Using gameplay semantics to procedurally generate player-matching game worlds

Ricardo Lopes
r.lopes@tudelft.nl

Tim Tutenel
tim.tutenel@gmail.com

Rafael Bidarra
r.bidarra@tudelft.nl

Computer Graphics and Visualization Group
Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands

ABSTRACT

The use of procedural content generation to support adaptive games is starting to gain momentum in current research. However, there are still many open issues to tackle, namely the reusability of methodologies. Our research focuses on reusable and generic methods for linking the procedural generation of 3D game worlds with gameplay, as measured by player modelling techniques. As the interface for that link, we propose the use of *gameplay semantics*, a knowledge representation technique that allows our case-based generator to match content to player models. We present and discuss the implementation of our proposed method in an existing game, *Stunt Playground*. Gameplay semantics is created by designers in a generic way and is then used to procedurally generate player-matching *Stunt Playground* game worlds, both at the design and game stage. Current results show that our approach can automatically create such adaptive game content, thus effectively bridging game world designers, procedural generation and gameplay.

Categories and Subject Descriptors

I.2.4 [**Computing Methodologies**]: Knowledge Representation Formalisms and Methods; I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling

General Terms

Algorithms, Design

Keywords

adaptive game worlds, procedural content generation, semantics

1. INTRODUCTION

Several researchers agree that adaptive games are a valuable contribution for further engaging players in more fun, balanced and effective gameplay experiences [2, 8, 24]. Accordingly, the game industry also shows an increasing interest in dynamic player-centered adaptation, as demonstrated by games like Remedy's *Max Payne*, Valve's *Left 4 Dead* or Bethesda's *The Elder Scrolls V - Skyrim*.

The main idea behind adaptive games is to recognize and comprehend their players' interaction and, ultimately, intelligently adapt game elements, in a dynamic fashion, to improve the gameplay experience. When adaptive in such a way, games can not only favor a more unpredictable and flexible game experience, but also widen its appeal to a larger audience, and even increase its replayability.

In this field, we are especially interested in the adaptation of complex 3D game worlds. Specifically, our focus lies on making the procedural generation of game worlds mutually dependent on individual gameplay. This means that the content generated and offered to players responds to the variation on their performance and behavior. As an example, consider a role playing game where the dungeons being progressively generated include more or less complex topology and props, depending on the measured player performance.

We recently proposed the use of *gameplay semantics* as the interface to match the player behavior and experience with the game content generator [6]. With this semantics, *i.e.* embedded gameplay knowledge about virtual world objects, beyond their geometry, our framework can support the generation of player-matching game worlds, in games where this is applicable and valuable.

This paper demonstrates the suitability and power of this approach, discussing our first results and the implementation of a modification of an existing game, *Stunt Playground*. We explain how our semantic approach can be easily applied to create a game with player-matching game worlds. This is achieved by integrating our semantic framework with a generation process for game worlds and with player and experience modeling techniques. In this case, we added semantic layout solving techniques for generating *Stunt Playground* game worlds and also created new player and experience models. We showcase our results, *i.e.* game worlds generated at game stage, discussing the added value of the new adaptive *Stunt Playground*.

This paper is structured as follows: in Section 2 we briefly survey game adaptivity and semantics. In Section 3, we give a brief overview of our previously proposed semantic generation framework and highlight the new contributions for the actual generation process. In Section 4 we explain the integration of our methods with *Stunt Playground* and the player modeling and content correlation methods. Section 5 presents and discusses our results, preceding our conclusions, on Section 6.

2. RELATED WORK

The architectural principles that should be behind game adaptivity have already been discussed by several researchers [2, 8, 24]. In essence, the players' performance is recorded and used to create a model of the player. Given a game state, these models can also be used to assess and predict the players desired experience of the next game state. Depending on the approach, both behavior and experience models can be used, in conjunction or not, to steer an adaptation and generation engine. This engine adjusts or generates the appropriate game components to better suit the player, *i.e.* adapted to the models' data. Adaptive games typically require this two-step methodology: player modeling and content generation. It is therefore relevant to survey not only adaptation and generation techniques, as done in our recent and more in-depth survey [7], but

also what steers them, *i.e.* behavior and experience modeling, as well as what we propose to support them, *i.e.* semantics.

2.1 Player behavior and experience modeling

With player behavior modeling, gameplay information and metrics are processed to create knowledge about the behavior and performance of the player. Behavior modeling has recently been broadly surveyed in [14]. Several techniques are already considered successful in capturing and modeling some aspects of player behavior. Supervised machine learning has been used to model player skills [3] and preferences [15]. Unsupervised machine learning has also been used: player clustering has been applied to classify player styles and preferences [11]. Other approaches beyond machine-learning have been used in domains where performance is a stronger requirement. Case-base reasoning and heuristic vector-based methods are able to model, respectively, player preferences [13] and player skills [23].

With player experience modeling, gameplay information and the game state are analyzed to determine and predict how the game was or should be experienced. It makes sense to look at these two angles separately: behavior modeling relates to player behavior (execution) and experience modeling relates to player experience (perception). Experience modeling was recently thoroughly surveyed in [24]. The same type of techniques used in behavior modeling can typically be applied to experience modeling. What changes is what is being measured and modeled. Recent contributions have been made to model the difficulty perceived by players and predict the optimal challenge level [3, 16]. More complex experience quantitative models can already predict: fun, challenge, boredom, frustration, predictability, anxiety, [9] or other affective states like engagement [1].

