





Creating Layouts for Virtual Game Controllers Using Generative Design

Gabriel F. Alves, Anselmo A. Montenegro , and Daniela G. Trevisan ^(✉) 

Universidade Federal Fluminense, Niteroi, Rio de Janeiro, Brazil
gabrielferreiraalves@id.uff.br, {anselmo,daniela}@ic.uff.br
<http://www.ic.uff.br>

Abstract. Video game controllers have a high influence factor on players, as they are responsible for the fun, motivation, and personality of a game. The organization and arrangement of the buttons are one of the relevant factors when developing new controllers since they are responsible for serving as an input of actions within the games. This work presents the construction of a generative design model to support game designers finding different and innovative layouts of virtual controllers for their games. The generative design produces many valid designs or solutions instead of one optimized version of a known solution. This solution was developed by linking genetic algorithms to generate a large number of layouts and machine learning techniques (SVN) to classify individuals between valid and invalid, seeking to facilitate the exploration of the design space by the designer. The tests performed sought to measure the variability of the results generated by the proposed model, showing that several solutions of different controllers with different configurations can be developed for a game.

Keywords: Generative design · Gamepad · Virtual controller · Genetic algorithm · Machine learning

1 Introduction

With the advancement of technology and thanks to its dynamism, touchscreen devices like smartphones and tablets have also become game controllers: the virtual gamepads. They are dynamic because they have components and devices such as Bluetooth and wireless, accelerometers, gyroscopes and cameras. These components may contribute to the development of different forms of interaction with digital games [2], either in games played on the mobile devices themselves or in games played on another device. Because these controllers can access device components by touching the screen and/or by moving and rotating the device, the buttons on the virtual controller can follow the rules of the game, so it is not necessary to include several buttons that will not be used during the gameplay.

Granted by CAPES, Brazil.

© IFIP International Federation for Information Processing 2019

Published by Springer Nature Switzerland AG 2019

E. van der Spek et al. (Eds.): ICEC-JCSG 2019, LNCS 11863, pp. 237–249, 2019.

https://doi.org/10.1007/978-3-030-34644-7_19

Unfortunately, this is not possible in physical controllers, because once the controller has been built, all the buttons will always be available to the user, even if they have no use in certain games. Each game has a finite number of actions and interactions that are not necessarily the same as in other games and can vary between different genres of games, such as action and adventure games; sports and shooting; race and platform; and even within the same genre of games. These actions are usually associated with a single button on the controller. Therefore, there will be a maximum number of buttons required to perform all interactions in the game. On the other hand, generative design is a promising approach to solve various aspects related to User Interface [15]. It offers new opportunities for conception and creation problems in areas such as engineering, architecture, and design, making it possible, through the use of meta-heuristics, to determine the layout of UI components, informing only a limited amount of parameters and producing new and sometimes, unexpected results [15].

Based on all this knowledge, the goal of this work is to develop a tool that can help game designers to find different and innovative layouts of virtual controllers for their games, since a virtual controller can be implemented especially for a specific game, containing only the actions necessary for it. Therefore, it is not the purpose of this paper to provide layouts for physical video game controllers. Thus, we defined a generative design model [4], where the designer must tell to the system what actions the player should be able to perform within the game, the number of layouts he/she wants the model to deliver as output and how many iterations the algorithm will perform before constructing these layouts. A genetic evolution-based approach has been used to generate a huge variety of layouts and, based on information added to a pre-trained database, the model can generate and suggest a set of valid controller configurations for the desired game. A machine learning approach has been applied to filter the most usable and appropriate set of layouts to be presented to the game designer. This approach was chosen because the generative design provides a wide range of solutions, but it is not feasible for the professional to analyze all of them. Based on the generated and filtered results, the designer will be able to choose which one suits his/her game and, if necessary, make adjustments and refinements to its design. A generative design-based solution can be used as a robust starting point where the results can be refined by software engineers and artists [15]. Therefore, it is important to point out that this tool is a software that provides creative support to the designer, the true responsible for the development of the controller of his/her game.

2 Related Work

In order to help game designers to develop one or more layouts of ideal and innovative virtual controllers for their games, we sought to find approaches, techniques, and methodologies that would aid in the process of creating ideas and supporting the design process. After researching in academic libraries such as Google Scholar and the ACM library, no results were found on the use of

Generative Design applied in the area of digital game controllers, so this section will present works and knowledge that have been served as the basis for the development of our solution.

