Towards Flexible Task Environments for Comprehensive Evaluation of Artificial Intelligent Systems & Automatic Learners

Kristinn R. Thórisson[1,2], Jordi Bieger[1], Stephan Schiffel[1], and Deon Garrett[1,2]

[1] Center for Analysis and Design of Intelligent Agents / School of Computer Science, Reykjavik University, Menntavegur 1, 101 Reykjavik, Iceland
[2] Icelandic Institute for Intelligent Machines, Uranus, Menntavegur 1, 101 Reykjavik
{thorisson,jordi13,stephans,deong}@ru.is

Abstract. Evaluation of artificial intelligence (AI) systems is a prerequisite for comparing them on the many dimensions they are intended to perform on. Design of task-environments for this purpose is often ad-hoc, focusing on some limited aspects of the systems under evaluation. Testing on a wide range of tasks and environments would better facilitate comparisons and understanding of a system's performance, but this requires that manipulation of relevant dimensions cause predictable changes in the structure, behavior, and nature of the task-environments. What is needed is a framework that enables easy composition, decomposition, scaling, and configuration of task-environments. Such a framework would not only facilitate evaluation of the performance of current and future AI systems, but go beyond it by allowing evaluation of knowledge acquisition, cognitive growth, lifelong learning, and transfer learning. In this paper we list requirements that we think such a framework should meet to facilitate the evaluation of intelligence, and present preliminary ideas on how this could be realized.

Keywords: task-environment, automation, intelligence evaluation, artificial intelligence, machine learning

1 Introduction

A key challenge in the development of artificial intelligence (AI) systems is how to evaluate them. Valid measurements are necessary to assess progress, compare systems and approaches, and understand their strengths and weaknesses. Most evaluation methods in use today yield only a single performance score that brings little qualitative insight, and is incomparable to performance on other tasks. Furthermore, few if any proposals exist for evaluating fundamental aspects of intelligence like learning capacity, transfer learning, deterioration of learned skills, as well as cognitive development and growth.

Evaluation of AI systems is traditionally done by measuring their performance on one or more tasks instantiated in an environment. A *task* is the transformation of a world state into a goal state, or the maintenance of a goal state

in light of perturbations. Tasks may be compound, have one or more explicit goals, sub-tasks, constraints, and call for continuous or intermittent action. A task is performed by an agent whose atomic actions can *in principle* perform it. An *environment* is the instantiation of the task and its context, and may include some form of body for the agent, as well as distractors/noise, that complicate the task but are not strictly a part of it. We use the term *task-environment* to refer to the tuple task+environment.

Most task-environments cannot easily be classified – let alone freely modified – along a large number of dimensions, making it difficult to systematically assess an AI system's strengths and weaknesses. This rigidity limits any chosen metrics to a small subset of systems, and complicates their comparison. Tasks such as pole-balancing or video game playing, for instance, are not sufficient for evaluating systems that can operate on a diverse set of data or under multiple high-level goals, but may be fine for certain single-goal learners.

At the other end of the spectrum, task-environments for evaluating human-level intelligence – e.g. the Turing test [22] – cannot be compared easily to those appropriate for simpler learners. Human "intelligence quotient" measures, developed by psychologists, use a set of tasks normalized by their distribution in a social group and are highly species- and society-specific – and thus not a good match for intelligent machines. Another problem with most measures proposed for higher intelligences is that they assess only single point in time [18]. Assessing a system's learning capacity, however, requires time-based measures in task-environments with adjustable complexity. A framework supporting incremental and predictable changes to compound task-environments, on appropriate features, could measure a system's learning rate. This would enable evaluation of lifelong learners and transfer learning capacity: the transference of acquired knowledge to new domains/tasks. Assessing these important aspects of intelligence calls for multiple similar task-environments that can easily be compared.

Another aspect of truly intelligent systems is capacity for cognitive development – the ability to improve the very cognitive apparatus enabling the learning. This ability can itself benefit greatly from a gradual increase in complexity and other tutoring techniques that could enhance task-environments [4]. Measuring cognitive growth (and meta-cognition) capacity might be enabled through mechanisms similar to those used for evaluating transfer learning.

Since general intelligence enables systems to (learn to) perform a wide range of tasks that they have not seen or been prepared for, it cannot be assessed in only a single task or environment. Evaluating lifelong learning – systems that continually adapt and learn new things – calls for either a wide range of task-environments, or a single (large and complex) dynamically changing multi-task environment. In both cases one would like to automatically generate such task-environments, given a high-level specification for certain features and constraints.