2.2 Adaptation and Generation

Generation methods for the adaptive games domain are still less researched than modeling techniques. It makes sense that modeling methods should be the first ones to establish, since they are the first problem needing solution. However, since player modeling is already establishing itself, some adaptive generation research has already been done. Game elements such as NPC behavior, game narratives, quests and scenarios have been subject to related work and research. We extensively surveyed such approaches in [7].

As for the adaptive generation of game worlds, to the best of our knowledge, research has so far, been exclusive to simple and linear level structures, typically 2D. This tendency is explained by the already mentioned stronger research focus on the player modeling methodologies. Still, procedural content generation has already been proposed for adapting racing tracks, using evolutionary algorithms [17] or platform game levels, using exhaustive search of generated content [12] and recombination of annotated level segments [3].

2.3 Virtual world semantics

Semantics in virtual worlds is a recent research field, raised for the need of more complete and ubiquitous information on virtual worlds and objects. In this direction, smart objects were proposed [4], containing information about the possible interactions that can be executed on them. Peters *et al.* [10] took the notion of smart objects further by creating objects with information about their functionality, how NPCs can interact with them, and where important features of the object are situated.

Our own previous work on semantic modeling explores these ideas further and allows building upon it to generate player-matching game worlds. We define semantics, in the context of game worlds, as all information about the world and its objects, beyond their geometry [18]. This includes object properties, high-level attributes and functional information, as well as interrelationships (geometric, functional, etc) among different objects. Each object of the game world typically carries all its semantics. A semantic library [5], *i.e.* a hierarchical class (relational) database partly based on the WordNet ontology, is responsible for storing all game objects and its semantics. Game designers use this library to specify semantics, atop geometry, on game worlds and its content, in a generic and reusable way.

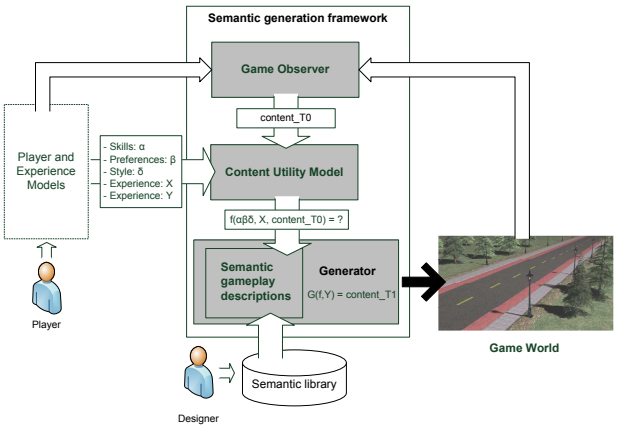


Figure 1: Semantic generation framework: Semantic gameplay descriptions are derived from the semantic library and are retrieved to match the behavior/experience models and correlated content.

Semantics imported from this library can be used as knowledge to automatically control and constrain algorithmic procedures that generate specific world content. This approach has already been successfully deployed, *e.g.* in interior layout solving [20], procedural filters [21] and building generation [22].

3. SEMANTICS AND GENERATION

In a previous article [6], we outlined our proposal for a framework aimed at generating player-matching content for complex and immersive game worlds. In this section we will first summarize this framework, to give context to the rest of this paper. For more details on gameplay semantics and the semantic library, we refer to [6]. In this section we will also outline new contributions on the generation process (layout solving), not presented before.

3.1 Semantic framework

Our semantic generation framework is responsible for integrating behavior and experience modeling with procedural content generation, possible through the use of game world semantics. Fig.1 illustrates the components of the semantic generation framework.

Designers use the semantic library to deploy information on several semantic classes (or entities), *e.g.* objects, attributes or groups. Gameplay semantics is one of the many layers of information that can be deployed. It describes the gameplay value of the different entities and can be specified in terms of: (i) gameplay experiences, (ii) player behavior features and (iii) involved game actors. For example, a designer can specify that a car ramp (the semantic entity) entails a certain level of fun (experience) for a reckless driver (behavior), when used by the player (actor) in a racing game. Gameplay semantics are defined by designers, are reusable and can be as generic as the designer wants them to be.

Semantic gameplay descriptions are automatically converted from the gameplay semantics layer in the library and are inspired by case-based reasoning. They are knowledge containers which encode valid

combinations between semantic entities (the car ramp) and enabled player experiences (fun). They are individually organized by the case preconditions each one applies to, *i.e.* the player behavior features (reckless driver). This way, adaptive content generation becomes a retrieval problem where the solution is to find the most appropriate semantic gameplay description, *i.e.* the best case, for a given moment.

The input of such a retrieval process is given by the behavior and experience models, integrated with the framework. Behavior models supply the behavior features (a certain level of reckless driving) to retrieve the matching gameplay descriptions. Experience models help retrieval by supplying not only the next expected gameplay experience (a certain level of fun), but also the previous measured gameplay experience (*e.g.* was not having fun). Additionally, we developed a game observer component to further help the retrieval process. It correlates which game content enabled the previous gameplay experience. The assumptions are that: (i) if a correlation function associates the measured behavior, the previous gameplay experience and the observed content that enabled it, and (ii) if a semantic gameplay description exists with that same association, then (iii) the remaining content-experience combinations in that description can be considered valid, namely the ones that include the next desired gameplay experience.

