Master's Degree Dissertation
2014/2015

Final Report

Crowdplay - Crowdsourcing Gameplay Data

Game Instrumentation

Author:
João Soares
jtalm@student.dei.uc.pt

Supervisor:
Dr. Licínio Roque
lir@dei.uc.pt

July 5, 2015

FCTUC DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Crowdplay - Crowdsourcing Gameplay Data

Game Instrumentation

Author:
João Soares
jtalm@student.dei.uc.pt

Supervisor:
Dr. Licínio Roque
lir@dei.uc.pt

Jury:
David Fonseca Palma
palma@dei.uc.pt

Marco P. Vieira
mvieira@dei.uc.pt

FCTUC DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Abstract

This report describes the development project of an author centric procedural content generation tool for videogames. The approach underlying this tool is to provide human designers access to the capabilities of procedural content generation techniques, whilst simultaneously affording them as much creative control over the creative process as possible.

The goal of this project was to finish designing and developing the tool and its procedural generation approach, and test it in a design scenario. To achieve this, three different tasks were carried out: first, a new game prototype was integrated into the procedural generation engine architecture; second, a user interface was implemented in a web server to allow designers' to use the tool; and finally, the tool was tested and evaluated.

The report describes the development process, the design and architecture of the tool, as well as all tests that were done. Usability test results show that the tool is in a usable state and preliminary data from a designer case study highlights the tool's potential.

Resumo

Este relatório descreve um intuito de projeto de investigação para o desenvolvimento de uma ferramenta de geração procedimental de conteúdo para jogos de vídeo. A abordagem subadjacente a esta ferramenta pretende disponibilizar o acesso a técnicas de geração de conteúdo procedimental a humanos, enquanto lhes permite tanto controlo sobre o processo criativo quanto possível.

O objetivo deste projeto era terminar o design e a implementação da ferramenta e a sua abordagem de geração procedimental, e testá-la num cenário de estudo. De forma a o conseguir, três tarefas diferentes foram executadas: primeiro, um novo protótipo de jogo foi integrado na architectura do motor de geração procedimental; segundo, uma interface para os utilizadores foi desenvolvida num servidor web de forma a permitir que os designer's utilizassem a ferramenta; e finalmente, a ferramenta foi testada e avaliada.

O relatório detalha o processo de desenvolvimento, o design e a architectura da ferramenta e todos os testes realizados. Os testes de usabilidade demonstram que a ferramenta se encontra num estado utilizável e os dados preliminares dum caso de estudo sublinha o potencial da ferramenta.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the years, designers have been creating custom content adapted not only to the user's needs, but also with the desire to stimulate certain emotions on them. In the game industry, whose size has been considerably increasing along the years, custom content importance is also rising as players are given more and better content and are continually waiting for more. When it comes to videogame content, methods that supply better information have risen over the past years, such as gameplay metrics [Drachen and Canossa, 2009a, Marczak et al., 2012]. This information can be used in various ways: to make gameplay experience closer to the intended[Mirza-Babaei et al., 2013], to understand how players are using the provided elements[Drachen and Canossa, 2009b] or how predominant gameplay styles are[Tychsen and Canossa, 2008].

Although there is progress, even with the information given by the data sources, there is the problem of how to make better content based on that information. One way of utilizing this data while aiming to improve videogames, is by employing procedural content generation methods that can use gameplay data to generate better content for players. This report describes the development of one such method: an Author-Centric Approach to Procedural Content Generation.

1.1 Objectives and Motivation

Whilst there are many novel procedural content generation approaches [Yannakakis and Togelius, 2011], they focus on providing players a predefined gaming experience, such as delivering game content that is fun and not frustrating. The focus of the author-centric approach [Craveirinha et al., 2013] is distinct: to provide game designers a powerful way to define and mediate their game agenda for player experience, instead of a pre-packaged solution. Thus, Author-Centric Approach to Procedural Content Generation aims to help game designers who want to use procedural content generation methods without straying from their intended game designs.

