Anza Island: Novel Gameplay Using ASP

Kate Compton
Expressive Intelligence Studio,
University of California, Santa Cruz
kcompton@soe.ucsc.edu

Adam Smith
Expressive Intelligence Studio
University of California, Santa Cruz
amsmith@soe.ucsc.edu

Michael Mateas
Expressive Intelligence Studio
University of California, Santa Cruz
michaelm@cs.ucsc.edu

ABSTRACT

Procedural content generation (PCG) has the potential to create unique artifacts, levels, and gameplay mechanics. However, it remains challenging to generate content that satisfies gameplay constraints: methods to achieve this include generate-and-test, search-based generation, and constructive methods. In this paper, we present a prototype, a simple game, which demonstrates the use of an off-the-shelf logic program solver, Clingo, as an easy and expressive way to model these constraint problems, and find solutions that satisfy gameplay constraints. By delegating the difficult search optimization problem to an external program, we were able to quickly prototype PCG in a low-effort way by expressing the desired content as a set of rules and constraints, keeping the focus on the designer's intentions for the generated content, rather than specific methods used to create or find it. The expressiveness and versatility of this approach is demonstrated by applying this technique to two areas of PCG in the game.

Categories and Subject Descriptors

K.8.0 [**Personal Computing**]: General—*Games*

Keywords

procedural content generation, answer set programming, puzzle games

1. INTRODUCTION

Procedural content generation (PCG) is a way of creating content with automated methods, rather than by hand. Although PCG has potential to add value to games, it can be difficult to author these methods such that they reliably produce output that can satisfy the gameplay constraints. Fortunately, tools for satisfying constraints, even complex ones, already exist. Our contribution to the field is an extension of Smith and Mateas's work on using Answer Set Programming (ASP) solvers to create PCG levels that satisfy gameplay requirements [3]. ASP is a declarative programming paradigm that allows declaratively describing a design space. A designer can define their desired PCG in a readable syntax, and use the solver to generate PCG without writing search algorithms. We extend this work by applying it not only to level generation, but creating a new game mechanic, logic crafting, which would not be possible without the use of ASP.

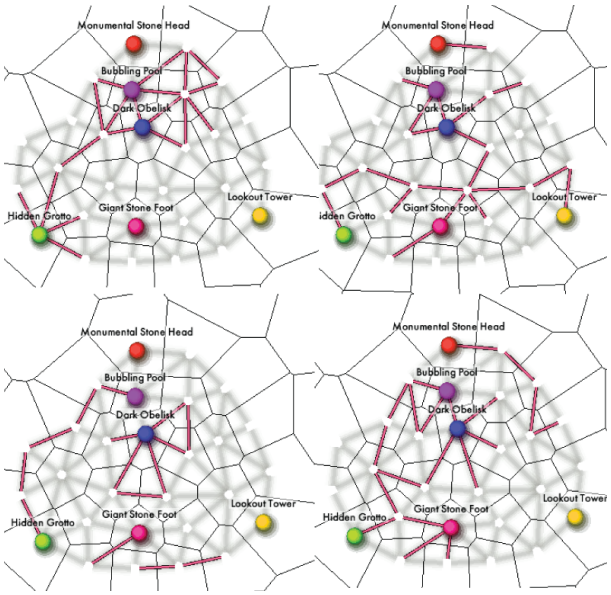

Figure 1: A selection of four possible solutions for the constraints: "Dark Obelisk can't be connected to Monumental Stone Head. Monumental Stone Head can't be connected to Hidden Grotto. Bubbling Pool must be connected to Hidden Grotto."

This motivated the development of a new game, Anza Island, which was created as a sandbox in which to test the versatility of ASP for PCG. In Anza Island, the player creates logical constraints that can be used to control an AI character's gameplay moves. Creating this new form of gameplay allows not only an ASP-specific game mechanic, but it makes the process of constraint-solving the center of gameplay.

2. RELATED WORK

2.1 Types of Procedural Content Generation

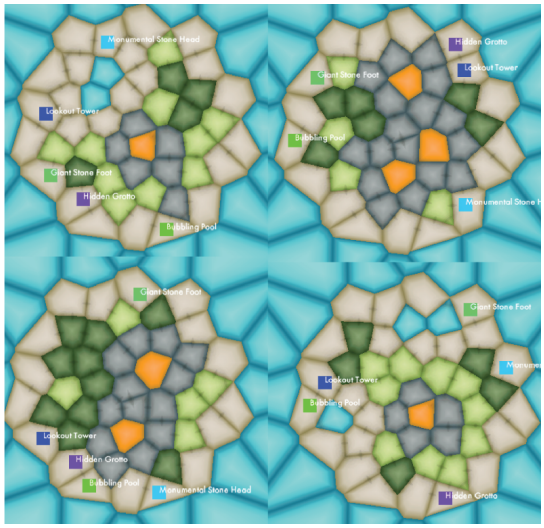

Figure 2: A variety of possible islands generated with a short list of constraints.

