the other way (if you normally walk right, but decide to walk left or vice-versa), how can you flip your character's orientation so that they are facing the direction they are moving in? (\*Hint\*: either think vertices of your triangles, vertices of your texture, or think modelMatrix).