To use the method, a game designer first needs to have an agenda for player experience in mind. This agenda has to be verifiable by looking at gameplay data. For example: a game designer might want players not to die a lot during a level; this can be tested by checking the average number of player deaths for that

level. Second, he needs to have a working prototype that can be iterated upon so that, once coupled with the procedural generation system, it generates the desired variety of content (levels, scenarios, agent behaviors and representations, etc). To direct the generation of new content, the system must be capable of defining key features of the generated content; this means designers forfeit control of these features so as to achieve their end goal. For example, if the designer wants to optimize the number of deaths in a level, he will have to allow the system to define the number for that same level. Finally, the designers' prototype has to be integrated into a metrics collection system, so as to provide gameplay data for analysis, that will then be used to verify how close the system is to achieving the designer's agenda.

Being a new method, this still lacks intensive experimentation. Therefore, our motivation is to aid this method's development and help verify its application and usage in real design cases. The drawback is that, as it is now, the methodology is hard to use by designers, as it lacks a working user interface.

Our main objective is to give our contribution by:

- Creating a tool, in the form of a web platform, that can make the usage of this method transparent and simple for the user
- Verifying the usability of said platform through usability tests
- Performing an actual case test for this procedural content generation method, for validation purposes. This includes:
  - the adaptation of a game to use in the test
  - getting a game designer to use the platform, to adapt the prototype to a design intent of his choosing
  - draw a conclusion over this particular case

1.2 Chapters Overview

The following chapter contains the state of the art in procedural content generation, from which the main concepts of this project come from. It also draws a conclusion on why this particular method was chosen over a more popular one. Chapter 3 describes the used methodologies along with the hypotheses the project wants to help prove. Although their development was not a part of this project, in chapter 4 we describe the design process behind the used interface prototypes. This is important to understand the assumptions and choices behind their creation. Chapter 5 describes the architecture proposed for the solution and the expected functionalities for the various elements developed. Chapter 6 contains the information relative to the development activities: what was done, how and why for each part of the project. It also describes the used game's choice, modification and connection to the methodology's server. Details of all the tests realized, the conclusions drawn and issues found are described in chapter 7. Finally, chapter 8 contains the list of issues that need to be solved after this project is over.

Chapter 2

State of the Art

In order to fully understand the concept of the experimentation, a deeper explanation about the used methodologies is needed. In this chapter such an explanation is given about how those methodologies came to be, in what they were based and what they are used for, from the Procedural Content Generation to the used Author-Centered Approach. Methods used in both the collection and usage of metrics in video games are also explained and so is the actual proof of concept for the proposed case.

2.1 Procedural Content Generation

Designing videogames is a complex job. For each different application, its author has to come up with an idea, design the game and its components, and then tune them so they fit into the desired play styles of many different types of players, making this a very time consuming task. Tuning each parameter's value requires each configuration to be iteratively tested by several users, so as to make sure that the results are as expected. This is why the use of procedural content can be helpful, as content is created in an automatic or semi-automatic way. There are several works based on procedural content generation [Togelius et al., 2011a,b, Smith et al., 2010], but in this report two of them are given special focus. These are the Experience-Driven Procedural Content Generation and the Author-Centric Approach to Procedural Content Generation, each described in the following sections.

2.1.1 Experience-Driven Procedural Content Generation

The EDPCG was a novel procedural content generation methodology developed in which the generated content is directly connected the player experience [Yannakakis and Togelius, 2011]. The methodology's goal is to create high quality content, with which the player experience would be near optimal. Of course this would be highly dependent on the content type being developed, and would be based on the designer's content representation and each player's gaming style. The methodology creates a model based on the player experience in order to evaluate content quality. As listed by the authors, this model can be created using three different approaches, as well as a combination of any of these, creating

a hybrid approach [Yannakakis and Togelius, 2011].

The subjective player experience modeling uses data collected directly from the players, through questionnaires, reports written by them or by registering their speech during play. This approach has a few limitations such as the player's memory, in case of lengthy play sessions, or having to interrupt said sessions to progressively collect data.

On the other hand, objective player modeling is based on a third party view of the player. This can be achieved through play session recordings analysis or physiological responses, referred as biometrics. *"The core assumption of such input modalities is that particular bodily expressions are linked to basic emotions and cognitive processes."*[Yannakakis and Togelius, 2011].

