Adapting the mapping diffculty for the automatic generation of rescue challenges

Zwennes, O.; Weiss, A.; Visser, A.

Publication date
2012
Document Version
Author accepted manuscript
Published in
Proceedings of the RoboCup IranOpen 2012 Symposium (RIOS12)

Adapting the mapping difficulty for the automatic generation of rescue challenges

Olaf Zwennes
Center for Advanced Gaming and Simulation, Universiteit Utrecht, Princetonplein 5, Utrecht, The Netherlands
Astrid Weiss
ICT&S Center, University of Salzburg, Sigmund-Haffner-Gasse 18, 5020 Salzburg, Austria
Arnoud Visser
Intelligent Systems Laboratory Amsterdam, Universiteit van Amsterdam, Science Park 904, Amsterdam, The Netherlands

***Abstract*—This paper focuses on the method and implementation of a map generator for the USARSim environment, which is capable of generating indoor environments. The generator adapts to a difficulty measure, which signifies how difficult the generated map should be, when mapped by a robot. Both the method of the procedural generation process as well as the knowledge on the difficulty measure are explained, followed by the implementation of the generator. Multiple maps with various difficulties are generated and mapping runs are simulated by experienced robot operators. Then the difficulty is assessed by these operators and compared to the difficulty level of the maps. The rules of the generator turn out to be able to influence the difficulty of the maps, but due to the complexities of 'difficulty' it is difficult to do this consistently.**

I. INTRODUCTION

For many a roboticist, the ultimate thrill is to have your programmed robot successfully fulfilling a task in front of your eyes. But before a real life robot is capable of impressive feats like mapping out rooms and finding its way around a campus, a lot of testing has to be performed. It is not uncommon for this testing to be performed in a simulated environment, in order to make testing reproducible and making analysis of the experiments more accurate. In order to best simulate the sensor input of a real-world robot, sophisticated video-game engines are used to simulate the environments and the robot itself. A popular software package used for these kinds of robot simulations is the USARSim package [1], build on the Unreal Engine 3^{TM} game engine.

USARSim offers various environments with its default package. Users of USARSim can use the game engine's tools to create arbitrarily many and diverse indoor maps to offer a representative collection of maps to test the algorithms for simultaneous mapping and localization of the robot. The experiments in a simulated environment are usually used to test an algorithm in as many situations as possible, in order to prove the performance of the algorithm in general. While an algorithm may be written for a specific type of environment, like indoor maps, it always needs to be tested on a wide variety of maps in order to confirm that the algorithm performs well on an arbitrary environment of that type. It would be beneficial if the variety of maps could be controlled with a single parameter; in this paper this parameter is the *mapping difficulty*.

Procedural generation is a technique from the game developer community which can be used to generate diverse maps of various types in a game engine. The process has already been used to generate both outdoor and indoor maps, as well as assets, like vegetation in outdoor maps or complete rooms in indoor maps, to fill those maps with a rich variety of objects [2], [3], [4]. Procedural generation is the use of a set of rules, usually with one or more randomized factors, to generate graphical assets, like the examples mentioned before. In procedural generation, there is a balance between randomizing parameters of the generation algorithm and having these parameters set by strict rules that encode knowledge about the type of asset to be generated. Indoor environments procedurally generated for use in USARSim could possibly have implicitly a variable difficulty for robot mapping tasks. The influence of the different rules of the generation algorithm with respect to mapping difficulty should be identified, in order to ensure that the difficulty of a generated map become explicit; i.e. could be controlled. The central question of this paper is: Can the rules used in an algorithm for generating indoor environments be used to make the generator adaptive to a difficulty measure for a robot mapping task? This means the algorithm will have to be able to generate indoor maps that reflect a user-set difficulty, where an increase in difficulty results in an indoor environment that is harder to map for the simulated robot. This will a require knowledge on the difficulties of mapping an environment, as well as procedural generation techniques. In this paper an initial attempt is made to incorporate this knowledge in a set of rules.

In the next section, related work is discussed and compared to this research. Following the related work is a section on the method and application of the adaptive indoor map generator, explaining the choices made in the application. The next section discusses how the experiments with the generator are performed and the results of these experiments are given afterwards. Finally, a conclusion is drawn from those results and the research as a whole is evaluated in a discussion section.

