Report from Dagstuhl Seminar 15051

Artificial and Computational Intelligence in Games: Integration

Edited by

Simon M. Lucas[1], Michael Mateas[2], Mike Preuss[3], Pieter Spronck[4], and Julian Togelius[5]

1 **University of Essex, GB,** sml@essex.ac.uk
2 **University of California – Santa Cruz, US,** michaelm@cs.ucsc.edu
3 **TU Dortmund – Dortmund, DE,** mike.preuss@cs.uni-dortmund.de
4 **Tilburg University, NL,** p.spronck@uvt.nl
5 **New York University, US,** julian.togelius@gmail.com

Abstract

This report documents Dagstuhl Seminar 15051 "Artificial and Computational Intelligence in Games: Integration". The focus of the seminar was on the computational techniques used to create, enhance, and improve the experiences of humans interacting with and within virtual environments. Different researchers in this field have different goals, including developing and testing new AI methods, creating interesting and believable non-player characters, improving the game production pipeline, studying game design through computational means, and understanding players and patterns of interaction. In recent years it has become increasingly clear that many of the research goals in the field require a multidisciplinary approach, or at least a combination of techniques that, in the past, were considered separate research topics. The goal of the seminar was to explicitly take the first steps along this path of integration, and investigate which topics and techniques would benefit most from collaboration, how collaboration could be shaped, and which new research questions may potentially be answered.

Seminar January 25–30, 2015 – http://www.dagstuhl.de/15051
1998 ACM Subject Classification I.2.1 Artificial Intelligence – Games
Keywords and phrases Multi-agent systems, Dynamical systems, Entertainment modeling, Player satisfaction, Game design, Serious games, Game theory
Digital Object Identifier 10.4230/DagRep.5.1.207

1 Executive Summary

Simon Lucas
Michael Mateas
Mike Preuss
Pieter Spronck
Julian Togelius

The research field of artificial and computational intelligence in games focuses on the wide variety of advanced computational techniques used to create, enhance, and improve the experiences of humans interacting with and within virtual environments. By its nature the field is broad and multidisciplinary. People working in it include academic researchers from a

Artificial and Computational Intelligence in Games: Integration, *Dagstuhl Reports*, Vol. 5, Issue 1, pp. 207–242
Editors: Simon M. Lucas, Michael Mateas, Mike Preuss, Pieter Spronck, and Julian Togelius
Dagstuhl Reports
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

variety of disciplines, corporate researchers in the games industry as well as in other industries, and independent game developers. The methods used include symbolic AI techniques such as reasoning, constraint-solving and partial-order planning as well as biologically-inspired techniques such as evolutionary computation and neural networks, statistical techniques such as support vector machines and clustering, as well as special-purpose techniques such as behavior trees. These are applied to games ranging from board and card games to first-person shooters and real-time strategy games as well as abstract mathematical games, and are used to, for instance, play games, model players, tell stories, generate levels and other game content, and match players. Different researchers have different goals, including developing and testing new AI methods, creating interesting and believable non-player characters, improving the game production pipeline, studying game design through computational means, and understanding players and patterns of interaction. Often several goals overlap in the same project.

Recent years have seen considerable progress in several of the techniques used in the field, as well as rapid development in what kind of research questions are asked and what kind of games are studied with which methods. It has become increasingly clear that many of the research goals require a multidisciplinary approach, or at least a combination of techniques that, in the past, were considered separate research topics. For instance, with respect to the behavior of virtual agents, ten years ago researchers mainly aimed at making such behavior more "effective," which can often be achieved with straightforward computational reasoning. Nowadays, however, researchers aim at making the behavior of virtual agents more "natural" in their interaction with humans, which requires contributions not only from computer science, but also from psychology and social sciences, and which requires a wide variety of techniques, such as player modeling, adaptation, reasoning, and computational linguistics.

To move the research field forward, it is therefore of crucial importance to facilitate the integration of the disciplines and techniques that are involved in this research. The various strands, methodological approaches, and research directions need to inform each other and collaborate, to achieve a whole that is more than the sum of its parts. The goal of the second *Dagstuhl Seminar on Computational and Artificial Intelligence in Games* was to explicitly take the first steps along this path of integration, and investigate which topics and techniques would benefit most from collaboration, how collaboration could be shaped, and which new research questions may potentially be answered.

The seminar was held between January 25 and January 30, 2015. To stimulate interaction between the participants, which is essential in this context, the seminar was structured around workgroups rather than presentations. The organizers started the seminar on Monday morning with a series of brief presentations on potential discussion topics, after which the participants formed their own workgroups around a variety of topics, not necessarily those brought up by the organizers. Workgroups typically consisted of 3 to 10 people from different backgrounds, who worked together for no more than one day. At regular intervals workgroups reported on their findings in a plenary session, after which new workgroups were formed.

At the start of the seminar it was announced that Thursday would be set aside for practical work. Participants could use that day to implement some of the ideas that had come up in the previous days, in the form of a game, a competition, a design document, or a research proposal. While the organizers deliberately gave the participants the option to simply continue with the workgroups if they so wished, all participants actually got involved in the practical work, some of them even working on multiple projects in parallel.

The results of the workgroups and the practical sessions are briefly related in the remainder of these proceedings. The 13 abstracts on workgroups cover automated and AI-based game

design; game analytics; interdisciplinary research methods; design of believable characters; general video game playing; creativity facet orchestration; methods and formal design for procedural content generation; design of "fun" gameplaying bots; communication on game AI research, computers that play like humans; and neural networks for games. The 11 abstracts on practical sessions cover the Planet Wars competition; the automatic generation of games, mazes, and text; Twitter bots; sonification of character reasoning; MCTS and representation learning for procedural content generation; two AI-based games; and the design for a board game.

A special issue of the *IEEE Transactions on Computational and Artificial Intelligence in Games* will be published on the topic of this Dagstuhl Seminar. While this issue is open for submission for any researcher in this field, it is expected that several of the workgroups of the seminar will submit papers on their results.

As organizers, we knew that the first seminar that we organized in 2012 was considered a great success, and we had expected more people to accept our invitations for this second seminar than for the previous one. However, demand for attending the seminar was even greater than we expected. Almost everyone we first invited immediately accepted our invitation. Moreover, everybody who accepted their invitation indeed showed up at the seminar. We were forced by capacity concerns to not invite many people who, by their strength of contribution in the field, should have been present. We are certain that we could easily have doubled the number of participants visiting the seminar, and that each of those participants would have made a strong contribution.

The value of these Dagstuhl Seminars is indisputable. Considering the large number of researchers that should be invited to a seminar that attempts to cover the whole, very broad research field of Computational and Artificial Intelligence in Games, we believe that it is wise for a future seminar to narrow down the topic, so that it can be restricted to a smaller number of participants that are active in the selected subfield. Naturally, considering the fact that "integration" is such an important issue in the research field, care must be taken to ensure that every discipline interested in and involved in the subfield is represented.

2 Table of Contents

3 Workgroups

3.1 Bold Automatic Game Design

Dan Ashlock (University of Guelph, CA)

Joint work of Ashlock, Daniel; Colton, Simon; Eladhari, Mirjam; Van Kreveld, Marc; Lanzi, Pier Luca; Sipper Moshe

"Bold Automatic Game Design" was the title of a session at Dagstuhl Seminar 15051 intended to explore farther reaching and more daring approaches to automatic game design. This abstract summarizes the recommendations and debating points located by the participants. These include downloading some of the design to the players, incorporating environmental information into games, the need for agents to behave irrationally, the need for difficulty estimation and adjustment with a game, the value of explicit versus implicit tutorial levels within a game, and regularizing the presentation of automatically generated content for automatic game design.

Automatic Game Design. This session identified a number of issues, some contentious and some consensus, in the area of automatic game design. Automatic game design is any process that automates some or all of the process of producing a new game. At its weakest is consists in a model for assembling available game elements into a particular instance of a particular type of game. At its strongest it represents a version of solving the strong AI software in which a software system comes up the the mechanics and details of a game and writes the code to implement it.

Leveraging Players as Data. The group felt that the ability to incorporate and leverage the behavior of players was a capability that should be explicitly left into, or at least left in as an option, for automatically designed games. This can take many forms from player modeling intended to match the player with an enjoyable game experience to using player behavior and actions to trigger subsequent branches of the game design process.

Location Based Games. For mobile devices with GPS capability there is potential for incorporating information about the player's current location into games. Since the supply of locations is quite large, this in turn makes games incorporating location information natural targets for partially or completely automatic game design.

Irrational Agents. There are many definitions of "rational" behavior from survival to the accumulation of treasure. Agents that behave according to a single notion of rationality are, potentially, predictable. Agents that change their notion of rationality or, in fact, sometimes act irrationally or from obscure motives can enhance the game experience by reducing the degree to which game agents and automata become predictable.

Automatic Difficulty Adjustment. The problem of adjusting game difficulty to a player's skill level is a current, active research topic outside of automatic game design. The group believes that best-practice dynamic difficulty techniques should be part of any automatic game design system.

Tutorial Levels: explicit or implicit. It is sometimes practical to leave players to work out game mechanics for themselves. At the other extreme there are games that come with novel length instruction booklets or files. Between these extremes lies the in game tutorial. The group felt levels designed to introduce and school a player in a given game mechanic were

superior to more explicit techniques like text screens or pop-ups. We thus recommend that levels introducing game mechanics and, when distinct game mechanics interact, introducing them as a group be something that an automatic game design system can produce.

Snap Games. The idea of small simple games that vanish if not played within a given time limit, in analogy to SnapChat, might be an interesting venue for automatic game design. The impermanence of such games almost demands automatic generation and such games could reasonably be small enough not to be too challenging as a test environment for automatic game design.

Element Libraries or Generators. There is a great deal of current research on automatic content generation (ACG). Any automatic game design system should have access to and incorporate appropriate ACG material. This suggests that VGL or some similar standard should be one of the possible output methods for ACG software.

Leveraging Mathematical Objects. This subject was somewhat contentious. Vector spaces, Cayley graph based on finite groups, and other mathematical objects can be used to generate level maps or terrain maps with complex but controllable properties. Some members of the group felt that understanding advanced mathematics would be too great an effort for game designers and that such techniques were complex enough to defy encapsulation by experts. Since this is a debate subject to resolution by research, some members of the group have undertaken to investigate the issue and publish the results.

3.2 Analytics and Game AI

Christian Bauckhage (Fraunhofer IAIS – St. Augustin, DE)

Joint work of Bauckhage, Christian; Bouzy, Bruno; Buro, Michael, Cowling, Peter; Kendall, Graham; Lanzi, Pier Luca; Lucas, Simon; Samothrakis, Spyridon; Schaul, Tom; Spronck, Pieter; Winands, Mark

The goal of this session was to fathom how or in how far modern data science can be integrated into game design and game development so as to assist in these processes. A particular focus was put on analytics for game AI and the session attempted to structure the landscape of ideas in this area.