Usually, the virtual gamepads follow some rules and are created based on existing organizational elements in physical controllers, such as the grouping of directional buttons, the possibility of incorporation of d-pads, the existence of buttons action, among others. Looking for solutions to support the design process of game controllers, some researchers have tried to use a set of Design Thinking techniques, such as a day in the life, empathy map, persona, ideation workshop and paper prototyping to design a game controller that pleases its players [1]. The developed prototypes have sought to soften and even eliminate the problem of the lack of tactile feedback in touchscreen devices by using personalized skins over smartphones. However, although the work relates to this in the way they look for new solutions that support the design process, no software support for the design process has been provided. Baldauf et al. [3] has developed four virtual controllers based on existing controllers configurations. A comparative laboratory study was conducted where four gamepad smartphone designs were selected. Each was tested in two popular games: Pac-Man and Super Mario Bros. The construction of different configurations of virtual controllers for games is the main relationship between our work and that of the authors, although our work aims to assist and give creative support to the designer, and not to create the controller itself. Regarding approaches that use artificial intelligence techniques to support the design of game controllers, we point out the work of Torok et al. [14] that deals with the design of adaptive game controllers for touchscreen devices. Their solution introduces an adaptation that derives the user's personal preferences from a series of basic events, such as button presses or internal gameplay changes. The main disadvantage of this approach is that the controller layout adaptation takes place during the player interaction time. At times, many changes on the fly to the controller layout may confuse the player, but on the other hand, this can be used by the designer as a way of adding different levels of difficulty to their game. Our work bears similarities to the work of Torok et al. [14], mainly in the quest to find solutions for virtual controllers. However, his work builds the solutions in interaction time, while this work produces them in time of design.

In his work, Krish [8] proposes a Computer-Aided Design (CAD) Generative Design model suitable for complex multi-criteria design problems in which important performance criteria are incomputable. This method is based on the construction of a genotyping project within a parametric CAD system based on history and then its parameters undergo random variations within predefined limits to generate a set of distinct projects. After the designs are generated, they are filtered through multiple constraint envelopes, representing geometric feasibility, manufacturability, cost, and other performance-related constraints, thus reducing the design space in a smaller, feasible design space, represented by a set of different designs.

By contrast, many of the technologies that masquerade as generative design such as topology optimization, lattice optimization, parametric optimization or similar technologies are focused on improving a preexisting design, not creating new design possibilities as in the generative design. The confusion arises because the inputs to generative design are similar to the inputs to many optimization tools. However, generative design produces many valid designs or solutions instead of one optimized version of a known solution. Genetic Algorithms are metaheuristic methods inspired by natural selection [5] and are unquestionably the most dominant in computational design exploration [8].

3 Methodology and Implementation

The architecture of the proposed model consists of three stages: parameter specification, exploring the design space (by the genetic algorithm) and filtering the most relevant solutions (by machine learning). In the first part, the user - a game designer - informs the model of the essential parameters for the creation of layouts that will be given as results. The model, in its current state, receives the following parameters: the number of buttons, the actions that these buttons execute within the game, the number of layouts that should be generated by the model and the number of iterations it should run. The second part is responsible for creating layouts based on the parameters entered by the user. These layouts will be randomly defined at the beginning of the model execution, pass through genetic operators, will be evaluated by a fitness function and some of the selected individuals will generate the next population. The algorithm will perform these operations n times, where n is the number of iterations defined by the designer. At the end of the iterations, the final set of layouts, also called the final population, will be defined. Many individuals may be produced at this time. The third and last part of the model was precisely defined so that it was possible to present to the designer just those who had some use for him/her. Therefore, the machine learning Support Vector Machine (SVM) algorithm was used to classify individuals between valid and invalid, separating them so the user could analyze a smaller range of results.

The solution development was done using the Java-based language called ‘Processing’ [11], highly recommended in the generative design community because it is a free, easy-to-use and flexible software sketchbook and language. Figure 1 illustrates the pipeline of the proposed generative design model which is detailed in the following subsections.