II. RELATED WORK

Related and relevant work can be found in both the field of game development and the field of robotics. In the field of game development, there is little work in the procedural generation of indoor environments. An interior layout is the

application of Tutenel *et al.* in rule-based layout solving [4]. Their rule-based system defines classes consisting of a particular type of object and rules associated with these classes which determine their placement in an interior layout. Our approach works with rooms, which allows to generate larger worlds, but with less resolution. In addition, in this article the rules are extended to encode the abstract notion of a difficulty measure for each entity. The rule-based system of Tutenel et al. is also fairly rigid, solving the rules to one best solution. By including random numbers, a more diverse set of interiors can be generated. This approach is used in this report.

Having an environment, like a program or a game, adapt to a difficulty measure, has been researched in a variety of contexts. Carro, Breda, Castillo and Bajuelos presented a framework for creating adaptive educational games [5]. The framework requires a set of parameters, one of which is a difficulty of each game. A game of a specific difficulty (among other features) is then considered, given the progress of the student/player. The system thus uses annotation and rules to adapt an educational game to the skill level of the player. In the context of game entertainment, a system for procedurally generating levels for platform games has been proposed by Compton and Mateas [6]. The difficulty of such a platform level is calculated by determining the timing and the spatial window required for successfully landing a jump from one platform to the next. So contrary to the previously noted research, in this case a simulation of the player's behavior is used to calculate the difficulty, rather than annotations and a rule-based system. A rule-based system is used in this report, instead of simulation as a method to calculate difficulty, because the freedom of movement of a robot is too large to reliably calculate the difficulty of a map using simulations alone.

While there has not been any research in adapting virtual environments to a simulated robot's skill at tasks like mapping, there have been reports noting the specific difficulties particular algorithms have when mapping environments. The book by Thrun, Burgard and Fox notes four important mapping problems [7]. The first problem is size, which means the larger an environment is, compared to the range of the robot's sensors, the more difficult the mapping task becomes. This problem is relevant to this report, because a map generator can vary the size of elements of the map (or the entire map) in order to vary the difficulty of that map. The second problem is noise in sensors, because when noise increases, localization and recognition becomes harder and thus mapping becomes harder. USARSim could be easily configured to vary the noise level of its sensors [8], but this is only relevant for the world generation when a dependence of sensor measurements on surface characteristics [9] is included. The third problem is perceptual ambiguity. When different places in an environment look a lot alike, it is more difficult for a robot to distinguish between these places. This problem can be used by the map generator, by creating ambiguity inside a map; it will generate a more difficult map. The last problem Thrun *et. al.* note is cycles in the environment, where a robot can come back to the same location using a different path. The odometry errors accumulate when a robot moves through a cycle, which results in mapping inaccuracy. This last problem is also relevant to the indoor environment generator, because cycles can be generated or prevented in order to increase or decrease map difficulty.

III. WORLD GENERATION

USARSim includes several tools to make certain tasks required for robot simulation easier. One of those is the so called World Generator[1], which is a simple map creator; an interactive tool where users can drag pre-made rooms and hallway pieces to a grid of arbitrary size and then export the created map to a format that can be converted to a 3D environment for use in USARSim. The tool also includes a procedure to automatically generate a map using simple generation rules. For instance, the hallways are laid out using a Manhattan-like grid with a constant distance between hallways. This automatic procedure is expanded in this report to adapt to a difficulty that the user sets when generating a new map.

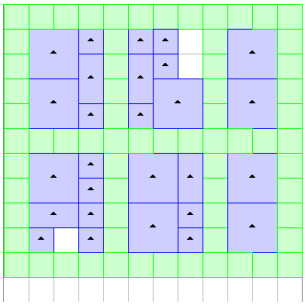


Fig. 1: A standard map generated by the World Generator tool.

There are three main rules in the existing World Generator, which encode the geometric properties of the environment. The first is the layout of the hallways in the map, where horizontal and vertical hallways are placed a specific distance apart, which span the entire width and height of the map respectively. In Figure 1 this hallway layout is clearly visible. The light-green colored grid positions are hallway pieces and the light-blue colored boxes are pre-made rooms inserted into the map with the triangle in the center determining their orientation. The white grid positions are empty (solid wall) and not accessible by the robot. The pseudo-code of the hallway rule is given in Algorithm 1.

