Towards generating novel games using conceptual blending

Jeremy Gow and **Joseph Corneli**
Computational Creativity Group
Department of Computing
Goldsmiths, University of London, UK

Abstract

We sketch the process of creating a novel video game by blending two video games specified in the Video Game Description Language (VGDL), following the COINVENT computational model of conceptual blending. We highlight the choices that need to be made in this process, and discuss the prospects for a computational game designer based on blending.

Introduction

This paper outlines the process of creating a novel video game by blending two video game descriptions given in the Video Game Description Language (VGDL). Several issues need to be considered: which games to blend? Which elements of the game will be blended and which removed? Do we need to adjust the resulting game in order to make it playable? Taking two games from the GVG-AI Competition (Perez et al. 2015), we provide a worked example based on a recent computational, domain-independant model of conceptual blending (Bou et al. 2015).

Our account provides a high-level sketch of how a computational system could blend game designs, and does not propose a concrete architecture. Instead, our intent is explore how a game blending system could work. We highlight the specific challenges presented by various steps in the blending process, and describe how certain steps provide the opportunity to add or close off new potential mechanics and meanings. The implementation and evaluation of this approach will be the focus of future work.

Background

Recent research (Treanor et al. 2012; Cook and Colton 2014) has examined computational generation of games from provided concepts. The current exploration of blending game descriptions is motivated by our observation that game designers often build new games by combining ideas from existing games.

Conceptual blending is the notion that new concepts can be generated by combining elements of existing concepts. **COINVENT** is an EU FP7 project that aims to develop a "computationally feasible, formal model of conceptual blending" (Schorlemmer et al. 2014)[1]. The project takes Goguen's formalisation of conceptual blending as a starting point (Goguen 1999). Early implementations draw on the related notion of amalgams (Ontañón and Plaza 2010). The core ideas in the model are:

Conceptual spaces may be mathematical theories, ontologies, or, in our case, video game descriptions.

Signature morphisms between conceptual spaces are mappings from the symbols of a source conceptual space into the symbols of target conceptual space. In the diagram below, ϕ_1, ϕ_2, ψ_1, ψ_2, and ι are all signature morphisms.

Generic spaces are a particular kind of conceptual space that possess maximum commonality with a number of given input spaces. In the diagram below, ϕ_1 and ϕ_2 represent the inclusion of the elements of the generic space G in both X and Y.

Blends are unique conceptual spaces that preserve structure from the input spaces and a generic space G, via a specific arrangement of signature morphisms; namely, such that the diagram is commutative, and that 'axioms' in the input spaces X, Y are sent to 'theorems' in the unique colimit B, known as *the blend*:

$$\begin{tikzcd}
 & B & \\
X \arrow[ur, "\psi_1"] & & Y \arrow[ul, "\psi_2"'] \\
 & G \arrow[uu, "\iota"'] \arrow[ul, "\phi_1"] \arrow[ur, "\phi_2"'] &
\end{tikzcd}$$

Within the COINVENT project the model has been primarily applied to examples from mathematics (Bou et al. 2015) and music (Cambouropoulos, Kaliakatsos-Papakostas, and Tsougras 2014). The constituent processes have been automated to differing degrees, and for some simple examples complete automation is feasible. One of the key outstanding issues is the evaluation of blends, both from the point of view of judging the final outcome, and from an 'online' point of view that can help constrain the potential combinatorial explosion as the collection of possible input spaces is explored, and possible blends are computed.

Preprint version.

[1]`http://www.coinvent-project.eu/`

VGDL: The Video Game Description Language allows the specification of simple arcade-style games (Schaul 2014). It was recently developed to support research into artificial intelligence and games, in particular general game-playing agents, and now forms the basis of the General Video Game Playing AI (GVG-AI) Competition (Perez et al. 2015).

VGDL conceptualises a game as having five components:

Sprite Set The entities in the game, their types and properties. The predefined types are Avatar, FromAvatar, NPC, Static, Movable, Resource and Portal, plus a large number of specialised subtypes, e.g. RandomNPC. The sprites are arranged in a hierarchy, where subsprites (here denoted $\sqsubseteq$) inherit types/properties of their ancestors.

Interaction Set The mechanics of the game. Each interaction rule defines the consequences of a particular pair of sprites overlapping.

Termination Set The win/lose conditions, based on either a timeout or the value of counter, e.g. player health.

