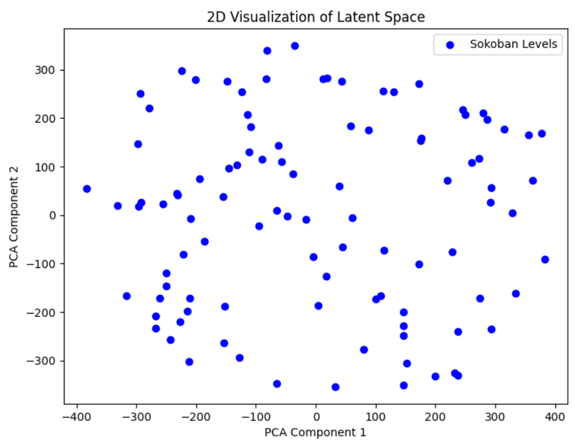
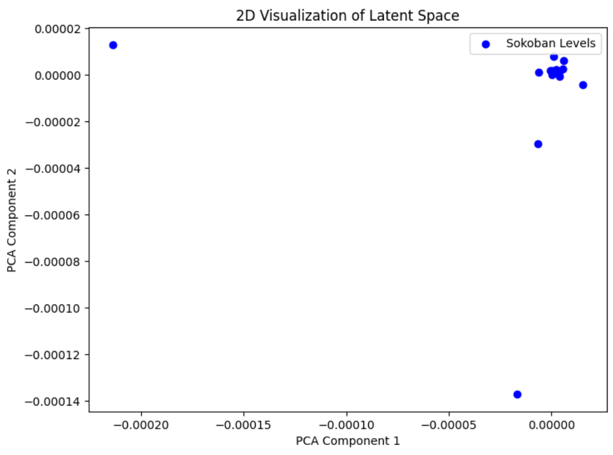
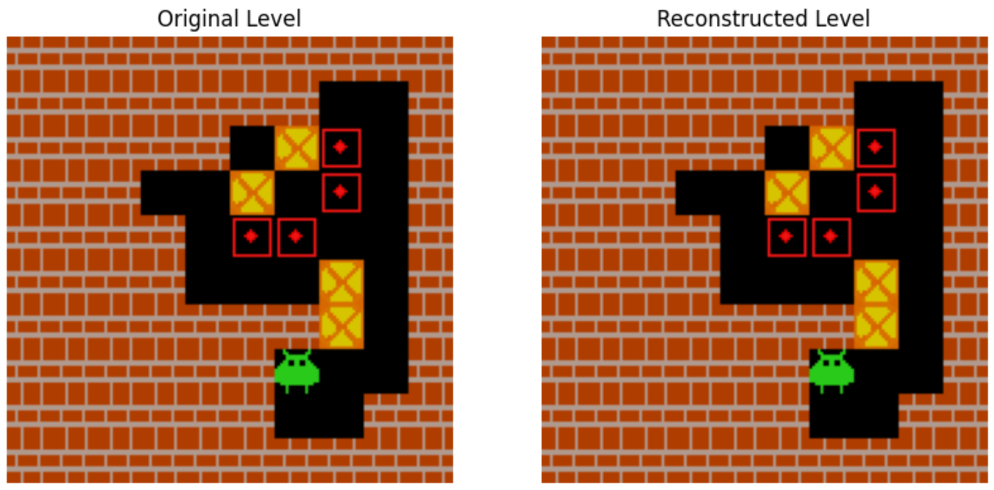
I my project, I used Jumanji Sokoban environment to create new levels from sampling latent space using autoencoder and variational autoencoder. In my hand in folder the autoencoder and variational autoencoder are in separate files, and there is also a file with utils functions that are used in both autoencoders. To create autoencoders I used JAX framework with Optax for optimization.

Both autoencoders were trained on 3D tensor representations of randomly sampled Sokoban levels, where each object type was mapped to a specific channel. They both encode 100 levels into 64-dimensional latent space and then reconstruct them. Then they are trained for 500 epochs and once the training is done, they generate 50 levels by sampling from latent space. The VAE has an additional step that involves reparameterization of sampling from latent space. Below is a visualization of the latent space, autoencoder on the left and VAE on the right.