The second rule places randomly the rooms in the empty space between the hallways, as illustrate in Algorithm 2. The World Generator contains a library of rooms (currently 20) with different sizes (ranging from 1x1 to 6x6). The function *fits*(room) checks if the empty space between the hallways is large enough for the randomly selected room.

The third and last rule places the doorways to the rooms, as indicated in Algorithm 3.

These decidedly simple rules encode all geometric properties of the generated map, with the exception of the content of the rooms, which is pre-made by human designers. Together,

[1] http://usarsim.sourceforge.net/wiki/index.php/World_Generator

Algorithm 1: The Hallway rule

Input: horizontalSpacing; verticalSpacing
Data: The map M with gridcells $\{r_i, c_j\}$
forall j **do**
  if modulo$(j, horizontalSpacing) == 0$ **then**
    $\forall_i M_{i,j} = hallwayPiece$
  end
end
forall i **do**
  if modulo$(i, verticalSpacing) == 0$ **then**
    $\forall_j M_{i,j} = hallwayPiece$
  end
end

Algorithm 2: The Room rule

Input: roomList
Data: The map M with gridcells $\{r_i, c_j\}$
forall i, j **do**
  if $\{r_i, c_j\} == \varnothing$ **then**
    repeat
      $\{r_i, c_j\} = room \in_{random} roomList$
    until fits*(room)* ;
  end
end

these rules generate the entire environment, but they do not create very varied maps, as variation is only achieved at two locations (by choosing rooms at random and randomly placing doorways).

IV. ADAPTIVE WORLD GENERATION

As previously mentioned, the generator is an adaptation of an existing tool for USARSim, as described in the previous section. This tool is written in Java and provides a valuable base for the generator to be built upon. Firstly, the World Generator dictates a grid structure that the generator can build its map on. More importantly, the tool includes processes to convert the grid map to a plain text file that can be converted to a proper 3D environment file, usable by USARSim. The tool also already includes a variety of pre-made rooms that can be placed in the map by the generator. Finally, the World Generator already has a simple map generator, whose rules can be expanded to build an adaptive map generator.

Algorithm 3: The Doorway rule

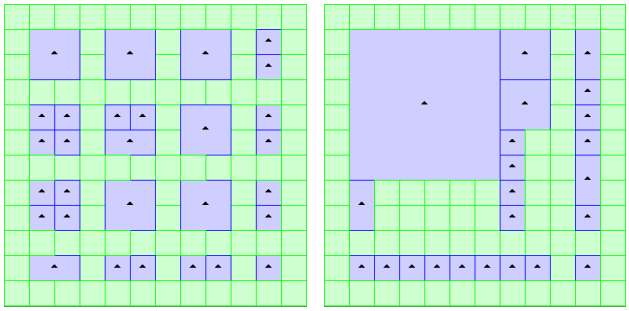
Input: doorwayPercentage
Data: The map M with gridcells $\{r_i, c_j\}$
forall $rooms \in M$ **do**
  forall $sides \in room$ **do**
    if random() $< doorwayPercentage$ **then**
      remove a doorway-sized part in the middle of the $side$
    end
  end
end

To create the adaptive generator, five adaptations to the existing generator have been made. The user is required to set the difficulty to a value between 0 and 10.

The first of the adaptations is a variable distance between hallways. The distance between hallways was set to an arbitrary value in the unmodified World Generator tool. The adaptation consists of increasing the distance between hallways in both directions (independently), when the difficulty increases. This adaptation is illustrated in Figure 2 with the lowest and highest difficulty, respectively. The distance between hallways hs is calculated independently for horizontal and vertical spacing (the **Input** of Algorithm 1) using the following equation:

$$hs = (0.06 \times d + 0.2) \times s + r \times 0.1 \times s \qquad (1)$$

The hallway spacing hs is a function of the user-defined difficulty d(range: 0-10), the size of the map s (either the height or width of the map in grid units). Variation in maps of the same difficulty is accomplished by including a random number r, which is a number drawn from a standard Gaussian distribution. The hallway spacing hs is kept within a range of 3 to s, to ensure there is enough space between hallways for the generator to place rooms and there are always at least two hallways per direction.



(a) lowest difficulty ($d = 1$), where the distance between hallways is small.

(b) highest difficulty ($d = 9$), where the distance between hallways is large.

Fig. 2: A grid-view of maps with varying hallway layouts.