Level Mapping A translation from ASCII characters to sprites.

Level Designs Multiple levels can be defined in ASCII, interpreted using the Level Mapping.

A VGDL specification defines the first four elements, with level designs being specified in a set of accompanying text files.

The sprites and rules are described in terms of a predefined ontology, e.g. a sprite can be a RandomNPC which wanders around the screen. The default ontology is fairly limited, and the VGDL versions of well-known titles are often quite simplistic. Nevertheless, it can be used to define an impressive range of games which present a serious challenge for a general game-playing agent, as demonstrated by the GVG Competition examples. In the context of game design, VGDL presents a large space of possible games. Random instances from within that space are unlikely to be coherant, playable games, let alone high quality ones (Barros and Togelius 2014). Hence it also presents a research challenge for automated game design, and researchers are now beginning to look at VGDL generation (Nielsen et al. 2015b).

Blending VGDL

Taking VGDL specifications as our 'conceptual spaces' for games, we wish to show how a computational agent could conceivably create new designs by blending existing game elements. Our proposed approach closely follows the general COINVENT process model:

Select two input games to blend.

Generalise the games by matching selected elements from each, resulting in a *generic game concept*.

Blend the games by combining their constituent elements, reformulated in terms of the generic game concept.

Evaluate whether the (perhaps nonsensical) blend is a viable game.

Weaken the blend by removing/adjusting elements to achieve viability.

Run the blend by elaborating the design to exploit any new structure which it has introduced.

The following account is a thought experiment, but is, we argue, computationally plausible. We expect that the majority of games developed by such a process would not be playable, and indeed there will be many partially developed blends that will fail. For non-trivial examples, each stage presents a huge number of choices — a combinatorial explosion which any automated designer needs to tackle in order to extract playable needles out of a haystack of 'frankengames'. We draw attention below to some of the decisions a computational designer would have to confront.

The term "designer" is used to refer to this imagined agent, but one might also conceive a co-creative approach where a human designer takes some of the key decisions, and perhaps takes inspiration from the blends the system produces.

Previous work in automated game design has suggested that there are few intrinsic criteria for game quality (Nelson et al. 2015). Playtesting, whether by humans or bots, is an essential component of game design. An implementation could use feedback from automated playtesting of the games to guide this process, as in e.g. (Nielsen et al. 2015a; 2015b).

We completely sidestep the difficult issues of blending level designs and blending audiovisual elements, focusing instead on blending game entities and rules. Of course, the design of all these elements is intimitely connected, and a full account of game blending would encompass these as well.

Selection

The first question is which games *should* we blend? Input game selection can be thought of as taking part in some wider game design context: perhaps an agent is elaborating a particular design, and decides to blend in another game concept in order to fix a perceived problem with the original. Alternatively, an agent could be simply exploring the creative possibilities of mashing together diverse game designs.

For this paper, we have just hand-selected two games from the GVG Competition example set: *Frogger* (AKA *Frogs*) and *Zelda* (Perez et al. 2015). Both games are inspired by well-known commercial titles and have simple specifications — the VGDL is shown in Figure 1 and 2 respectively. The level and visual design are both outside the scope of this paper, so for simplicity we have omitted the `LevelMapping` information and `img` and `color` properties. We have also renamed a few sprites and made minor alterations for clarity, without affecting the game mechanics.

It is possible to intuit a lot about how these games work from the VGDL descriptions: the player controls an Avatar, Missiles travel in given direction etc. Due to lack of space, we do not include a full description of the games: see Schaul (2014) for more details, or play

```
SpriteSet
  forest > SpawnPoint stype=log
    dense > prob=0.4 cooldown=10
      sparse > prob=0.1 cooldown=5
  structure > Immovable
      home > Door
      water
      wall
  log > Missile orientation=LEFT speed=0.1
  safety > Resource limit=2
  frog > MovingAvatar
  truck > Missile img=truck
      rightTruck > orientation=RIGHT
          fastRtruck > speed=0.2
          slowRtruck > speed=0.1
      leftTruck > orientation=LEFT
          fastLtruck > speed=0.2
          slowLtruck > speed=0.1

InteractionSet
  home frog  > killSprite scoreChange=1
  frog log   > changeResource resource=safety value=2
  frog log   > pullWithIt
  frog wall  > stepBack
  frog water > killIfHasLess resource=safety limit=0
  frog water > changeResource resource=safety value=-1
  frog truck > killSprite scoreChange=-2
  log   EOS  > killSprite
  truck EOS  > wrapAround

TerminationSet
  SpriteCounter stype=home win=True
  SpriteCounter stype=frog win=False
```

Figure 1: *Frogger* VGDL specification.

them using the Java emulator at `https://github.com/EssexUniversityMCTS/gvgai`.