Correspondingly, participants set out to discuss what analytics for game AI could be, why it would be helpful, and how it can be accomplished? Key points that were identified as general paradigms or research directions for game analytics included learning from human game play, learning from self play, i.e., reinforcement learning like approaches for AI algorithms, analyzing game results, e.g., game tree structures, and the problem of understanding player behaviors.

Regarding the question as to why corresponding research would be beneficial for game AI programming, participants agreed that it could help to improve player experiences, automatize game bot programming, automatize game design, develop actionable metrics for game analysis, and balance games.

In order to accomplish these goals, methods to extract direct and indirect features would be needed which in turn would allow for the application analytic methods. In this context, the term direct features refers to measurements or observations such as durations of games, actions taken by players, or information available during play. Indirect or derived features, on the other hand, refer to measures of, say, skill depth, frustration, addictiveness, or even

game aesthetics. Once a game has been played by one or many players and features have become available, simple statistics as well as more elaborate methods could be used to look for patterns in the data. Examples of more elaborate techniques that were reviewed in detail included ELO-like measures, cluster analysis, game tree analysis, A/B testing, and neural networks.

Participants then discussed industrial and academic objectives for game analytics. With respect to industrial interests in game data mining, they identified the goals of maximizing revenue, maximizing retention, maximizing the number of premium players in freemium games, maximizing player experience (e.g., through balancing and content adaptation) as well as the goal of automatically maximizing the quality of software. Academic objectives, on the other hand, appeared to be more closely related to basic AI research and included progressing towards human-like AI through behavior analysis, automatizing map layouts and content generation, as well as the question of how to "gamify" scientific problems so that games could be used more easily in genetics research (protein folding) or in simulations for financial and social research.

In the second part of the session, participant decided to put these ideas into practice. To this end, the following games were chosen to guide the discussion in smaller working groups:

- "planet wars", a simple RTS game of perfect information
- "resistance", a multi-player card game of imperfect information
- "retry", a simple platform game
- "skiddy", a simple puzzle game.

For each of these games, the group identified direct and indirect features as well as methods that would allow for their analysis. In particular, the game "planet wars" was then considered for further practical studies. Participants set out to program simple rule-based as well as more advanced neural network and Monte Carlo tree search (MCTS) based game bots for this game whose parameterizations were informed by or learned from game logs. In addition, practical experiments in behavior modeling and analysis were carried out where behaviors were represented in terms of sequences of prototypical actions taken by either human players or game bots. In preliminary results it was found, that even simple learning-based approaches, i.e., an ensemble of linear perceptrons, lead to more engaging game bots and that even simple behavioral models are capable of distinguishing styles of game play and can thus provide hints for the dynamic adaptation of bot behavior. Training of a deep neural network for "planet wars" and the implementation of an MCTS bot were begun during the seminar but could not yet be evaluated; however, results are expected to be available and published soon.

In conclusion, the practical work done in this session demonstrated in an exemplary manner that the integration of game analytics into game development can indeed contribute to the general goals and objectives identified above. This confirms that the idea of game analytics is an auspicious direction for future research and, in the concluding session of the seminar, the participants of the Analytics and Game AI working group could raise awareness for this emerging topic among the other participants of the seminar.

3.3 Interdisciplinary Research Methods

Ian Horswill (Northwestern University – Evanston, US)

Joint work of Horswill, Ian; Mateas, Michael; McCoy, Josh; Paiva, Ana; Rudolph, Günther; Smith, Gillian; Young, Michael; Zhu, Jichen

Game AI research is inherently interdisciplinary, involving not only computer science, but also design, psychology, sociology, graphic design, theater, filmmaking, creative writing, and a host of other areas. A system can be successful from a technical computer science standpoint, while still being an abject failure as a piece of art or entertainment. There is thus an open question as to the appropriate ways to evaluate Game AI research contributions.

While the traditional evaluation methodologies of computer science, including HCI methodologies such as controlled user studies, are undeniably useful, we will likely need to look to other disciplines, particularly in the arts and humanities, for methods for evaluating the aesthetic, thematic, or ideological dimensions of a work.

Evaluation of this kind of interdisciplinary work is complicated by the differing notions of problem adopted by the various disciplines. Engineers, including computer scientists, typically work on well-posed problems – problems with a clear formulation and natural performance metrics. Designers on the other hand, often work on so-called "wicked problems," ones without definitive formulations or metrics, and little ability to generalize solutions across problems. Artists may not even conceptualize their work as solving a problem, but merely as exploring an interesting region of design space to see what they encounter within it.

Another complication lies in the different intellectual and social purposes of evaluation. Although evaluation sections of papers are often written as if answering the question of how well a system worked, they are often read by reviewers and other gatekeepers as answering the question of how well the researchers worked, and thus, whether the work should be published. And yet, they might be most usefully written and read as trying to articulate what insights can be derived from the system for the guidance of future system builders.

We can find resources for evaluation from the component disciplines. The use of performance metrics and correctness proofs from engineering; quantitative methods such as survey instruments as well as qualitative methods such as ethnography from the social sciences; artist talks, critique sessions, and juried shows from the fine arts; postmortems and commentary tracks from the game industry; close reading and critical theory from the humanities; symbiotic interaction between critical reflection and algorithmic development from critical technical practice; and the use of the case study, common in areas such as medicine, law, and business.

However, we don't currently have good evaluation methods for characterizing many of the issues that come about in game AI, and acknowledge the need to create new evaluation methods. How do we evaluate architectures as opposed to systems? How do we evaluate generation systems in PCG? Or the authorial effort involved in using a particular system?

Understanding these issues is important to the future development of the field. We look forward to a rich dialog on evaluation, both within the field of game AI and in conversation with sibling disciplines from which it can learn.

3.4 GVGP Competition Revisited

John M. Levine (University of Strathclyde, GB)

Joint work of Ashlock, Dan; Levine, John; Ontañón, Santiago; Thawonmas, Ruck; Thompson, Tommy

During this Dagstuhl Seminar session, we analyzed the current status of the General Video Game Playing (GVGP) competition, discussed its potential biases toward some families of techniques, and produced a list of suggestions for future directions of the competition.

Throughout the history of artificial intelligence, competitions and challenge tasks have often driven a significant amount of the research being done in the field. Publicly available and open competition domains allow for: the consolidation of a relevant research problem, fair comparison of rival techniques, increase the visibility of research, attract new researchers to the field and are often incubators of new techniques. The GVGP competition is an effort to incubate general AI techniques for video games, and move away from game-specific competitions, which might result in solutions that are too tailored to specific domains.

Specifically, the GVGP competition provides a general framework where 2D, tile-based, single-agent, full-information video games can be specified via the Video Game Description Language (VGDL). A forward model is provided, with which entries can "simulate" sequences of actions in the world. The goal of the competition is to create algorithms that can play any video game defined using the VGDL. Simplified versions of a collection of classic video games are provided as "training" games, for researchers to test their algorithms in, and a hidden collection of "testing" games is used during the competition to compare the algorithms submitted.

The most common family of algorithms in this competition is game-tree search, with UCT being the algorithm underlying most submissions. Given this fact, the inclusion of a forward model in the competition seems to favor this types of algorithm. Other families of algorithms struggle in this setting: for example, reinforcement learning struggles with the large state space of these games and the fact that it needs to be trained for each new game; supervised learning cannot be applied in the absence of training data; heuristic search suffers sue to the difficulty of devising general heuristics for a large class of games; and evolutionary algorithms suffer from the lack of a training period.

Given the initial goals of the GVGP competition, we identified a set of lines for consideration in future editions of the competition:

- Forward model vs no-forward model tracks: the inclusion of a forward model favors game- tree techniques. Moreover, assuming the existence of such model is too strong an assumption in general, and thus, would prevent the competition from spurring development of general algorithms that can play new games for which such model is not available.
- Inclusion of training time: allowing an initial training phase would enable a larger diversity of algorithms to be applicable.
- Partial observability track: most real video games are partially observable, and thus algorithms that can work under such assumption could be encouraged with such a track.
- Adversarial track: all games in the GVGP are currently single player. However, multi player games introduce unique challenges that could result in interesting algorithmic developments.
- Team-coordination track: another interesting possibility would be to include a track containing team games where more than one agent need to collaborate together to play a game against another team of agents.

Finally, it is worth mentioning that the organizers acknowledged at the seminar that that some of these issues would be addressed. In fact, it has been recently announced that at the CIG 2015 competition, the organizers will be introducing two new tracks: Learning Track and Procedural Content Generation Track. Although their information has not yet been revealed at the time of writing this report, the former track apparently will allow a training phase.

3.5 Creativity Facet Orchestration: the Whys and the Hows

Antonios Liapis (IT University of Copenhagen, DK)

Joint work of Bidarra, Rafael; Liapis, Antonios; Nelson, Mark; Preuss, Mike; Yannakakis, Georgios

Creativity facet orchestration aims to combine generation across the multiple creative domains that comprise game design. Literature identifies six core facets existent in games: level design, game design, audio, visuals, narrative and gameplay (involving NPCs or not) [1]. The first two facets are *necessary* for a game to be instantiated whereas the remaining four are *optional*. While there have been a few attempts to integrate more than one facet during the generative process (e.g. Game-o-Matic [2], Angelina [3]) these have been limited to mere hierarchical (linear) procedures. It is only very recently that research on computational game creativity has focused on ways in which more that two facets are interweaved during their generation (such as the work of Hoover et al. [4] orchestrating visuals, audio and gameplay elements).

We argue that to generate novel and valuable games that lie on unexplored regions of the game design space, an *orchestration* approach is needed to automate game generation in a truly integrated manner. We view this approach as an *iterative refining process* particularly suited for the **generation of playable prototypes** for designers to consider and get inspired from. Orchestration requires that the human designer specifies the desired semantics for a query within a (large but manageable) space of possible games. For example, a designer might request a horror game (directly affecting the mechanics of the game), with open-space-style levels (affecting level design), with a warm ambiance (affecting visuals), relaxing music (affecting audio), linear narrative and aggressive NPCs. The generative system blends available concepts (using e.g. ConceptNet [5]) with the aim to deviate from the query in the game design space (e.g. this process could involve searching for novel games from semantically annotated databases of existing games). At this point each facet operates independently, constrained by the semantic information the designer has provided. The facet-specific generator operates using the semantics of all generators, thus providing a high-level context to guide its generative processes. The result is a set of prototypical games with unconventional combinations of facets' outputs, all matching the same underlying semantics. Those games are then presented for designer consideration, along with information about their distance (dissimilarity across several dimensions) to typical games, in order to choose which are to be refined. Refinement tailors parameters of the game space and polishes facets such as visuals and audio.

