Utrecht University

Game and Media Technology
Department of Information and Computing Sciences

# Graph-defined Dungeons: Exploring Constraint Solving for the Generation of Action-Adventure Levels

**A Master Thesis project by Twan Veldhuis**

## Introduction

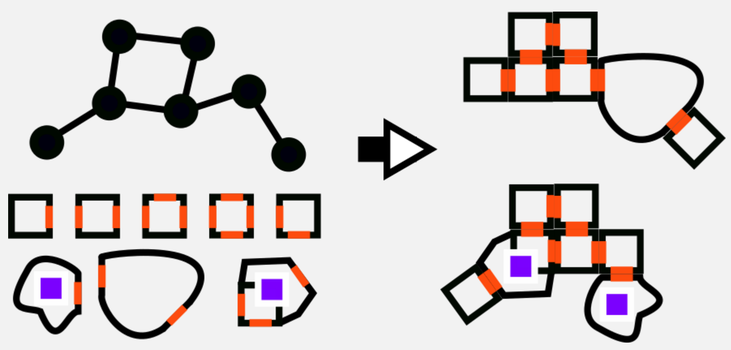
Procedural Generation is an effective way to increase the replayability of games as well as increase the productivity of the content creation process. We look into the generation of dungeons. An interesting type of generation systems involves constraint solving. These systems allow for a declarative specification of content. We introduce a constraint solving method to create 3D dungeons that are not restricted by a grid.

## Input

As input, we have a graph. This graph specifies the topology of the dungeon, and may, in turn, be generated by a different system. The graph maps directly to a constraint network and a designer may also specify additional constraints on the nodes and edges. We also provide pieces of a dungeon, which should fit together seamlessly and respect the topology specified by the graph. We call these pieces modules. Modules specify collision information, a number of connectors and the actual geometry.

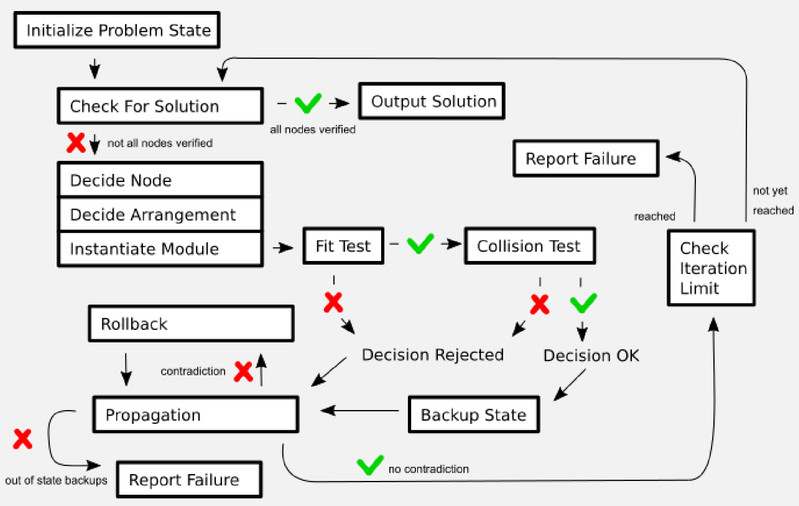
## Method

We created a constraint solving system that is designed around the problem of fitting pieces together. Connectors specify where, and in which orientation modules can connect to each other. They also specify plug types, which allow distinguishing between structures that connect modules, such as a tube or a hallway. Arrangements specify the combination of which module to use, and which connector should connect to which neighbor. We use backtracking and a modified version of Mackworth's AC3 to find an arrangement for each node in the graph. The system is implemented in Unity.



### Input and Output

We use a graph to specify topology and we define a set of modules. These modules are then fitted to create dungeons that have the same topology as the graph. In this figure, we show two example outputs for a specific graph and module set.




### System Overview

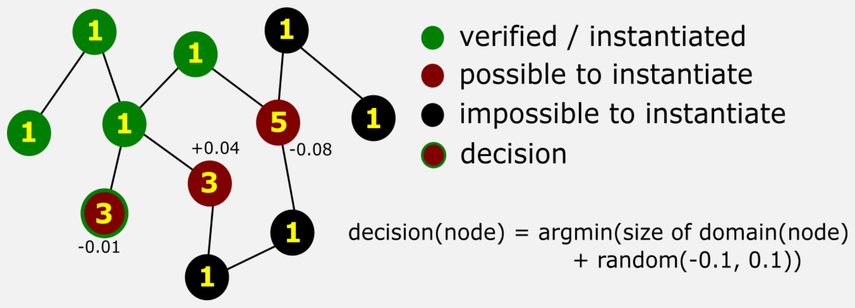
This diagram shows an overview of our constraint solving system.

