Experiments on Computer Game Development Methodology

Andrei Gabriel Morar
Technical University of Cluj-Napoca
Cluj-Napoca, Romania
andreigabrielmorar@gmail.com

Dorian Gorgan
Technical University of Cluj-Napoca
Cluj-Napoca, Romania
dorian.gorgan@cs.utcluj.ro

ABSTRACT
This paper describes the methodology of creating an interactive computer game. It analyzes the necessary phases by which the functional and non-functional requirements related to a computer game are transformed into a final product. The presentation of the game development methodology is based on YAMS interactive multiplayer dice game. The description of each development phase includes both different implementation ideas for some features and changes based on the objective analysis of the previous phase.

Author Keywords
Interactive Games; Unity; Development Methodology.

ACM Classification Keywords
H.5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous.

General Terms
Human Factors; Design; Methodology.

INTRODUCTION

Interactive games have experienced a significant evolution in the themes and interaction models, due to the evolution of graphics and generally the high processing and storage capacity of the devices that such applications can run. Game scenarios are often complex, and the objects scene can easily simulate the real environment. However, although the complexity of the games has increased significantly, certain user experience requirements have to be respected in order for the game to reach its goal, namely to attract and maintain as many users as possible [1].

Also, the development of a game requires that specialists in areas such as computer science, art, media design and business work together on the same goal. This means they need a common methodology to guide them. They need to identify very clearly the development phases of such a project and understand the requirements that video games must meet in order to be able to have real success on the market.

In the development process of a game, it is important to always make a parallel between the complexity of the functionalities and the usability degree of the application. A game creation methodology aims to ensure the dependence of these two steps at every stage of development, regardless of the platform the game is addressed to (PC, gaming console, mobile phone).

Usability is a highly important component that should be considered in all the methodology steps of development. According to [2], usability is the extent to which a specific set of users manage to use a particular product to achieve specific goals, thereby guaranteeing the user a high level of efficiency, effectiveness and satisfaction.

Efficiency refers to the ratio between the amount of resources required to perform a task by the system and the extent to which users are able to accomplish their purpose through those actions. Effectivness refferes at the accuracy of certain tasks which are performed by users. Satisfaction is the degree of comfort and acceptance that users experience using the product, which is a subjective measure, often difficult to predict.

The purpose of this paper is to present a methodology for developing video games and to prove the importance of using it. This article proves the benefits of using an iterative game development methodology. The main goal of this article is to explain and exemplify the cyclical steps of a complete development methodology that begins from general specification of a game theme and results in a finished product that meets high usability standards.

In the next chapter of this paper there are presented the main steps necessary for a game development methodology, along with the advantages of using an iterative method of implementing the steps. The general phases that make up the methodology are described too.

In the chapter "Experimentation of the Development Methodology" there are given relevant examples from the development process of the YAMS game. It will show the advantages that the methodology has brought in the composition of the game scenarios and in establishing the interaction between the users and the game.

The "Methodology Evaluation" chapter describes a process for evaluating the quality of the methodology used for game implementation. There are presented the main improvements that this methodology has brought to the development of the game.

The last chapter will contain various relevant conclusions related to the process described in the game development methodology.

Related works

The description of a complete methodology for developing interactive games is described in [3]. The paper outlines the importance of a unified methodology through which a team of specialists from different fields can collaborate to carry out a joint project. This article introduces the creation of a series of educational games called "Candy Depot" designed to help general education students to improve their knowledge base in various chapters of mathematics. The article presents the benefits of using a methodology to simplify the interaction between target audience (general school students) and application functionalities. In the article, it was noted that the complexity of the development process was reduced in the case of the first version of the "Candy Depot" game for which a development methodology was used. The article also presents decisions to improve the development methodology, based on the analysis of the interaction between the first versions of the application and the users.