In our framework, this generation process can be emergent, since we allow retrieval to be somewhat loose. Although descriptions can have multiple behavior features as case preconditions, we allow retrieval to find descriptions for an input which includes only a subset of those. The same partial retrieval applies for the input correlations of observed content and measured gameplay experience. This means that finding multiple descriptions and combining their retrieved content is possible and increases variability beyond what designers declared. Solving retrieved content is up to the post-retrieval generator.

3.2 Post-retrieval content generation

After retrieval, it is still necessary to synthesize such player-matching content into a meaningful game world (segment). The post-retrieval generation we describe here is responsible for this.

For post-retrieval generation, we use our own previous work on semantic layout solving (see [19] for details). This choice was motivated by two reasons: (i) the input of the semantic layout solver fits very well with the output of the retrieval generation process, and (ii) the solver is already fully integrated with the semantic library. The use of other generators should be possible, as long as the generator can synthesize large scenes from individual content.

Given an initial layout, the semantic layout solver can stochastically position individual objects in that layout, complying with a set of rules. These placement rules take into account what is defined in the semantic library. Each entity in the library includes a semantic representation consisting of *object features*, *i.e.* tagged 3D shapes. The object features we used are *Clearance* and *OffLimits*. A *Clearance* feature cannot overlap with any other features except for other *Clearance* features (shared open areas) and an *OffLimit* feature cannot overlap with any other feature type (solid areas). Additionally, we used object features *back*, *front*, *top*, *bottom*, *left* and *right*, which define the respective 3D shape of each of those object boundaries. These features are used in placement rules to derive valid locations for each entity in any layout. The layout solver includes rules for *against*, *around* and *on*. As an example, if the semantic entity *plate* has an *on* rule with the feature *top* of the semantic entity *table*, the solver places the plate somewhere on the table top.

Each placement rule requires its own individual procedure, within the layout solver, to calculate all valid locations. These procedures are the geometric realization of the respective placement rule and return a group of multiple possible locations. As an example, the individual procedure for the *on* rule uses the Minkowski subtraction between both features of both semantic entities to calculate the polygonal shape that contains all possible valid locations. The layout solver is also responsible for selecting the best order in which to solve placement rules. The order to pick entities to place is defined by a dependency graph, with the following sorted criteria : (i) least outgoing rules to entities not yet picked, (ii) most incoming rules, (iii) most outgoing rules to already picked entities, (iv) largest entity. If semantics are correctly defined in the library, then they, together with the ordering algorithm, are enough to solve layouts, and if there are no available valid locations left, an entity is simply not placed.

3.3 Contributions for semantic layout solving

Regarding the semantic layout solver, the existing object features and placement rules, as defined by designers, are the basis for post-retrieval generation to work. The retrieval process feeds its result, *i.e.* the player-matching content, to the solver who, supported by the semantics of the library, creates a valid game world.

However, upon analysis of the layout solver, we identified additional placement rules and object features necessary for player-matching game world generation. As a result, these were created and added to the semantic layout solver.

The new *empty when placed* object feature defines a shape which cannot overlap with any other object feature, but only at the moment when the entity possessing this feature is being placed. Afterwards, that shape can overlap with any entity or feature. This feature is typically used to express a precondition to a placement rule with another entity. For example, if a ramp should be placed before some obstacle, then the placement of the ramp should consider and reserve some empty space in front of it for future occupation by an obstacle. The implementation of such a new feature builds upon the *Clearance* and *OffLimits* features. For each placement rule and its two semantic entities, a Minkowski sum is used to calculate an area containing all locations for which the *empty when placed* shape would overlap with the already present shapes (objects, *Clearance*, *OffLimits*, etc). This area is then subtracted from the possible placement locations.

The new *follow on* placement rule constrains the possible locations of a source and a target entity. This rule is used to specify that a source entity should be placed in a layout, aligned with an already placed target entity, *i.e.* where both form a trajectory. For example, an obstacle should be placed following on an already placed car ramp, in a certain trajectory. This new rule implied the creation of new additional features: *mid-axis x, y* and *z*. These features define a plane shape which cuts an object in half, in each of the 3D coordinates. For flexibility purposes, we also added similar spaces for cuts in quarters. The new and existing features can be used to define the trajectory of the source and target entities.

The implementation of this new rule involved a new dedicated procedure, easily explained. For a target feature (*e.g.* mid-axis x of a car ramp), a new extruded line is defined by calculating two incremental points (the incremental range is either defined from designers parameters or, if not present, randomly) in the same direction as the feature. The center point of the source entity is then randomly placed somewhere on that line, rotated to respect the source feature plus an optional rotation parameter. Even though this *follow on* procedure constrains the placement of the source entity, there are still some degrees of randomness and, thus, variability: the initial placement of the target entity, the range of the placement line (if not defined by designers) and the selected point in the placement line.

Figure 2: Stunt Playground gameplay

4. PLAYER-MATCHING GAME WORLDS

This article documents the integration of our semantic generation framework into an adaptive game. In this section we will outline its implementation and research issues.

We chose to integrate our approach with an existing game, in order to assess how generic and applicable the semantic generation framework is. The chosen game was *Stunt Playground* [1], illustrated in Fig.2. In this single-player game, players can drive around, free to do stunts in an arena with a variety of props. It is an open sandbox game with no scoring, progression or goals. It includes a game world editor, where the user can create, save, load and play arenas, apart from the ones already included in the game.

For this research, we developed and added a new game mode. In this adaptive game mode, the player starts in a predefined initial arena. After playing a game cycle of five minutes, a new arena is generated each time to better fit the player's style. Unique arenas are generated every game cycle for every player. This new mode entailed the creation of methods for player and experience modeling, content correlation, gameplay semantics and procedural content generation, which will now be described.