The next adaptation changes the random room placement of the unchanged World Generator tool. In order to adapt the map to a difficulty level, the rooms inside the map also need to be of a difficulty corresponding to that level. The adaptation consists of annotating the rooms that can be placed inside the environment, with their own difficulty. The rooms are assessed, ideally by an expert in the field of robot mapping, and given a difficulty value $d(room)$ in the same range (0-10) as the difficulty values d the user sets for generated maps. A list of difficulties set to each room can be found in the appendix of [10]. The probability of a room to be placed on the map depends on how well the room's difficulty matches with the map's difficulty. This correspondence is calculated with

as the inverse squared error ($ISE(room)$) of the difference $d - d(room)$ between the requested map difficulty d and the estimated difficulty $d(room)$ of the selected room. The equations that calculate this probability are as follows:

$$ISE(room) = \frac{1}{(d - d(room))^2 + 1} \tag{2}$$

is calculated for each room, where $d(room)$ is the difficulty annotation of room. The probability of selecting this room is calculated by normalizing the inverse squared error:

$$P(room) = \frac{ISE(room)}{\sum_r^{rooms} ISE(room)} \tag{3}$$

The +1 in equation 2 prevents a possible division by zero. The Room rule (Algorithm 2) is now adapted, in the sense that each room has a chance of $P(room)$ of being placed in that space. Using probabilities ensures that maps of the same difficulty are still varied, because all rooms have at least some probability of being placed. This method also ensures that on average, the difficulty of the rooms inside an environment is the same as the difficulty of the environment, as set by the user.

The third adaptation is a variable doorway probability. The unmodified World Generator tool already places doorways on the sides of rooms according to a probability and this probability is made adaptive to the difficulty level. The reasoning behind this adaptation is that having more doorways to a room on average, results in more small-scale cycles, where the robot can enter and exit the same room through different doorways. This can result in the same room being mapped from different directions and the two views of the room being misaligned on the map. So, a lower doorway probability $P_{doorway}$ is assigned to a map with a lower difficulty level, according to this equation:

$$P_{doorway} = \frac{d}{16} + 0.375 \tag{4}$$

This equation ensures that the probability of placing doorways is never 0, because that would make every room inaccessible.

The last adaptation adds a new rule to the generation process, as opposed to changing the existing rules to become adaptive. This rule is designed to prevent cycles in a generated map. A cycle in a hallway can result in a misaligned hallway on the robot's map. The robot builds up an odometry error when moving through the cycle, that is insignificant at a local level, but when the robot completes the cycle the odometry error is large enough that the cycle is misaligned and the map reflects that error [7]. The new rule blocks off horizontal hallways at specific positions, so the hallways do not form cycles anymore. This rule is applied with a probability $P_{PreventCycles}$ of:

$$P_{PreventCycles} = 1 - \frac{d}{10} \tag{5}$$

At low difficulties, there is a high probability of cycles being blocked off, and that probability drops linearly with increasing difficulty. The pseudo-code of the rule itself is given in Algorithm 4.

Algorithm 4: The PreventCycles rule

```
Input: horizontalSpacing; verticalSpacing
Data: The map M with gridcells {r_i, c_j}
forall j do
    forall i do
        if modulo(i, verticalSpacing) == 0 and
        modulo(j, horizontalSpacing) == 0 then
            Remove hallwayPiece
        end
        Place random hallwayPiece
    end
end
```

This rule is inspired by the Hallway rule (Algorithm 1) but instead of placing hallway pieces, it removes them to block of the horizontal hallways. In order to ensure that the entire map is still accessible, one hallway piece is placed back per column. Figure 3 shows the effect of this rule (note the white grid positions), when compared to Figure 2a for instance. When this rule is applied it is no longer possible to move in a cycle around the map, when moving through hallways. However, because rooms can have multiple doorways, moving through a room can still result in a cycle. But since the doorway probability is lower at the difficulties that this rule is likely to be applied, it is less likely for rooms to have multiple doorways that connect different hallways together.