Although many AI evaluation frameworks exist [13], none address all of the above concerns. In Sect. 3 we attempt to collect in one place the full set of requirements that such a comprehensive framework should address, and present some preliminary ideas on how this could be realized in Sect. 4.

2 Related Work

In a comprehensive and recent survey, Hernández-Orallo argued that the assessment of general "real" intelligence – as opposed to specialized performance – should be oriented towards the testing a range of cognitive abilities that enable a system to perform in a *range* of tasks [11]. One way to accomplish this is to procedurally generate task-environments that require a suite of abilities, and appropriately sample and weight them. Hernández-Orallo takes this approach, but focuses on discrete and deterministic task-environments [12, 10]. Legg & Veness's Algorithmic IQ approach posits a similar framework for measuring universal AI with respect to some reference machine which interprets a description language to run the environment [14]. The choice of this description language remains a major issue and deeply affects the kinds of environments that are more likely to be generated. The BF programming language used in their work closely resembles the operations of a Turing machine, but cannot easily generate complex structured environments and is opaque to analysis. A wide range of description languages has been proposed for coordination and planning tasks (e.g. TÆMS [7] and PDDL [17]), but tend to focus on static, observable domains and specify things in terms of agent actions and task hierarchies which can then drive the development of AI systems specialized to the specified task.

Games have long been considered a possible testbed for the evaluation of intelligence [20]. In the General Game Playing competition, AI systems play previously unseen games after being provided with the rules in the very analyzable Game Description Language, but the games must be finite and synchronous [16]. More recently, there has been a lot of interest in the automatic play of Atari-era video games. An extensible, user friendly description language for such games has been proposed that relies heavily on opaque built-in functions and should in the future be amenable to procedural generation [8, 19]. Much work has been done on procedural generation in specific video games, but more general work is still in its infancy [21]. Lim & Harrell were able to automatically generate variants for video games written using the PuzzleScript description language, but the simulation-based approach they used for the evaluation of candidate rulesets is not feasible for very difficult games since it requires an agent that is intelligent enough to perform the task [15].

Some research has tried to relate problem structure to heuristic search algorithm performance, including efforts to use a wide variety of problem types to increase the generality of algorithms [2, 5]. Some of this work, notably that on hyperheuristics [6], has focused on algorithms that try to learn general search strategiesand don't only perform well on a few specific problem types. Understanding the impact of problem characteristics on learning has been key in these efforts, but so far only search and optimization domains have been addressed.

Similar work has been done in the field of generating random Markov Decision Problems (MDPs) [1, 3], focusing on a rather limited domain of potential task-environments. Our own **Merlin** tool [9] supports various methods for the procedural generation of discrete and continuous multi-objective MDPs, but does not adequately address the full set of requirements below.

3 Requirements for Intelligence Evaluation Frameworks

The goals of evaluating AI systems are to measure research progress, compare systems and approaches, and understand their strengths and weaknesses. We wish to achieve this for a wide range of AI systems, from very simple to very complex, where the systems may be built with different background assumptions. The framework we envision must support evaluation of intelligence on a number of aspects such as *skill*, *knowledge*, *transfer learning*, *cognitive development and growth*, *lifelong learning* and *generality*. All combined this calls for multiple task-environments, selected for appropriate amounts of similarity and complexity. Note that here we are not attempting to propose *particular benchmarks*: we are interested in identifying requirements for a framework that *can be used* to construct benchmarks for the above cognitive skills.

We have identified the following high-level properties that we consider important for a flexible task-environment framework as described above:

(A) Offering **easy construction** of task-environments, and **variants** with a wide range of features and complexity dimensions. This would include the ability to (a) **compose** and **decompose** desired task-environments and parts thereof, and (b) to **scale** and **tune** them, in part and in whole, along various parameters and properties, with predictable effects, especially for increasing and decreasing their complexity along known dimensions.
(B) Ability to specify, at any level of detail, the **procedural generation** of task-environments with specific features, constraints, etc., and how they should (automatically) grow, possibly depending on the progress of the system under evaluation.
(C) Facilitation of **analysis** in terms of parameters of interest, including task complexity, similarity, observability, controllability, etc.

