Shopping for Game Mechanics

Tiago Machado, Ivan Bravi, Zhu Wang,
Andy Nealen, Julian Togelius
New York University
{tiago.machado, ivan.bravi, zhu.wang, nealen, julian.togelius}@nyu.edu

ABSTRACT

Recommender systems are very common nowadays, from shopping websites to social networks, from map routing systems to entertainment stream services. We use recommender systems as an inspiration to create an AI Game Design Assisted tool which recommends game elements, such as sprites and mechanics, during the development process. Suggestions are based on similarities between games and freely inspired by game analysis studies. The tool is based on the Video Game Description Language.

Keywords

Recommender Systems, AI Game Design Assisted Tool, Game Analysis Studies

INTRODUCTION

Recommender systems are very common nowadays. Practically every system has its own way to suggest users to be friends with, movies to watch, or something to buy. It is possible to find recommender systems in the game industry as well, although they focus on suggesting games to the user, based on the games she or her friends played before (newgrounds.com 2016).

In this work, we describe a recommender system designed to assist developers, by suggesting VGDL game mechanics (Ebner et al. 2013). Our methods are inspired by Game Analysis studies, which describe some methods commonly used in games' pre-production phase. Our system uses the games in the GVG-AI framework game library as its knowledge base (Perez et al. 2015). Every time a user requests suggestions, it provides recommendations by comparing the current game with the games in the library. Two types of suggestions are provided based on two search paradigms: the item-based search and the user-based search. In the former, the sprites in each game will be treated independently as items, and ranked by how they fit the user-authored-game. In the latter, each sprite and interaction are considered in the context of its own game, conveying design choices from other games.

The main purpose of this study is to prototype a potential functionality for AI-assisted game design tools, where a system helps a human designer by suggesting game elements and mechanics that might fit well into the game design that is being developed. It is also interesting to compare the recommender system with certain processes in game design, where designers seek inspiration (or get ideas) from other similar games for the games they are developing.

In the following section we present a brief background. Afterwards, a section describes our system, followed by a study. Finally, we discuss our conclusions and future work.

Proceedings of 1st International Joint Conference of DiGRA and FDG

BACKGROUND

Many online services provide content suggestions, like friends, movies, or products. For example, Roth et al. describe a system that gives friend suggestions to users (Roth et al. 2010). Their idea is to cluster contacts into appropriate groups for users, using implicit social graphs like interactions between the users and their contacts, or interactions between groups of contacts. Paquet et al. use a Bayesian generative process to suggest movies to users, by estimating how likely a user will pick a movie, and by predicting the probability that the movie will be viewed or liked (Paquet and Koenigstein 2013). Leino et al. obtain user activities in Amazon online store and use the data to provide shopping recommendations from applied ethnography, on-location interviewing and observation (Leino and Räihä 2007). Our recommender systems provides game mechanics by analyzing pre-designed games.

There are several methods to analyze games. Some of them categorize elements, such as interface, rules, and mechanics (Zagal et al. 2005; Nealen et al. 2011). Others are concerned with how heuristics can improve particular elements in a specific game genre (Rodio and Bastien 2013). Game analysis in general is a recurrent subject in classical game design literature (Salen and Zimmerman 2004; Brathwaite and Schreiber 2009), and it is portrayed as an effective way to learn and understand games (Adams 2013). Normally, a game analysis consists of perform a game breakdown in order to understand it from its atomic elements to its entire design (Schuytema 2007). It consists of perform a game breakdown in order to understand it from its atomic elements to its entire design (Schuytema 2007). This approach is used by designers as a research method to get inspiration for new titles that combine elements from other games, like (Soup Games 2007) whose core mechanics is based on mechanics from similar games inside the genre of tile-matching (Juul 2007).