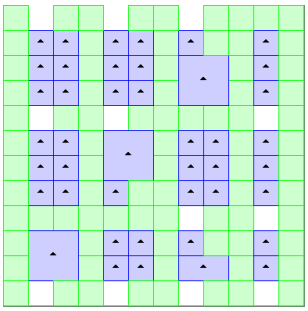
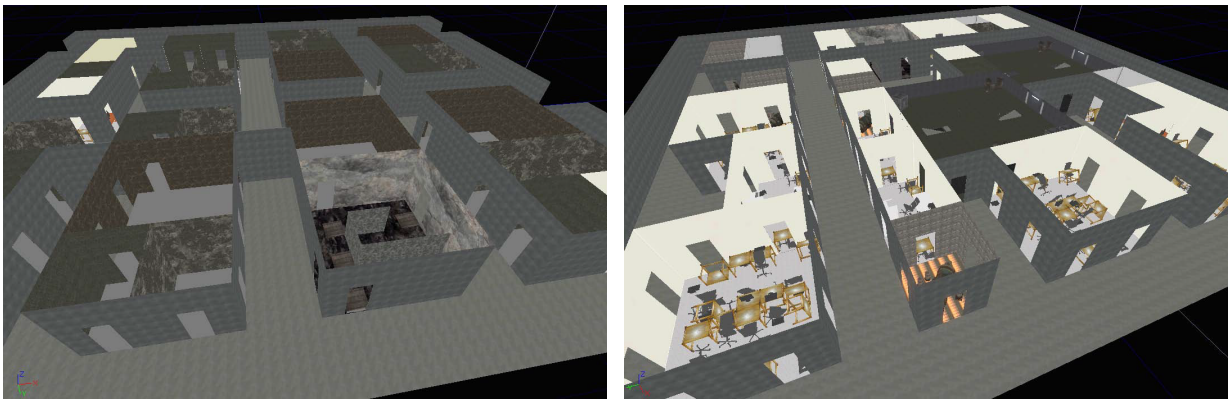


Fig. 3: A grid-view of a generated map with 'cycle prevention'.

Together, all the above adaptations of the World Generator tool result in a map generator that adapts to a difficulty that the user sets, without requiring further user intervention. It generates diverse maps with clear differences in geometry between different difficulties. Figure 4 shows just how different a map of a low difficulty is compared to one with a high difficulty, with a clear difference in room complexity, as well as in general layout.

V. EXPERIMENTS

Ideally, the difficulty of the generated environments is evaluated by having a quantitative measure of the quality of



(a) map generated with a low difficulty ($d = 1$).

(b) map generated with a high difficulty ($d = 9$).

Fig. 4: An eagle-eye view of generated maps with different difficulty settings.

a robot's map, after performing a run through the generated environment. Then, the quality of a map decreases as the difficulty increases, if the generator correctly adapts to the difficulty measure. However, determining the quality of a map, even in a simulated environment where the ground truth of the environment is known, is an open research issue [11]. And since this issue is beyond the scope of this research, a qualitative assessment of the performance of the map generator is performed.

To get a representative assessment of the generated maps, several USARSim operators with different experience levels were asked to run simulations on maps generated with varying difficulty levels. Maps were generated with five difficulties (1, 3, 5, 7 and 9), with two different maps per difficulty, for a total of ten maps. The participating operators were asked to run simulations on three maps with different difficulties. They were then asked for detailed feedback on how difficult they perceived each environment to be, when mapping the environment. They were also asked more specifically what elements in the environment made the mapping either difficult or easy. The operators were free to use the operating environment and settings of their choice, as well as choosing how many robots they used to map the environments. One reason for not specifying a simulation configuration is that the focus is on the environment and the map's specifics can be assessed regardless of the exact simulation configuration. Another reason is that the assessment of the difficulty can vary between operators, so operators are compared to themselves by comparing their feedback on the different maps they evaluated.

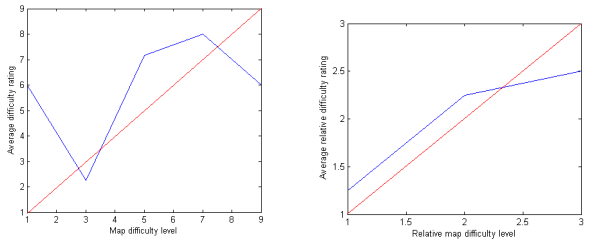
If the participants in the experiments rate the maps in the same order of difficulty as the difficulty levels at which they were generated, then the map generator encodes at least some element in the maps that gets more difficult to map as the difficulty rating of the environments themselves increases. If on top of the rating, the operators also note the effects of the adaptable rules to be the cause of the difference in difficulty, then the rules can be considered to correctly adapt to the difficulty measure. If the participants find no distinguishable difference in difficulty between the maps, then the map generator is not capable of adapting to a difficulty measure. If the operators' ratings are in reverse order compared to the supposed map difficulties, then an adaptable rule can be reversed to show the behavior at the opposite end of the difficulty scale, in order to make the map's difficulty properly scale. This does however require that the rule that caused the incorrect behavior is identified by the operators.