Generalisation

The generalisation step proposes a *generic game concept* which can be mapped onto elements of the two input games. In the case of VGDL, we need to propose one or more generic sprites, which can then be mapped on to a subset of the two input sprite sets. For example, if game X features *aliens* and game Y *ghosts*, we could propose a generic game concept G with *enemies*. We then have two *signature morphisms*, i.e. mappings from the sprites of G to the sprites of the input games: $\phi(G, X)$: *enemy* $\mapsto$ *alien* and $\phi(G, Y)$: *enemy* $\mapsto$ *ghost*.

Even for quite small sprite sets the number of possible mappings is considerable. We can imagine quite radical mappings, e.g. where a sprite is mapped to both the player avatar and a ghost, or both an alien and an ammo pack, and these clearly have a lot of creative potential. However, for now we make the additional simplifying assumption that the mappings preserve the VGDL sprite categories: Avatar, NPC, Resource etc. So the generic player avatar is mapped to the player avatars in both input games, and so on.

In our running example, the designer could propose the following category-preserving generic sprite set that maps

```
SpriteSet
  door > Door
  key > Immovable
  wall > Immovable
  sword > Flicker limit=5 singleton=True
  movable >
      link > ShootAvatar stype=sword
          nokey
          withkey
      monster > RandomNPC
          monsterQuick > cooldown=2
          monsterNormal > cooldown=4
          monsterSlow > cooldown=8

InteractionSet
  movable wall  > stepBack
  nokey door    > stepBack
  door withkey  > killSprite scoreChange=1
  monster sword > killSprite scoreChange=2
  link monster  > killSprite scoreChange=-1
  key link      > killSprite scoreChange=1
  nokey key     > transformTo stype=withkey

TerminationSet
  SpriteCounter stype=door win=True
  SpriteCounter stype=link win=False
```

Figure 2: *Zelda* VGDL specification.

as follows:

Generic	**Frogger**	**Zelda**
`avatar`	`frog`	`link`
`goal`	`home`	`door`
`wall`	`wall`	`wall`
`enemy`	`truck`	`monster`
`fastREnemy`	`fastRTruck`	`monsterQuick`
`fastLEnemy`	`fastLTruck`	`monsterQuick`
`slowREnemy`	`slowRTruck`	`monsterSlow`
`slowLEnemy`	`slowLTruck`	`monsterSlow`

Note that both input games have additional unmapped sprites, e.g. *Zelda*'s `key` and `sword` don't correspond to anything in the generic game.

Blending the Sprites

An initial version of our blended game — let's call it *Frolda* — can now be generated. The blended sprite set combines those from both games, merging sprites identified by the signature morphisms: this is shown in Figure 3.

In generating this blend, we have retained as much structure as possible from both games: `goal`$\sqsubseteq$`structure` is based on the `door` from *Zelda*, and the `monster` and `truck` sprite hierarchies have been integrated, retaining the `rightTruck` and `leftTruck` sprites from *Frogger* (renamed `rightEnemy` and `leftEnemy`).

The input games used two separate mechanisms to control NPC speed — `cooldown` (rate of action) and `speed` (distance traveled per movement action) — hence all enemies are slower in the blend than either of the inputs. This is an example of a undesirable interaction between game el-

```
forest > SpawnPoint stype=log
   dense > prob=0.4 cooldown=10
   sparse > prob=0.1 cooldown=5
structure > Immovable
   goal > Door
   water
   wall
   key
sword > Flicker limit=5 singleton=True
log > Missile orientation=LEFT speed=0.1
safety > Resource limit=2
movable >
   avatar > ShootAvatar stype=sword
      nokey
      withkey
   enemy > Missile
      rightEnemy > orientation=RIGHT
         fastREnemy > cooldown=2 speed=0.2
         slowREnemy > cooldown=8 speed=0.1
      leftEnemy > orientation=LEFT
         fastLEnemy > cooldown=2 speed=0.2
         slowLEnemy > cooldown=8 speed=0.1
   monsterNormal > RandomNPC cooldown=4
```

Figure 3: Initial sprite set for *Frolda*.

ements that will need to be dealt with later in the blending process.