4.1 Behavior and experience modeling

As stated in Section 3, our framework needs to integrate behavior and experience modeling, in order to generate player-matching game worlds. For Stunt Playground, we developed our own behavior and experience models. Due to the simplicity of the game mechanics, we opted for heuristic vector-based models, which seem to be effective enough, as demonstrated by Westra *et al.* [23]. All the heuristics described below, for both models, were the result of empirically observing various informal game sessions with several player types.

For modeling player behavior, we defined two simultaneous scales able to capture a stunt driver's playing style. The *Evel Knievel* scale measures how much of a reckless stunt driver a player is. On the other side, the *Sunday Driver* scale measures how much cautious a player is in performing stunts. We used two simultaneous measurements because we aimed to capture overlaps between one behavior and the other. Our initial idea was to investigate the existence of a dual complex behavior where a player might drive around acting as cautious and reckless simultaneously. For example, using only one measurement would be less expressive in characterizing a player who drives very fast, but does not interact with any prop or does any jump.

For both measurements, we use the following heuristics, which are logged and measured at run-time, and stored in a vector:

- i : ratio between the distance spent in the air (jumps) and the total driven distance;
- j : ratio between the time spent in the air (jumps) and the total game time;
- k : ratio between the measured average speed and the maximum possible speed;
- l : ratio between the number of flips which were made, and a maximum number of flips.

Heuristic k is particularly interesting since it hides a fifth measurement. On each arena, the player can choose from a group of vehicles with different maximum speeds, ranging from a bus to a racing car. However, the maximum speed in heuristic k is fixed to the maximum speed achieved by the fastest vehicle. This way, this heuristic indirectly reflects the impact of choosing a faster or slower vehicle, an important factor in characterizing a player. As for the maximum number of flips in heuristic l, they are calculated proportionally to the props available in the arena, with an average of flips per prop per time experimentally determined.

The following formulas define the final values calculated at the end of each game cycle, and determine, respectively, the measurements for the *Evel Knievel* and *Sunday Driver* scales:

$$EK = \frac{\frac{(\sqrt{i}-\tilde{d})}{(1-\tilde{d})} + \frac{(\sqrt{j}-\tilde{t})}{(1-\tilde{t})} + \frac{(\sqrt{k}-\tilde{s})}{(1-\tilde{s})} + 2\left(\frac{(\sqrt[3]{l}-\tilde{f})}{(1-\tilde{f})}\right)}{5}$$

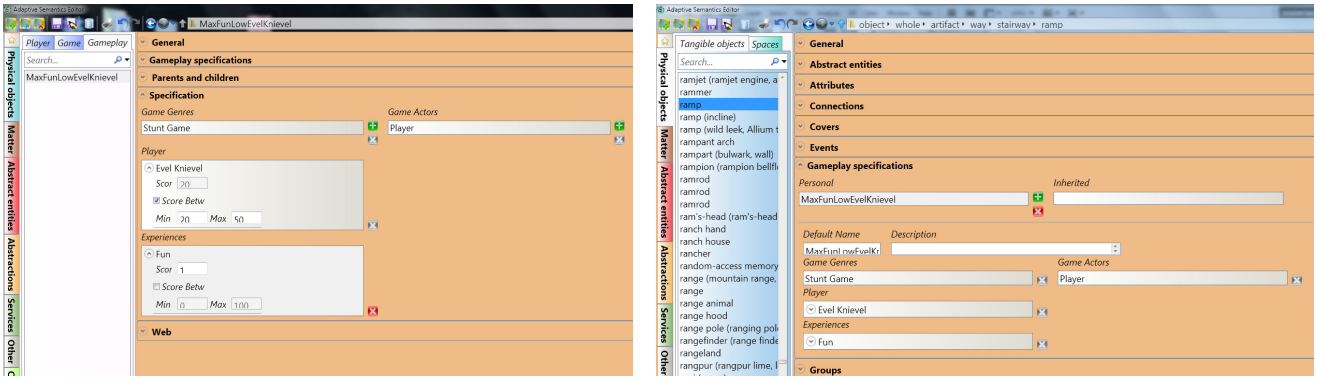
$$SD = \frac{\sqrt{1-\frac{i}{\tilde{d}}} + \sqrt{1-\frac{j}{\tilde{t}}} + \sqrt{1-\frac{k}{\tilde{s}}} + 2\left(\sqrt[3]{1-\frac{l}{\tilde{f}}}\right)}{5}$$

where $\tilde{d}, \tilde{t}, \tilde{s}, \tilde{f}$ represent, respectively, reference values for ratios i, j, k and l, expected for the ideal neutral player and dependent on the props in the arena. Thus, these formulas capture the normalized average deviation from the experimentally determined neutral behavior. We choose to apply quadratic functions to each heuristic to better capture the faster growth rate observed for each measurement. We applied a cubic function and weight 2 to heuristic l to give additional importance to the number of flips, a maneuver which is both harder to perform and a better indicator of recklessness.

Due to the pure sandbox nature of Stunt Playground, we decided that the adaptation goal would be the maximization of the game fun factor, in every scenario. Although fun is a complex concept to measure, the limited nature and goal of Stunt Playground's game mechanics encouraged us to simplify it to a more attainable level. Therefore, for modeling the player experience we defined a measurement for capturing the fun experienced by players. For this, we used the following heuristics:

- m : initial fun value, for finalizing an arena (game cycle);
- n : difference between both behavior modeling scales;
- o : number of re-spawns (if stuck somewhere or bored, the player can choose to be relocated back to the initial position);
- p : ratio between the time spent totally stopped and the total game time.