VI. RESULTS

Six operators provided feedback on one or three of the ten generated maps each, with experience levels ranging from no experience as an operator to experience as an operator at multiple RoboCup competitions. All operators used the same type of robot for mapping the simulated environments, namely the P3AT robot present in the USARSim package. The P3AT simulated robot is modeled after the real-world Pioneer 3-AT four-wheeled robot, with a laser range scanner for mapping purposes and a camera for operator convenience (so the operator can navigate using the camera image instead of the map). All operators used the same control software [12], with a direct connection to USARSim (without the Wireless Simulation Server in between). Some operators choose to map the environments using one robot, while others navigated two robots simultaneously through the environment. There was a 20 minute timer for the simulation, but the operators never reached this limit, instead reaching a natural ending when either the robot(s) got stuck or the robot(s) ran out of battery power.

The first of the results is the overall difficulty rating that each operator assigned to each environment they mapped. The actual maps they made can be found in [10]. These difficulty ratings have been plotted both using absolute and relative values, in Figure 5. The absolute graph plots the difficulty with which the maps were generated against the average difficulty that the operators assigned to those maps. For the relative graph, only the feedback of operators that finished runs on three maps were processed. The three maps that each operator evaluated were ordered based on the difficulty that they were generated with and then they were given a value of 1, 2 or 3

based on the order of the difficulty ratings that the operator assigned to these maps. Then the average value of each of the three relative difficulties was plotted.



(a) Absolute difficulty level against difficulty rating.

(b) Relative difficulty level against difficulty rating.

Fig. 5: Graphs of the absolute and relative difficulty ratings, against the difficulty level the maps were generated with.

While the difficulty rating is a quantitative value and as a result can be plotted in graphs, it should be noted that the sample size is small (14 samples spread over 5 difficulty levels) and the plots offer no guarantee of statistical significance. They do however give some indication of the performance of the map generator.

The red line in both graphs shows the best possible result, where the difficulty that the operators assigned to the maps equals the difficulty level the maps were generated with exactly. The blue line is the actual result, which in the absolute graph shows that the lowest and highest difficulty levels are not rated as such by the operators.

Maps that were generated with a difficulty of 1 are rated 6 on average, which is significantly higher and it is even higher than the difficulty rating for maps with difficulty 3. An experienced operator that has participated in multiple competitions, notes that the dead ends created by the cycle prevention rule result in mapping difficulties, as written in his feedback form: ”The dead ends at the corners made the mapping difficult...” and ”Only one central crossing, which forces you to repeatedly come back at the same location...”. Another operator with experience at a single competition, notes the lack of doorways in rooms as a difficulty when mapping an environment generated with difficulty 1: ”No doors found: need to leave through the door I entered.”, is his response to the question which rooms were difficult to map. Turning, especially in a small space, is a time-consuming and difficult job for a robot operator and it results in an odometry error building up, says another operator.