References

1 Antonios Liapis, Georgios N. Yannakakis, Julian Togelius. *Computational Game Creativity*. In Proceedings of the Fifth International Conference on Computational Creativity, 2014.

2 Michael Treanor, Bryan Blackford, Michael Mateas, Ian Bogost. *Game-o-matic: Generating videogames that represent ideas*. In Procedural Content Generation Workshop at the Foundations of Digital Games Conference, 2012.
3 Michael Cook, Simon Colton, Azalea Raad, Jeremy Gow. *Mechanic miner: Reflection-driven game mechanic discovery and level design*. In Proceedings of Applications of Evolutionary Computation, volume 7835, LNCS, 284–293, 2013.
4 Amy K. Hoover, William Cachia, Antonios Liapis, Georgios N. Yannakakis. *AudioInSpace: A Proof-of-Concept Exploring the Creative Fusion of Generative Audio, Visuals and Gameplay*. In Proceedings of Evolutionary and Biologically Inspired Music, Sound, Art and Design (EvoMusArt), 2015.
5 Hugo Liu, Push Singh. *ConceptNet – A Practical Commonsense Reasoning Tool-Kit*. BT Technology Journal 22, 4, 211–226, 2004.

3.6 Believable Characters

Brian Magerko (Georgia Tech – Atlanta, Georgia, US)

Joint work of Champandard, Alex; Eladhari, Mirjam; Horswill, Ian; Magerko, Brian; McCoy, Josh; Spronck, Pieter; Treanor, Mike; Young, Michael; Zhu, Jichen

The goal of believable characters in computer games (and other media) is to achieve dramatic believability. In other words, "audiences do not pay for reality." This task group considered two views of the problem of creating dramatically believable characters, as has been considered in film, theater, and academic games research in decades prior: the requirements for "believable behavior" and the outstanding issues in computational architectures for creating said behavior.

As a driving example, we considered a specific dramatic scene of "asking someone out at a bar" and what perceivable individual and social cues were related to the dramatic content of the scene. These externally visible cues included: gaze, feeling, posture, proxemics, dialogue, and action selection & execution. The internal drives that could therefore enable these cues in the scene are functions like: anxiety, a theory of mind, a concept of social contract & cultural norms, character histories, and idiomatics.

The concept of procedural idiomatics seemed to be an avenue of particular interest. Within a system that explores the "asking out on a date" scene (or, conversely, the "Little Red Riding Hood" story that has been a prevalent example story in interactive narrative systems), one could consider automatically reasoning about narrative discourse, genre conventions and tropes, character archetypes, and character status as a means of exploring the scene in dramatically believable fashion with highly different dramatic results.

In terms of architectures, there are a number of important outstanding issues. One important issue is the mid-level consistency problem: arbitrating between high-level plans and low-level action selection so as to avoid trashing behavior. The authorial consistency problem is a related issue. While many character architectures can produce consistent and intelligent behavior within a given character, and high level narrative sequencers such as beat systems and drama managers can coordinate long-term narrative consistency, coordinating between the two systems in a way that produces sensible behavior is difficult. Finally, there is a problem with achieving thematic consistency: preventing low-level systems from choosing locally rational dramatically inappropriate action, such as a Sims character washing the dishes immediately after the death of a loved one.

3.7 ASP versus EAs: What Are We Really Searching For in PCG?

Adam M. Smith (University of Washington – Seattle, US)

Joint work of Nelson, Mark; Mateas, Michael; Smith, Adam; Stanley, Kenneth

Answer-set programming (ASP) and evolutionary algorithms (EAs) have emerged as powerful building blocks for search-intensive procedural content generation (PCG) systems. The two traditions (each with a rich history outside of PCG) imply very different problem formulations and, subsequently, very different algorithmic approaches. With the intent to provide guidance to the PCG research community, a working group with representative experts from both traditions was assembled to assess the alternatives. The surprising outcome was that, despite the revelation that neither tradition was particularly well aligned with the needs of PCG systems, both traditions offered oft-undiscussed variations that mapped well to PCG needs.

In the discussion, ASP stood as the representative for a broad class of approaches based on applying exhaustive search to constraint satisfaction problems (CSPs). In such formulations, the goal is to find any solution that violates zero constraints or terminate with a proof that no solutions exist. Typically, these approaches search choice-by-choice, making incremental elaborations to a partially-defined solution, pruning subspaces of possibilities on the basis declaratively-specified constraints. Despite the strong guarantees that can be made about these algorithms and their outputs when they are run to completion (in finite time), little clues are offered to practitioners about what happens when they are forcibly terminated early (as is sometimes needed in practical systems). Additionally, PCG practitioners often struggle with formulating their design concerns as declarative constraints.

EAs, on the other hand, represented approaches that apply stochastic local search to optimization problems. In these formulations, the goal is to find the best solution possible in an anytime fashion. Typically, those approaches follow a generate-and-test paradigm where complete candidate solutions are repeatedly constructed and then assessed (often to produce a single fitness score). The most promising candidate solutions usually become the prototypes for the next round of candidates. Because these approaches treat the assessment of candidates as a black-box process, PCG practitioners are offered significant flexibility in how they formalize optimization criteria (however these criteria must sometimes be perturbed in order to achieve acceptable search performance). Owing to the anytime formulation, it is difficult to characterize if and when a given type of solution will be produced. Nevertheless, the stream of candidate solutions provides observers with a clear sense of progress.

Understanding the typical properties of these two traditions does not immediately clarify the situation in PCG for two reasons. First, there exists systems which subvert these expectations; there are constraint solvers that use stochastic local search and exhaustive global optimizers. Second, most PCG systems do not simply want the first feasible solution or the best-scoring solution, they seek collections of content artifacts that faithfully represent the solution space. Neither problem formulation directly expresses our interest in producing a diverse archive of high-quality solutions (from which solutions can be later selected on-the-fly without re-running the resource-intensive search process).

In *Automatically Categorizing Procedurally Generated Content for Collecting Games*, Risi et al. [1] described how they used a self-organizing map (SOM) to produce a visually intuitive, finite categorization of the space of generated flowers in the *Petalz* game. Their most effective SOM grouped content according to emergent (phenotypic) features rather than defining (genotypic) features, and supported a novel game mechanic where players

used an open-ended content generator (based on EAs) to incrementally explore the finite categorization.

In *Automatic Game Progression Design through Analysis of Solution Features*, Butler et al. [2] generated "a large and diverse database of puzzles" for *Refraction* (using ASP) by grouping those puzzles according to properties of player- constructed solutions those puzzles implied (an emergent property). The creation of this diverse database directly powered their online progression generation system which systematically introduced the player to every combination of concepts that were present in the database (allowing progression design to be reshaped by puzzle generation concerns).

Is the idea of building diverse archives of high-quality content where solutions are distinguished by their emergent properties somehow foreign to ASP and EAs? Apparently not. Within the realm of EAs, "archive-based" algorithms (such as AMGA [3]) explicitly build and maintain such archives, returning the whole archive as their final result rather than just the best individual in the working population. For ASP, the "projected enumeration" feature in some solvers [4] allows users to explicitly request the generation of an example from every one of the possible classes of solutions, where classes are defined by some subset (projection) of the solution features. In the *Refraction* puzzle generator, projected enumeration was used to ensure each puzzle had a distinct laser graph (implying the puzzles differed by more than a trivial spatial adjustment).

References

1 Risi, S. and Lehman, J. and D'Ambrosio, D. B. and Stanley, K. O. *Automatically Categorizing Procedurally Generated Content for Collecting Games*. In: Proc. of the Workshop on Procedural Content Generation in games (PCG) at the 9th Intl. Conf. on the Foundations of Digital Games (FDG-2014). 2014.

2 Butler, E. and Andersen, E. and Smith, A. M. and Gulwani, S. and Popović, Z. *Automatic Game Progression Design through Analysis of Solution Features*. In: Proc. of the SIGCHI Conf. on Human Factors in Computing (CHI'2015). 2015.

3 Tiwari, S. and Koch, P. and Fadel, G. and Deb, K. *AMGA: An Archive-based Micro Genetic Algorithm for Multi-objective Optimization*. Cybernetics, IEEE Transactions on, volume 45 issue 1, pp. 40–52. Jan. 2015.

4 Gebser, M. and Kaufmann, B. and Schaub, T. *Solution Enumeration for Projected Boolean Search Problems*. In: Proc. of the 6th Int'l Conf. on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'09), pp. 71–86. 2009.

3.8 Fun Resistance Bots

Pieter Spronck (Tilburg University, NL)

Joint work of Champandard, Alex; Cowling, Peter; Spronck, Pieter

During the 2012 Dagstuhl Seminar on Computational and Artificial Intelligence in Games, a workgroup focused on intelligence in modern board and card games. The work of this group culminated in a one-day programming jam session, in which several participants created intelligences for the game "The Resistance."

"The Resistance" (http://www.indieboardsandcards.com/resistance.php) is a game in which each player is either a Spy or a Resistance fighter. The roles are secret but the Spies know each other. One player is the leader (a role which rotates between the players

continuously), who has the responsibility to propose a team of a specific size to go on a mission. Everybody gets to vote on the team, and if the majority agrees with the leader's selection, the team gets to execute the mission. During the mission, the members of the team each play a card that indicates whether they support or sabotage the mission. These cards are played in secret, shuffled, and revealed. If one of the cards says that the mission is sabotaged, the mission fails. Otherwise it succeeds. As soon as three missions have succeeded, the Resistance fighters win. If, however, before that time the Spies manage to sabotage three missions, the Spies win. The Spies also win if five proposed teams are rejected in sequence.

The bots developed for "The Resistance" in 2012, and in the two years after that, have all been focused on playing the game as strongly as possible. This means that they try to make a solid estimate of which player is on which team, and of how opponent players play the game. The majority of the bots use expert-systems backed by statistical analyses. The setup of the competitions was aimed at allowing players to build opponent models if they wanted, with matches consisting of tens of thousands of games, and all communication between players happening via method calls. However, the original implementation of the framework encouraged bots to use a learning-centric based approach rather than a search-based approach. A game state that can easily be searched exhaustively was added before the meeting in Dagstuhl.

For the 2015 seminar, we aimed to investigate whether Resistance bots could be developed that are interesting or entertaining for human players to play against, rather than "just" effective. Effective bots were deemed to be a relatively straightforward problem to solve given time, but the Dagstuhl environment seemed better suited to building bots with more creative behaviors. To create such bots, three requirements must be met:

1. The bots must be able to play a reasonable game from the onset, and not need dozens of rounds of training, because when playing against humans there is no time for extensive training. Existing bots were refactored into easily extensible base classes for this purpose, e.g. Bounder and Invalidator.
2. The bots must be able to express their opinions on the game and on the other players, to allow for in-game discussion (which is an essential game element when humans play). This often required storing the information in a different format than when used purely for optimal play.
3. The bots must be able to communicate in human-understandable form.