3.1 Input Parameters

In order to start the model’s development, it was necessary to carry out a previous study on games, their styles and the ways of controlling them. It is said by Rogers [12] that all games fit into a previously established genre, and a game can fit into one or more genres, where one is dominant. An example is the game

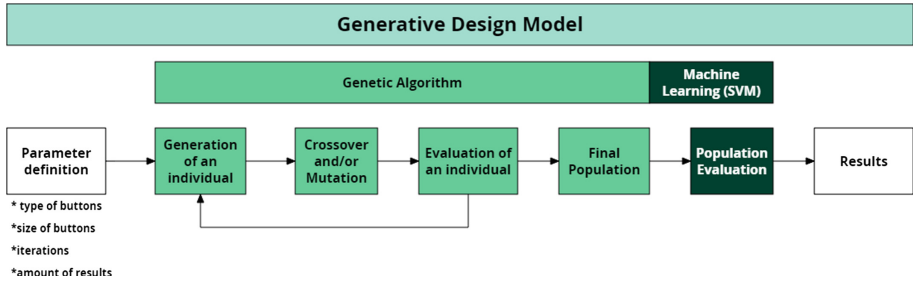


Fig. 1. The proposed generative design model pipeline.

The Last of Us [10] released for PlayStation3 and remastered for PlayStation4, classified as an action game containing features of a puzzle and shooting game.

To classify a game in a given genre, it must be possible to perform some specific actions during gameplay. For example, one of the genres that a game can be classified is the “Platform” genre. In Platform games, the player must be able to perform the actions “walk”, “run”, “jump” and “act”, the latter being a variable determined by the game designer, such as throwing an item, triggering a device or performing a hit.

With the knowledge about game genres, it was possible to list the actions that can be performed within them. These actions - from moving to pausing a game - are the first input parameters that the designer must inform the model. Other defined parameter is related to the genetic algorithm: population size and the number of iterations.

3.2 Exploring the Design Space

In order to explore the design space for the creation of virtual controllers, we used a genetic algorithm. To understand the logic of the system, it is necessary to understand some terms, such as ‘population’, ‘individual’, ‘fitness’ and ‘roulette’. A population consists of two or more individuals, while an individual consists of a set of configurations of a controller. By controller configuration, we are talking about a layout and its components, such as the number of buttons, button types, button classifications and their positions (X and Y axes) on the screen. For the development of the system, some data structures were defined. In the data structure associated with each individual, we define three arrays called chromosomes, and each of them stores essential information for the creation of individuals:

- chromosomeX: contains the X-axis positions of all buttons requested by the designer;
- chromosomeY: contains the Y-axis positions of the buttons;
- classification: contains the classification of the button (the role that such button assumes within a control, what action it assumes).

The most basic button classes defined in the model are presented in Table 1.

Table 1. Basic button classes and actions.

Classes	Buttons			
Directional buttons	Up	Down	Left	Right
Action buttons	Jump	Shoot	Run	Defend
Extra buttons	L1	L2	R1	R2
System buttons	Start	Select	–	–

The size of the chromosomes is defined at the beginning of the process, when the user informs the number of buttons that the controller should have, making it possible to perform the actions within the game. In addition to these three vectors, each individual also carries some attributes:

- fitness: a numerical value responsible for representing how apt an individual is to follow to a new population;
- fitnessPercent: the fitness value represented in percentage;
- rouletteTrack: the range that the individual holds over the roulette of individuals' selection. This will be detailed later.

When an individual is created, the position chromosomes containing the X and Y positions of the buttons are filled with random values, causing each button to assume a random position within the controller area and the classification chromosome is filled with labels for the actions listed on Table 1. The value of the fitness attribute is defined by the fitness that an individual has to maintain in the next generations of the algorithm. Let L be a layout candidate, i.e., an individual in the population, with n buttons. Let bx and by be two arrays that store, respectively, the x and y coordinates of each button i , $0 \leq i < n$. Moreover, let us define bt as an array that identifies the type or class of a given button i . The objective function that defines the fitness value of a layout L is given by:

$$f(bx, by, bt) = \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} \left[\left(\frac{1 - \delta(bt_i, bt_j)}{l} \text{dist}(bx_i, by_i, bx_j, by_j) \right) + \alpha \delta(bt_i, bt_j) \left(\frac{1}{1 + \text{dist}(bx_i, by_i, bx_j, by_j)} \right) \right] \quad (1)$$