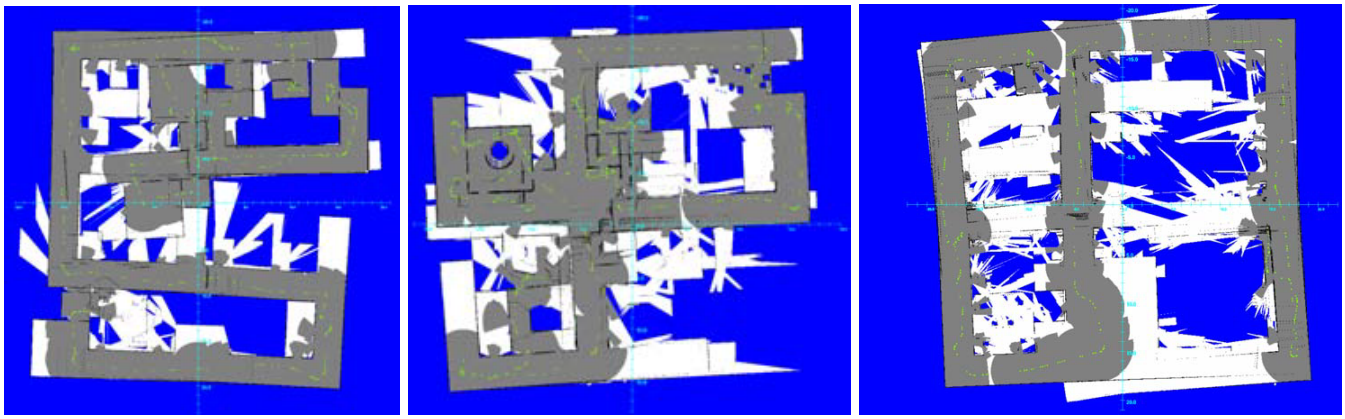
On the other end of the spectrum, the maps that were generated with a high difficulty (9) were not rated as being very difficult by the operators, as those maps got an average rating of 6. One operator with no experience notes in his feedback form that the many doorways to each room, as well as the long hallways, results in an easy to map environment: ”Long straight corridors and square rooms with openings on several sides made it easier to map.”. The probability for many doorways per room is significantly higher at higher difficulties, because a rule lowers the amount of doorways at low difficulty to avoid cycles in the environment. The hallways are also not cut off at higher difficulties, because the rule that prevents cycles has a very small chance at activating at higher difficulties. The same operator did also note that the objects inside rooms were a primary cause of difficulty in the map and rooms with obstructing objects have a higher probability of being placed on a map with a higher difficulty, due to their higher difficulty annotation (for instance the office-type rooms).

Between the extreme difficulties however, the ratings by the operators matched the desired difficulty levels to a reasonable degree, as seen in Figure 5a. This is backed up by the feedback given by the operators, who consistently noted the contents of the rooms as a reason for the maps to be either easy or difficult to map. One operator with experience in a single competition notes that a map with difficulty level 7 is difficult specifically because of the contents of the rooms: ”Although a general overview of the map is easily found by following the corridors, exploring the rooms turned out to be exceedingly difficult.”. This comment also shows that hallways were a predictable element in the environments and following those gave a very good idea of the overall layout of the entire map, which was also noted by other operators. The same operator also notes that on an easier environment (with difficulty level 3), the rooms are the reason for being easier to map: ”Relatively large rooms with little furniture are very easy to navigate.”. This comment also shows that besides the contents, the size of the room is also a factor in its difficulty, as larger rooms are noted to be easier to map, likely because the robot can more easily navigate a larger room.

VII. CONCLUSION

In this paper, the challenge was to adapt an indoor map generator of USARSim environments to generate maps according to a user-defined difficulty level, and the generator should generate maps that were roughly that difficult to map for a (simulated) robot. The focus was on the following research question: Can the rules used in an algorithm for generating indoor environments be used to make the generator adaptive to a difficulty measure for a robot mapping task? The adaptive map generator was implemented by adding five simple additions to an existing generator, to make the generated maps different, based on the difficulty level. The adaptive rules resulted in significant differences to the layout of the maps of different difficulty levels. However, operators that mapped the generated environments found that the hallways were a predictable structure in most of the generated maps. And while it is usual that hallways have a clear pattern, even in real-world indoor locations, the generator could only generate one type of hallway layout, despite adapting that layout by scaling it based on the difficulty level. This predictable structure resulted in the maps being noticeably easier to map by operators, which likely counteracted the effects of the other rules to make the maps more difficult (when the difficulty level was high).

There were two specific adaptive rules, namely cycle prevention and doorway probability, which were designed to



(a) $d = 1$ (b) $d = 3$ (c) $d = 7$

Fig. 6: Explorations performed by two different operators for three levels of mapping difficulty d, using two robots.

prevent cycles in the environment at low difficulties. And while they did succeed in that respect, they had the unforeseen effect of making the map more difficult, by creating dead ends and rooms with few doorways that required extensive maneuvering by the robot operator to back out of. So while these rules did prevent cycles and the potential mapping errors associated with them, they introduced a navigation difficulty that had a negative effect on the map. This shows that the rules encoded the abstract notion of difficulty very naively; not taking into account the complexity of difficulty and the many ways it can manifest itself.