About YAMS

YAMS is a multiplayer dice game. It is divided into rounds and in one round each player is entitled to three consecutive throws of the dice[4] . Five dice are used for the game and their throws aim at creating a game configuration. The configurations are predefined and represent different combinations of dice that result from their discarding: simple, full, four-card, five-of-a-kind (similar to the different categories found in the poker game). If a player makes a valid configuration, then the predefined score for that configuration is passed to a score table. Otherwise, the player chooses another category that is marked with 0 points. The game is finished when all the rounds have been exhausted (there are no categories to be dotted). The player who scores the maximum is declared winner.

GAME DEVELOPMENT METHODOLOGY

According to [5], the development of any game must take into account the following major phases: pre-production, production, and post-production.

The pre-production phase includes the deciding process of the main scenarios and task which would be used by the players in order to accomplish the application required functionalities. On the business side of the project, it may include requirements for marketing strategies.

The production phase involves the actual implementation of the game. As long as the main functionalities has already been established, this phase is centered on the implementation of the scenarios directly into the game scenes, along with sound and graphics. This stage involves planning decisions (establishing methods for interacting between objects in game scenes, the flow of certain operations that make up a scenario, decisions about user interaction with the game).

The post-production phase involves testing, marketing, and game advertising. In this last phase, a usability validation of the implementations from the previous phase is done. This ensures that the game meets the originally set requirements and at the same time can be marketed to the target audience (checking compatibility of the game with the notion of usability).

The three main phases described above aim to transform game requirements into an implementation that focuses on game scenarios and can be accepted by a final user. The process focuses on getting the correct gaming scenarios, detailing their implementation in various objects scenes and checking them to find out the impact that the game may have on the market.

The development of an interactive game with a high level of complexity encounters various problems when it comes to implementing the phases described earlier through a classic abstraction. The use of the Waterfall model [6] is inappropriate because it requires explicit assessment of the requirements and then they are solved in the following stages (production and post-production) through clear and preset procedures. Scenarios can change as game developers receive feedback from test users. Also, some details about the interaction between objects can be adapted, so a more appropriate method in developing an interactive game is proven to be one that accepts regular changes in planning and development.

Agile development methods [7] are more suited to game development as it allows changing pre-production specifications. As during production, it can be concluded that certain tasks or scenarios are planned inefficiently, agile allows for an initial version that can sustain further developments in an iterative way. Designing and solving certain implementation details (graphical, functional, interaction details between the user and the application) can be included in the iterative modification process. The feedback received from test users is otherwise much easier to be integrated into the production process of the game. This method relies on the idea that scenarios, tasks and details of object scenes can be planned, implemented and tested in incremental cycles. Pre-production does not define a complete version of the game to be implemented at production level and later tested in post-production. Agile development methods are better suited to the need adaptive games (to the usability requirements).

The methodology I chose to use in implementing YAMS is based on an incremental development of the game, based on a set of initial requirements that have been set according to the rules of the game. The phases of the methodology are based on the three principal stages: pre-production, production and post-production. The development system chosen for the YAMS game, however, divides development

cycle of the project into a larger number of phases, including: establishing the main theme and initial requirements (pre-production), developing tasks and game scenarios, designing scenes and interaction techniques, development of interaction and control algorithms (production), heuristic evaluation of the game (post-production). Each of the production and post-production phases begin with a feedback received from a group of evaluators who analyze the previous scene and write recomandations about the implementation details that should be redesigned or totally changed in a future iteration cycle. This is a very important step in every phase and should not be neglected in order to ensure a high degree of usability for the game.

Establishing the main theme and initial requirements

The first phase focuses on determining the main theme of the game and what are the fundamental requirements that the application needs to focus on. This is important because the main part of the implementation will derive from this. Even though the details of planning and implementation can change in a cyclical way, erroneous setting of basic requirements will result in an application that is inconclusive to the idea of the game. From this stage, it is determined whether the game creation project is feasible, and it contains the requirements that the final application must meet. Also this describes the target audience of the game, so developers take into account the user profile when setting interaction methods. Requirements do not have to cover all the gameplay from the beginning but must clearly define the main functionalities that users should be able to find in the game. Also, details about the device specifications on which the game is running, how users can use the application (single player, multiplayer) are set in this phase. For a successful ending of this phase the team that plans and develops the game needs to understand the rules of the game. Determining the main requirements derives to a great extent from these rules.