where $\delta(bt_i, bt_j)$ is 1 if buttons i and j belong to the same class and 0 if they are in different classes. The Euclidean distance between two buttons is given by $\text{dist}(bx_i, by_i, bx_j, by_j)$, $l = \sqrt{(w+h)}$ is a normalization factor, where w and h are the width and height of the layout, and α is a weight. In our model, we set the α value as six. The first term in the sum is responsible for creating a separation between buttons from different classes, and the second term tries to approximate

buttons in the same class. Having defined a fitness value for the individual, the percentage relative to his fitness is calculated and, according to it, the rouletteTrack is defined. The roulette is a technique of selecting individuals. Through it, all individuals have attached to them a range of a roulette wheel, this range is defined based on their fitness: individuals with higher fitness values will have larger range bands in it, thus having a greater chance of being selected to be part of the new population. Although individuals with lower fitness scores have smaller roulette intervals, they still have a chance of being selected to participate in the new population, causing diversity among individuals. Determining the roulette tracks for each individual is done using the fitnessPercent value to assign the ranges: (1) the first step is to sort the population in ascending order using the fitnessPercent values; (2) after that, each individual is linked to a roulette track - the first receives the initial range, which ranges from zero to his fitnessPercent. The next individual will go from the fitnessPercent value of the previous individual to the sum of this value with their percentage; (3) until 100% of the roulette wheel is distributed. With fitness and fitnessPercent values calculated and having defined the ranges of roulette that each individual possesses, the execution of the genetic operations are triggered. By the number of iterations required, the following steps will be performed: (1) a new population is defined; (2) for each individual in the population, a value between 0 and 100% will be drawn. The individual whose track in the roulette contains such value will be selected to undergo the following operations; (3) mutation operators may be triggered to modify the individual; (4) the individual will be added in this new population and old individuals will be removed.

In this work, we did not use any Crossover operators. Instead, we used two mutation operators with 50% probability of a mutation type 1 to occur, while the probability of a mutation type 2 occurring is 80%. The first mutation is responsible for changing the position of any button within the generated layout. This operation calculates an offset vector in any direction and moves the button to a maximum distance of 100 pixels from its original position. The second mutation is responsible for exchanging the position of one or more buttons with other buttons on the same layout - for example, the directional button ‘up’ exchanges its position with the ‘jump’ action button.

Finally, it is necessary to check if there is no collision between the buttons, that is if one button does not overlap another within a layout. At the end of n iterations, the final population will be defined and can proceed to the next step: filtering the design space using the Support Vector Machine (SVM) algorithm.

3.3 Filtering the Design Space

The search for a solution that applied labels to the results given by the genetic solution was necessary because evaluating all the generated layouts is a tiresome task and the objective function used in the genetic algorithm does not capture all aspects of good design. This was a strategy used to keep the originality and variability of the individuals generated. By embedding too much rules and

constraints in the objective function we could drastically reduce the variability of the individuals.

Each button in a controller carries with it a semantics that gives a meaning to its presence in the layout. An ‘up’ directional button is expected to be positioned above the other directional buttons, which makes the interaction between player and controller more dynamic, easy, and intuitive. Thus, the classification of individuals from different populations was performed considering the position of the buttons in the controller area, the groupings between them and the distances between the buttons of the same class - directional, action or system buttons. This knowledge, originated in the designer rationale, is learned by the Support Vector Machine (SVM) classifier during the training set.

Based on a previously trained database (*training set*), the SVM algorithm can classify other databases (*test set*), automating the process of selecting individuals. For the model, two classifications are made: valid individuals and invalid individuals. The first are those that may be chosen by the designer as candidates for game controller configurations, given the position of the buttons in the controller area, while the second classification is formed by layouts that do not represent configurations that can be used by the final user. After that, the model displays the results for the designer, separating them according to the classification that the SVM has produced. To training the dataset to classify three-button layouts, 301 individuals were labeled manually by a designer, where 150 were labeled ‘valid’ and 151 as ‘invalid’. The SVM training resulted in correct classification of 259 individuals, 86.05% of the entire population. Another population with 306 individuals were manually classified to construct the dataset responsible for classifying controllers configurations with nine buttons. From that, 131 individuals were classified as ‘valid’ and 175 as ‘invalid’. At this case, 273 individuals were correctly classified, corresponding to 89.21% of the settings.

4 Tests and Results

The experiments were carried out to find out if there is any relation between the number of iterations and the number of valid individuals generated by the model and, in addition, to measure the variability of the results. It is important that the model provides a range of useful results for the designer and that the layouts are not all similar to each other.