PCG is often offered as a potential solution to a number of game design problems. As Togelius et al enumerates, these include: storing large quantities of content in small amounts of memory so that it can be regenerated as needed, speeding up the process of making content, so that artists can create larger amounts of content in less time, and finally, creating new kinds of gameplay by allowing infinitely long levels, which could never be generated by previous hand-authored techniques [5].

Smith et al [4] describes a similar list of uses for PCG techniques, but has a more expanded taxonomy of the new gameplay mechanics that PCG allows. They propose that some uses of PCG provide replayability; that is, they do not create new mechanics, but just allow the player to apply the same mechanics in an endless set of new levels.

They contrasts this with adaptive PCG which takes the player's actions as input, and uses that information to guide the generation of new content. While PCG for replayability does not create novel game mechanics, as the generator is not making anything that a human designer couldn't have made, adaptive PCG allows mechanics that could never exist otherwise, because a traditional artist or level-designer could not respond in real-time to a player's actions. Although PCG levels that are generated to match a player's skill or preferences could be considered adaptive, the player is not consciously making choices to affect the PCG, so while it affects the gameplay experience, it does not enable new mechanics.

Understanding these different categories of PCG informs the prototype that we have built. If using an ASP solver is to be expressive enough, it should be able to create examples of all of these kinds of content.

2.2 Procedural Content Generation for Tile-based Level Generation

One of the most common types of PCG is terrain generation. Patel's meticulously documented process for creating a procedurally generated island [2] provides a good illustration, both because it shows the many constraints that are important for terrain generation, and how these are maintained in a feed-forward generation algorithm, but he also identifies points at which these constraints were hard to satisfy, and what had to be done to correct it. This is exemplified when his system creates the elevation of the terrain: though the generator may have the knowledge of how many mountains are desirable, the height-creation stage of the algorithm may create either too few or too many mountains. In that case, the generator needs to know that it is in this problematic state, and redistribute the terrain heights to create the desired end condition.

This problem, and its solution, is a common pattern in PCG: content is generated, and it doesn't satisfy the constraints needed, for gameplay or for aesthetics, and some corrective algorithm must be applied. Of course, if many different kinds of corrections need to be made, these corrections become prohibitively complex and might undo each others work, or create new situations that must be corrected.

Another solution would be to generate many possibilities for that content, and choose the best one of those [5]. However, if the search space is sufficiently large and multi-dimensional, and there are many constraints, the generator may need to create thousands of examples before it finds a satisfactory one, and it isn't guaranteed to find a satisfactory one at all.

In this prototype, we show that similar landscapes could be created with ASP by just expressing these constraints directly, and with no post-generation corrections needed.

2.3 Answer Set Solving for Games

This work has been inspired and guided by Smith and Mateas's work on using ASP for PCG [3]. He establishes a method in which a design space is modeled by a logic program, which is in turn solved by the solver to return answer sets, which can be interpreted as artifacts. He applies this to the problem of generating chromatic mazes, with success: this problem is easy to describe in ASP, and the ASP solver is able to find solutions. In the taxonomy by Smith et al, this would be PCG for replayability. ASP generates the same sort of mazes that a human could have generated, though it would have been a tedious task for a human designer, but it can generate many of them. This shows ASP's potential as for generating replayability, but does not yet explore the potential of ASP for any adaptive PCG. This presents an opportunity for us to apply ASP for replayability, and also to explore the ways in which ASP enables direct and indirect control of the PCG.

3. PROTOTYPE

3.1 Gameplay

To effectively demonstrate the versatility of ASP for gameplay, a simple turn-based game was devised that could use ASP in multiple ways. A human player is playing against an AI character, Anza, that can control whether or not different zones are reachable by turning on and off the bridges that connect them. The player must move her pawn to a

number of landmarks in order to win. Each time the player moves to a new zone, she collects a logic card, which contains descriptions of the zones, such as "all beaches", "the tower", and "the player's current tile", or a phrase describing how they may be connected such as "must be connected", "must not be connected", "must be connected with no fewer than 3 bridges". The player will often be prevented from moving to where she wants: Anza can activate and deactivate bridges each turn to try to lock the player. So, to create paths between these landmarks, the player must create constraints such that Anza is forced to connect the zones in a way that satisfies the constraints, and the player creates these constraints by "logic crafting", that is, playing 3 logic cards such that they make a logical statement: "All beaches must be connected to the player's current time". Playing that statement would force Anza to make enough bridges active so that the player could move freely to any beach.