Analysis of various non-explicit features of such task-environments could facilitate an understanding of their function in evaluating various systems, and thus help with their automatic generation, robustification, and standardization. Decomposition can tell us about a task-environment's structure and help find commonly used building blocks. Composition allows for the construction of (much) larger structured task-environments. Scaling helps with the assessment of progress and growth, and tunability can facilitate the systematic assessment of a system's strengths and weaknesses. These can all result in variants that are similar but different in specified ways, which allows transfer learning. Finally, automatic generation can provide us with a virtually unlimited supply of fresh task environments with which to test cognitive abilities and general intelligence.

The framework should support the gradual construction and tunability of task-environments with the following properties:

1. **Determinism**: Both full determinism and partial stochasticity (for realism regarding, e.g. noise, stochastic events, etc.) must be supported.
2. **Ergodicity**: The reachability of (aspects of) states from others determines the degree to which the agent can undo things and get second chances.

3. **Controllable Continuity**: For the framework to be relevant to e.g. robotics, it is critical to allow continuous variables, to appropriately represent continuous spatial and temporal features. The degree to which continuity is approximated (discretization granularity) should be changeable for any variable.
4. **Asynchronicity**: Any action in the task-environment, including sensors and controls, may operate on arbitrary time scales and interact at any time, letting an agent respond when it can.
5. **Dynamism**: A static task-environment's state only changes in response to the AI's actions. The most simplistic ones are step-lock, where the agent makes one move and the environment responds with another (e.g. board games). More complex environments can be dynamic to various degrees in terms of speed and magnitude, and may be caused by interactions between environmental factors, or simply due to the passage of time.
6. **Observability**: Task-environments can be partially observable to varying degrees, depending on the type, range, refresh rate, and precision of available sensors, affecting the difficulty and general nature of the task-environment.
7. **Controllability**: The control that the agent can exercise over the environment to achieve its goals can be partial or full, depending on the capability, type, range, inherent latency, and precision of available actuators.
8. **Multiple Parallel Causal Chains**: Any generally intelligent system in a complex environment is likely to be trying to meet multiple objectives, that can be co-dependent in various ways through any number of causal chains in the task-environment. Actions, observations, and tasks may occur sequentially or in parallel (at the same time). Needed to implement real-world clock environments.
9. **Number of Agents**: It should be possible to add any number of (intelligent) agents to the task-environment without specifying their behavior explicitly. This would allow for the testing of the AI in isolation, in social situations, or with a teacher [4], and the evaluation of systems of systems (e.g. simulators). Other agents can greatly affect the difficulty of any task-environment.
10. **Periodicity**: Many structures and events in nature are repetitive to some extent, and therefore contain a (learnable) periodic cycle – e.g. the day-night cycle or blocks of identical houses.
11. **Repeatability**: Both fully deterministic and partially stochastic environments must be fully repeatable, for traceable transparency.

4 Flexible Task-Environment Framework: A Proposal

To meet the stated requirements we propose a description language for task-environments containing a low number of small atomic building elements (the base operators); few atomic units – as opposed to multiple types – means greater transparency since superstructures can be inspected more easily than larger black boxes can, facilitating comparison between task-environments. This also lays the foundation for smooth, incremental increase in complexity, as each addition or change can be as small as the smallest blocks. Sect. 4.1 gives an example of what

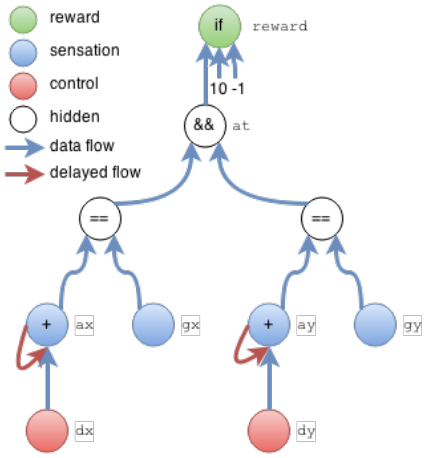
this might look like. On top of this methods for modification (Sect. 4.2), analysis (Sect. 4.3), construction (Sect. 4.4), and execution can be developed.