[1] Stunt Playground was developed by Tim FitzRandolph using the Ogre3D engine. Available as freeware at: http://walaber.com



(a) (b)

Figure 3: The semantic library allows the definition of gameplay semantics blocks (*e.g.* MaxFunLowEvelKnievel in *(a)*), to apply and "tag" different entities (*e.g.* ramp in *(b)*)

Heuristic *m* was fixed as half of the maximum fun value (1), to represent an average game session. Heuristic *n* was introduced to capture an implicit goal of gameplay: to experience the progression towards a gradual specialization in one preferred type of behavior (in this case recklessness vs. cautiousness). The calculation of the fun factor in the end of each game cycle is easily explained: heuristics *m* and *n* are added, and heuristics *o* (normalized against an experimental maximum reference value) and *p* are subtracted from that value. The additions and subtractions are weighted to reflect the experimentally observed relative importance of each heuristic. Additionally, the result is normalized between 0 and 1.

Validation of both the behavior and experience models was performed empirically by both observing game sessions and informal questionnaires. Both models were initially considered valid and expressive in capturing the intended information.

4.2 Correlation with content

As illustrated in Fig.1, the measured behavior and gameplay experience need to be correlated with the game content that enabled them. These correlations are used as inputs for the retrieval of semantic gameplay descriptions. In our previous article [6], we proposed a set of criteria to be implemented by a game observer. Our hypothesis is that these criteria can be sufficient to correctly establish those correlations. The proposed criteria were: (i) content interacted by players, (ii) by NPCs, (iii) by the game engine, and (iv) content which is used in the measurements or game metrics considered by the behavior and experience models (*e.g.* the heuristics above).

In Stunt Playground's case, the implementation of the game observer's criteria was straightforward. The simple nature of the game mechanics, and the behavior and experience models explained above, both show that the content interacted by players is enough to account for the correlations. Therefore, the Stunt Playground game observer keeps track of the arena props a player interacts with and correlates that list with the behavior and experience models' final values.

4.3 Gameplay semantics

Gameplay semantics is defined by designers, using a semantic library editor for that purpose. For Stunt Playground, we defined all the required semantics, not only for the gameplay semantics layer, but also all remaining necessary knwoledge (*e.g.* placement rules and features for semantic layout solving). Although semantics is specified manually, there are some mechanisms in place to prevent this task from becoming too tedious. The use of WordNet, inheritance between semantic entities or the creation of stand-alone gameplay semantics (to then re-apply to different entities faster) are examples of this. Fig.3 illustrates a screenshot of the semantic library editor, *Entika* [5].

For this case study, we created all semantics before designing the player models. The goal was to keep semantics independent from the modeling methods, able to capture all the knowledge the semantic library offers. The only constraint was the mandatory integration requirement: gameplay semantics should use the same vocabulary as what is being modeled. In this case, the player styles for *Evel Knievel* and *Sunday Driver* and a *Fun* parameter.

As explained, we decided that the goal of adaptation would always be to maximize fun. This meant to apply a Fun parameter, at maximum value (100% or 1), to different semantic entities, depending on the type of player they apply to, *i.e.* a value for the *Evel Knievel* or *Sunday Driver* player styles. We also applied a lower fun value to semantic entities, depending if they were not deemed appropriate for the corresponding player styles. We devised this binary behavior (fun vs. not fun) due to the simplicity of Stunt Playground and the research goals for this paper (implement, integrate and test our framework).

We created six levels of player styles: low, medium and high proficiency as *Evel Knievel*, and low, medium and high proficiency as *Sunday Driver*. Each level was quantified as rates: 20% to 50% (low), 51% to 74% (medium) and 75% to 100% (high). These rates were defined following conversations with game designers and players.

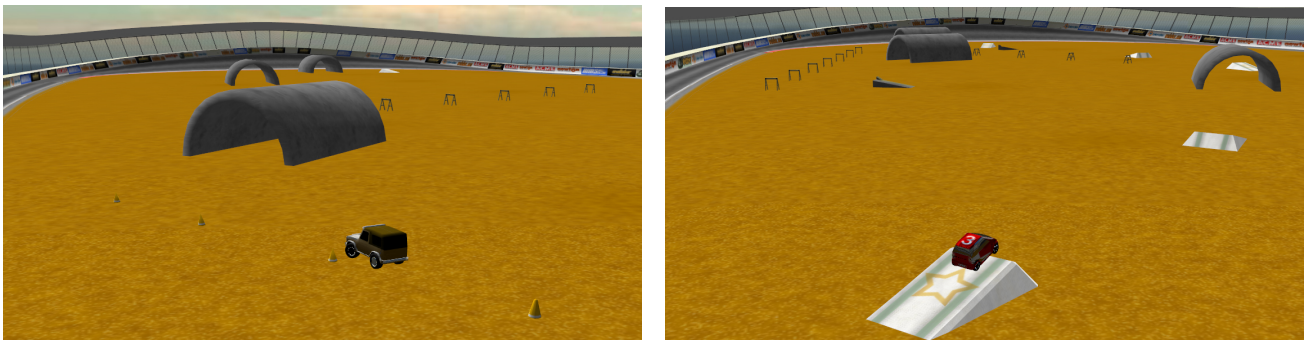
The described gameplay semantics originated six semantic gameplay descriptions. They refer to the six independent levels of player styles and contain different correlations of semantic content and fun levels. Although these descriptions are disjoint in nature (due to the six independent player style levels), they can still be combined during retrieval. This happens because, as explained before, the input of the retrieval process always includes simultaneous player modeling measures for the *Evel Knievel* and *Sunday Driver* styles.