Developing tasks and game scenarios

In the previous phase, a set of main requirements and certain details related to the typology of the target audience of the application have already been established. This can determine some of the actions that game users should be able to access in a first release. The game's operating rules and set of requirements established in the first stage serve as input for this phase. It is advisable to start this phase directly with checking the previous one. The team planning the basic tasks and scenarios must have a set of initial requirements that cover the rules set for that game. In other words, the description of the game from the first phase must be found in the set of requirements, otherwise the tasks and scenarios required for a first version can't be fully accomplished. It is not necessary for the first phase to fully describe the game (changes may be made along the way, depending on the feedback received from test users), but the specifications in the initial description must be fully covered. This requires that users who validate the first phase verify the extent to which the initial requirements cover the description of the game. In case of incompatibilities or incomplete description, some aspects of the first phase need to be restored (or clarified) before the planning and development team can move to the second phase.

The tasks set up at this stage are simple actions. A set of tasks which are performed into a certain order compose a scenario. A task is usually lead to the execution of a simple command (clicking, pressing a button or a group of buttons that performs an indivisible action). An action associated to a task cannot be split into sub-actions. These tasks must cover the needs indicated by the set of initial requirements. The set of tasks resulting from this phase allows the creation of scenarios that cover the description of the game from the previous stage. The task description should also specify the interaction between the user and the system. It must be specified which commands should be accessed by the user (and in what order, if there are several) to execute the action associated with the task. It will also be specified whether the task execution will generate a particular response from the system (a text message displayed on the screen, a particular sound).

During the iterations of this phase, it is highly recommended to create a prototyping of the scenes in which scenarios based on tasks will be developed. The prototype does not need to be very precise, its purpose being to help developers more easily understand the context in which certain game features will unfold. By making a prototype it can be much easier to find out which tasks are to be performed for each scenario and especially what would be their logical execution order.

As soon as we have a set of basic tasks and a prototype of the game scenes (at least the main scene of the game), it is possible to create scenarios that satisfy the functional requirements of the application. Scenarios will step-by-step describe how a user can interact with certain objects in the object scene. This objects should be present in the prototype, even though their design and specifications will be highly different. Also, the series of actions the user will execute (or that will notice as being executed by the system) will be represented as a set of tasks, in the logical order of their execution.

The interaction metaphors should be chosen in such a way that the development of a scenario is intuitive and similar to certain actions of the same type that the user knows from everyday life. A scenario should be divided into tasks in such a way that the logic of execution is intuited by the user, even if that user is not experienced enough. Scenarios of high complexity are usually easier to be understood when the tasks of which they are composed of sub-scenarios groups, divided by a certain element of the scene. Thus, adding intermediate objects to the prototyping of the game scene can help reduce the complexity of some

scenarios (the first sub-scenario will result in a set of simple actions performed over some objects from the scene which eventually interact with the link object, while the second sub-scenario will take place through tasks which act directly on that object).

Designing scenes and interaction techniques

It's important to determine from how many scenes the game will be composed. Their number may vary depending on the platform for which the game is designed, but the complexity of the game and the number of game-related options contribute decisively to this detail. Typically, games are designed in such a way that the user has a first contact with the options the application offers. The easiest way to do this is by creating a menu for the users to understand what the general options are (how to access the actual game scene, how to change the settings in order to gain a better playing experience, how to see details related to the rules of the game or how to access the history of the user's results in the game).

The main scene of the game must be built in such a way that the end user of the application can identify in an intuitive manner what are the objects needed to perform the specific actions related to the game theme. A user must also be able to understand which are interacting metaphors used to control those objects. The first decisions related to the creation of a scene are centered on establishing an environment in which the scenarios described in the previous stage can be implemented as easily as possible. The details linked to the graphic design of the objects in the scene are a little less important for the first iterations of this phase. The accent at first should be put on creating objects that can be placed and interact with other objects in such a manner that the scenarios can be easily implemented.