The Resistance engine used for competitions has been expanded with the the ability for the bots to express messages on three levels:

1. Game moves: e.g., "I vote against the proposed team."
2. Structured opinions: e.g., "I estimate the probability that player 3 is a spy at 37.5%."
3. Free-form text: e.g., "That'll teach ya, ya scurvy dogs!"

Several bots have been implemented which make use of these features. The engine has also been connected to a chat system, so that mixed human/bot groups can play, using the chat-system features, and using messages for their communication. Also, a text-to-speech engine via a Web API and the built-in speech-to-text were added as features as well. With such a setup, we hope to investigate the following questions:

- Will the extra communication abilities change bot behavior at all? This is not immediately clear. If the bots do not get stronger by using the extra features, then increased playing strength provides no reason to use them, and the behavior of bots which aim only to maximise their win rate will remain unchanged. Ideally the artificial intelligence must be able to exploit the extra information to increase its win-rate. Experimentation will have to show whether or not that is possible,

- Will the extra communication abilities make the bots more fun to play against as a human? Naturally, as long as the bot is new, players might like occasional banter from the side of a bot, but if such banter is repetitive or not related to the game situation, it will not take long before it gets boring or even annoying. Experiments can be run to test out different bot implementations against humans in a webchat-based environment, and query the humans on their enjoyment of the bots.
- Can bot personalities help in creating more interesting bots? Once a good bot has been created with a strong communication system, it might be possible to create variants of it using a personalization system, which slightly colors behaviors and messages, for example to tailor the "fun" aspects of a bots behaviour to the moves and messages of the other players.
- Can bots be created that have human-reasonable opinions of whether the other players behaviour is "fun"?

The intention was to run an initial competition at the 2015 seminar, but different projects took precedence since this topic was already addressed at the previous Dagstuhl Seminar on AI/CI in games. However, some prototypes based on language-based interaction proved very successful, and further follow-up work has been scheduled for later in 2015.

3.9 AI Game Research

Kenneth O. Stanley (University of Central Florida – Orlando, US)

The website http://www.aigameresearch.org was conceived at the original Dagstuhl session on Artificial and Computational Intelligence in Games in 2012. Several researchers subsequently worked together to make the website a reality. The idea behind the site is to provide a focal point where video games that are based on artificial intelligence or computational intelligence research can be showcased to the general public. By providing such a venue, aigameresearch.org makes it possible for researchers to reach a broader audience with their games than would otherwise be possible. Furthermore, the more games the site showcases, the more interest it attracts. In that way, it amplifies the creative energy of the community by allowing all of us to benefit from each other's work. Each game submitted to the site is reviewed by an Editorial Board (http://www.aigameresearch.org/editorial-board/) in a manner similar to a journal. The current editor-in-chief is Kenneth Stanley of the University of Central Florida.

3.10 Algorithms That Learn To Play Like People

Julian Togelius (New York University, US)

Joint work of Paiva, Ana; Samothrakis, Spyridon; Schaul, Tom; Shaker, Noor; Sipper, Moshe; Togelius, Julian; Yannakakis, Georgios; Zambetta, Fabio

This workgroup was concerned with algorithms for learning to play games like humans. Learning to play a game like a human goes over and above learning to play a game in general, and has potential applications in game recommendations and personalization and content

evaluation for procedural content generation. It is also an interesting scientific question in its own right.

We assumed that different people have different game playing skills, just like the have different cognitive profiles in general. Some people have fast reactions, others are good at assessing large quantities of information, planning far ahead in the future, or maybe spatial reasoning. There is likely to be a player skill/performance profile that to some extent carries over between games.

With learning to pay like a human, one could mean several different things. We identified at least the following ways: mimicking the style of a particular player, mimicking the performance profile of a player, or mimicking the way a player learns. To illustrate the latter concept, an algorithm that learns to play Super Mario Bros while mimicking human learning style should learn skills in the same order: if the human learns to jump over gaps first and then learns to pick up power-ups, the algorithm should learn to learn to do this in the in the same order.

To further analyze the question, the various ways in which an algorithm could learn to play like a human could be described according to three axes: (1) Data which we are learning from: game profiles (e.g. from Steam accounts), human play-traces or interactively; (2) The resolution at which the model operates, from a single human to a player type to an average human; (3) the goal of the learning, i.e. predicting performance, predicting learning, mimicking learning, mimicking performance or mimicking style. The group agreed that predicting performance for classes of people based on interaction and human play traces is almost certainly doable with extensions of existing algorithm. Mimicking style (across games) is almost certainly doable. Mimicking learning is thought to be doable, but hard.

Finally, a proposal for a research project for predicting performance and predicting learning was sketched out. This included recording multiple humans' performance on multiple games, and using singular value decomposition to learn performance embeddings of humans. This would essentially identify a number of principal components of skill. It was proposed that the General Video Game Playing framework (based on the Video Game Description Language which was largely designed in the previous Dagstuhl Seminar) could be used for this.

3.11 AI-Based Game Design

Michael Treanor (American University – Washington, US)

Joint work of Cook, Michael; Eladhari, Mirjam; Levine, John; Magerko, Brian; Smith, Adam; Smith, Gillian; Thompson, Tommy; Togelius, Julian; Treanor, Michael; Zook, Alex

In our working group, we created a model for designing games around Artificial Intelligence (AI). AI-based games put AI in the foreground of the player experience rather than in supporting roles as is often the case in many commercial games. To develop the model we investigated the use of AI in a number of existing games and generalized our findings into several design patterns for how AI can be used in games. From there, we created a generative ideation technique where a design pattern is combined with an AI technique or capacity to result in an AI-based game. Next, the technique was put into practice in the creation of two AI-based game prototypes. From this work, a paper was created, submitted and is in review for the upcoming Foundations of Digital Games Conference (2015).

3.12 Neural Networks for Video Game AI

Mark Winands (Maastricht University, NL)

Joint work of Bauckhage, Christian; Bouzy, Bruno; Buro, Michael; Champandard, Alex; Cowling, Peter; Kendall, Graham; Lucas, Simon; Samothrakis, Spyridon; Schaul, Tom; Winands, Mark

Recently, deep convolutional neural networks (DNNs) have made huge advances in non-search based Go engines, playing a strong game without any lookahead, easily defeating GNU Go (a traditional search-based Go program). Though they are not defeating leading Monte Carlo Tree Search (MCTS)-based programs, they are performing respectably against them.

The goal of this work group was to further investigate how these Deep NNs can be applied for Game AI. In particular the work group was interested of its application to Real-Time Strategy (RTS) games. As a test case the RTS game Planet Wars was chosen, which was the Google AI Challenge of 2010. An advantage of this game is that its rules are straightforward, but the game possesses quite an amount of strategic depth. Also many bots are available to serve as a benchmark.

As an additional challenge is that the DNNs should be able to handle variable input sizes (e.g., variable planets) and to handle the game dynamics (e.g., generalize from different number of planets). Two solutions ideas for the architecture were discussed.

The first idea was a designed feature set. It contained concepts such as the number of ships, total growth, growth of the source / destination planet, etc. The core idea is a pairwise treatment of planets. An action consists of sending ships from a source planet to a destination planet. All possible actions are evaluated by the NN, and the one with highest score is chosen.

The second idea was a DNN that takes as input a session of frames. It would extend the architecture to image analysis. The game is encoded in terms of a stack of matrices so that convolutions would be applied.

For integration, optimal ways of combining deep learning with MCTS were discussed. As tree policy the DNNs could be applied only in promising nodes (i.e., which have been visited frequently). Another option is to use it for progressive widening (i.e., only to investigate interesting actions). For the rollout policy, the DNNs serve as action selection. There is though a tradeoff between the power of the standard DDN versus the speed of the pairwise treatment.

3.13 Procedural Content Generation and Formal Design

Alex Zook (Georgia Institute of Technology, US)

Joint work of Bidarra, Rafael; Cook, Michael; Liapis, Antonios; Smith, Gillian; Thompson, Tommy; Van Kreveld, Mark; Zook, Alex

With the consolidation of Procedural Content Generation as an academic area of study, there now exists both a need for greater theoretical foundation for design and simultaneously an opportunity to build these theories. We posit that all procedural content generators are themselves encoding a formal theory of game design, in terms of both the process being followed and the products that are being generated. Understanding formal design theories (including their construction, analysis, and evaluation) requires integration of research and

practice across several disciplines: the arts, humanities, design studies, and computer science. Game design and AI stand to benefit greatly from the varied perspectives on content generation by systematizing, generalizing, and deepening existing knowledge, while also broadening the range of topics addressed through procedural content generation.

Generators build upon design theories both explicitly and implicitly. Explicit models result from deliberate choices of a system designer to encode a form of the design knowledge for a domain (e.g., mazes) or a design process (e.g., the Mechanics, Dynamics, and Aesthetic framework). Implicit models are encoded within unacknowledged commitments expressed through algorithmic details (e.g., data structures, representation, generative methods) that may require critical analysis to be uncovered. Uncovering and acknowledging both the explicitly and implicitly encoded theories about the design process is key to learning from generators. Comparing them to each other and generalizing lessons learned from individual systems will lead to a broader, more inclusive, formal theory of game design. By examining the gap between generated products and exemplars made by human designers, it is possible to better understand the nature of the artifact being procedurally designed, thus building a formal theory of the artifacts being designed as well as the process taken to design them.

There is therefore value in understanding the design theory behind generators in terms of their goals, metaphors, methods, and ethics. Such a conscious commitment to an epistemological view on formal design theories in generators can lead to a better understanding of the generative process and the generated products. Formal approaches to design theories can support the analysis, interpretation and dissemination of PCG as an academic field and integrate practitioners including artists, designers, players, programmers, and researchers. For generative systems where the goal is primarily to produce a particular kind of playable experience, design theories allow artists to communicate their aesthetics, ethics, or message and reflect on it (in conjunction with the responses of a wider audience). For generative systems where the goal is primarily the system itself, design theories allow the creators of the system to fine-tune the algorithms as well as challenge current conventions.

In this working group we discussed motivations, high-level research topics, and initial research projects at the intersection of procedural content generation and formal design theory.

References

1 Shaker, N. and Togelius, J. and Nelson, M. J. *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer, 2015

2 Togelius, J. and Yannakakis, G. N. and Stanley, K. O. and Browne, C. *Search-based Procedural Content Generation: A Taxonomy and Survey*. IEEE Transactions on Computational Intelligence and AI in Games (TCIAIG), volume 3 issue 3, pp. 172-186, 2011.