4.4 Generation

We integrated our approach with Stunt Playground (developed in C++) using both the semantic generation framework and the semantic layout solver (both developed in C#). The behavior/experience modeling and content correlation methods were developed directly in Stunt Playground's code (due to their dependence to the game's nature) and acted as input for the retrieval process. In the end of each game cycle (each arena), the models' values are used to retrieve content from the appropriate semantic gameplay descriptions.

Table 1: Behavior and experience model data used as input for the generation of each game world in the corresponding example. The meaning of each parameter is described in Section 4.

	Fig. 4a	Fig. 4b	Fig. 5a	Fig. 5b	Fig. 6a	Fig. 6b
i (m/m)	18.1/1013.5	39/892.6	211.2/1206.9	780.6/1505.2	105/859.6	84.2/851.3
j (s/s)	2.3/300	10.3/300	35.6/300	69.6/300	41.3/300	29.3/300
k (kmh/kmh)	52.7/150	49.1/150	60.7/150	79.4/150	58.2/150	66.3/150
l (#flips/#props)	0/6	0/10	4/14	24/18	1/6	1/6
EvelKnievel	0	0	**0.62**	**0.75**	**0.21**	**0.2**
SundayDriver	**0.79**	**0.65**	0	0	**0.4**	**0.4**
o (#)	6	10	19	12	6	2
p (s/s)	25	18.5	22.3	6.7	15.6	9.1
Fun (before)	0.73	0.7	0.66	0.83	0.49	0.54



(*a*) (*b*)

Figure 4: Stunt arenas generated for players modeled as high (*a*) and medium (*b*) *Sunday Drivers*.

Although descriptions already include knowledge about the quantity of each semantic entity (in this case, stunt props), we also made that quantity proportional to the values of the player models. The semantic entities contained in the gameplay descriptions refer to the same content used in the game since they include the same identifiers and the same model meshes (Ogre3D models, in this case).

After content retrieval, the semantic layout solver is executed to generate a new player-matching stunt arena. The layout solver places the retrieved content within the inputted mesh of the arena (which is always the same), following the semantic placement rules. As explained before, Stunt Playground includes a game world editor to create, save and load stunt arenas. Since this editor uses XML as the format to represent game worlds, we are able to store all generated arenas for each player.

Concerning generation, we developed a game design prototyping tool to generate player-matching stunt arenas [2]. This tool works outside the game, at the design stage, and within a clone of the semantic library editor. Designers can input values for gameplay semantics levels (as explained in the previous subsection) and a matching arena is generated as an XML file. This arena can then be loaded in Stunt Playground and played. This prototyping environment is a valuable contribution for creating game worlds at the design stage, fitted for input player types and game experiences. It can be used both for testing purposes or to deploy worlds in a game where there are no player modeling methods available, but where the player/gameplay profile is known beforehand.

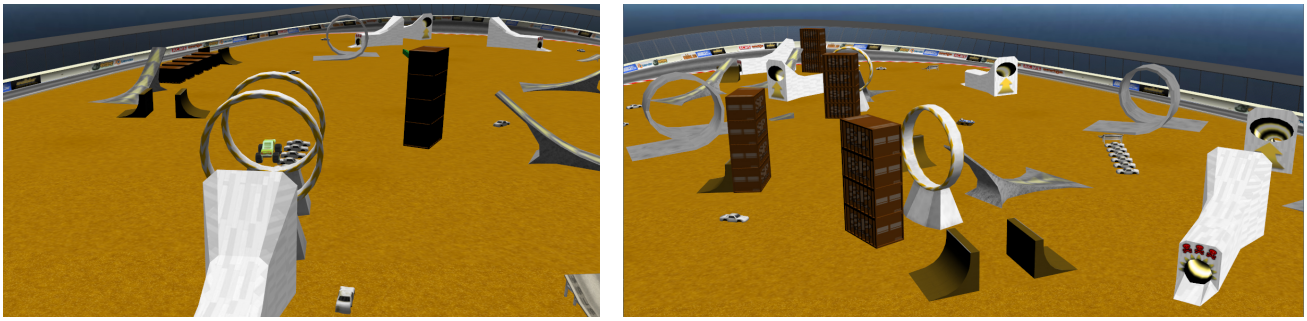
[2]Demo videos of the design tool and the game adaptation available at http://graphics.tudelft.nl/~rval

5. RESULTS

Our results for this case study relate to the success in procedurally generating player-matching game worlds. Due to the lack of space, we decided to exclude analysis of the design prototyping tool since its results are analogous to the game. The difference is that the input for game-time generation is supplied automatically by behavior and experience models and for the prototyping tool by gameplay semantics parameters manually inserted by designers.