3.2 Integrating Processing and Clingo

Similar to the color-maze generator by Smith and Mateas, Anza Island uses ASP to turn a logic program into an answer set, one answer of which is taken and turned into artifacts. Only in this case, the answer set program has itself been generated from the gameplay, so that it can respond to the player's input and the changing game state.

We are using Processing, a Java-based programming language, for its ease of use and graphical interface, which allows us to create an appealing game world. Processing creates and stores the information about the game world, and the constraints that have been played.

For our ASP solver, we are using Clingo [1]. Processing calls the Clingo executable, feeds it the answer set program that has been generated, and then parses Clingo's output to identify the changes that need to be made to the game world.

3.3 Terrain generation

ASP is first used to create the terrain layout of the island on which the game is played. Figure 4 shows the full terrain generation process. First, a distribution of points (Fig 4:1) are used to create a Voronoi mesh of multiple zones and a Delaunay mesh of potential connections between all neighboring zones (Fig 4:2). This information is used by Processing to create a game world of zones and bridges that are not yet fully specified (Fig 4:3). Information about this game world, along with some further constraints about island layout, are compiled into a single answer set program (Fig 4:4) which is given to Clingo to solve, and a single answer from the set is selected. That answer is parsed, and the information from it is used to finish specifying the zones and bridges (Fig 4:5).

As an example, assume that Processing has already output the bordering information for each zone into the Answer Set program.

```
borders(zone32, zone33).  borders(zone30, zone33)....
```

If we want between 20 and 30 zones to be land, this is easy to express:

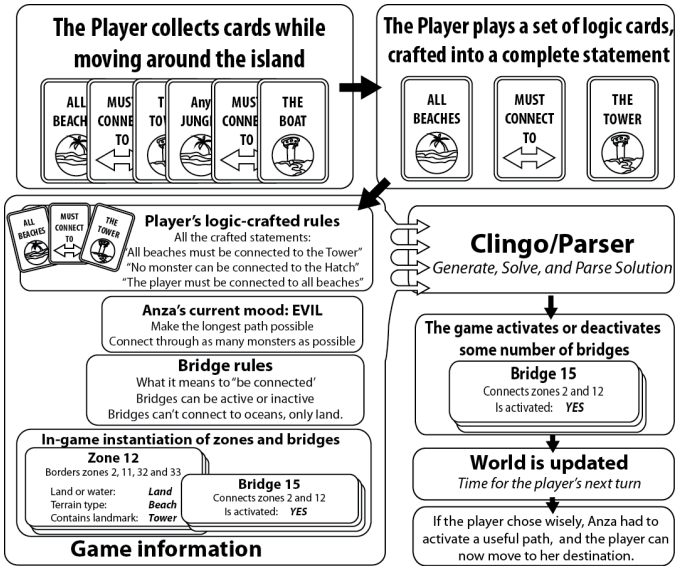

Figure 3: Adaptive PCG with ASP: the prototype responds to a player's crafted logic statement.

```
20{ land(Z) :zone(Z) } 30.
```

Notice that we do not have to write a separate algorithm to correct it if we get the wrong number of zones of land, as Patel [2] did. Error correction and constraint satisfaction are left to Clingo.

Since that was so easy, we can start complicating our generator:

```
:- hasTerrain(A, lava), zone(B), borders(B, A),
not hasTerrain(B, mountain).

hasTerrain(A, beach) :- land(A), borders(B, A),
hasTerrain(B, ocean).
```

This expresses the constraints: if a land zone touches water, it is a beach, and if there is a volcano, it must be surrounded by mountains. These constraints were added without having to consider how they would affect previously defined constraints; we could just add them and let Clingo sort through it. This declarative style of programming enables us to add and remove constraints, without ever having to modify or rewrite a search algorithm, enabling extremely fast prototyping. Figure 2 shows an example of generated islands.

3.4 Choosing Anza's Moves

Though ASP can be used to more rapidly prototype content generation that could be done using traditional methods, it can also be used to create content and modes of interaction that would be impossible to create otherwise, such as adaptive PCG. The method for creating adaptive content with ASP is exactly the same as for non-adaptive content, only the player's added constraints and Anza's current goals are also added on to the list of constraints for Clingo to solve. Figure 3 shows the cycle for one turn of gameplay.

In Anza Island, the adaptive application of PCG is logic

crafting, a new mechanic enabled by the use of ASP. In the course of play, the player collects logic cards, each of which represents a logical concept, such as "All beaches", "The Tower", or "Are Connected To". Once she has collected a useful set of cards, these can be crafted into a valid logic statement, like "All beaches are connected to the Tower". In this game, the player is directly controlling the adaptive PCG by explicitly expressing these demands.