First, it was necessary to define which genre of games we will generate the layouts of the controllers. We assume that the game designer wants to build virtual controllers for games of the platform genre, where some specific actions must be made available to be executed in it. One of the controllers should have only three buttons: two directional buttons (left and right) and one action button (jump) and another controller setting should have nine buttons: the four directional buttons (up, down, right and left), four other action buttons (jump, shoot, defend and run) and a system button (start). After choosing the configurations as a function of the number of buttons we define the number of iterations of the evolutionary algorithm and the number of individuals in the final population. Ten different types of iterations were defined: 1000, 2000, 3000, up to

10,000 iterations. The number of individuals was established as 1000, 2000 and 3000 individuals for each controller configuration with three and nine buttons. A total of 60 populations were obtained. After creating all the settings, each population was submitted to the evaluations of the SVM classifier that, according to a previously trained database, classifies each layout between ‘valid’ or ‘invalid’. In this case, specific classification databases were used for three and nine buttons (Fig. 2).



Fig. 2. Samples of layouts with three buttons classified as ‘valid’ (first row) and invalid (second row).

4.1 Variability of Layouts

The variability implies the diversity of results so that it is possible to test several valid configurations and to find those that better suit the designer’s need to create an ideal controller for his game. That said, we seek to find a solution that can group similar layouts to make it possible to measure the variability of the valid results generated. The K-means [7] clustering algorithm proved to be a possible solution to our problem because it is an algorithm that partitions data into clusters. The K-means clustering algorithm is a commonly used method for automatically partitioning a data set into k groups [16]. The purpose of data grouping is to discover the natural grouping of a set of patterns, points or objects so that it is possible to do analyzes and absorb knowledge intrinsic to the data [7]. The more traditional definition of this technique is not able to determine the ideal number of clusters, and it is necessary to previously inform the number of clusters that it is desired to separate the data. To define the number of clusters and find the number of clusters that might be ideal for measuring the variability of results, we use a K-means extension that uses the Bayesian Information Criterion (BIC) [6]. It is based in part on the likelihood function and can be used to compare models with different parametrizations, with different number of components, or both. Therefore, we use this technique to find the ideal number of clusters and, finally, to separate layouts into similar clusters. As an input to the algorithm, we provide a CSV file containing the positions on the x and y axes of each button, concatenated with the distances

between each of the buttons. The algorithm performs clustering and organizes layouts that have been output by the machine learning classifier. It has been observed that both the number of clusters and the number of valid individuals oscillate when the number of iterations varies for each population. This can be explained in part by the fact that the objective function is not able to capture all aspects of a good layout, so this result justifies the use of machine learning to help the designer identify a good design. The stochastic nature of the selection of the individuals can propagate, albeit less likely, individuals with low fitness for subsequent generations, which can then be wrongly classified as valid by the SVM since the technique does not guarantee 100% accuracy in the classification of individuals. One measure that can help us understand the nature of the layout generation process is the ratio between the number of clusters and the number of valid individuals. The largest the ratio, the more efficient is the process is in creating variability inside a group of high quality individuals. As we stated earlier, we are interested in presenting a range of results that are distinct from each other to the end-user. Figure 3 graphically illustrates the calculated ratios in each population created by the model. Analyzing the graphs, we can see that, in most cases, the ratio of clusters by valid individuals is higher in the population of 1000 individuals. This happens in all two configurations containing different amounts of buttons, except in the following iterations: iteration 4000 in layouts with three buttons; iterations 4000, 6000, 7000, and 10,000 in the layouts with nine buttons. As this behavior occurred in a few cases, most likely caused by the randomness of the genetic algorithm when generating and selecting individuals, we realized that a population with 1000 individuals is sufficient to generate sets of valid individuals with greater variability.

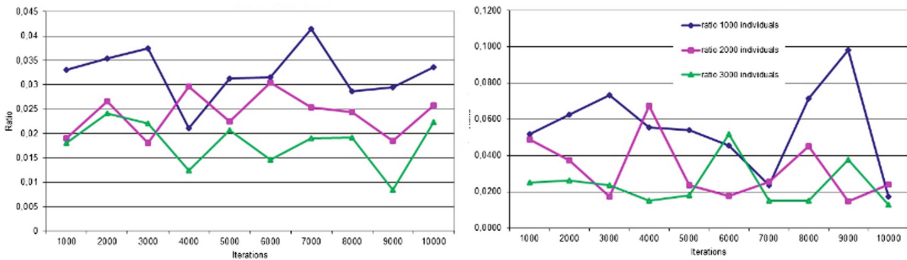


Fig. 3. Graphical representation of the value of the ratio between the number of clusters and the number of individuals classified as valid in a population of 1000, 2000 and 3000 individuals - layouts containing three (on the left) and nine (on the right) buttons.

