Exploring a Large Space of Small Games

Gabriella A. B. Barros
IT University of Copenhagen
Rued Langgaards Vej 7
2300 Copenhagen, Denmark
gbar@itu.dk

Julian Togelius
IT University of Copenhagen
Rued Langgaards Vej 7
2300 Copenhagen, Denmark
julian@togelius.com

***Abstract*—We explore the soundness and playability of randomly generated games expressed in the Video Game Description Language (VGDL). A grammar is defined for VGDL, which is able to express a large variety of simple arcade-like games, and random expansions of this grammar are fed to a VGDL interpreter and played with off the shelf agents. We see this work as the first step towards generating complete, playable games.**

I. Introduction

Artificial intelligence can be used not only to play games, but also to design them. While research in procedural content generation aims to find methods to generate content for games (levels, textures, items etc) [1], the related research goal or *automatic game design* seeks ways of procedurally generating complete games [2]. In particular, this includes generating the rules and mechanics of the game. Automatic game design has now been attempted by several researchers, using methods as different as grammar expansion, evolutionary computation [2], [3], [4] and answer set programming [5]; for an overview of such attempts see [6].

In order to generate a complete game, one first needs to define a space of games which can be sampled or searched in some way. Most of the automatic game design attempts so far have designed a domain-specific language for describing the games (a Game Description Language or GDL), and then explored the space of games that can be expressed in this language through generating strings. Thus, the limits of the design space become very explicit, and the properties of this space can be controlled through changing the details of the language. A lower bound on the expressivity of a GDL can be given by showing that certain games can be expressed within the language – for example, Browne showed that a number of classic board games could be expressed with his Ludi GDL [4]. If the langauge can express games A and B, it can also express a large number of games on the line segment in design space between these two games. Intuitively, some of these should be interesting new games.

The Video Game Description Language (VGDL) is a GDL designed to allow the expression of a large variety of 2D arcade games, of the types commonly found on early 80's game consoles such as the Atari 2600 [7], [8]. One of the design goals for the language was that it should be possible to do a reasonable approximation of most classic arcade games that feature an avatar moving on a plane, and interacting with enemies, power ups, walls etc. During the design process of VGDL, games such as Frogger, Lunar Lander, Space Invaders and Boulder Dash were implemented, and language features were added when needed to make these implementations possible. Separate game engines are available in Python and Java, that can take any valid VGDL definition and produce a complete playable by a human or an agent.

The language was developed with the dual purposes of creating a parameterisable benchmark for game-playing algorithms, what we call "general video game playing", and for allowing the automatic generation of new games. We expect work towards these two purposes to exhibit considerable synergy, where game generation algorithms develop ever more sophisticated games for ever more sophisticated game playing algorithms to solve. A competition based on general video game playing, featuring dozens of human-designed games, is currently underway. The current paper, which is a short work-in-progress report, represents the very first step towards generating games in the VGDL space.

The medium-term plan is to use evolutionary algorithms to find games that are playable but non-trivial for general game-playing algorithms, and base evaluation functions on metrics from how the games are played out. However, the first steps are to ensure that we can generate valid VGDL definition at all, and to characterise the space of valid VGDL definitions through random sampling. In particular, we are interested in what percentage of the space represents playable games.

II. Random VGDL expansions

In order to generate random VDGL expansions, we developed a grammar. A small part of this grammar, which contains a few dozen lines in total, is shown in Figure 1. The grammar structure closely models VGDL, and also consists of four main blocks, in the same order as described below:

The *Sprite Block* contains game objects' information, in a hierarchical form based on the VGDL ontologys Sprite representation. An avatar is first defined, followed by the remaining sprites. The *Level Mapping Block* defines how to represent each sprite in game levels. Each one is associated with a single character, e.g. "A" is, by default, associated with the avatar. The *Interaction Block* describes what happens when two game objects collide, e.g. in the classic Space Invaders, avatar shoots missiles, missiles kill aliens, and aliens kill avatar. Finally, *Termination Block* indicates how the game ends, e.g. if the player survives for 2 minutes, he would win.

978-1-4799-3547-5/14/1.00

⟨*start*⟩ ::= BasicGame ⟨*eol*⟩ INDENT ⟨*sprite_block*⟩ ⟨*level_block*⟩⟨*interaction_block*⟩ ⟨*termination_block*⟩

⟨*sprite_block*⟩ ::= SpriteSet ⟨*eol*⟩ INDENT ⟨*create_avatar*⟩ ⟨*eol*⟩⟨*create_sprites*⟩ DEDENT

⟨*create_sprites*⟩ ::= {⟨*identifier*⟩ > ⟨*sprite_class*⟩ NEWLINE}

⟨*sprite_class*⟩ ::= ⟨*conveyor*⟩ | ⟨*flicker*⟩ | ⟨*immovable*⟩ | ...

⟨*conveyor*⟩ ::= Conveyor [⟨*option_sprite_color*⟩] [⟨*option_sprite_img*⟩] ...