Also, in the construction process of the objects scene, it will be taken into account the freedom of movement degree that must be assigned to the user. There are applications in which the user can move freely across the entire surface of the object scene, while other applications restrict the user to a limited number of movements. This is determined by the degree of user involvement in the scenarios that make up the application. If the user interacts with many dynamic objects whose position in the scene varies in a way that is not entirely controlled by the application, then the user's mobility degree must be high. If the user can control interaction with various objects in the scene through minimal commands, accessible regardless of the position of objects in the scene, then the user's position can be fixed.

Some of the objects that make up the scene are static objects, keeping a fixed position throughout the game. Objects whose position can be modified are dynamic objects, and for these, a logic of interaction must be defined in order to understand how their movement or state is influenced by situations in which they are manipulated by the users or situations in which they interact with other objects in the scene. It is important for dynamic objects to establish certain additional properties (collision, weight, specification of translation and rotation of objects when interacting with other objects), depending on the possibilities offered by the program through which the game is created. Also, for objects whose status can be modified, it is necessary to define the set of controls (mouse clicks, keyboard buttons, joysticks buttons) through which the user gets to manipulate the objects. This set of controls is attributed to certain functionalities that can be equated with the tasks previously defined in the second phase of the game development methodology. In practice, users interact with the objects in the scene through the set of controls defined in this phase, and the result of the interaction is represented by a previously set task. A sum of this tasks allow the user to perform a certain scenario in which are involved one or more of the objects created in this phase.

The process of assigning certain controls to the objects in the scene is done based on the tasks that are performed on the objects in question. Their definition allows game developers to create the logical implementation on which different aspects of the game are decided. Techniques of interaction between the user and the objects in the scene allow the game development team to manage the input the game receives from the user and develop various complex algorithms to manage the evolution of the game as the user performs various tasks.

The feedback received from the testing group that has verified the tasks and scenarios described in the previous phase should be taken into account before creating any of the scene's objects. If the scenarios are inconsistent, a mistake has probably been made in understanding the requirements of the first phase, or the prototyping on which the scenarios were set did not take into account the main flow that the game must follow. In such situations, certain scene objects created at this stage may not involve the necessary functionalities of the game and thus reach a situation in which they can not be used.

The scenes made at this stage may need to be modified in the following iterations and the probability that their condition will remain unchanged throughout the development cycles is very small. However, objects that can be used very easily (minimum interaction) and that match the theme of the game (can be used in many of the game's scenarios) are more likely to remain in the game scene. It is understandable that advanced customization (complex graphic design) of objects recently added to the scene is not recommended because their state is due to be changed during the next few iterations.

Development of interaction and controll algorithms

At this stage, it is assumed that there is already a scene containing the objects necessary for the scenarios described so far. Also, the basic methods for object control (setting controls) have already been done at the previous stage, and

so the tasks defined so far for the game can be applied to various objects in the scene. This phase consists in achieving the complex logic of interaction. Interaction between objects and various actions made by the user generally have various direct effects on the game scene.

This phase contains algorithms that define complex movements of certain objects in the scene. This movements are usually triggered by the users but controlled by the application. This phase also contains the answer that the game returns to the user when it tries to act a certain command. It is very important to keep in mind that the users need feedback from the system to understand if the actions they are doing are good. This feedback can come in various forms: displaying warning / error messages, emitting various sounds, changing the status of objects that the user has directly acted, changing the scene or even ending the game.

This phase must include the control logic of the scene (developers should implement some details planned during some of the previous phases): limit the user's posibility to move outside the scene, strategy of generating special motion effects for certain objects (to increase the visual effect in the case a specific task is performed on a particular object), implementing a control logic to allow game objects to return to a previous state in the event of non-concordance. As well, this phase defines the management techniques used to controll the whole flow of the game (how to move from one scenario to another, how to measure the scores, how to skip certain steps, how the game manages certain actions / scenarios which are incompletely executed).