Figure 4 shows examples of layouts generated by the model and grouped into certain clusters (smaller figures) and its potential layout proposed by a designer (bigger figure) after analyzing those results. At the end of the tests, we look for other designers’ opinions about the ‘ideal layouts’ constructed after analyzing the model outputs. Five game designers, four male and one female, aged 18–30

years analyzed the four candidate layouts. Those designers have between one and five years of experience with game design. The layouts were presented to them on a smartphone ASUS Zenfone 3 Zoom with a 5.5-inch screen. Their opinion was requested about those layouts, asking them to consider the arrangement and grouping of the buttons in the virtual controller area. In short, the layouts that separated the directional buttons into two groups divided opinions among the designers. The position of the action button in the three-button settings has been reported as too far away, making it difficult to combine with other actions. There were also comments on the user’s finger range and button height.

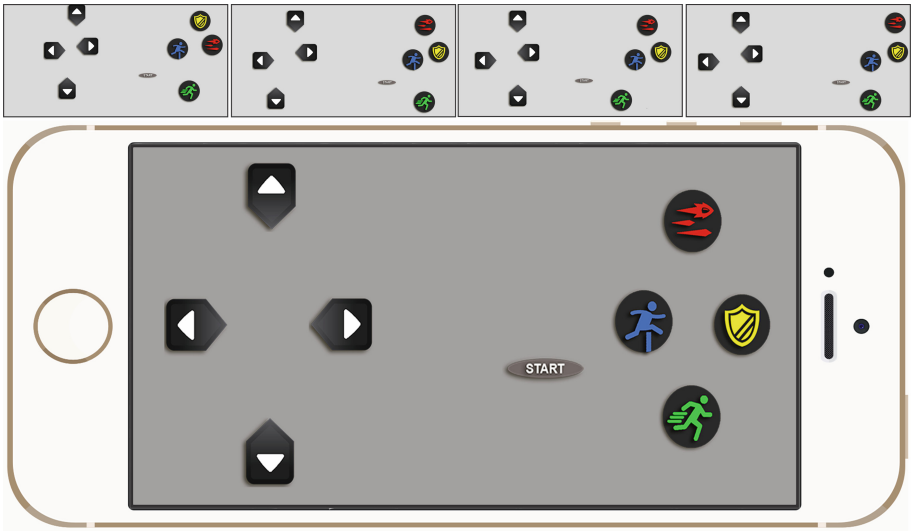


Fig. 4. Layouts with different semantic values inserted in the same cluster because of geometric proximity.

5 Conclusion and Future Work

In this work, we addressed the problem of assisting a game designer in the process of creating different gamepad layouts using generative design. The main contribution of this work is a generative design pipeline for gamepad design based on evolutionary algorithms and machine learning techniques. The evolution algorithm is responsible for creating a large number of gamepad configurations based on the inputs provided by the user: the number of layouts to be generated, the number of iterations to be executed, quantity, and type of buttons that should be present in the final layout. After execution, the model outputs the exact number of layouts requested by the user. As this is generally a large number, it becomes impractical for the designer to analyze them all. As certainly many of these will be discarded by him because video game controllers must respect some organizational rules, we a Support Vector Machine classifier to divide individuals as

‘valid’ or ‘invalid’. In addition, we propose a way of measuring the variability of the candidate solutions using an extended version of the K-means algorithm, which can automatically determine the number of clusters in the group of individuals. Static database training can be seen as a limitation. Since the data that fed the training base of the SVM classifier have been classified only once, the outputs of the generative design model are, to a certain extent, restricted to those that the classifier knows to be valid or not. In future works we intend to implement a self-feeding of the classifier training base so that it is possible to further increase the variety of results that can be classified as valid. Regarding the analysis of the variability of layouts, the technique used may erroneously group some layouts of controllers in the same cluster, as shown in Fig. 5. This occurs because the K-means algorithm does not interpret the semantics present in videogame controllers since it analyzes only the positions of the buttons within the controller’s layout space. In this case, the layouts placed in the same cluster are very similar when considering the positioning of the buttons within the gamepad area, but in the context of the player experience they are very different, and that can generate a high cognitive effort on the part of the user when trying to perform actions within a game. Looking to address this issue, in future work, we intend to apply techniques such as graph matching approaches [9] or computational vision-based techniques (e.g image subtraction) [13], to improve the way we can measure variability of the solution.