The recommender system we are proposing works like an automatic game analyzer. It looks for patterns in a game library that matches with a game in development by a user. Our goal is to build an AI-assisted game design tool which can help developers on prototyping creative content by enhancing their capabilities of game's research and analysis. According to (Smith and Mateas 2011) in his work about Computational Caricatures, this system claims to be an asynchronous collaborative tool between humans and machines that modifying a shared design. Examples of similar tools can be found on the work by Davis et al (Davis et al. 2013) in which it is discussed tools and methods to create a Machinima assistant. The main goal is help novice filmmakers to avoid beginner mistakes on their first productions. Strictly related to game creation, we can find systems like Tanagra, that uses rhythm based analysis to assist users with level design for platformer games (Smith et al. 2010). Stream Levels also assists with level generators, however it allows the user to draw strokes on a canvas or upload it from real data (Ferreira 2015). The strokes represent a player trace, and the system uses it to create levels. Another example is Ropossum supports level design for the physics based game *Cut The Rope* using simulations (ZeptoLab 2010; Shaker et al. 2013). Finally *Sentient Sketchbook*, a tool that helps to design maps using map analysis and evolution (Liapis et al. 2013). In addition to generating levels, *Sentient Sketchbook* can assist designers with heat maps that indicate in which map areas the player has a higher probability to win or lose a game. These systems provide feedback and suggestions for level design in topological sense. On the other hand, our system focuses on suggesting mechanical components during the game development process.

A RECOMMENDER SYSTEM FOR GAME MECHANICS

Video Game Description Language - VGDL

VGDL is a game description language that allows users to specify games within a few lines, basically by describing objects and by describing how the objects interact with each other (Ebner et al. 2013). It is a fundamental part of the General Video Game AI framework (Perez et al. 2015), which allows users to create agents for general video game playing. It comes with 60 implementations of VGDL games, and permits that users to add new VGDL descriptions themselves. Most of them are versions of famous arcade or Atari 2600 games, like Pacman and Frogger. These games represent our knowledge base: every time a user requests a suggestion when designing a game, the game is compared with all the games in game library. In VGDL, game mechanics are mostly represented by sprites and interactions. Sprites are the objects associated with types that specify their behaviors. Interactions describe what happens when two sprites overlap (see Figure 1).

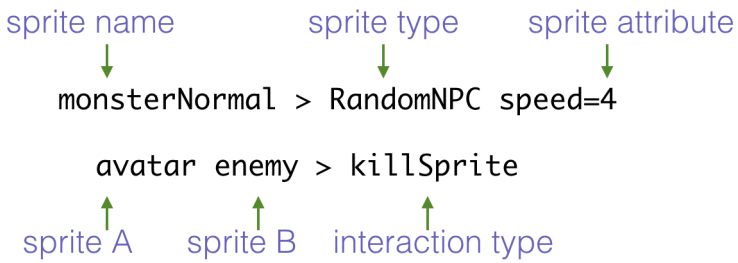

Figure 1: The sprite type definition in the first line means it will move randomly with speed of 4. The interaction type in the second line means that the avatar will be killed when it overlaps an enemy sprite.

Comparing Games and Sprites in VGDL

VGDL represents games as a tree, where the main root always has four children: sprite set, interaction set, level mapping and termination set. Each one plays a specific role in the game description. Our system applies transformations to the original VGDL game, in order to ease the searching process. Among the children described above, our adaptations refer to the sprite set and the interaction set, because they embed the game core mechanics. The sprite set is a collection of sprites, its attributes and type define its behavior. The interaction set is a collection of interactions, which use actions and attributes to specify what happens when two sprites collide with each other. The level mapping represents how the sprites are encoded in the map. The termination set contains the termination conditions of a game.