Heuristic evaluation of the game

This phase is very important because it checks the usability level of the game version obtained at the end of each iteration of planning and development. This methodology uses the heuristic evaluation model proposed by Jakob Nielsen. The reason why this evaluation method is so popular is the fact that Jakob Nielsen's set of rules is easy to use, detects many of the common usability issues and requires low costs. The method is composed of a set of ten general rules.[8]

The first rule assumes that the status of the game should be available any time.

The second rule refers to the need for a correspondence between the real world and the virtual world of the game.

The third rule refers to the level of control and freedom that the user has in the game.

The fourth heuristic states that there must be certain standards in the game that are respected in a consistent manner.

The fifth rule claims that errors should be prevented and treated so that the user has a pleasant experience.

The sixth rule specifies that the application should be intuitive so that the user does not have to memorize it.

The seventh rule assumes that the application can be easy and flexible in use.

The eighth rule says designing should be simple and aesthetic

the ninth heuristic implies that the user must be able to identify error states and be helped to recover from them.

The last rule assumes that the application is accompanied by a help menu and an explicit documentation

EXPERIMENTATION OF THE DEVELOPMENT METHODOLOGY

YAMS theme and requirements

YAMS has been developed with the help of Unity technology. This allows the creation of interactive scenes. Objects composing the scene can be customized by adding textures for graphic design, adding collisions for interaction management, and adding scripts written in C # language to control their movement and state.

The first development phase of the YAMS game began with the clear definition of the theme of the game. Subsequently, the general rules of the game were taken over. The understanding of these rules identified the main requirements. These requirements include: creating an arena containing the dice as objects of central interest, creating a score table on which dice configurations can be marked, allowing individual dice to be manipulated by the user, creates a method by which the dice selected can be mixed and discarded separately from the dice that the user did not want to select.

It was also inferred from the regulation that the game must be able to be played in multiplayer mode. The target audience of the game is formed by people of all ages (the rules of the game are easely understood). It is thus inferred that the application must be implemented in such a way that it can interact with very different age users (common, simple language, with clear examples and explicit rules).

The variety of user ages (some of them may not be very familiarized with computer games techniques) requires another fundamental requirement: development team should choose interaction metaphors from a list of those which are known by the users from everyday life (simple clicks, movement using keystrokes, drag and drop technique).

To ensure a high level of interactivity in the game, a requirement is to add a bonus dice object that will be collected by users so they can gain extra points.

YAMS tasks and game scenarios

From the requirements described in the first phase, it was understood that the tasks and scenarios for YAMS must be

closely related to lifting, throwing and mixing the dice. Also, tasks that allow movement through the scene and tasks that allow the dice configuration marking on the score table are required. As interacting metaphors have to be simple, intuitive, I chose that individual dice manipulation (dice lifting) should be done by drag-and-drop technique (mouse click and mouse movement). Movement through the scene was projected in an intuitive way (using the arrow keys on the keyboard). The scoring task was accomplished by using the mouse click (the mouse was already used for the dice lifting and moving task, so the number of controls used remains small).

Initial prototyping consisted of an outer arena surrounded by four walls. The dice were placed in the center of the arena, being the main objects of the game. The scoreboard will be placed on the front wall of the arena so that it is easily accessible to users. It is understood from the requirements analysis that there must be a distinction between the individual lifting of the dice to be thrown and their simultaneous discarding. Thus, the initial prototyping contains another important object: the dice thrower. It is projected as a mug in which the user throws the individual dice that should be considered in a throwing step. This object (Figure 1) will be used to perform the throwing scenario.