3 Hunicke, R. and Leblanc, M. and Zubek, R. *MDA: A formal approach to game design and game research*. AAAI Press, 2004

4 Practical Sessions

4.1 The Dagstuhl Planet Wars Hackathon

Michael Buro (University of Alberta – Edmonton, CA)

Joint work of Bouzy, Bruno; Buro, Michael; Champandard, Alex; Cowling, Peter; Lucas, Simon; Samothrakis, Spyridon; Schaul, Tom; Spronck, Pieter

This abstract describes the Hackathon event that followed up on the Dagstuhl working group on neural networks for real-time game AI, in which neural networks for Planet Wars were developed. Planet Wars was the subject of the Google AI challenge in 2010 (http://planetwars.aichallenge.org/). It was chosen as the application domain for this Hackathon project for its simple rule set, existing programming frameworks, real-time decision complexity, and our curiosity about how state-of-the art AI techniques could be successfully applied to the game by AI experts who only have a day to implement a complete player. After introducing the Planet Wars game, we describe each approach in turn, what worked and what did not, experimental results, and why we as a group feel that this was one of the best workgroup experiences we ever had. We conclude the paper with a discussion of future research directions.

Planet Wars

Planet Wars is a two player zero-sum simultaneous move game in which players aim to destroy opponent's fleets in real-time. Figure 1 shows a screenshot. Circles represent planets with fixed positions and rates at which they generate ships. The numbers displayed inside planets indicate how many ships are currently stationed there. The planet's color indicates ownership (grey = neutral). Moves consists of picking a planet under your control and sending a number of ships to another planet. Ships cannot change course while in transit and do not interact with other ships (another version that was implemented during the workshop considered on-transit fleet collisions, but no AI systems were specifically designed for this variation). Once ships arrive at their target planets their count is either added to the planet's ship count (if friendly), or deducted from the count if the planet is neutral or hostile, in which case whoever has more ships left owns the planet. Neutral planets never send ships or create any ships. A player wins if all opponent's ships are destroyed. In the game version we used, planets and fleets are visible at all times.

The Planet Wars Google AI challenge was won by a bot written in LISP implementing alpha-beta search (http://quotenil.com/Planet-Wars-Post-Mortem.html). We first considered to using Google's server-client framework so that we would be able to compare our bots with the state-of-the-art, but then ended up using a simple Python implementation because we couldn't find Google's server code, and the Python project we found lent itself to quick development in the short time we had for the project. Alex Champandard set up a git repository (https://github.com/alexjc/planetwars) which contains all code that was generated during the Dagstuhl Seminar. The software is based on a Python project (https://github.com/maxbogue/planetwars) which provides game simulation code and a web server for game visualization.

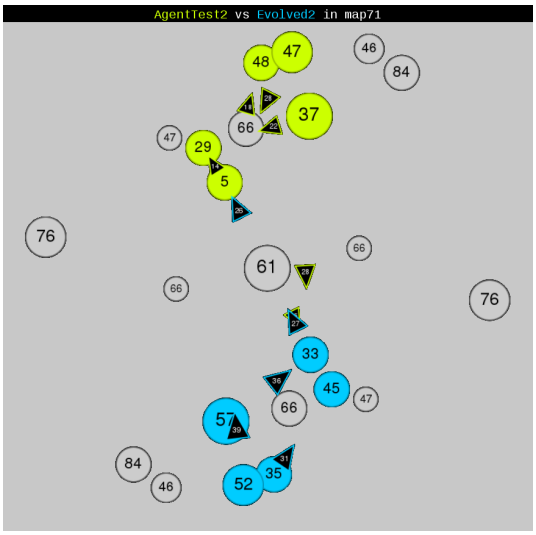

Figure 1 Planet Wars Screenshot.

Our Planet Wars Bots

In this section each group describes their approach for building a Planet Wars bot in 1.5 days using different AI techniques.

Evolutionary Algorithm (Lucas and Buro). We wanted to see how far we could get with a simple but interesting approach: the main idea was to use a simple evolutionary algorithm (EA) to learn a set of weights to play the game. We took the following steps:

1. Define how the AI would interface to the Planet Wars game
2. Define a feature set
3. Implement an initial set of weights for a linear classifier
4. Write a simple evolutionary algorithm in Python, for ease of interfacing to the game engine
5. Run the algorithm, both using random initial weights and hand-tuned initial weights
6. Evaluate the results

Steps 1 and 2 had been sketched out during an earlier session in the week.

For interfacing to the game (step 1), the approach was to consider a single possible action per time step, although the game allowed a list of actions to be executed each step. We defined an action as a planet pair, with the first element listing the source planet (which must be owned by this player) and the second planet being the destination planet, which could be any planet. In the event that the source was the same as the destination, this would have no effect and hence allowed for null moves. Otherwise, the action would be to transfer 50% of the ships on the source planet to the destination planet: a straight transfer in the case the planet was owned by this player, or an attack in the event that it was owned by the opponent.

Step 2: for each possible action, a feature vector was constructed based on the properties of each planet in the pair (for example, the number of ships, the size of the planet), and also on the relationship between them (e.g., the distance between them). Although the features had been sketched out in a previous session, during the hackathon it became clear that some

of them were ill-defined and needed some more work to implement in a meaningful way. Mapping the outlined features to a Python implementation was an interesting and satisfying part of the process.

Step 3: We designed an initial hand-tuned set of weights. There was an obvious intuition to some of them (e.g., give a high rating to choosing a source planet with lots of ships). Ones without an obvious way to set them were set to zero. We then tested this and found (very satisfyingly) that it beat all but the strongest of sample bots.

Step 4: Here's where we discovered just how rusty our Python skills were. Michael had left the EA team to try to improve feature weights manually together with Bruno Bouzy. So, Simon's initial approach was to take an existing Java EA code base and port it to Python: although EAs are quite simple to implement, to get a bug free version running in Python still took a couple of hours. This was tested on some toy problems, and then run to evolve feature weight vectors for the controller described above. With a population size of 10 running for just 100 generations from initial random weights (and even 50 generations) the evolved bot was able to defeat many of the sample bots. A limitation here was that fitness was evaluated by playing against a random bot: the problem is that after some initial evolution the evolved bot beats the random agent every time, and so there is no incentive for it to improve beyond this (there is no search gradient). Co-evolution was also implemented, which can lead to a more open-ended evolutionary process. Reasonable results were obtained with this, but there was insufficient time to see the full extent of what could be learned in this way.

Seeding the population with the hand-tuned weights led to a bot that was able to beat all of the sample bots, and were defeated only by the super hand-tuned bot (more about the actual tournament results later).

The only frustrating part of running the EA was the time taken for each fitness evaluation: each evaluation takes several seconds, and this severely limits what can be evolved on a single machine in a short period of time. This could be ameliorated by using a Java implementation of the game server.

One satisfying aspect of the simple evolutionary approach is that the idea of evolving a linear heuristic function had been roundly mocked earlier in the week as being too simple and therefore of no interest. Yet this turned out to be the leading approach of all the adaptive / learning methods tried during the hackathon. It reinforces the wisdom of starting with a simple method and then building on it.

There are many more experiments that can and should be made along these lines, and it will be interesting to see which approach wins out in the long run, and just how far simple methods can be taken.

"Professor Descent" (Bouzy and Buro). For the Hackathon day we chose to work on Planet Wars. We were considering to implement an MCTS bot together with Mark or Pieter. But given that we only had one day to produce a working player, we didn't know Python very well, and the anticipated playout speed obstacle for Planet Wars, we gave up on the MCTS idea. The previous day, Simon and Michael had written code for feature extraction and designed a "one-neuron bot" called AgentTest for a Python Planet Wars framework. Between lines 20 and 40 of this bot there were a dozen features evaluated and weighted by values that were easy to modify. So, we decided to play around with these values and having fun observing Planet Wars games with our manually tuned one-neuron bot playing against the existing bots all afternoon.

We created **AgentTest2**, a copy of the reference bot. Some features were sensitive to slight weight changes, and others were not. Manually changing the values of the sensitive features made AgentTest2 beat AgentTest most of the time rather quickly. That was great –

our novel manual method worked, and we were ready to give a name: "Professor Descent", a variation on the "Graduate Descent" method which professors use to solve hard research problems by assigning tedious work to a number of graduate students. However, at that point we discovered that while professor descent enabled AgentTest2 to soundly defeat the reference bot, it had lost its ability to beat an existing bot that AgentTest consistently defeated. Not worrying too much, we solved this problem by executing one more iteration of professor descent. Watching a few more games and further tweaking parameters, we came up with a weight set that was able to defeat all bots that shipped with the Python Planet Wars framework and the AgentTest reference bot.

Hackathon for us was the most enjoyable part of this year's Dagstuhl Seminar. We learned how to get root access without knowing the root password on Bruno's university Linux notebook to install Python, we refined our Python programming and Git skills (which were rather rudimentary when we started to be honest – thanks Alex!), enjoyed watching Planet War games – leaning back, sipping coffee, and cheering for AgentTest2, while watching other teams frantically typing, trying to improve their bots. The professors were rather pleased with the results of their "descent" method. Little did they know that another strong competitor was in the making ...

Machine Learning (Champandard, Samothrakis, Schaul). Our goal was to implement a system that learns approximate state-action values using variations of Q-Learning and Monte Carlo methods in the game of Planet Wars. State-action values are approximated using ReLU-type neural networks. The learning process involves fixing opponents and maps and training for a predetermined set of episodes. The process worked reasonably well, but requires extensive hyper-parameter tuning on the following fronts: (a) neural network weight initialisation, (b) neural network learning rates, (c) exploration rates, (d) network topology, (e) input and output scaling, (f) exploration rate decay, (g) discount rates. The number of hyper-parameters that need tuning makes standard neuro-control methods hard to apply out of the box in games and we think this is a vital shortcoming for the widespread adoption of these methods. Possible ways ahead include, but are not limited to, using already existing knowledge to guide exploration (similar to default policy playouts in Monte Carlo Tree Search) and incorporating learning rate adaptation techniques from supervised learning.

Rule-Based AI (Spronck). The **Hotshot** Planet Wars bot takes a parameterized rule-base approach. At every turn, for each planet, it will decide whether or not to launch ships. It has a "max-in-flight" parameter that determines the maximum percentage of the total ships owned that is allowed in flight. This way, it intends to ensure that there are always enough defenses. Moreover, there is a minimum percentage that must always be left behind on a planet, and if the number of ships on the planet is less than that, it will not send ships from the planet.

If it decides to launch ships, it may send them to neutral and/or enemy planets. Its potential targets are the closest, lowest-growth, highest-growth, and "easiest" of the target planets. In this case, "easiest" means easiest to conquer, i.e., the least defended planet (if there are multiple choices, it will take the closest). To determine how many ships are to be sent to each of the chosen targets, parameters are used.