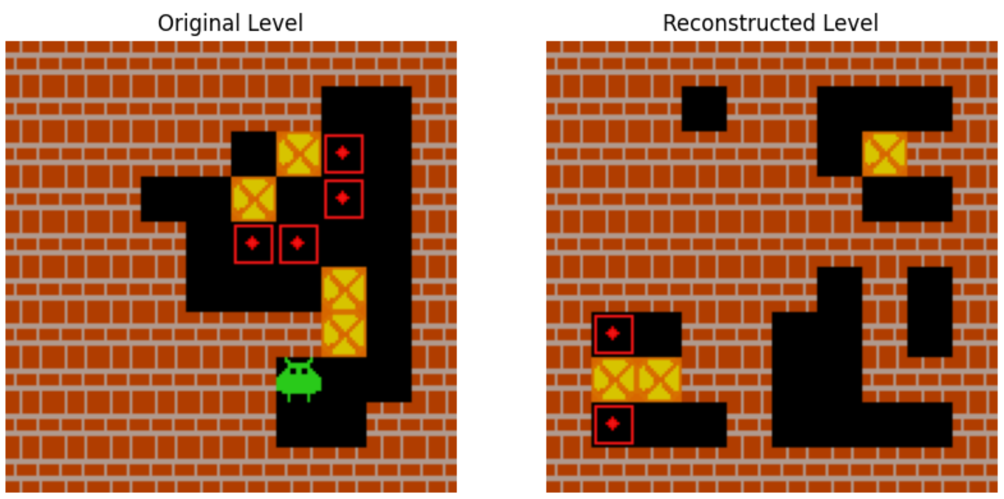




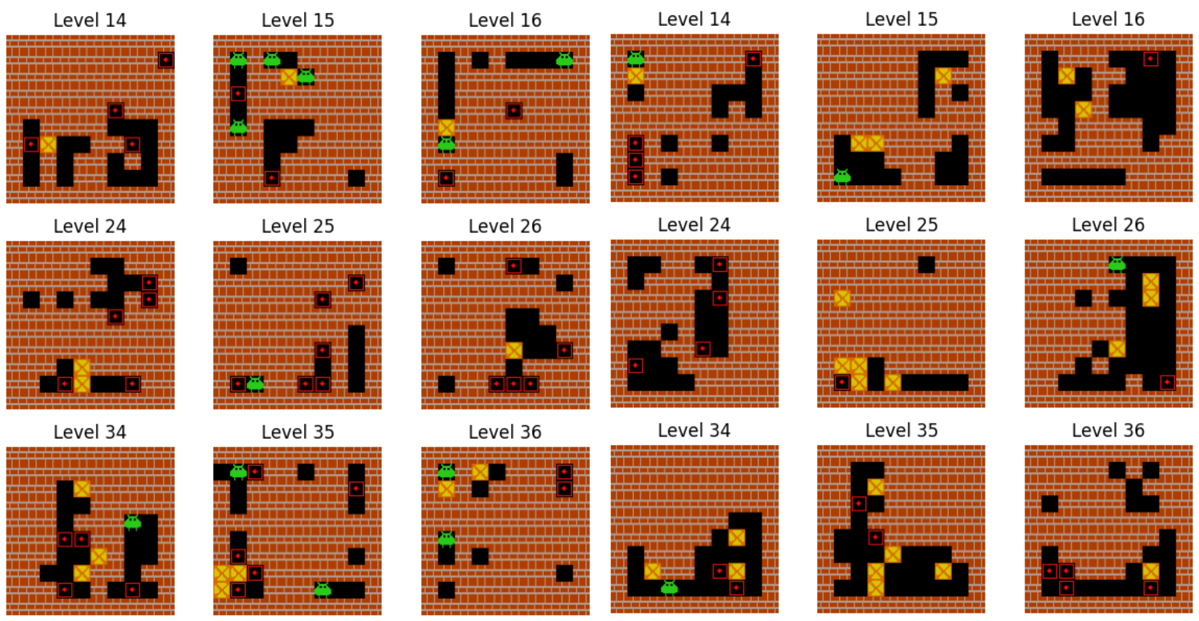
In terms of decoding sampled levels, autoencoder seems to give more accurate results. After sampling 100 levels and training for 500 epoch, it was able to recreate pretty much perfect level after decoding.



The VAE on the other hand produces less accurate reconstruction after the same number of sampled levels and the same training. I assume it is like this because of the randomness introduced in latent space, so it is expected outcome.



As for the generated levels, both autoencoders produce diverse but unplayable levels. Below is a comparison of some of the generated levels, autoencoder on the left and VAE on the right.



Because both autoencoders have no rules or constrains, generated levels are unplayable. Although, for the levels generated with VAE, the aspect of randomness can be noticed a little bit. The levels seem to be more spacious and have more variety.

If I'm being honest, working on this project was quite frustrating. It's probably because of my lack of experience with jax and implementing autoencoders, but it didn't help that there was no clear starting point that I could use to build on top of. Throughout the development I had no idea whether what I'm implementing is good or not, and overall, most of the time I spent on trying to figure out how to solve bugs related to the code or autoencoders implementation. I really wish we had some code examples available, not to copy for the assignment but to get a better understanding of how autoencoders should be implemented in jax, before we make our own. But with that being said, I definitely learned a lot about autoencoders. I've tried multiple approaches to solving this assignment and while most of them are probably completely wrong, I think I figured out one that works the way its supposed to.