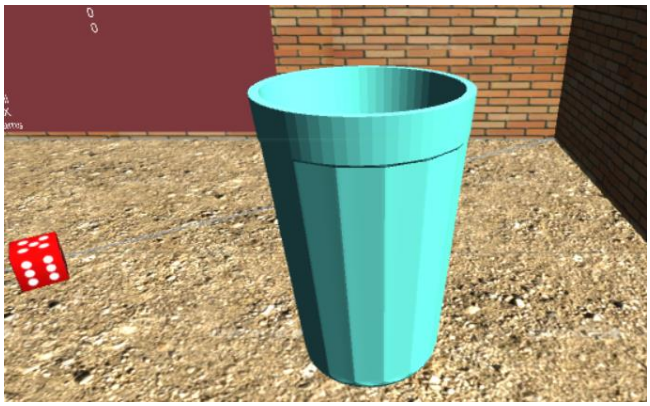

Figure 1. Throwing dice container

The main scenarios identified in this phase are: selecting dice to be thrown, blending the dice (using the throwing mug object), marking scores on the scoreboard and collecting bonus dice. A special situation is represented by the scenario of mixing dice and throwing them. Initially, the user started picking up the mixing container and simulating dice mixing (drag and drop technique). Subsequently, the user had to press a certain button on the keyboard to throw dice on the playing surface. Feedback from users was negative because the move was complicated and difficult to synchronize. In addition, dice mixing was not done in a random manner (the drag and drop technique did not allow the mixing mug to rotate). In a second iteration, the scenario was changed: the user presses a simple command (mouse click) and the actual mixing is done by the game.

Designing YAMS scenes and interaction techniques

The original prototype was used to create the main stage of the game. The dice used for throwing are placed in the center of the stage and are small enough to fit in the throwing mug. Bonus dice (Figure 2) have been designed differently (larger sizes and different colors) than normal ones, in order not to mislead the user. In order to be able to inform the user about the current state of the game, the scene must also contain a set of information to be permanently visible (in the top - left corner of the screen): the current player, the last throw configuration, the number of remaining rounds, the number of remaining throws. The throwing mug should conveniently be placed close to the dice (so users can easily add the dice inside) and be sized enough for the dice to blend in. Scoreboard object must be built in such a way that users can easily identify the categories they can play and see from any part of the scene the scores they have recorded inside the table up to a certain point in the game.The tasks defined at the previous stage should be attached to objects, so control scripts for those objects have been created. Controls defined for task action have been attached to the scene objects so they can be manipulated directly by the user. Colliders were used for the interaction between the dice and the rest of the scene (floor and walls of the arena, the throwing mug, the other dice in the scene).

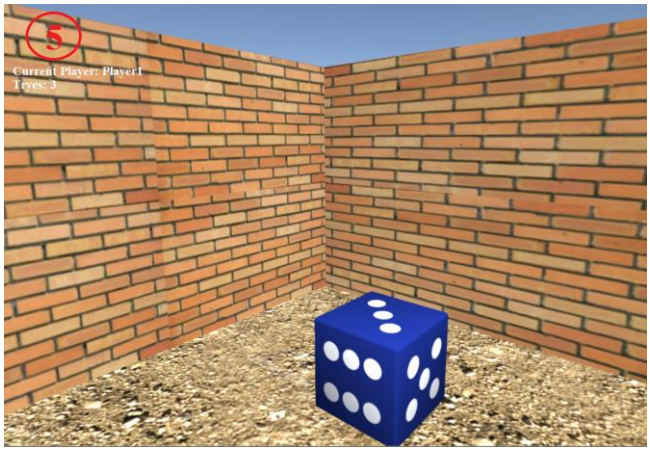

Figure 2. Bonus dice

YAMS development

The development part of the application was made in successive interactions. It started from the logic of interaction between the user and the objects. Shuffling and throwing the dice involves synchronizing the dice movement with the rotating motion of the dropper. At this stage, you can also simulate the random dice throw. The scenario is available only after the user has added the dice to the recipient. By pressing a click on the mixing mug, random rotation vectors will be generated to be assigned to each dice. Rotational angles will also be generated randomly. The direction of moving the dice when they leave the container has a random component (motion on the

oz axis). Thus, throwing is hard to predict. In this phase, the dice throw scenario is simplified, as the dice mixing logic is shifted from the user to the application. However, the user remains involved in the process (choose which dice should be discarded and access the throwing mug). The throwing process is presented in Figure 3.