Once the basic rule-base was in place, default values for all the parameters were chosen. Then the Hotshot bot was pitted against several of the default bots, in particular the stronger ones, such as "all to close or weak". It fought on all 100 maps. A hill-climbing mechanism was used to change the parameter values after each 100 skirmishes, in order to increase the number of victories. As soon as 100% victories was reached, another opponent was chosen.

This way, Hotshot tweaked its parameters to get to a stage that it would win against any of the default bots (nearly) 100% of the time. At that point, the parameters were frozen.

Using the parameters learned in this way, Hotshot is a very fast bot, that indeed defeats all the "beginner" bots nearly 100% of the time. Unfortunately, it does not do so well against more advanced bots. When examining the parameters values that it learned, we see some of the reasons why. The parameters show that Hotshot heavily favors attacking enemy planets with low growth, followed by the closest enemy planet, and the easiest-to-conquer neutral planet. This does not seem to be a strong tactic, as low-growth planets, once conquered, do not contribute much to an empire, while leaving high-growth planets in enemy hands gives the enemy an advantage. There is no denying that the strategy works against the beginner bots, but it is no surprise that it is pretty bad against more advanced bots.

It is not hard to implement more "rules" in the bot, and let it use hill-climbing to learn more parameters. The set of rules it uses now are probably insufficient to defeat more advanced bots in any case, regardless of parameter learning. The basic approach is sound, though, and leads to fast bots.

MCTS (Cowling). The **MCTS2** bot uses UCT to make decisions at each iteration. Doing this at 60 decisions per second poses some difficult challenges, which we addressed to some extent via the time-slicing approach of our successful PTSP player (where we use a time slice consisting a many frames and only make a decision a few times per second). However, in the hackathon there was too little development time for this – the main pressure there was on producing something that worked, made sensible decisions, and embodied an idea that could be enhanced later.

The MCTS2 bot builds on Michael Buro's useful Planet Wars class to capture state and issue commands. It uses the Python implementation of UCT which Ed Powley, Daniel Whitehouse and I wrote a couple of years ago – which can be found at http://mcts.ai/code/python.html. Hence the functions that are needed for a basic implementation are simply Clone(), DoMove(), GetMoves() and GetResult(). Here GetResult() returns the player that has won from a terminal state (and otherwise throws an exception). The Clone() function proved tricky – it had to create a changeable version of the world state, then once a move was chosen, it had to refer back to the original world state. Early versions did not really work – essentially the simulations based on random moves would take ages to terminate. The trick that made it all work was to consider only moves where exactly half of the ships on a planet were sent to a planet where they would be able to successfully conquer (based on the ships currently on the planet), and allow only one move per iteration. This way of pruning moves both allowed simulations to terminate in a reasonable number of moves, and actually provided stronger play than all other players available at the time of testing even with a single iteration. With 1000 iterations the player seemed very strong, but a single move took several seconds. In order to run at competition speed 5 iterations per move were chosen (and for very small numbers of iterations play was better with n+1 iterations than with n. 5 iterations was about right – and that was the competition bot. I wasn't there for the final playoffs, but MCTS2 did seem very strong, and stable, in testing. This in spite of Alex twice adding an "MCTS is not an algorithm" exception ☺

The time pressure of the hackathon, and working together under pressure in the same room as a group of like-minded colleagues, made the hackathon one of the highlights of the week.

Table 1 Tournament 1 results after 25 round-robin rounds. At this point MCTS2 choked and had to be stopped after spending 15 minutes on one game:

	Player	total%	0	1	2	3	t-avg (ms)	t-max (ms)
0	MCTS2(5)	57	–	64	36	72	93.2	347.4
1	AgentTest2	55	36	–	56	72	5.0	18.9
2	Hotshot	52	64	44	–	48	0.1	1.4
3	Evolved	36	28	28	52	–	3.5	13.7

Table 2 Tournament 2 results after 100 round-robin rounds (MCTS2 excluded):

	Player	total%	0	1	2	t-avg (ms)	t-max (ms)
0	AgentTest2	65	–	68	62	6.1	56.4
1	Evolved	48	32	–	64	4.2	15.4
2	Hotshot	37	38	36	–	0.1	1.1

Table 3 Tournament 3 results (100 rounds) with Evolved2:

	Player	total%	0	1	2	t-avg (ms)	t-max (ms)
0	Evolved2	83	–	84	83	4.8	15.4
1	AgentTest2	39	16	–	61	4.9	21.7
2	Hotshot	28	17	39	–	0.1	1.1

Table 4 Tournament 4 results with Evolved2 and MCTS(1) after 50 rounds, when MCTS(1) encountered a list index out of range error:

	Player	total%	0	1	2	3	t-avg (ms)	t-max (ms)
0	Evolved2	76	–	61	84	82	4.4	15.2
1	MCTS2(1)	53	39	–	45	76	19.0	149.2
2	Hotshot	38	16	55	–	43	0.1	1.1
3	AgentTest2	33	18	25	57	–	4.5	21.1

Tournament Results

By Thursday night, four bots were ready for the final competition. Later, two more entries were added to study the effects of parameter changes:

- MCTS2(5) uses move pruning and five playouts per move
- AgentTest2 is based on the "one-neuron" linear move evaluator with hand-tuned weights,
- Evolved uses the same feature set, but evolved weights starting with random values and training against built-in bots.
- Hotshot is a fast rule-based bot.
- MCTS2(1) uses one playout per move
- Evolved2 is identical to Evolved, except for the weights which were evolved by starting with the hand-tuned weight set.

Tables 1–3 show the results of the four bots playing round-robin tournaments using the 100 symmetrical maps that come with the Python Planet Wars framework. The time columns indicate that the submitted MCTS player used considerably more time than the other bots. In fact, the first tournament had to be suspended for lack of progress for 15 minutes in a single game caused by MCTS2(5). For this reason, we removed MCTS2(5) and reran the tournament. Among the remaining bots, AgentTest2 ended up winning the main contest, followed by Evolved and Hotshot. We then entered Evolved2 – the evolved player which

started from manually tuned weights, to see how good "professor descent" really is. Not that good, as it turned out, looking at Table 3. Lastly, we re-entered MCTS2, now using just one playout per move, to see how well it plays in a stricter real-time setting. Unfortunately, halfway into the tournament it crashed. At that point Evolve2 which is more than 4 times faster prevailed again.

Conclusion

In this abstract we reported on an exciting Hackathon Dagstuhl event whose goal it was to find out which state-of-the-art AI techniques work best to create AI systems for simple real-time games within 1.5 days.

Sparked by recent advances in deep learning we were optimistic that we could create a strong player based on reinforcement learning. Unfortunately, this effort failed, due to slow game generation. We also witnessed EA running into issues when seeded with random weights. The weight set obtained by starting with our hand-tuned weights performed much better, in fact, winning the final tournament.

There are a lot of venues that can be explored from here. For instance, it would be interesting to see how well MCTS performs when using playout policies based on Evolved2. Comparing our bots with the ones that performed well in the original Planet Wars competition would shed light into how MCTS and learning bots fare against classical minimax based players. Finally, exploring the trade-off between acting fast and acting well in real-time games is worth studying to help us create strong AI systems capable of defeating human expert players.

4.2 What The Hell Is Going On?

Simon Colton (University of London/Goldsmiths, GB)

In the MetaMakers Institute at Falmouth University, we are developing an automated game generator within a Computational Creativity context. On the technical side, this is the first system we have developed from scratch which is intended to automatically alter its own code, in order to increase the yield, variety, and unexpectedness of games, and address some of the higher level issues of autonomy and intentionality in software. On the gaming side, we aim to build a community of players who enjoy and share the games, with the added benefit of sometimes being the first person in the world to solve a game level. On the philosophical side, we are interested in the correspondence between an open conjecture in pure mathematics and an unsolved game level, especially in balancing the joy of solving a previously un-solved game with the futility of attempting to solve a game that might not be solvable, and the strategies of knowing when to give up that this leads to.

At the start of the Dagstuhl Seminar, the first prototype of the game generation system was demonstrated individually to people, and feedback was gratefully received and processed. The system produces puzzle mini-games, where a strategy for placing pieces on board cells has to be devised by the player and then used to solve the game. As both the pieces and the board move around, there is a physical element to each game which increases difficulty. Currently, games can be randomly generated and hand-finished, and ten such games were demonstrated, with the intent of showing that the game space contains difficult puzzles.

Many seminar attendees played the games and found them in part enjoyable and in part frustrating, with some people attempting more than 100 times to solve particular games over a substantial period spanning several glasses of wine. Lessons were learned about player interaction, making the games more like toys to increase the fun, and how people feel about playing a game that might be unsolvable.

During the sessions on whole game generation and through the interaction with seminar attendees, the approach was improved so that the next stage of development could take place during the game-jam session of the seminar. Here, the aim was to show that, by including images in the games, the space includes games which have contextual meaning. To this end, five games with a Dagstuhl theme were produced in a semi-automated way and showcased at the end of the seminar. The games were as follows: "Let it snow" (where the best tactic is just to the let the snow fall); "Too many Michaels, Too many ideas" (where the aim is to enable a researcher to concentrate on one topic at a time); "Fabulous Dan Ashlock" (where the aim is to get to the coffee without being sidetracked by interesting conversations); "Random Lunchtime Shuffle" (where the player must seat all the seminar attendees on the moving tables); and "Tired and Emotional" (where the aim is to herd the researchers to their bed, even though they want to be elsewhere). In the context of the Dagstuhl Seminar, these games had much meaning, and were well received by the seminar attendees.

4.3 Twitter Bot Tutorial

Michael Cook (University of London/Goldsmiths, GB)

In this session participants were invited to set up their workstations for the development of Twitter bots, small programs that run at regular intervals usually on external web servers. Bots normally produce some kind of content when they run, which they post to Twitter either as a regular tweet on the service (a message of 140 characters, possibly including image URLs or other media) or an interaction with another user (a semi-private message directed at one of Twitter's users). Twitter bots are an emerging medium for art and experimentation, and are also being used to design and augment games and related technology. The session was organised to enable more people to engage with the medium, as it has a lot of potential for researchers.

Many games interact directly with Twitter by allowing their users to post information to it such as high scores, and some games such as Hashtag Dungeon use Twitter data to generate content within the game dynamically. In addition to design, Twitter is also a useful platform for experimentation: researchers are already using Twitter as a way of conducting surveys, gaining user feedback, or writing lightweight applications that allow users to submit information (such as location data) with technology they already have on their phones. Because Twitter is restricted to short messages, it's an ideal medium for communicating with simple apps, and Twitter bots are an excellent way to build software that facilitates this.

All participants successfully set up bots, and some used it to connect other code they had written at Dagstuhl to bots which posted to Twitter. We hope to see the fruits of this work in research projects in the future.