Two significant decisions have been made during the sprite set blend:

1. The player is a `ShootAvatar`, following *Zelda*.
2. An `enemy` is a `Missile`, following *Frogger*.

This is a form of weakening (see below) which it makes sense to apply at this stage, adjusting the blend to ensure the initial blend is valid VGDL.

Some elements of the input games are unaffected by the blend: the walls, the river system from *Frogger* (`forest/water/log/safety`), and the `sword`, `key` and `monsterNormal` from *Zelda*.

Blending the Rules

We continue the blend by combining the interaction sets: an initial version is shown in Figure 4. All interactions can be included without modification, except for `avatar-enemy`, which reduces the score by 2 in *Frogger*, but only 1 in *Zelda*. By choosing the latter value we again weaken the blend in order to obtain valid VGDL.

Finally, we blend the termination sets. In this example, the structure happens to be identical: the player loses if the avatar is killed, and wins if the goal is ‘killed’ by being touched by the avatar.

Evaluation/Weakening

The initial blend includes all the sprites, interactions and termination conditions from both games, with a unique blended structure defined by the chosen signature morphisms. Adjustments were made to the blend to ensure we had valid VGDL, which introduced choices about how the unique blend should be weakened. The next step is to consider whether we need to further weaken the blend to improve the game’s coherance and playability.

```
goal avatar  > killSprite scoreChange=1
goal withkey > killSprite scoreChange=1
nokey goal   > stepBack
key avatar   > killSprite scoreChange=1
nokey key    > transformTo stype=withkey

movable wall > stepBack
avatar wall  > stepBack
avatar log   > changeResource resource=safety value=2
avatar log   > pullWithIt
avatar water > killIfHasLess resource=safety limit=0
avatar water > changeResource resource=safety value=-1
log   EOS    > killSprite
enemy EOS    > wrapAround

enemy sword  > killSprite scoreChange=2
avatar enemy > killSprite scoreChange=-1
monsterNormal sword  > killSprite scoreChange=2
avatar monsterNormal > killSprite scoreChange=-1
```

Figure 4: Initial interaction set for *Frolda*

First, we can see that the initial interaction set (see Figure 4) contains some redundancies:

- `avatar` and `movable` have the same interaction with `wall`, and $\texttt{avatar}\sqsubseteq\texttt{movable}$. Also note that $\texttt{truck}\sqsubseteq\texttt{movable}$ and if trucks are blocked by walls then they will be prevented from wrapping around, i.e. all the trucks will end up stacked against the wall. Hence, we remove the `movable` interaction and retain the more specific rule.
- `avatar` and `withkey` have the same interaction with `goal`, and $\texttt{withkey}\sqsubseteq\texttt{avatar}$. If we retain the more general rule then the player could always get to the goal, making the key redundant. Hence we again choose to retain the more specific rule, and drop the `avatar` interaction.

Another principle we can apply is parsimony: does the game contain unnecessary elements? This is very much context-dependent. A designer who was working on *Frogger*, but had blended it with *Zelda* in order to add a key- door mechanism might regard the wandering `monsterNormal` as an unnecessary addition. Likewise, a *Zelda* designer who had blended in the ‘crossing the road’ structure from *Frogger* could regard the log/river system as unwanted. If we are interested in mashing up the two games, we could regard any additional structure outside the core of the blend (i.e. the generic shared structure) as unnecessary. Here we adopt the latter position, and remove both the wandering monsters (`monsterNormal`) and river system.

Finally, we may wish to retune values in the blended game, such as the speed of the enemies, which may not be appropriate in their new context. As noted above, the enemies speed in the initial blend is being modulated by both `speed` and `cooldown` values. Here, we decide that we can use the default `cooldown` value.