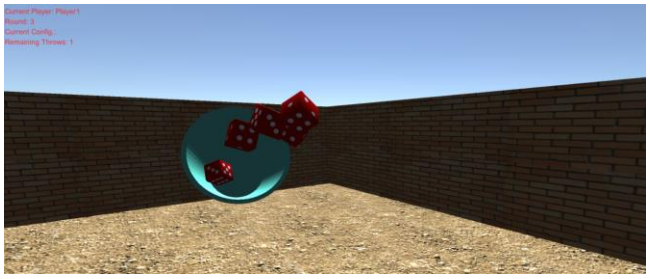

Figure 3. Throwing process

In subsequent iterations of the implementation process, the methods by which the system sends an answer to the user as a result of its actions have been added. Also, the implementation process also manages to display the game's status, which is important for usability. After going through certain scenarios (throwing the dice, marking the result in the table), the current user data, the number of remaining rounds, the result of the last throw will be changed. These will be updated in the scene using the information panel added in the previous step.

In this phase, messages have been added to guide the user in the process of the game. If a user tries to use the throwing mung without some dice inside, the system will display a corresponding message on the center of the screen. Also, if a user runs out of throws, the system will not allow a new dice throw, but will ask the current user to choose a category to be scored. To simplify the user's work, the score calculation process is done automatically when the user chooses the category.

METHODOLOGY EVALUATION

The usefulness degree of the methodology previously described must be evaluated according to certain defining criteria for the resulting game. It is therefore necessary to determine an evaluation metric to determine the impact that the methodology has had on the development process of the game.

At the beginning of the article it was described the argument of the high complexity of technology that allows the creation of video games with complex scenes and advanced interactions. The problem of meeting high quality standards in terms of usability has also been identified. As such, this application development model has been proposed to involve multiple development cycles and to focus on creating user-friendly versions. The initial analysis of the game has been divided into sets of requirements that have been developed interactively in well-defined phases, precisely because the game can be constantly changed, but the end result of the implementation cycle is stable and user-friendly.

Degree of functional coherence

A first evaluation criterion involves determining the degree of functional coherence. It is considered that the methodology was appropriate for implementing the game when the basic functionality of the application respects the original description of the game. While users can use the application to reproduce the features for which it was designed, it means that the planning and development iterative cycles have had a positive effect on the project. Such a result would also prove the necessity of the first two phases which are included in the methodology, namely the requirement specification phase and scenario building phase.

In the case of YAMS, the initial requirement was to create a game that complies with the rules of the real YAMS game. The to-do game allows you to select dice that can be thrown to create YAMS-specific configurations. Those configurations had to be marked on a scoreboard. Also, the resulting game should adapt to certain rules imposed by the regulation (maximum three consecutive throws per player, marking the score after each throwing round). The final version of the game exactly respects the game flow: Users can individually select the dice, dice can be thrown to create a configuration and the scoreboard allows marking all the categories described by the game's rules. Thus, it can be said that the level of functional coherence in this project is high, so the methodology was useful from this point of view.

Degree of usability

Among the main reasons to use a project development methodology was to reach a user-oriented end-product. Complex functionalities and advanced graphical design have to go through the end-user acceptance filter to be able to talk about the success of the project. If the game's requirements were transformed into scenes with complex interaction and advanced design, but it respects the usability rigors described in Jakob Nielsen's heuristics, then the iterative phases like design and implementation had a positive impact. Moreover, each iterative cycle provides an assessment of the current version, while each phase of the game development process begins with a brief evaluation (with test users) of the previous phase. Thus, the development team has the ability to constantly modify the game to accommodate user needs through feedback.