Once the logic cards are crafted into a statement, the player give this to Anza as a command. Behind the scenes, this logic statement is converted into a few lines of ASP declarations. These generated lines are appended to the rest of the program that describes the world. When the solution set comes back from the solver, it contains which bridges Anza should activate and deactivate. This information updates the world and the player can continue moving.

Figure 1 shows a selection of the many possible moves that Anza could make in response to a given set of constraints. This allows us to further constrain these sets by giving Anza a personality, either helpful, or unhelpful, that will add constraints to the solutions that Clingo will find. This blurs the line between PCG techniques and AI techniques: the same ASP methods are used to generate terrain, and generate the opponent's game moves.

4. CONCLUSION

In this paper, we describe a work-in-progress PCG-based puzzle game using constraint solving as a gameplay mechanic, a mechanic that would not be possible without ASP. In addition, using ASP to create Anza Island allowed us to easily prototype, modify, or add new kinds of procedurally generated content, without writing laborious search algorithms for each new kind of content. At the same time, the constraint solving power of ASP made it easy to ensure that all the generated content would work with the gameplay requirements. As a prototype, Anza Island demonstrates the versatility of ASP for game creation.

5. REFERENCES

[1] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and S. Thiele. A user's guide to gringo, clasp, clingo, and iclingo. *University of Potsdam, Tech. Rep*, 2008.

[2] A. Patel. Polygonal map generation for games. http://www-cs-students.stanford.edu/ amitp/game-programming/polygon-map-generation/.

[3] A. Smith and M. Mateas. Answer set programming for procedural content generation: A design space approach. *Computational Intelligence and AI in Games, IEEE Transactions on*, 3(3):187–200, 2011.

[4] G. Smith, E. Gan, A. Othenin-Girard, and J. Whitehead. Pcg-based game design: enabling new play experiences through procedural content generation. In *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games*. ACM, 2011.

[5] J. Togelius, G. Yannakakis, K. Stanley, and C. Browne. Search-based procedural content generation: A taxonomy and survey. *Computational Intelligence and AI in Games, IEEE Transactions on*, 3(3):172–186, 2011.

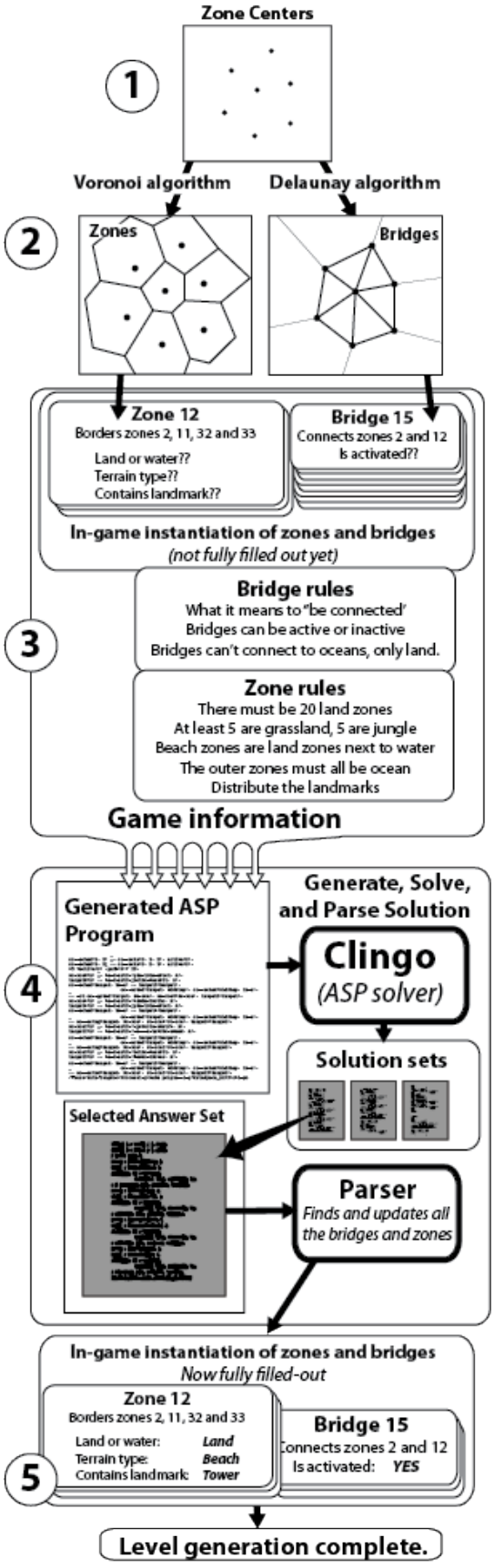

Figure 4: ASP for replayability, generating a level