4.1 Example Syntax and Task

Fig. 1a shows a description of an extremely simple task where the agent must reach a goal position in a 2-dimensional space. We describe a task-environment by a set of (time-) dependent variables with causal relations. The *Initialization* section provides the initial values for the variables. In our case these are goal and agent position. The *Dynamics* section defines how variables change over time by allowing us to refer to the past variable values using time arguments and the reserved variables `t` (for the current time) and `dt` for the size of the (arbitrarily) smallest atomic time step. Unlike other languages in Sect. 2 we allow the specification of arbitrary expressions.

```
 1. Initialization:
 2.   gx = 3  // goal x
 3.   gy = 3  // goal y
 4.   ax = 4  // agent x
 5.   ay = 10 // agent y
 6. Dynamics:
 7.   dx(t) = 0  // step x
 8.   dy(t) = 0  // step y
 9.   ax(t) = ax(t-dt) + dx(t)
10.   ay(t) = ay(t-dt) + dy(t)
11.   at(t) = ax(t) == gx(t) &&
              ay(t) == gy(t)
12.   reward(t) = 10 if at(t) else -1
13. Terminals:
14.   reward(t) > 0
15. Rewards:
16.   reward(t)
17. Observations:
18.   ax(t), ay(t), gx(t), gy(t)
19. Controls:
20.   dx(t) = [-1, 0, 1]
21.   dy(t) = [-1, 0, 1]
```

(a)



(b)

Fig. 1: Example description (a) and extracted graph (b) for a task where the agent must reach a goal location, including causal connections with latency.

Lines 7 and 8 in the example set `dx` and `dy` to 0 by default. However, these variables can be controlled by the AI, as we can see on lines 20 and 21. Line 9 says that the value of `ax` at the current time `t` is equal to the previous value of `ax` plus the current value of `dx`. Line 12 uses conditional statements and refers only to variables in the current time step. The arithmetic and comparison operations make up the (extensible) set of base operators which are not further defined.

While the *Initialization* and *Dynamics* sections mostly describe the environment, the *Terminals* and *Rewards* sections can be said to describe the task in

terms of environment variables. They consist of zero or more lines which each specify an expression that evaluates to a terminal (Boolean) that ends the task or a reward (a number). Like everything else, rewards can depend on time and other variables, which allows tasks to have e.g. time pressure, deadlines, start times, and complex preconditions and interactions – perhaps even modulating other dependencies such as time.

Finally, the sections for *Observations* and *Controls* describe how the agent interacts with the environment. Observations consist of a set of sensations that occur simultaneously and are described on their own line with a comma-separated list of expressions. Controls are described as assignments of a collection of acceptable values to environment variables whose value is overwritten when specified. Non-deterministic responses of an agent's body can be modeled by making the causal connections following the controls more complex.

4.2 Example Tuning

The task-environment as described is fairly simple, being discrete, fully observable, deterministic, and static. To make the **space continuous**, we can add a controllable angle. We add `angle = 0` to the Initialization section, replace existing controls with `angle(t) = [-pi..pi]`, and modify the Dynamics section like so:

```
 7. dx(t) = dt * cos(angle(t))       8. dy(t) = dt * sin(angle(t))
11. reward(t) = 10 if (ax(t)-gx(t))^2 + (ay(t)-gy(t))^2 < 1 else -1
```

Making the space continuous in this way requires relatively significant changes. It is much easier to go from this continuous representation to one that appears discrete to the agent by discretizing controls and sensations (e.g. by rounding to the nearest integer). In the new lines 7 and 8 we have started making the environment (more) **continuous in time** as well. *dt* would ideally be infinitesimal for continuous environments, but a small value will have to suffice in practice.

To make the task more **dynamic** and **periodic** we can have the goal move a little. We replace the initialization of `gx` with `gx(t) = 4+3*sin(t)` and move it to the *Dynamics* section. The environment can easily be made **stochastic** by the use of random number generators that are provided as base operations.

We can further decrease **observability** by adding delays into the causal chain and changing refresh rates. For example, to let observations of `ax` and `ay` occur with a delay of one time step and allow observation of the goal position only at time steps 1, 3, 5, ...:

```
17. ax(t-dt), ay(t-dt)       18. gx, gy @ [1:2:]
```

In similar ways we can affect **controllability** by introducing delays or letting controls block each other for a period of time, for example, when they share actuators which may then be busy for some time.

