I started this project on 14th of October and worked it basically every day until 31th December. That's 2.5 months or 11 weeks to be precise. I'm really proud of my work but there are definitely things that I could do better.

What went right:

1. **Animation System:**

I've based system on spritesheets which are cut based on coordinates passed in code. There are a few variables to set to create animation – row (all spritesheets are iterated over columns), number of frames (columns), frames per one sprite position, is looping. After setting up system once, creating new characters, adding animations and editing them were simple and fast. There isn't a possibility of changing time for chosen frame, which is a little downsize.

2. **Map creator in Tiled**

Creating maps in tiled and importing them as a .txt file greatly improved speed of creating levels. Also there is possibility for potential players and moders to add new levels. Due to using Tiled I could use different tiles fully randomly which slightly improves visual aspects of game.

3. **Collision detection**

Physics engine, even though it's pretty simple, works great. Ground collision, enemy collision, drone's spears collision are spot on. Base case is basing collider on model's AABB (frame WIDTHxHEIGHT specified on importing spritesheet), but I've changed it slightly many times. For example arrow's basic collider is a square which isn't desired – I've changed it to long rectangle with long X and short Y coordinate values.

4. **Enemies**

Enemies were recreated the same way as in original Ancient Drone and the same way as I wanted them to work. Basing them on class EnemyBase and then on MovingObjectPrototype was great because I've reduced amount of work on each enemy to gameplay programming. Everything related to model initialization, sprites, animations and collisions are resolved with few initialization methods.

5. **ModelClass**

ModelClass is class that I've based all models on. 4 vertices, 6 indices, translation, rotation, scale, collision detection. That's all what I needed it to do. It was a base for every shader to work on. Why am I treating it as something that "went right"? Because it was something similar to "SpriteRenderer" in Unity. I could just add component, put sprite and everything works perfectly as I wanted. It also included "BoxCollider2D" (speaking in Unity terms again), Transform. Creating ModelClass was basically like creating whole new GameObject which I could work on. But having such power came with a great downsize…

What went wrong:

1. **ModelClass**

… I had to use everything in model-space which were based on screen-size (800x600). At some point I've realized that I can no longer work in [0, 1]-Space and I had to accept this fact. Also basing everything through two months on as I called model-space caused me to accept and ship game in only one resolution (800x600). It was too late to fix this shameful mistake on mine without risking many errors and failing to ship at the end of year.

2. Optimization

Game doesn’t work bad. On my setup that’s solid 60FPS (i3-4160, GTX 750Ti 2GB) but during development I didn’t thought through collision detection enough. Therefore every ground is checked if collides with object every physics tick (20ms). For player I’m checking only visible ground which isn’t really helpful. More profit comes with frustum culling I’m doing for ground tiles outside of camera (even though it’s not real frustum culling, I’m doing CPU ModelClass distance calculation). No other significant optimization or heuristic algorithms were implemented. With a few times larger levels there would be probably awful FPS drops making game unplayable.

3. Using .DDS files

Working with this format was really hard. For some reason changing .png files using transparency to .dds using Paint.NET caused errors so I had to use online converters and they also didn’t work sometimes. Due to using D3DX11 I was forced to use this format. As I was just learning and doing first larger project using DirectX11 I coped with .dds files in order to be able D3DX11. In next projects I hope to use newer libraries like DirectXTex or DirectXTK.

4. Rendering order

Even though it’s rather minor problem, I regret not implementing rendering order. In case of wrong rendering order I had to manually change order of rendering in main process by swapping order of rendering execution. It wasn’t a big problem and almost everything works as it should be (I didn’t resolve UI-crow bombs only).

5. Spritesheet filtering

Using point filtering and spritesheet at the same time caused some glitches when sprites are moving rapidly (jumping for example). On the other hand, linear filtering was causing even more serious glitches. Potential fix is to slice spritesheet and treat each sliced peace as independent texture.

Everything that I’ve mentioned in “What went wrong” is something that I’d like (and hopefully will) avoid in my future project. I hope you’ve enjoyed reading my postmortem, weekly progress and Ancient Drone Remake. Thank you!