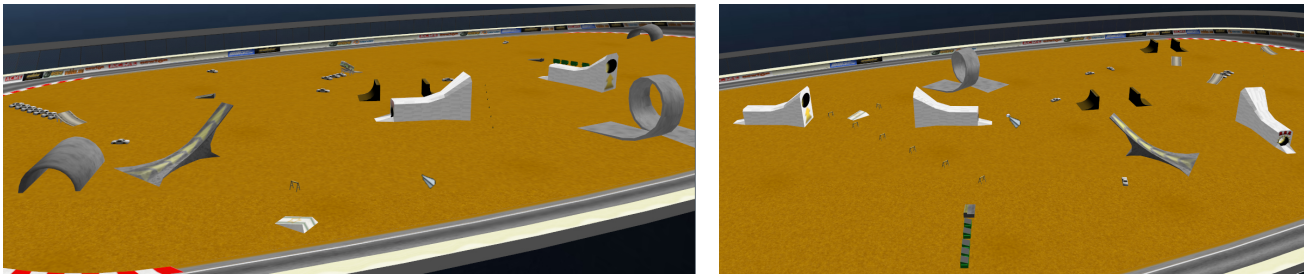
In this section we present our early results, *i.e.* some examples of typical player-matching game worlds generated during gameplay. We saved all generated stunt arenas and logged all the data, both collected from players and calculated by the models. For this section, we showcase some examples that we identified as representative of the global results. Table 1 shows which data matches with the examples chosen.

Fig.4a and Fig.4b show game worlds that were generated in response to a detected non-maximized fun experience of players modeled with, respectively, a high and a medium value of the *Sunday Driver* style. Each of these game worlds used different retrieved semantic gameplay descriptions, containing different semantic content. The main variations, besides the quantity of content (*e.g.* the amount of hurdles), resides in some of the content. The less extreme Sunday Driver arena (Fig.4b) does not include cones, but adds new types of small ramps and new possible layouts for them (notice the liftoff-landing ramp layout in Fig.4b). In these examples, the amount of variability is limited by the number of stunt props available in the game. Of the 15 available props, we only deemed 8 as appropriate for this player type. This ends up not hindering gameplay that much, since the game is naturally biased towards encouraging players to



(a) (b)

Figure 5: Stunt arenas generated for players modeled as medium (*a*) and high (*b*) *Evel Knievels*.



(a) (b)

Figure 6: Two stunt arenas, both generated for a similar input of a player modeled as low *Evel Knievel* and medium *Sunday Driver*

be *Evel Knievels*. Since the goal of the game is to perform stunts, it was unusual for players to assume a Sunday Driver performance, even with less props.

Fig.5a and Fig.5b are examples of game worlds generated for, respectively, a medium and a high value of *Evel Knievel* player style. The variations observed between them are also explained by the retrieval of different semantic gameplay descriptions, which include different quantities and different (more complex) layout possibilities (whereas the props nature is approximately the same). Fig.5b shows the typical upper limits (in terms of complexity and richness) of exclusively *Evel Knievel*-based game arenas, since it tries to maximize fun for a high *Evel Knievel* player. This is also the the performance lower limit of our framework: generation took, in average, 5.1 seconds, a value acceptable while players wait between arenas. This value is mostly dependent on the performance of the semantic layout solver.

Even though low proficiency levels are not exemplified here, the progression of Fig.4a through Fig.5b demonstrates that our framework is able to generate game arenas that are highly dependent on the player behavior and experience. They were generated for different players, but always for the third game cycle (*i.e.* as the second generated arena), showing that the game content at that stage is strongly adapted to how the game was played until then.

Fig.6a and Fig.6b illustrate arenas that were generated for players modeled as part *Evel Knievels* and *Sunday Drivers*. They demonstrate the emergence in our generation framework and the variability of the generator for a similar input. The emergence is possible through the retrieval of multiple semantic gameplay descriptions, for both *Evel Knievel* and *Sunday Driver*, and is illustrated by the presence of content appropriated to both player types. These arenas are emergent in a sense that the rules of their generation (*i.e.* the gameplay descriptions) were not entirely described by designers, but are derived from a combination of those. The semantic layout solver is responsible for the variability between both examples. This is illustrated not only by the quantity of props and their layout (notice the white tunnel ramps on both cases), but also the content that was actually placed, *e.g.* the tunnels of Fig.6a and the hurdles of Fig.6b.

Finally, we point out that these emergent cases did not occur as much as expected. We can identify two reasons for this observation. First, this can be explained by the nature of Stunt Playground, with a gameplay naturally biased towards high values of *Evel Knievel* models. The second reason relates with our player behavior modeling goals. In our case, emergence is only dependent on the dual simultaneous classification of a player as both *Sunday Driver* and *Evel Knievel*. We find two related reasons for this lack of emergence: (i) both measurements (see Section 4.1) are not disjoint enough (as our aim was) and actually act as one single scale, and (ii) players simply do not behave like that. We plan to improve our models to better consider these observations.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we presented our results in applying gameplay semantics within a game, to generate player-matching game worlds. We demonstrated that our semantic-based methods are generic and re-usable enough to be integrated with an existing game, in this case Stunt Playground. This approach, based on designer-defined gameplay semantics, can use procedural content generation to create player-matching game worlds. These game worlds are not only fitted to the players behavior and experience, but also include the expected variability of procedurally generated content. We also confirmed that our generation process can have some emergent behavior, beyond what was specified by designers. Our semantic framework can therefore not only bridge procedural content generation, gameplay and designers, but also add some emergent but controllable aspect to the generation of player-matching game worlds.

As for future work, we intend to perform a formal user evaluation of our Stunt Playground adaptive modification. We are currently integrating our approach with other different games and player modeling methods, in order to investigate: (i) its generic applicability to other domains, (ii) its ability to function with another generator, fully on-line (i.e. at run-time), and (iii) its integration with an existing and more complex player modeling method.