4.3 Analysis

Analysis of task-environments can help measure their similarity, which is highly useful for evaluating learning capacity and transfer learning, and elucidate fea-

tures such as complexity, observability and difficulty that may shed light on the "why" behind the performance of various systems. For structural analysis a graph representation may be useful (see Fig. 1b). The edges show (possibly delayed) data flow; and nodes represent base operations and can be annotated with their role: reward, sensation, control, or hidden. Our description language makes important features like spatial and temporal resolution, delays, and blocking controls, readily apparent and easy to tune. The relative positions of observation, reward, and control nodes says a lot about difficulty, observability, and controllability of a particular task-environment. For instance, a task may be easier when these are grouped closely together; sensations and controls might be distractors if they are off the critical path to a reward or goal state.

The defining high-level characteristics of task-environments have yet to be identified, but will most likely include features like complexity, difficulty, observability, controllability and dimensionality. Graph algorithms such as compression, similarity detection and frequent subgraph mining can be leveraged to help determine these. A butterfly effect – where small changes in code have a large effect – may complicate purely structural analysis of some features. Tracing the construction from a small known task-environment through known transformations and compositions is likely to help.

4.4 Construction: Addressing the Range From Q-Learning to AGI

The easiest way to construct a new task-environment is to make variants of existing ones by changing initial conditions and other constants, which in our case include important concepts like resolution, delays, observability, and constraints on controls and sensors. One can also start from approximations of known tasks, although we find more important the easy construction of a variety of task-environments whose properties can be easily compared on key dimensions.

A natural way for scaling task-environments up or down is to modify the range of variables (e.g. board size in a game, or `ax` to `gx` distance in our example) or by changing the dimensionality. In most simple tasks, such as pole balancing, only a handful of variables need to be observed at a sufficient update frequency, and only a few need to be controlled. More complex tasks for the evaluation of more capable systems can be constructed in a number of ways. Tasks appropriate for human-level intelligence often have a high number of (possibly irrelevant) observable variables, and hidden variables whose state can only be inferred by observing a different set of partially and/or conditionally correlated variables. Our formalism facilitates this through easy definition of dependencies between variables, and their (un)observability. Similarly, tasks can be made harder by introducing latencies between causal connections. Much of the tuning in Sect. 3 and 4.2 can be done automatically using such techniques.

Manipulation of rewards is another obvious way to make tasks more challenging, for instance moving them further away from controls (making the causal chains longer). Adding time-dependent functions, e.g. by replacing a constant, is a natural way to increase complexity through tunable levels of dynamism.

Truly large and complex multi-goal tasks can be created in many ways by composing tasks together, requiring the AI to solve them sequentially or in parallel, especially if they share sensors and controls. This could e.g. be achieved by duplicating a single task and changing the initial state of one, and/or the ranges of some variables. Variables in one task may be made co-dependent on values of (different or same) variables in the other. There is no limit to how often this process could be repeated, with different or duplicated tasks, low-level or high-level, to create large, structured and complex task-environments.

So far we have created tasks of comparable logical complexity to Pac-Man and Pong, as well as mazes of arbitrary complexity. Their graph representations can easily be modified in various ways, creating increasingly complex, dynamically varying composite tasks, where sequential and temporal dependencies can be freely introduced. Comparing and modifying them is much easier than if using completely hand-crafted tasks with no underlying common base.

5 Conclusions & Future Work

We have identified requirements that a framework ideally must meet to allow flexible construction of task-environments for evaluating artificial learners and AI systems, and proposed a preliminary formalism to meet these requirements. In our proposed approach, defining simple tasks requires a few lines of code; scaling is straightforward. In future work we plan on completing a first version of our task-environment description language and start on the development of methods for the automatic construction, analysis, and execution of evaluating AI systems, which is important for addressing the full range of requirements identified.

Acknowledgments

This work was supported by the School of Computer Science at Reykjavik University, by a Centers of Excellence Grant from the Science & Technology Policy Council of Iceland, and by EU Marie Curie CIG #304210.