A final version of *Frolda* is shown in Figure 5. The game

```
SpriteSet
  structure > Immovable
      goal  > Door
      wall
      key
  sword > Flicker limit=5 singleton=True
  movable >
    avatar  > ShootAvatar stype=sword
      nokey
      withkey
    enemy > Missile
      rightEnemy > orientation=RIGHT
        fastREnemy > speed=0.2
        slowREnemy > speed=0.1
      leftEnemy > orientation=LEFT
        fastLEnemy > speed=0.2
        slowLEnemy > speed=0.1

InteractionSet
  goal withkey > killSprite scoreChange=1
  nokey goal   > stepBack
  key  avatar  > killSprite scoreChange=1
  nokey key    > transformTo stype=withkey
  avatar wall  > stepBack
  avatar enemy > killSprite scoreChange=-1
  enemy EOS    > wrapAround
  enemy sword  > killSprite scoreChange=2

TerminationSet
  SpriteCounter stype=goal   win=True
  SpriteCounter stype=avatar win=False
```

Figure 5: Final specification for *Frolda*.

features enemies that move laterally left and right across the screen, but which can be killed by using a sword, and requires the player to collect the key before reaching the goal. Figure 6 shows the game running in the GVG-AI Java emulator, with our choice of sprite images and level design. This level takes advantage of the key-door mechanic to force Frolda to cross the 'road' of monsters twice before exiting the level.

'Running the blend' refers to further elaborating the blended game to take advantage of the synthesis of game elements. For example, we may decide to make the `key` a Missile so that as well as 'crossing the road' of enemies, the player has to pick the key out of the moving traffic.

Alternative Blends

One can imagine many alternative choices to the ones made above. Consider these examples, out of the many possible blends:

- A generic game which doesn't distinguish the speed of enemies, resulting in single-speed monsters which travel left and right.
- A generic game without a unified class of enemies. Trucks and monsters would both be included in the blend as a separate sprites, and the designer could run the blend by inventing new interactions for them.
- Blending `home` with `door`$\sqsubseteq$`structure` was straightforward. But what if *Frogger* had specified `home`$\sqsubseteq$`building`? One option would be to blend `building` and `structure` during the generalisation phase. Another would be to create a new relationship in the blend, e.g. `building`$\sqsubseteq$`structure`. A third option would be to remove, say, `building` entirely.
- When blending the sprite set we could have selected `MoveAvatar` instead of `ShootAvatar`, leaving the sword without a role in the game, to be later removed during weakening.
- Choosing `RandomNPC` enemies instead of `Missile` would have given us a *Zelda*-style free-roaming monsters instead of the horizontally-moving trucks that can be used in level designs to create a dangerous 'road'.

Next Steps

The account above has glossed over many details, although we hope that each step is computationally plausible. Towards this end we intend to implement a basic VGDL blender using the Java VGDL library (Perez et al. 2015). We envision the primary research topic for blended games to surround *experimentation with heuristics*: can we manage the massive combinatorial possibilities to produce a diverse range of non-trivial, coherent, novel, playable games? A subsequent phase of work would integrate the implementation with the COINVENT software stack, as that matures. The COINVENT system currently accepts input in the Web Ontology Language (OWL) or the Common Algebraic Specification Language (CASL), so one likely approach would be to write a translator, for instance between VGDL descriptions and OWL.

A more immediate next step to support system-building is to work out further compelling examples in detail, e.g.

Pac-Man + Zombies The *Survive Zombies* game that ships with the GVG-AI collection has some similarity to *Pac-Man* (Perez et al. 2015). Both games give the protagonist the task of collecting a certain resource while avoiding a population of `Chasers`. However, in *Survive Zombies* another sprite is introduced (bees) which transforms the chasers into honey. One possible blend with *Pac-Man* would allow the protagonist to place fruit that is toxic to ghosts in strategic locations.

Sokoban + Snake The protagonist's tail grows when boxes are pushed to the goal[2].

Sokoban + Sokoban Two copies of *Sokoban* hooked together, so that boxes cleared from one side appear on the other. Perhaps best conceived of as a two-player game, with each player in the role of "Maxwell's demon". VGDL would have to be extended to multi-player games.

Two other directions that we did not address here include *level design* and *audiovisual design* for games. Rather than directly blending the example levels that accompany

[2]It seems challenging to implement Snake in VGDL, since the snake's tail would have to be made of a chain of `chasers` that are programmed to follow each other.