System core functionalities

The core of our recommender system consists of four simple comparisons that allow for a higher level search abstraction. These comparisons involve sprites and interactions, and are described as follows:

- *sprite type check*: verifies that the type of the two sprites compared is the same. If so it returns 1, otherwise 0;

- *sprite similarity*: checks if the sprite types are the same, if so it computes the normalized vector distance between the parameters vectors, and subtracts the result from a *maximum pre-defined value*. Otherwise it returns a *minimum pre-defined value*. The parameters can either be floating point numbers or sprite name. In the former, it performs a euclidean distance. In the latter, it uses the *sprite type check* comparison on the sprites referenced;
- *interaction similarity*: verifies that the sprite types involved in two interactions are the same, if not it returns the *minimum pre-defined value* otherwise it subtracts the normalized vector distance between the parameters vectors from the *maximum pre-defined value*. The distance is computed as in the *sprite similarity* case;
- *interaction fitness in sprite set*: measures how much an interaction is suitable for a set of sprites;

From the point of view of the recommendation system, we are interested in performing the following comparisons: 1) game similarity measurement; and 2) sprite fitness in a game.

Game similarity measurement

A high level comparison can be done by measuring the similarity between two games. The quality of the recommendation improves as the similarity between games increases, and vice-versa. This is an asymmetric comparison between two games, a reference game and the compared game.

The similarity measurement is a two-fold process. For each sprite in the reference game, we select the similar sprites in the compared game. Then we pick the highest interaction set similarity measurements between the sprites previously selected. Those values are summed and averaged on the number of sprites in the reference game so to equalize between long and short game specifications.

Algorithm 1 depicts the pseudo-code for the above-mentioned comparison.

In this way the system can measure how each sprite in the reference game is represented in the compared game's mechanics.

Sprite fitness in a game

This high level comparison tries to understand how well a sprite and its interactions can fit in a game. The system iterates through the sprite's interactions and sums the *sprite similarity* metric between the sprites in the interaction and the sprites in the game we are trying to fit the sprite into. The result is then averaged by the number of sprites in the game. See the pseudo-code in Algorithm 2.

Sprite Suggestions

The system can provide a set of suggestions which are selected after analyzing the game library using the core functions. Each suggestion is a sprite that comes with its own interactions extracted from the game in which the sprite is declared. Thus the user can choose

Algorithm 1 Game Similarity

```
procedure MeasureGameSimilarity(Game ref, Game other)
    similarity ← 0
    for all Sprite s1 ∈ ref.SpriteSet do
        bestMatch ← 0
        for all Sprite s2 ∈ other.SpriteSet do
            if SpriteSimilarity(s1, s2) >threshold then
                currentMatch ←InteractionSetSimilarity(s1.InteractionSet, s2.InteractionSet)
                bestMatch =max(bestMatch, currentMatch)
            end if
        end for
        similarity ← similarity + bestMatch/ref.SpriteSet.Size()
    end for
    return similarity
end procedure
```

Algorithm 2 Sprite fitness in game

```
procedure SpriteFitnessInGame(Sprite sToFit, Game g)
    fitness ← 0
    for all Interaction i ∈ sToFit.InteractionSet do
        other ← i.OtherSprite(sToFit)
        for all Sprite s ∈ g.SpriteSet do
            fitness ← fitness+SpriteSimilarity(other, s)
        end for
    end for
    return fitness / g.SpriteSet.Size()
end procedure
```

a sprite in the set of suggestions taking all, some, or none of the related interactions. The suggestion set is obtained by applying some selection policies to a vector of sprites, which is ordered according to the system's metrics.

Currently, the system can provide suggestions based on two ranking types: 1) item based ranking; 2) user based ranking.

Item based ranking

This ranking measures the fitness of all the sprites contained in the game library with respect to the game currently being edited by the user. It uses the *sprite fitness in game* metric to rank all the sprites in the library. Then, it applies the selection policies to filter out the most interesting results. The first selection policy takes the sprite with the highest fitness value, in order to exploit the current game's configuration and to suggest the sprite that is more suitable for it. On the other hand, the second selection policy picks the sprite with the lowest fitness, aiming at expanding the current game mechanics.