7. ACKNOWLEDGMENTS

This work was supported by the Portuguese Foundation for Science and Technology under grant SFRH/BD/62463/2009. We thank Tim FitzRandolph, the creator of Stunt Playground, for letting us use his game source code, for this research.

8. REFERENCES

[1] G. Chanel, C. Rebetez, M. Bétrancourt, and T. Pun. Boredom, engagement and anxiety as indicators for adaptation to difficulty in games. In *Proceedings of the 12th International Conference on Entertainment and Media in the Ubiquitous Era*, pages 13–17, New York, NY, USA, 2008. ACM.

[2] D. Charles, A. Kerr, M. McNeill, M. McAlister, M. Black, J. Kucklich, A. Moore, and K. Stringer. Player-centred game design: Player modelling and adaptive digital games. In *Proceedings of DiGRA 2005 Conference: Changing Views - Worlds in Play*, pages 285–298, 2005.

[3] M. Jennings-Teats, G. Smith, and N. Wardrip-Fruin. Polymorph: dynamic difficulty adjustment through level generation. In *PCGames '10: Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, pages 11:1–11:4. ACM, 2010.

[4] M. Kallmann and D. Thalmann. Modeling Objects for Interaction Tasks. In *Proceedings of the Eurographics Workshop on Animation and Simulation*, pages 73–86, 1998.

[5] J. Kessing, T. Tutenel, and R. Bidarra. Designing Semantic Game Worlds. In *PCGames '12: Proceedings of the 2012 Workshop on Procedural Content Generation in Games*, Raleigh, North Carolina, USA, 2012. ACM.

[6] R. Lopes and R. Bidarra. A Semantic Generation Framework for Enabling Adaptive Game Worlds. In *ACE '11:* 8^{th} *International Conference on Advances in Computer Entertainment Technology*, New York, 2011. ACM.

[7] R. Lopes and R. Bidarra. Adaptivity challenges in games and simulations: a survey. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(2):85 –99, june 2011.

[8] B. Magerko. Adaptation in Digital Games. *IEEE Computer magazine*, 41(6):87–89, June 2008.

[9] C. Pedersen, J. Togelius, and G. N. Yannakakis. Modeling player experience for content creation. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(1):54–67, 2010.

[10] C. Peters, S. Dobbyn, B. MacNamee, and C. O'Sullivan. Smart objects for attentive agents. In *Proceedings of the International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, 2003.

[11] D. Ramirez-Cano and S. Colton. Player classification using a meta-clustering approach. In *Proceedings of the 3rd Annual International Conference Computer Games, Multimedia & Allied Technology*, pages 297–304, 2010.

[12] N. Shaker, J. Togelius, and G. N. Yannakakis. Towards Automatic Personalized Content Generation for Platform Games. In *Proceedings of the Sixth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, pages 63–68, October 2010.

[13] M. Sharma, S. Ontañón, C. Strong, M. Mehta, and A. Ram. Towards player preference modeling for drama management in interactive stories. In *Proceedings of the Twentieth International FLAIRS Conference (FLAIRS07)*, pages 571–576, 2007.

[14] A. M. Smith, C. Lewis, K. Hullett, G. Smith, and A. Sullivan. An inclusive view of player modeling. In *6th International Conference on Foundations of Digital Games*, Bordeaux, France, July 2011.

[15] P. Spronck and F. den Teuling. Player Modelling in Civilization IV. In *Proceedings of the Sixth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, pages 180–185, 2010.

[16] P. Spronck, M. Ponsen, I. Sprinkhuizen-Kuyper, and E. Postma. Adaptive game AI with dynamic scripting. *Machine Learning*, 63:217–248, June 2006.

[17] J. Togelius, R. De Nardi, and S. Lucas. Towards automatic personalised content creation for racing games. In *IEEE Symposium on Computational Intelligence and Games, 2007. CIG 2007*, pages 252–259, April 2007.

[18] T. Tutenel, R. Bidarra, R. M. Smelik, and K. J. de Kraker. The role of semantics in games and simulations. *Computers in Entertainment*, 6(4):1–35, 2008.

[19] T. Tutenel, R. Bidarra, R. M. Smelik, and K. J. de Kraker. Using semantics to improve the design of game worlds. In *Proceedings of the Fifth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, pages 100–105, 2009.

[20] T. Tutenel, R. Bidarra, R. M. Smelik, and K. J. de Kraker. A Semantic Scene Description Language for Procedural Layout Solving Problems. In *AIIDE '10: Proceedings of the* 6^{th} *Conference on Artificial Intelligence and Interactive Digital Entertainment*, Stanford, CA, USA, October 2010.

[21] T. Tutenel, B. Bollen, R. van der Linden, M. Kraus, and R. Bidarra. Procedural Filters for Customization of Virtual Worlds. In *PCGames '11: Proceedings of the 2011 Workshop on Procedural Content Generation in Games*, New York, NY, USA, 2011. ACM.

[22] T. Tutenel, R. M. Smelik, R. Lopes, K. J. de Kraker, and R. Bidarra. Generating Consistent Buildings: a Semantic Approach for Integrating Procedural Techniques. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3), 2011.

[23] J. Westra, F. Dignum, and V. Dignum. Keeping the trainee on track. In *IEEE Conference on Computational Intelligence and Games, 2010. CIG 2010.*, pages 450–457. IEEE, August 2010.

[24] G. N. Yannakakis and J. Togelius. Experience-driven procedural content generation. *IEEE Transactions on Affective Computing*, 99, 2011.