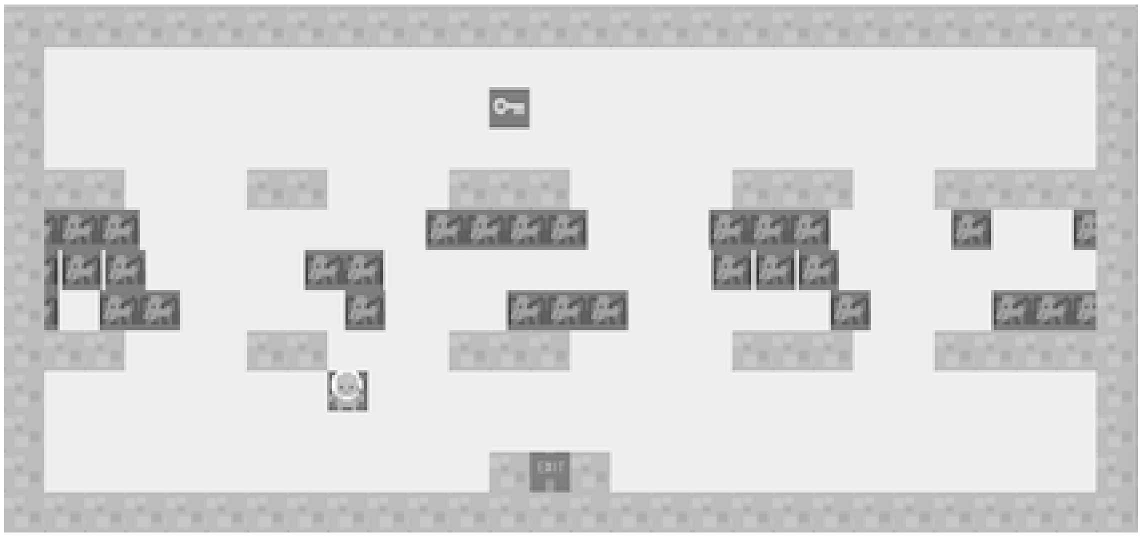

Figure 6: A screenshot of *Frolda* running in the GVG-AI Java emulator. The player (bottom left) has to cross the 'road' of monsters to get the key (top) and then cross again before exiting the level (bottom).

VGDL, we propose to express the *level idiom* in the form of a declarative level generator, e.g. in ASP (Smith and Mateas 2011), and blend these generators in concert with the rest of game description. Other aspects of visual design (e.g. sprites) could be developed through blending, building on recent work in icon blending (Confalonieri et al. 2015). Given its focus on game-playing agents, VGDL does not currently support any audio that could be blended, although it would be relatively easy to extend with simple audio effects to support research in that direction.

Discussion

One aspect in which our account is significantly lacking is how a computational designer can "run the blend" to take advantage of the new creative opportunities afforded by the newly combined elements. Consider the variant mentioned above where trucks and monsters remain separate sprites. How should they interact? We expect our initial implementation will use quite basic mechanisms, e.g. randomly selecting an event (or no event) for the interaction of the two sprites. A more radical move would be to elaborate new mechanics. For instance, monsters could commandeer and drive trucks. To give Frolda a chance to survive in this new *Mad Max* universe, it may be important to give her the opportunity to collect a bazooka that can launch missiles at the trucks. How this kind of elborative process can be realised computationally could be a direction for future research.

The player's imagination does a significant amount of work to make sense of the blend — perhaps especially in a game with limited graphics. Knowledge of the original games may help the player envision the new blended scenario. Thus, if we recognise the two source games, *Frolda* may be envisioned as an anthropomorphic frog equipped with a sword. We have only considered structural blending, but a broader view could include blending of audiovisual elements to achieve these kind of effects.

Brandt and Brandt offer a critique of conceptual blending that can help in thinking about what makes a blended game playable (Brandt and Brandt 2005). They argue that a purely *conceptual* approach to blending is not as cognitively useful as an approach that considers blends in their communicative context. In the case of games, there is both a narrative and an interactive component to the communication. This is consistent with the view from PCG that intrinsic criteria are insufficient for automated game design, and that some form of playtesting is required (Nelson et al. 2015).

Conclusions

We have shown that it is — in principle — possible to provide a computational approach to blending games specified in VGDL, although many such blends are possible. We highlighted the opportunities for emergent meaning making that are afforded to game designers using this approach. Nevertheless, it is difficult to point to anything resembling elaboration, evaluation, or meaning-making within VGDL games themselves: for instance, the observation that a given game is unduly difficult for the player would have to come from outside. Automated playtesting could provide some of the necessary feedback to guide the blending process, and will be integral to the implementation of the approach we have outlined here.