User based ranking

The user based ranking is more articulate than the above one. It consists of two steps, and treats each game as a user, trying to analyze the user's decisions. So a game from the library is seen as a user that has selected a specific set of game elements in the past. In the first step, games are ranked according to the *game similarity measurement*, so to understand how similar the game design editing is, the system compares with design decisions taken in other games. Afterwards, the selection policies pick the worst and best ranked games: this translates in selecting the most similar and most different user. In the second stage, the sprites from those games are ranked using the *sprite fitness in game* metric. At this point, the selection policies choose the best and worst sprites. What the system tries to do is, on one hand, to select the most appealing design decision taken by someone else in other games; while on the other hand, to select the least expected one, so as to introduce novel design decisions to the game.

The item-based aims to be used when a user wants suggestions about future sprites to add on her design. The user-based purpose is to be used when a user needs to add elements, sprites and interactions that can fit well on her current design choices. The item-based considers only how a VGDL sprite is configured. In the other hand, the user-based considers previous decisions made by users about sprites and its interactions.

To better understand these metrics, see Appendix.

THE GAME MECHANICS RECOMMENDER

This section describes the system's user interface, and provides a user case where the system is used to generate a game based on suggestions.

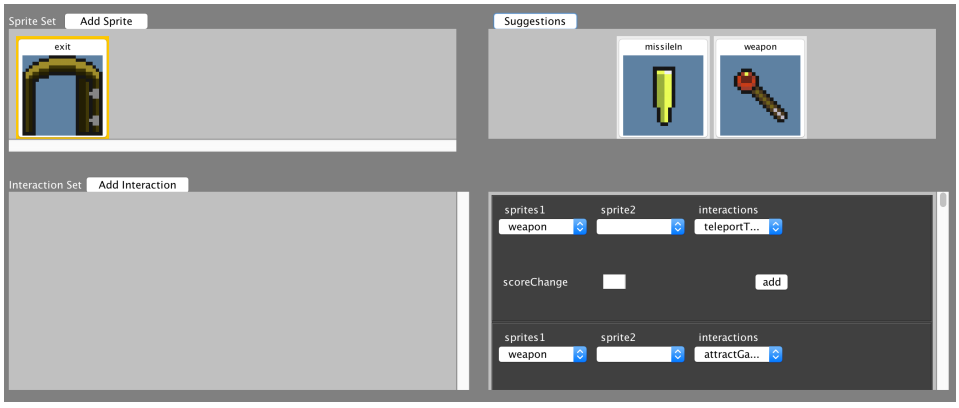

Figure 2: The game mechanics recommender's main interface. On the left, the panel has buttons to add sprites and interactions. On the right, the panel displays the sprites and interactions suggested.

The main interface of the game mechanics recommender consists of two group of panels: one for adding sprites and interactions by pressing the "Add Sprite" and "Add Interaction" buttons respectively; the other group shows the suggestions of sprites and interactions whenever the user presses the "Suggestions" button (see Figure 2).

When the user hits the “Add Sprite” button, a panel with empty fields appears. These fields can be used to specify the sprites parameters, such as speed, orientation and type. In the same way, when an user hits the “Add Interaction” button, a panel with empty fields appears to specify the parameters for the interaction. Every time the user requests a suggestion, the current game is compared with contents from other games in the VGDL repository. When the system finds a game with similar content, it retrieves and suggests sprites and interactions.

When the user accepts a suggestion, an edit panel appears as a pop-up. It shows the fields already filled. However, it is up to the user to accept the suggestion as it is or change parameters according to her convenience.

In this description, a simple game is generated according to the user based search to balance the choices between the best and the worst suggestions. The game’s goal is straightforward: the avatar needs to shoot at walls in order to reach the exit. It contains one sprite provided by the user, and two others suggested by the system. First we accept the best sprite suggested without performing any change. For the second, we pick up the worst sprite. All the interactions are suggested by the system but with some adjustments. The game design is created after suggestions, and the game is designed without any previous concept in mind.