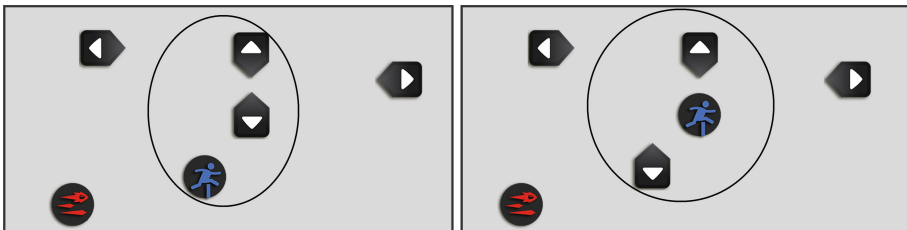


Fig. 5. Individuals inserted in the same cluster, however these layouts are not similar because they have buttons with different semantics.

References

1. Alves, G.F., Souza, E.V., Trevisan, D.G., Montenegro, A.A., de Castro Salgado, L.C., Clua, E.W.G.: Applying design thinking for prototyping a game controller. In: Clua, E., Roque, L., Lugmayr, A., Tuomi, P. (eds.) ICEC 2018. LNCS, vol. 11112, pp. 16–27. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-99426-0_2
2. Baldauf, M., Adegeye, F., Alt, F., Harms, J.: Your browser is the controller: advanced web-based smartphone remote controls for public screens. In: Proceedings of the 5th ACM International Symposium on Pervasive Displays, pp. 175–181. ACM (2016)
3. Baldauf, M., Fröhlich, P., Adegeye, F., Suetete, S.: Investigating on-screen gamepad designs for smartphone-controlled video games. *ACM Trans. Multimed. Comput. Commun. Appl. (TOMM)* **12**(1s), 22 (2015)

4. Bentley, P.: *Evolutionary Design by Computers*. Morgan Kaufmann, Burlington (1999)
5. Bjørlykhaug, E., Egeland, O.: Mechanical design optimization of a 6DOF serial manipulator using genetic algorithm. *IEEE Access* **6**, 59087–59095 (2018)
6. Fraley, C., Raftery, A.E.: How many clusters? Which clustering method? Answers via model-based cluster analysis. *Comput. J.* **41**(8), 578–588 (1998)
7. Jain, A.K.: Data clustering: 50 years beyond k-means. *Pattern Recogn. Lett.* **31**(8), 651–666 (2010)
8. Krish, S.: A practical generative design method. *Comput. Aided Des.* **43**(1), 88–100 (2011)
9. Oliveira, A., et al.: An efficient similarity-based approach for comparing XML documents. *Inf. Syst.* **78**, 40–57 (2018)
10. PlayStation: The last of us (2018). <https://www.playstation.com/en-us/games/the-last-of-us-remastered-ps4/>. Accessed 30 Dec 2018
11. Reas, C., Fry, B.: *Processing* (2018). <https://processing.org/>. Accessed 21 Dec 2018
12. Rogers, S.: *Level Up! The Guide to Great Video Game Design*. Wiley, Hoboken (2014)
13. Russ, J.C.: *The Image Processing Handbook*. CRC Press, Boca Raton (2016)
14. Torok, L., Pelegriano, M., Trevisan, D., Montenegro, A., Clua, E.: Designing game controllers in a mobile device. In: Marcus, A., Wang, W. (eds.) *DUXU 2017*. LNCS, vol. 10289, pp. 456–468. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-58637-3_36
15. Troiano, L., Birtolo, C.: Genetic algorithms supporting generative design of user interfaces: examples. *Inf. Sci.* **259**, 433–451 (2014)
16. Wagstaff, K., Cardie, C., Rogers, S., Schrödl, S., et al.: Constrained k-means clustering with background knowledge. In: *ICML*, vol. 1, pp. 577–584 (2001)