4.4 Sonification of Character Reasoning

Ian Horswill (Northwestern University – Evanston, US)

As with any complex program, character AI systems perform long chains of operations through time. And like other programs, understanding the operations of these systems can be difficult. This is an issue not only for debugging but for making the behavioral state of the system transparent to the player.

Take the case of debugging. The real-time nature of games limits the usefulness of typical debugging tools such as breakpointing and single-stepping, so game programmers often resort to exhaustive logging. However, log files can be difficult to work with. A human can look at only a small portion of the log at a time; even with filtering tools, understanding the log can still be a laborious process. Visualization techniques can help make patterns and anomalies visually apparent without requiring the programmer to fully digest the text being represented. But even they require the programmer to interrupt the game, or at least gaze away from it, so as to attend to the visualization.

An interesting alternative is the use of *s*onification: the rendering of logs as sound in time, rather than as text or image in space. The basic approach is simple. As with other techniques, one augments the code to "log" events as they happen. However, rather than logging the events as text in a file, one logs them as sound events in the audio stream generated by the game. This allows the "log" to be produced continually as the game runs, giving feedback to the programmer without having to interrupt execution.

As part of the seminar's hackathon, I added sonification to the experimental AI-based game *MKULTRA*. The sonification used granular sound synthesis [2] where each logged event generated a 1–10 ms "grain" of sound whose spectral characteristics depend on the event being logged. Consideration of a problem-solving operator produces one type of grain; resolving conflicts between competing operators, another; exception handling, another; and consideration of lying to another character, produces yet another type of grain. All grains were synthesized using a simple FM sound synthesizer [1] whose parameters (carrier frequency, modulation frequency, and modulation level) were adjusted based on the type of event being logged.

Although not a replacement for proper text logs, the sonification system can make apparent patterns in temporal behavior such as looping. It can also allow the programmer to notice anomalies such as differences in sound from one run to another, allowing a kind of passive debugging.

More interestingly, however, sonification also introduces the potential for novel gameplay mechanics where players rather than programmers are required to detect anomalies the sonification. In the fiction of *MKULTRA*, the sonification of NPCs is explained in terms of the player character having limited mind-reading capabilities. The player can gradually learn to recognize lying and other anomalous behavior on the part of an NPC by learning to distinguish the sound textures associated with different cognitive processes in the NPC.

References

1 Roads, Curtis (2001). Microsound. Cambridge: MIT Press. ISBN 0-262-18215-7.
2 Chowning, J. (1973). "The Synthesis of Complex Audio Spectra by Means of Frequency Modulation". Journal of the Audio Engineering Society 21 (7).

4.5 MCTS for PCG

Adam M. Smith (University of Washington – Seattle, US)

Monte-Carlo Tree Search (MCTS) and procedural content generation (PCG) were independently popular topics a the seminar, however little discussion involved combining the two. MCTS is usually seen specifically as a way define a game-playing agent's behavior while the choices made inside a PCG system, often a result of search, are not seen as manifesting the behavior of any specific agent. These default perspectives need not block us from imagining new integrations, however.

I presented a prototype system, partially developed during the seminar's hack day, that applied MCTS to a PCG problem. In my prototype, the tree-search algorithm explored alternative executions of an imperatively-defined nondeterministic procedure, receiving rewards according to how long the procedure could execute before encountering an exception.

The example domain, generating simple solvable algebraic equations, was derived from an earlier generator I created for *DragonBox Adaptive* (http://centerforgamescience.org/portfolio/dragonbox/) In the domain-specific parts of the generator, I wrote imperative Python code which used an abstraction with a similar interface to a random number generator to make nondeterministic choices, e.g. defining equation structure and selecting coefficient values. During search, whenever a nondeterministic choice was encountered, the `fork` syscall was used to branch executions in which a different value was returned. The tree-search algorithm directed exploration of these branches much more effectively than the random or bread-first selection policies I tried.

Although the project was prompted by the potential to combine MCTS and PCG, it also inadvertently explored another PCG-related discussion point: must constraint-based PCG systems necessarily be implemented in declarative languages? My demonstration showed one way for the constraints on a design space to be derived automatically from the execution of an imperative program (similar to the use of path-exploring symbolic execution in formal verification [1]) as well as suggesting how MCTS could be applied as a lightweight constraint solving algorithm (employing a fixed variable ordering). The use of MCTS for solving constraints [2] and the use of operating system call to build search spaces from the execution of native code [3] are not new, however their introduction to PCG researchers is a significant novelty.

References

1 Williams, N. and Marre, B. and Mouy, P. and Roger, M. *PathCrawler: Automatic Generation of Path Tests by Combining Static and Dynamic Analysis*. In Proc. 5th European Dependable Computing Conference (EDCC-5), Budapest, Hungary, April 20-22, 2005, Lecture Notes in Computer Science 3463 Springer 2005, ISBN 3-540-25723-3, Budapest, Hungary, April 2005, pages 281–292.

2 Loth, M. and Sebag, M. and Hamadi, Y. and Schulte, C. and Schoenauer, M. *Bandit-based Search for Constraint Programming*. In Proc. of the AAAI Workshop on COmbining COnstraint solving with MIning and Learning (COCOMILE), 2013.

3 Paige, B. and Wood, F. *A Compilation Target for Probabilistic Programming Languages*. In Proc. of the 31st Int'l Conf. on Machine Learning, pp. 1935–1943, 2014.

4.6 Exploring Embedded Design Theory in Maze Generation

Gillian Smith (Northeastern University – Boston, US)

As an offshoot of work discussed by the PCG and Formal Design Methods group, one concrete project is to examine the formal design theories that are embedded in a single, simple domain: both in terms of the process being followed by the generator and theories of the product being created. A domain for procedural generation that has seen a great deal of prior work is that of maze generation; some of the earliest generative systems created simple mazes and labyrinths. The aim with this project is to examine maze generators from a wide variety of authors: computer scientists and artists, students and professional developers, researchers and hobbyists. What aspects of mazes are prioritized in the representation, what is implicit in the specification, and what is left to fall out in emergent behavior? What kinds of approaches to the design process are modeled? What kind of design properties are considered required or desirable, and how are those properties enforced?

Several Dagstuhl participants created or documented their own maze generators to bootstrap collecting examples for this study. Approaches include generation via tree search, optimization (e.g. genetic algorithms), constraint satisfaction (e.g. answer set programming), and ad-hoc methods. All mazes thus far assume an underlying grid structure. The intent moving forward is to cast a wide net to find other existing maze generators, solicit for additional maze generators from a diverse audience, and study ways to evaluate the fitness of each maze and expressiveness of each generator for a given purpose.

4.7 Representation Learning for Procedural Content Generation

Kenneth O. Stanley (University of Central Florida – Orlando, US)

An important goal for procedural content generation (PCG) is to be driven by data [1, 2, 3, 4]. That is, in such data-driven PCG, the computer could be shown examples of the kind of content it should generate, and it would then on its own generate an entire content space automatically. While this capability does not presently exist, a recent paper [5] hints that deep neural networks may someday provide it. In the paper, *Learning to Generate Chairs with Convolutional Neural Networks*, a deep neural network is shown 50,158 chair images and learns from those to generate new chairs that were never seen before. Interestingly, this process is the reverse of what most deep networks do: instead of training on images and outputting classifications, in effect this network trains on classifications and outputs images. More broadly, generative models learned by neural networks suggest similar potential.

While the full potential of such systems remains to be explored, their realization would open up many promising applications in PCG. For example, sprite sheets and 3D models could be generated en masse for enemies, friends, characters, monsters, animals, objects, weapons, vehicles, dwellings, trees, etc. The technique could also potentially be extended to sound, for example to generate a space of audio explosions. Designers might also benefit by using such systems as a creative tool for inspiration. The capability to generate an image from a set of descriptive parameters also suggests more ambitious potential applications,

such as inputting an image and outputting an animated behavior for that image. Examples include walking gaits for body morphologies, explosions or appropriate physical properties for particular objects, growth processes for plants, or even skin for skeletons. Perhaps even a descriptive vector for a level could be input and yield an entire output level.

Beyond just applications, data-driven PCG raises several intriguing theoretical issues and questions. For example, is it possible to learn to extrapolate beyond the bound of given examples, such as to generate a level more difficult (yet still engaging) than anything seen before? More generally, how effective are such techniques with only limited data? Is it possible to ensure the feasibility of generated content, or is it necessary to prune suggestions away after training is over? One particular challenge might be to obtain different options for the same interpolation, given that a neural network often only outputs one result for any given input.

While the paper on generating chairs provides one possible approach to such data-driven PCG (a deconvolutional network), others are conceivable. For example, generative models like autoencoders or RBMs might provide similar capabilities. Evolutionary diversity methods such as novelty search or MAP-Elites might also provide similar capabilities, in addition to the potential to discover multiple diverse instances of the same interpolation. Transfer learning could also apply, that is, one domain might inform another. For all techniques, obtaining the necessary training data will of course always be an important prerequisite.

References

1 Steve Dahlskog, Julian Togelius, and Mark J Nelson. Linear levels through n-grams. Proceedings of the MindTrek Conference (2014).
2 Sam Snodgrass, and Santiago Ontañón. Experiments in map generation using Markov chains. Proceedings of Foundation of Digital Games (2014).
3 William Raffe, Fabio Zambetta, Xiaodong Li, Kenneth O. Stanley. An Integrated Approach to Personalized Procedural Map Generation using Evolutionary Algorithms. IEEE Transactions on Computational Intelligence and AI in Games (2014).
4 Noor Shaker and Mohamed Abou-Zliekha. Alone We can do so Little, Together We can do so Much: A Combinatorial Approach for Generating Game Content. Proceedings of Artificial Intelligence and Interactive Digital Entertainment (2014).
5 Alexey Dosovitskiy, Jost Tobias Springenberg, and Thomas Brox. Learning to Generate Chairs with Convolutional Neural Networks. arXiv preprint arXiv:1411.5928 (2014).

4.8 What Did You Do?

Julian Togelius (New York University, US)

Joint work of Cook, Michael; Eladhari, Mirjam; Smith, Gillian; Thompson, Tommy; Togelius, Julian; Zook, Alexander

What did you do? is a prototype game designed and mostly implemented during the course of the seminar. The design originates in the discussions of the "AI-based game design" group, where a number of design patterns used to make foreground AI useful in games were identified. In order to validate and exemplify these design patterns, two small game prototypes were hastily designed and very hastily implemented. One if them is "What did you do?". The game is designed to implement three of the identified patters: Player Trains AI, Player Edits AI and Visualize AI State.

The game is turn-based and plays out on a grid. There are five types of entities in the world: the player character (parent shrub), childshrubs, stones, strawberries, and ponds. The parent shrub can move in any of the four cardinal directions, eat or pick up whatever is in the direction it is facing, or drop something it is carrying. The parent has an energy level which can be recharged by eating. Child shrubs have mostly the same actions available, but they have somewhat different effects: in particular, if the child picks up stone it might be crushed under its weight, and it will drown if moving into a pool. The child's energy levels are also constantly decreasing. Both stones and ponds block the way, but stones can be picked up and moved elsewhere. In the course of the game, rivers can be forded by stones, strawberries can be eaten etc.