Figure 3 shows the main interface, with the final sample configuration and the game running on a simple level. A sample video showing the interface in action can be seeing here *

Figure 3: The final sample game configuration (left) and it running (right)

CONCLUSION AND FUTURE WORK

In this paper we present an AI-assisted tool inspired by recommender systems and based on the VGDL game library. This system contributes to suggesting mechanics, instead of assisting level design which is a common feature of many AI-assisted tools. This feature can be determinant to solve creativity blocks and to improve productivity. The set of metrics introduced allows to synthesize game features into numbers, an interesting feature, it allows mathematical comparisons and rankings between game elements and consequently games. The tool, however, needs an appropriate user study to validate the quality of the suggestions provided. Moreover, populating the game library with more games will improve its overall quality. Finally, the tool can be expanded to suggest termination conditions and level layouts as well.

*. Link to see the system video demonstration: https://goo.gl/JK0O6e

ACKNOWLEDGMENTS

Tiago Machado is supported by the Conselho Nacional de Desenvolvimento Cientfico e Tecnolgico (CNPQ), under the Science without Borders scholarship 202859/2015-0

BIBLIOGRAPHY

Adams, Ernest. 2013. *Fundamentals of game design.* Pearson Education.

Brathwaite, Brenda, and Ian Schreiber. 2009. *Challenges for game designers.* Nelson Education.

Davis, Nicholas, Alexander Zook, Brian O'Neill, Brandon Headrick, Mark Riedl, Ashton Grosz, and Michael Nitsche. 2013. "Creativity support for novice digital filmmaking." In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems,* 651–660. ACM.

Ebner, Marc, John Levine, Simon M Lucas, Tom Schaul, Tommy Thompson, and Julian Togelius. 2013. "Towards a video game description language." *Dagstuhl Follow-Ups* 6.

Ferreira, Lucas N. 2015. "StreamLevels: Using Visualization to Generate Platform Levels."

Juul, Jesper. 2007. "Swap adjacent gems to make sets of three: A history of matching tile games." *Artifact* 1 (4): 205–216.

Leino, Juha, and Kari-Jouko Räihä. 2007. "Case amazon: ratings and reviews as part of recommendations." In *Proceedings of the 2007 ACM conference on Recommender systems,* 137–140. ACM.

Liapis, Antonios, Georgios N Yannakakis, and Julian Togelius. 2013. "Sentient Sketchbook: Computer-aided game level authoring." In *FDG,* 213–220.

Nealen, Andy, Adam Saltsman, and Eddy Boxerman. 2011. "Towards minimalist game design." In *Proceedings of the 6th International Conference on Foundations of Digital Games,* 38–45. ACM.

newgrounds.com. 2016. `http://www.newgrounds.com/`.

Paquet, Ulrich, and Noam Koenigstein. 2013. "One-class collaborative filtering with random graphs." In *Proceedings of the 22nd international conference on World Wide Web,* 999–1008. International World Wide Web Conferences Steering Committee.

Perez, Diego, Spyridon Samothrakis, Julian Togelius, Tom Schaul, Simon Lucas, Adrien Couëtoux, Jeyull Lee, Chong-U Lim, and Tommy Thompson. 2015. "The 2014 general video game playing competition."

Rodio, Florentin, and JM Bastien. 2013. "Heuristics for Video Games Evaluation: How Players Rate Their Relevance for Different Game Genres According to Their Experience." In *Proceedings of the 25th IEME conference francophone on l'Interaction Homme-Machine,* 89. ACM.

Roth, Maayan, Assaf Ben-David, David Deutscher, Guy Flysher, Ilan Horn, Ari Leichtberg, Naty Leiser, Yossi Matias, and Ron Merom. 2010. "Suggesting friends using the implicit social graph." In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining,* 233–242. ACM.

Salen, Katie, and Eric Zimmerman. 2004. *Rules of play: Game design fundamentals.* MIT press.