References

1. Archibald, T.W., McKinnon, K.I.M., Thomas, L.C.: On the generation of markov decision processes. J. Oper. Res. Soc. 46, 354–361 (1995)
2. Asta, S., Özcan, E., Parkes, A.J.: Batched mode hyper-heuristics. In: Nicosia, G., Pardalos, P. (eds.) LION 7, LNCS, vol. 7997, pp. 404–409. Springer (2013)
3. Bhatnagar, S., Sutton, R.S., Ghavamzadeh, M., Lee, M.: Natural actor-critic algorithms. Automatica 45(11), 2471–2482 (2009)
4. Bieger, J., Thórisson, K.R., Garrett, D.: Raising AI: Tutoring Matters. In: Goertzel, B., Orseau, L., Snaider, J. (eds.) AGI-14. LNCS, vol. 8598, pp. 1–10. Springer (2014)

5. Bischl, B., Mersmann, O., Trautmann, H., Preuß, M.: Algorithm selection based on exploratory landscape analysis and cost-sensitive learning. In: Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation. pp. 313–320. GECCO '12, ACM, New York, NY, USA (2012)
6. Burke, E.K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Qu, R.: Hyper-heuristics: A survey of the state of the art. J. Oper. Res. Soc. 64(12), 1695–1724 (2013)
7. Decker, K.: TAEMS: A framework for environment centered analysis & design of coordination mechanisms. In: O'Hare, G.M.P., Jennings, N.R. (eds.) Foundations of distributed artificial intelligence, pp. 429–448. G. O Hare and N. Jennings (eds.), Wiley Inter-Science (1996)
8. Ebner, M., Levine, J., Lucas, S.M., Schaul, T., Thompson, T., Togelius, J.: Towards a video game description language. In: Lucas, S.M., Mateas, M., Preuss, M., Spronck, P., Togelius, J. (eds.) Artificial and Computational Intelligence in Games, Dagstuhl Follow-Ups, vol. 6, pp. 85–100. Schloss Dagstuhl (2013)
9. Garrett, D., Bieger, J., Thórisson, K.R.: Tunable and Generic Problem Instance Generation for Multi-objective Reinforcement Learning. In: ADPRL 2014. IEEE (2014)
10. Hernández-Orallo, J.: A (hopefully) non-biased universal environment class for measuring intelligence of biological and artificial systems. In: Baum, E., Hutter, M., Kitzelmann, E. (eds.) AGI-10. pp. 182–183. Atlantis Press (2010)
11. Hernández-Orallo, J.: AI Evaluation: past, present and future. arXiv:1408.6908 (2014)
12. Hernández-Orallo, J., Dowe, D.L.: Measuring universal intelligence: Towards an anytime intelligence test. Artif. Intell. 174(18), 1508–1539 (2010)
13. Legg, S., Hutter, M.: Tests of Machine Intelligence. arXiv:0712.3825 [cs] (Dec 2007)
14. Legg, S., Veness, J.: An approximation of the universal intelligence measure. In: Dowe, D.L. (ed.) Algorithmic Probability and Friends. Bayesian Prediction and Artificial Intelligence. LNAI, vol. 7070, pp. 236–249. Springer (2011)
15. Lim, C.U., Harrell, D.F.: An Approach to General Videogame Evaluation and Automatic Generation using a Description Language. In: CIG 2014. IEEE (2014)
16. Love, N., Hinrichs, T., Haley, D., Schkufza, E., Genesereth, M.: General game playing: Game description language specification. Tech. Rep. LG-2006-01, Stanford Logic Group (2008)
17. McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., Wilkins, D.: PDDL-The Planning Domain Definition Language. Tech. Rep. TR-98-003, Yale Center for Computational Vision and Control (1998), `http://www.cs.yale.edu/homes/dvm/`
18. Rohrer, B.: Accelerating progress in Artificial General Intelligence: Choosing a benchmark for natural world interaction. J. Art. Gen. Int. 2(1), 1–28 (2010)
19. Schaul, T.: A video game description language for model-based or interactive learning. In: CIG 2013. pp. 1–8. IEEE (2013)
20. Schaul, T., Togelius, J., Schmidhuber, J.: Measuring intelligence through games. arXiv preprint arXiv:1109.1314 (2011)
21. Togelius, J., Champandard, A.J., Lanzi, P.L., Mateas, M., Paiva, A., Preuss, M., Stanley, K.O.: Procedural content generation: Goals, challenges and actionable steps. In: Lucas, S.M., Mateas, M., Preuss, M., Spronck, P., Togelius, J. (eds.) Artificial and Computational Intelligence in Games, Dagstuhl Follow-Ups, vol. 6, pp. 61–75. Schloss Dagstuhl (2013)
22. Turing, A.M.: Computing machinery and intelligence. Mind 59(236), 433–460 (1950)