The room placement based on the room difficulty annotation did encode difficulty in a reliable way, as nearly all operators noted that easily mapped rooms could be found in maps with a low difficulty and harder to map rooms, mostly due to obstacles and hazards like water, could be found in maps with a high difficulty. However, the rooms were annotated with their own difficulty level by someone with insight, which means the annotations encode the knowledge that this person has about mapping difficulties and the room placement rule simply uses those annotations to place rooms with the correct difficulty in the environment. The generator is not able to somehow derive the difficulty of a room, based on the room's characteristics, but is simply handed the difficulty of each room and places the room with the right difficulty in the map.

In conclusion, the simple adaptive rules are not capable of reliably generating environments with a specific difficulty for robot mapping, as the feedback of the operators proved by not consistently assigning the highest difficulty rating to the map with the highest difficulty level, for instance. The rules do however significantly influence the difficulty of the map, as the operators note differences in difficulty that they attest to the effects of the rules.

Acknowledgement

We would also like to thank all the RoboCup Rescue Simulation operators that participated in the experiments for offering their expertise and sacrificing their time: Nick Dijkshoorn, Okke Formsma, Nguyen Nhu Hieu, Julian de Hoog, Seváztian Soffia Otárola and Arnoud Visser.

References

[1] S. Carpin, M. Lewis, J. Wang, S. Balakirsky, and C. Scrapper, "Usarsim: a robot simulator for research and education," in *Proceedings of the IEEE Conference on Robotics and Automation (ICRA'07)*, 2007, pp. 1400–1405.

[2] Y. Parish and P. Müller, "Procedural modeling of cities," in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques (SIGGRAPH '01)*, 2001, pp. 301–308.

[3] O. Deussen, P. Hanrahan, B. Lintermann, R. Měch, M. Pharr, and P. Prusinkiewicz, "Realistic modeling and rendering of plant ecosystems," in *Proceedings of the 25th annual conference on Computer graphics and interactive techniques (SIGGRAPH '98)*, 1998, pp. 275–286.

[4] T. Tutenel, R. Bidarra, R. Smelik, and K. de Kraker, "Rule-based layout solving and its application to procedural interior generation," in *Proceedings of the CASA Workshop on 3D Advanced Media In Gaming And Simulation (3AMIGAS)*, 2009, pp. 15–24.

[5] R. Carro, A. Breda, G. Castillo, and A. Bajuelos, "A methodology for developing adaptive educational-game environments," in *Adaptive Hypermedia and Adaptive Web-Based Systems*, ser. Lecture Notes in Computer Science (LNCS). Springer-Verlag, 2006, vol. 2347, pp. 90–99.

[6] K. Compton and M. Mateas, "Procedural level design for platform games," in *Proceedings of the Artificial Intelligence and Interactive Digital Entertainment International Conference (AIIDE'06)*, June 2006, pp. 109–111.

[7] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, September 2005.

[8] B. Balaguer, S. Carpin, and S. Balakirsky, "Towards quantitative comparisons of robot algorithms: Experiences with slam in simulation and real world systems," in *Workshop on "Performance Evaluation and Benchmarking for Intelligent Robots and Systems" at IEEE/RSJ IROS'07 conference*, 2007.

[9] O. Formsma, N. Dijkshoorn, S. van Noort, and A. Visser, "Realistic simulation of laser range finder behavior in a smoky environment," in *RoboCup 2010: Robot Soccer World Cup XIV*, ser. Lecture Notes on Artificial Intelligence (LNAI). Springer, March 2011, vol. 6556, pp. 336–349.

[10] O. Zwennes, "Adaptive indoor map generator for usarsim," Bachelor's thesis, Universiteit van Amsterdam, June 2011.

[11] C. Walraven, "Using path planning to grade the quality of a mapper," Bachelor's thesis, Universiteit van Amsterdam, June 2009.

[12] N. Dijkshoorn, H. Flynn, O. Formsma, S. van Noort, C. van Weelden, C. Bastiaan, N. Out, O. Zwennes, S. Soffia Otárola, J. de Hoog, S. Cameron, and A. Visser, "Amsterdam Oxford Joint Rescue Forces - Team Description Paper - RoboCup 2011," in *Proc. CD of the 15th RoboCup International Symposium*, June 2011.