Acknowledgments

Many thanks to Christian Guckelsberger and the anonymous reviewers for their comments on an earlier draft of this paper. Joseph Corneli was supported by the European Commision via the COINVENT project (FP7 FET-Open grant 611553).

References

Barros, G., and Togelius, J. 2014. Exploring a large space of small games. In *Proc. 2014 IEEE Conf. on Computational Intelligence and Games, CIG 2014*.

Bou, F.; Corneli, J.; Gómez-Ramírez, D.; Maclean, E.; Pease, A.; Schorlemmer, M.; and Smaill, A. 2015. The role of blending in mathematical invention. In Colton, S.; Toivonen, H.; Cook, M.; and Ventura, D., eds., *Proc. 6th Int. Conf. on Computational Creativity, ICCC 2015*. ACC. `http://computationalcreativity.net/`.

Brandt, L., and Brandt, P. A. 2005. Making sense of a blend: A cognitive-semiotic approach to metaphor. *Annual Review of Cognitive Linguistics* 3(1):216–249.

Cambouropoulos, E.; Kaliakatsos-Papakostas, M.; and Tsougras, C. 2014. An idiom-independent representation of chords for computational music analysis and generation. In *Proceeding of the joint 11th Sound and Music Computing Conference (SMC) and 40th International Computer Music Conference (ICMC), Athens, Greece*.

Confalonieri, R.; Corneli, J.; Pease, A.; Plaza, E.; and Schorlemmer, M. 2015. Using argumentation to evaluate concept blends in combinatorial creativity. In Colton, S.; Toivonen, H.; Cook, M.; and Ventura, D., eds., *Proc. 6th Int. Conf. on Computational Creativity, ICCC 2015*. ACC. `http://computationalcreativity.net/`.

Cook, M., and Colton, S. 2014. Ludus ex machina: Building a 3D game designer that competes alongside humans. In *Proc. 5th Int. Conf. on Computational Creativity, ICCC 2014*. ACC.

Goguen, J. 1999. An introduction to algebraic semiotics, with application to user interface design. In *Computation for metaphors, analogy, and agents*. Springer. 242–291.

Nelson, M. J.; Togelius, J.; Browne, C.; and Cook, M. 2015. Rules and mechanics. In Shaker, N.; Togelius, J.; and Nelson, M. J., eds., *Procedural Content Generation in Games*. Springer. chapter 6. `http://www.pcgbook.com`.

Nielsen, T. S.; Barros, G.; Togelius, J.; and Nelson, M. J. 2015a. General video game evaluation using relative algorithm performance profiles. In *Proceedings of EvoApplications 2015*.

Nielsen, T. S.; Barros, G. A. B.; Togelius, J.; and Nelson, M. J. 2015b. Towards generating arcade game rules with VGDL. In *Proc. 2015 IEEE Conf. on Computational Intelligence and Games, CIG 2015*. IEEE.

Ontañón, S., and Plaza, E. 2010. Amalgams: A formal approach for combining multiple case solutions. In *ICCBR10: 18th International Conference on Case-Based Reasoning*, volume 6176 of *Lecture Notes in Artificial Intelligence*. Springer. 257–271.

Perez, D.; Samothrakis, S.; Togelius, J.; Schaul, T.; and Lucas, S. 2015. The General Video Game AI Competition. `http://www.gvgai.net/`.

Schaul, T. 2014. An extensible description language for video games. *IEEE Trans. Comput. Intellig. and AI in Games* 6(4):325–331.

Schorlemmer, M.; Smaill, A.; Kühnberger, K.; Kutz, O.; Colton, S.; Cambouropoulos, E.; and Pease, A. 2014. COINVENT: Towards a computational concept invention theory. In *Proc. 5th Int. Conf. on Computational Creativity, ICCC 2014*. ACC. `http://computationalcreativity.net/`.

Smith, A. M., and Mateas, M. 2011. Answer set programming for procedural content generation: A design space approach. *IEEE Trans. Comput. Intellig. and AI in Games* 3(3):187–200.

Treanor, M.; Blackford, B.; Mateas, M.; and Bogost, I. 2012. Game-o-matic: Generating videogames that represent ideas. In *Proc. 3rd Workshop on Procedural Content Generation in Games*. ACM.