⟨*level_block*⟩ ::= LevelMapping ⟨*eol*⟩ INDENT {⟨*mapping_def*⟩ NEWLINE} DEDENT

⟨*interaction_def*⟩ ::= ⟨*identifier*⟩ ⟨*identifier*⟩ > ⟨*interaction_class*⟩

⟨*interaction_class*⟩ ::= ⟨*bounce_forward*⟩ | ⟨*change_resource*⟩ | ⟨*clone_sprite*⟩ | ⟨*kill_if_from_above*⟩ | ...

⟨*termination_block*⟩ ::= ⟨*sprite_counter*⟩ | ⟨*timeout*⟩ | ...

Figure 1. Small part of the generation grammar, in Extended Backus-Naur Form.

Games status	Quantity
Crashed during initialisation	71
Crashed during runtime	133
Did not crash	362

Outcome of games that did not crash	Quantity
Always ended	232
Never ended	124
Sometimes ended	6

Controller's outcome, among games that sometimes ended	Quantity
Always won	0
Sometimes won	2
Never won	4
Disqualified	2

TABLE I

TOP: AMOUNT OF GAMES THAT CRASHED OR NOT. MIDDLE: OUT OF THOSE THAT DID NOT CRASH, AMOUNT OF GAMES THAT ALWAYS, SOMETIMES OR NEVER ENDED BY GAME RULES. BOTTOM: OUTCOME OF GAMES THAT DID NOT CRASH AND SOMETIMES ENDED.

During initial development, it was noted that it would be necessary to constrain the creation process for each block, except Level Mapping. Parameter types would have to be defined according to the Sprite, Interaction or Termination type. For instance, a SpawnPoint object required that another object was chosen to be spawned. However, a RandomNPC object has no use for another object. Also, some types of sprites are incompatible, such as RandomNPC and Spreader. This means that their characteristics cannot be processed together. Therefore, a RandomNPC object cannot also be a Spreader, and vice-versa. Feeding the engine with a game with two sprites connected as above returns an exception.

The grammar is expanded directly as objects in the game engine, from which VGDL definitions can then be generated (rather than the other way around). This aimed at minimizing inconsistencies, e.g. a singleton sprite with a termination condition of "end when this sprite's count reaches 10", would be semantically inconsistent.

III. BUT ARE THEY PLAYABLE?

Using this grammar, 566 VGDL definitions were generated. These were constrained to be relatively short. Concretely, games were limited to 5 sprites, 13 interactions, and 2 terminal conditions. There definitions were fed to the Java engine, which produced playable versions of these games. For each game, a 15x15 map (surrounded by walls) was created. One object for each Sprite type was randomly positioned in it.

Each game was tested 10 times with the standard "random" agent, which simply chooses uniformly randomly from all available actions. Table I shows how many of these games crashed during setting, while creating game objects; during runtime, while the controller played the game; and did not crash al all. Also shows, out of those games that did not crash, how many always, sometimes or never finished according to game rules. A game that never ended by game rules was ended by the engine after 2000 game cycles passed. Table I also shows, on the bottom, the outcomes of games that sometimes ended: controller either always won, sometimes won, never won or was disqualified due to an invalid movement.

IV. CONCLUSIONS

This work described a method for generating vGDL expansions using a grammar in a game object class form. Although the majority of generated definitions are games that either crash or never finish, descriptions created are very diverse. We believe that this work is an important step torwards a larger and more ambitious goal, and there is much space for new ideas and improvement. In our next step, we will try to find practical ways of constraining the search space, and use more sophisticated agents for playouts. We also believe that the current method may serve as an initial effort torwards an interesting debbuging tool, since the wide range of possible descriptions generated allows for intensive engine testing.

ACKNOWLEDGEMENTS

Gabriella A. B. Barros acknowledges financial support from CAPES Scholarship. Bex 13727133

REFERENCES

[1] N. Shaker, J. Togelius, and M. J. Nelson, *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer, 2014.

[2] J. Togelius and J. Schmidhuber, "An experiment in automatic game design," in *Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On*. IEEE, 2008, pp. 111–118.

[3] J. Font, T. Mahlmann, D. Manrique, and J. Togelius, "Towards the automatic generation of card games through grammar-guided genetic programming."

[4] C. Browne and F. Maire, "Evolutionary game design," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 2, no. 1, pp. 1–16, 2010.

[5] A. M. Smith and M. Mateas, "Variations forever: Flexibly generating rulesets from a sculptable design space of mini-games," in *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*. IEEE, 2010, pp. 273–280.

[6] M. J. Nelson, J. Togelius, C. Browne, and M. Cook, "Rules and mechanics," in *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*, N. Shaker, J. Togelius, and M. J. Nelson, Eds. Springer, 2014.

[7] M. Ebner, J. Levine, S. M. Lucas, T. Schaul, T. Thompson, and J. Togelius, "Towards a video game description language," *Artificial and Computational Intelligence in Games*, vol. 6, pp. 85–100, 2013.

[8] T. Schaul, "A video game description language for model-based or interactive learning," in *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*. IEEE, 2013, pp. 1–8.