Schuytema, Paul. 2007. *Game Design: A Practical Approach.* Sirsi) i9781584504719. Charles River Media.

Shaker, Noor, Mohammad Shaker, and Julian Togelius. 2013. “Ropossum: An Authoring Tool for Designing, Optimizing and Solving Cut the Rope Levels.” In *AIIDE*.

Smith, Adam M, and Michael Mateas. 2011. “Computational Caricatures: Probing the Game Design Process with AI.” In *Artificial Intelligence in the Game Design Process.*

Smith, Gillian, Jim Whitehead, and Michael Mateas. 2010. “Tanagra: A mixed-initiative level design tool.” In *Proceedings of the Fifth International Conference on the Foundations of Digital Games,* 209–216. ACM.

Soup Games. 2007. *High Seas.* Copenhagen.

Zagal, Jose, Michael Mateas, Clara Fernández-Vara, Brian Hochhalter, and Nolan Lichti. 2005. “Towards an ontological language for game analysis.” In *Proceedings of the 2005 Digital Games Research Association Conference (DiGRA).* Vancouver, Canada, June.

ZeptoLab. 2010. *Cut The Rope.* London.

Appendix

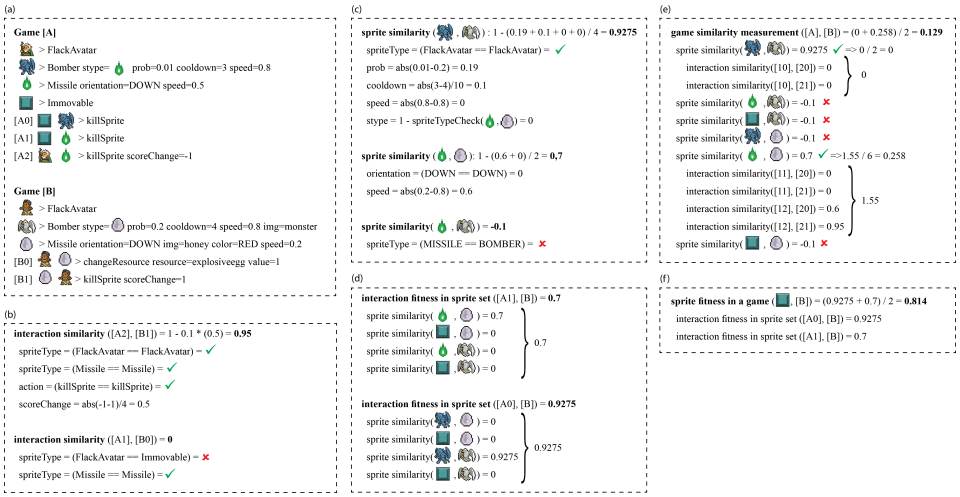

Figure 4: How to compute metrics using two short games.

In (a) we can see the partial implementations of two games, [A]: Aliens and [B]: Eggomania taken form the GVG-AI framework. In (b) are shown three examples of computation of *interaction similarity*, if the sprite types of the two interactions do not match the metric is 0; otherwise if also the action matches, the distance of the parameters is computed. In (c) is computed the *sprite similarity*: whenever the two sprite types match we compute the distance, otherwise we return -0.1 . (d) shows how to measure on what degree an interaction is suitable into a game [G]; for each sprite from the interaction we compute the *sprite similarity* with all the sprites from the game [G], these values are summed together. In (e) we can see how to compute the *game similarity*; for each coupling of sprites from [A] and [B] we compute the *sprite similarity*, whenever this value is over a certain threshold (0.6) we compute the *interaction similarity* and sum those values together. Finally, (f) show how to compute the *sprite fitness in game*, it consists on summing the *interaction fitness in sprite set* of the interactions of the sprite to measure, and the interactions of the game. The values obtained from *game similarity* and *sprite fitness in game* are not a percentages, they are used as heuristic to compare similarities. This is why they are averaged on the number of sprites in game [B] so not to penalize games with less sprites.