This is a backtracking and propagation system, with additional verification steps. Before fitting tests and collision tests may be performed, the module specified by the arrangement is instantiated and placed into the world.

### Decision Model

In our system, we need to decide on nodes and arrangements. We choose the node with the smallest domain of arrangements. We then select an arrangement at random.

We only consider nodes that are next to a node we already know the arrangement of.

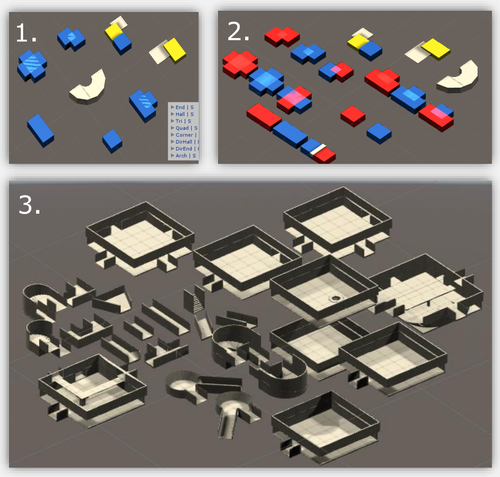


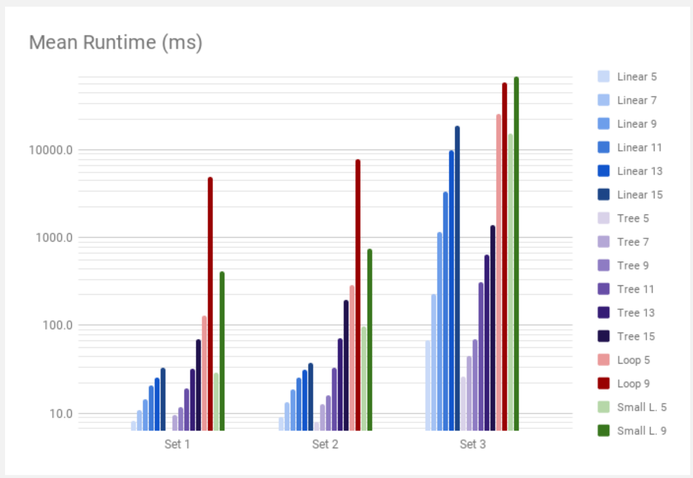

decision(node) = argmin(size of domain(node) + random(-0.1, 0.1))

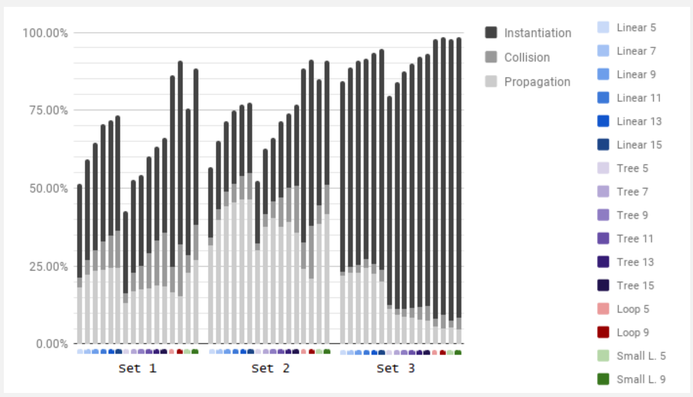
### Results

To the right are some of our results, including the average runtime and the distribution of time spent in different parts of the system.

Below are the module sets we used for our experiments.




## Experiments

We tested our algorithm for three module sets and sixteen different graphs. 100 runs were performed per test. A cutoff point of 25000 iterations was used. We measured the number of iterations, runtime and the time spent in different parts of the algorithm. We also logged the number of fitting failures and collision failures.

## Conclusion

Performance seems to scale exponential with the number of nodes. We also find that the different module sets have an impact on performance, in particular the third set.
The system still has some room for improvement. Most notably, instantiation of modules is a bottleneck, which may be resolved by delaying the instantiation of geometry to the end of the algorithm and re-using existing instances. Additionally, there is some overhead in creating backup states. This may be replaced by a patch stack, that only records changes.

We also consider some general improvements, such as adding support for partial generation. Additionally we may split the algorithm in two phases. The first phase would deal with fitting the geometry to the graph and the second phase would ensure plug types would match up.