In a nutshell, the game is about feeding and protecting your kids while also protecting them from the outcomes of the stupid things they while imitating your behavior. A rule learning algorithm runs in the background, and finds common situations the parent faces and what action the parent took. The children's brain (they share a brain) learn rules from this, and these rules are clearly displayed to the player in the GUI. Using resources gotten from eating strawberries, the player can remove certain rules, but this means that the same resources are not available for feeding the kids. There is thus at least two different tradeoffs required during gameplay: between feeding the kids and weeding the thoughts (rules) in their mind, and between defending the kids against the outside world and defending them against themselves. In both cases, this requires engaging with and understanding the workings of the AI system. Thus, we believe that this prototype shows how to combine several of these design patterns into a truly AI-based game.

4.9 The Grey Eminence: A Political Game of Conflict, Influence and the Balance of Power

Fabio Zambetta (RMIT University – Melbourne, AU)

Joint work of Lanzi, Pier Luca; Zambetta, Fabio

We propose an AI-based game designed around the Years of Lead [1], a period of political turmoil in Italy that stretched across the 1960s up to the 1980s. The period was characterised by the heavy influence of secret services of the two blocks in the Cold War in key events of the political life, as well as their interference in now infamous events, such as the bombing at the Bologna Station in 1980 or the Piazza Fontana bombing in 1969.

The player is indeed tasked with trying to implement the Strategy of Tension [1], what used to be the strategic centerpiece of Gladio [2] a so-called stay-behind organisation that was supposed to be reacting to a Soviet invasion in Europe but indeed started to proactively plan false-flag operations. The aim of the game is to spread terror and anxiety in the Public Opinion such that the Prime Minister is pressured into declaring the State of Emergency.

From a technical perspective, we endeavour to represent key actors in the game (the Prime Minister, the Public Opinion, the Leader of the Opposition, etc.) as agents that can be swung via specific operations, whose cost, paid out of a budget provided by Gladio, as well as their risk/reward trade-off can vary considerably. The relationships between such agents will be non-linear so that the player will not be able to easily predict patterns of behaviour that can elicit victory.

Another interesting technical aspect of the game is that feedback provided to the player at the end of a turn, will be in the form of TV news, newspaper articles, etc. detailing the current situation in the parliament as well as other "random" events (e.g., a new conflict in Middle East has started, a very big IT company has gone bust, the global economy is experiencing a recession). Such news will be procedurally generated out of templates and/or news stubs, which will then be customised on the fly via tokens, tags and annotations that will be instantiated based on the current context and gameplay progression.

References

1 Ganser, D. NATO's Secret Armies: Operation Gladio and Terrorism in Western Europe. London (2005).
2 Ganser, D. Terrorism in Western Europe: An Approach to NATO's Secret Stay-Behind Armies. ISN. Whitehead Journal of Diplomacy and International Relations, 6(1), South Orange NJ (2005).

4.10 One Word at a Time: a Magnetic Paper Bot

Jichen Zhu (Drexel University – Philadelphia, US)

Joint work of Zhu, Jichen; Ontañón, Santiago; Magerko, Brian

One Word at a Time (OWAAT) is a program based on an improvisation theater game of the same name. In the original game, two or more people compose a story by taking turns to add one word at a time. Sometimes used as an ice-breaker, an important aspect of the game is the unexpected direction toward which the sentence might be steered, often achieving humorous effects.

The goal of this project was to create bots that could play OWAAT to generate academic writings, while preserving the unexpected and humorous tone in the original game. In addition, we wanted each bot to exhibit certain linguistic characteristics of specific authors of choice, in the visual style of magnetic poetry.

To address this challenge, we designed and developed a bot that can generate sentences, one word at a time, based on a corpus of writings by a given author. In this way, the bot can play the game with a human, or it can play with other bots trained on a different corpus. To make the sentences as grammatically and semantically coherent as possible, we used the following techniques.

Given a corpus of academic papers written by a given author, the OWAAT bot plays the game thanks to two main components:

1. A low-level statistical model: this model captures the frequencies of the different words in the corpus relative to the context in which they appear. For our prototype, we used a first order Markov chain, which captures the probability distribution of a word given the previous word.
2. A sentence-structure model: this model tries to ensure that the generated sentences are syntactically correct (i.e., they have a subject, a verb and a predicate). In our prototype, we used a finite-state machine (FSM) where transitions between states correspond to "part-of- speech" tags ("verb", "noun", "preposition", "determinant", etc.), The FSM is designed in a way that traversals from the start state to any of the terminal states correspond to valid sentences, and was generated by hand.

For the bot to add the next word to the current partial sentence S, it first determines the state in the FSM that S is in. The transitions coming out of this state are the possible part-of-speech tags that the next word can have. The Markov chain is used then to generate the next word, only considering the words that have the appropriate part-of-speech tag. In this way, by combining both low-level and sentence-structure models, our bot can play the game, resulting in sentences that 1) resemble the sentences written by the intended author, 2) have local consistency (thanks to the Markov chain), and 3) have whole-sentence consistency (thanks to the FSM).

To demonstrate our bot, we gathered publications from several researchers at this Dagstuhl Seminar and trained bots to play OWAAT in the style of these authors. Underlining the comic effect, we designed the interface to visually resemble magnetic poetry, which shares the similar aesthetics of unexpected juxtapositions.

4.11 Contrabot

Alex Zook (Georgia Institute of Technology, US)

Joint work of Champandard, Alex; Cook, Michael; Smith, Adam; Smith, Gillian; Thompson, Tommy; Zook, Alex

Contrabot (Figure 2) is a game prototype for an AI-based game, developed at the 2015 Dagstuhl Seminar on Computation and Artificial Intelligence in Games. AI-based games foreground interaction with an AI system in the player's experience. Our goal with Contrabot was to build a game based on agents that learn, crafting gameplay around understanding, playing against and ultimately deceiving a machine learning system.

You play as a smuggler trying to secretly label boxes to communicate with a contact on the other side of a customs checkpoint. The smuggler is trying to learn the code you use to indicate a box is contraband – but an inspector is randomly checking boxes too. Can you design and redesign your secret codes to stop the inspector learning your patterns? Will you still manage to sneak your code through to your contact and smuggle your goods out of the country?

The game mechanics revolve around how the smuggler and inspector agents learn to check codes based on codes they have seen. These agents have two main processes: learning

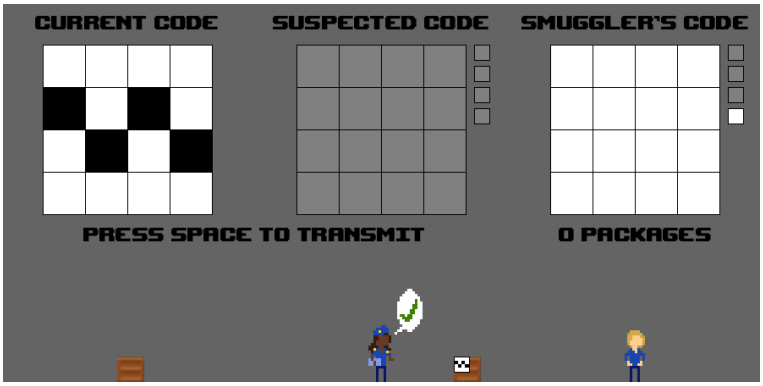

Figure 2 Contrabot.

codes and matching new codes against their learned code. Both agents generalize patterns from example codes – using a form of least general generalization – in their memory to then try to match new codes to these learned patterns. The inspector has a larger memory than the smuggler and gameplay is based on using reverse-engineering how learning works to take advantage of the smuggler forgetting old patterns more quickly than the inspector. The generalization process is simple, when comparing all codes seen the agents memorize exact matches for black or white tiles and generalizing to gray tiles if a position has been occupied by both colors. Despite this simplicity the design accommodates many levels of difficulty and risk-reward considerations for players based on the size of the codes used and memory capacities of each agent.

The game prototype is available to play and fork the design at https://github.com/gamesbyangelina/contrabot.

Participants

- Dan Ashlock
University of Guelph, CA
- Christian Bauckhage
Fraunhofer IAIS – St. Augustin, DE
- Rafael Bidarra
TU Delft, NL
- Bruno Bouzy
Paris Descartes Unversity, FR
- Michael Buro
University of Alberta, CA
- Alex J. Champandard
AiGameDev.com KG – Wien, AT
- Simon Colton
University of London/Goldsmiths, GB
- Michael Cook
University of London/Goldsmiths, GB
- Peter I. Cowling
University of York, GB
- Mirjam P. Eladhari
University of Malta, MT
- Ian Horswill
Northwestern University – Evanston, US
- Graham Kendall
University of Nottingham, GB
- Pier Luca Lanzi
Politecnico di Milano Univ., IT
- John M. Levine
University of Strathclyde, GB
- Antonios Liapis
IT Univ. of Copenhagen, DK
- Simon M. Lucas
University of Essex, GB
- Brian Magerko
Georgia Inst. of Technology, US
- Michael Mateas
University of California – Santa Cruz, US
- Joshua Allen McCoy
University of California – Santa Cruz, US
- Mark J. Nelson
IT Univ. of Copenhagen, DK
- Ana Paiva
INESC-ID – Porto Salvo, PT
- Mike Preuss
Universität Münster, DE
- Günter Rudolph
TU Dortmund, DE
- Spyridon Samothrakis
University of Essex, GB
- Tom Schaul
Google DeepMind – London, GB
- Noor Shaker
IT Univ. of Copenhagen, DK
- Moshe Sipper
Ben Gurion University – Beer Sheva, IL
- Adam M. Smith
University of Washington – Seattle, US
- Gillian Smith
Northeastern University – Boston, US
- Pieter Spronck
Tilburg University, NL
- Kenneth O. Stanley
University of Central Florida – Orlando, US
- Ruck Thawonmas
Ritsumeikan Univ. – Shiga, JP
- Tommy Thompson
The University of Derby, GB
- Julian Togelius
New York University, US
- Michael Treanor
American University – Washington, US
- Marc van Kreveld
Utrecht University, NL
- Santiago Ontanon Villar
Drexel Univ. – Philadelphia, US
- Mark Winands
Maastricht University, NL
- Georgios N. Yannakakis
University of Malta, MT
- R. Michael Young
North Carolina State Univ., US
- Fabio Zambetta
RMIT Univ. – Melbourne, AU
- Jichen Zhu
Drexel Univ. – Philadelphia, US
- Alex Zook
Georgia Inst. of Technology, US