According to the YAMS game evaluation, the status of the game is permanently displayed on the screen (in the panel in the left-hand corner). This has been achieved by following the instructions received from users over several iterations. Following the implementation of the score marking functionality and the precalculation of the score shown in the table, test users noted the correspondence between the original YAMS game and the one created on

the basis of this technology. Users have confirmed that the game provides explicit messages to help them prevent errors and avoid difficult situations. This was done during the implementation phase, also based on the feedback received during the iterative development of the game. The task and scenario planning stage has allowed the establishment of a set of interaction metaphors that could be reused, which was remarked by users through the high degree of recognition of object control modalities. The stages creation phase (placed before the implementation phase) has allowed for a high degree of freedom for the user, yet he has the possibility to control the objects with which he interacts.

Adaption to often changes

Certain requirements related to the interaction between objects or the role of certain objects in the scene are changed as games evolve. The methodology will prove useful if the implementation of the game has come to a regular occurrence in situations where certain stage or implementation details need to be changed.

During the implementation of the YAMS game, most changes were due to the adaptation of the game to the requirements of the test users. In the score marking scenario, an object of a pencil marking the results was initially used, but the test users considered that the use of the object is too complicated because it requires two additional steps compared to the current method (direct use of the mouse pointer for marking the result). Thus, the object was removed from the scene. The methodology proposes that at the first stages of scene development, an object is created only if there is a close connection between that object and the requirements of the game. Also, it is recommended that in the first iterations (until it can be determined exactly the utility of the object), the object is not personalized in terms of design. This has greatly reduced workloads to setting final scenes. Also, iterative development has allowed some functionality to be moved from the user's responsibility (mixing the dice) to the system. This has resulted in improved (simplified) user experience and a better algorithm for generating random values.

CONCLUSIONS

An interactive game with a high degree of complexity in playing scenarios and interaction techniques requires a well-defined development methodology. The phases of such methodology divide production steps to reduce waste and avoid unnecessary work. A methodology allows teams with diverse specializations to work together on a common project. Also, a methodology allows the theme of the project to be transformed into a set of requirements that can then be processed in the form of game scenarios. These scenarios can be implemented independently as a set of tasks that act directly on objects in the game scene. Thus, game scenes can be designed in such a way that the algorithm development team can directly implement the established scenarios. A methodology is useful when attempting to obtain a finished product that is mapped to the initial requirements and which can meet certain market-imposed expectencies.

Using an iterative methodology allows developers to focus from the very beginning on a small set of requirements that are considered the most important. Also, iterative design and development provide versions of the game whose usability degree can be estimated, controlled, and improved. An iterative methodology allows the design team to modify the object scene to simplify the interaction methods, which is beneficial to the end user. Also, a methodology based on cyclical implementation allows for better management of game flow logic (unwanted functionality can be eliminated and scenarios can be adapted to increase interactivity).

The methodology proposed in this article involves a pre-implementation analysis that focuses on the player's profile and the type of device the game is intended for. This is very useful as this information helps to establish interaction metaphors that users can easily understand and identify. Also, the structure of this methodology improves the game development process as it proposes creating the scene only after a rigorous analysis of the game theme. This will allow the design team to identify only those scene objects that are required in scenarios.

REFERENCES

1. Chad Hadzinsky, *A Look into the Industry of Video Games Past, Present, and Yet to Come,* 2014.
2. Adriana-Mihaela Guran, Grigoreta Sofia Cojocar, *Abordări în evaluarea automată a utilizabilităţii. Studiu comparativ*, 2008.
3. Serdar Aslan, *Digital Educational Games: Methodologies for Development and Software Quality*, 2016.
4. Yams Game Rules, http://www.stratozor.com/en/yams-rules.php
5. Saiqa Aleem, Luiz Fernando Capretz, Faheem Ahmed, *Game development software engineering process life cycle: a systematic review*, 2016.
6. Ann Osborne O'Hagan , Rory V. O'Connor, *Towards an Understanding of Game Software Development Processes: A Case Study*, 2015.
7. Fabio Petrillo, Marcelo Pimenta, *Is agility out there? Agile practices in game development*, 2010
8. Jakob Nielsen, R. Molich, *Heuristic evaluation of user interfaces*, 1990.