Gameplay-based player experience modeling is done based on the assumption *"player actions and real-time preferences are linked to player experience since games may affect the player's cognitive processing patterns and cognitive focus"*. Game-play based player experience models are created using elements from the interaction between the player and the game. This approach can use collected gameplay metrics as input data, since metrics are generated from the interaction of the player with the game.

The main reason player experience models are used in EDPCG is to evaluate the generated content's quality, in order to generate new and improved content. This evaluation can be done using three kind of functions: direct, simulation-based and interactive.

Direct functions extract features from the generated content and directly verify its quality value. This means that for a given content, its values are directly mapped to a quality range, be it in a linear or non-linear way. For instance, when creating mazes, the quality of said maze can be directly evaluated by the number of exits it has. This method is fast as, usually, it takes little computational power, but needs to be personalized to a specific game and content type.

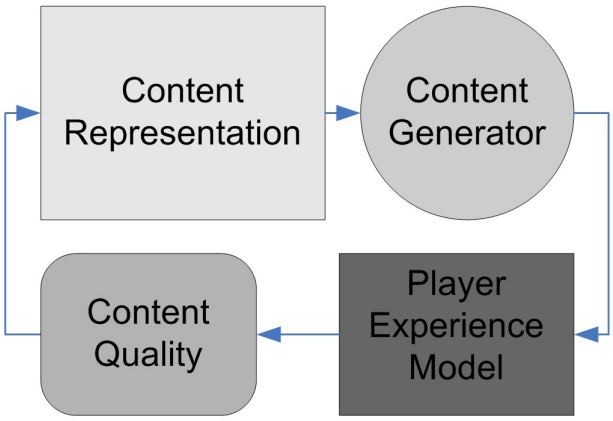


Figure 2.1: The main components of the experience-driven procedural content generator. Extracted from [Yannakakis and Togelius, 2011]

Simulation-based functions are used when content cannot be directly mapped

into a quality value, as it takes several experimentations to reach an evaluation. It uses an artificial agent that plays through the needed part of the game and, based on its gameplay (speed, success, behavior, ...), calculates its quality value. The used agent may or not vary during the evaluation. If it does, it is considered a dynamic simulation-based function, otherwise it is static. Dynamic functions are useful when creating content that stimulates the players learning curve or to simulate user fatigue. Using simulation-based functions is computationally expensive and, sometimes, very time consuming.

Interactive functions evaluate the quality of a content based on the interaction between a player and that content. Again, the data concerning the interaction can be collected based on the player's physiology, questionnaires or behavior. Interactive functions are especially useful if used to create content for a specific player or a specific play style.

2.1.2 Author-Centric Approach to Procedural Content Generation

When it comes to games, basing the content on the player experience alone can have unwanted results, such as altering the game's difficulty or not focusing several key components, leading to a potential loss of intended gameplay experience.

This led to the experimentation of another kind of Procedural Content Generation: an Author-Centric Approach to Procedural Content Generation (or ACPCG for short) [Craveirinha et al., 2013]. The point of the ACPCG is to give designers the *"choice to lead the end-result for gameplay, as opposed to subjective experience aspects"*[Craveirinha et al., 2013].

The first step in this methodology is for the designer to supply a game Archetype along with a definition of parameters and Target Indicators. The parameters are a *"set of design elements, implemented by designers, that they forfeit direct control of"*[Craveirinha et al., 2013], for which new values will be generated. The Archetype game is a version of the game deprived of the defined parameters, which will be filled with the generated values of each generation. Target indicators are the values expected for the content, although not the exact resulting values. This will consequently create variations of the base game, each called candidate games, ready to be tested.

Using the parameter definitions, the ACPCG uses a genetic algorithm to generate parameter sets, which will be combined with the Archetype to be tested. As players playtest the candidate game, gameplay metrics are collected and will help evaluate the quality of the generated content. Gameplay metrics are used because they are objective, direct and can be gathered in large quantities without interrupting gameplay sessions. The collected metrics are then converted to gameplay indicators and the quality of the content is evaluated based on the difference between the obtained and the target gameplay indicators. This process is repeated until the obtained gameplay indicators are near the target indicators. In order to avoid unnecessary repetition and to speed up the process, new generations use the previous values as base, giving them a fitness based on their quality. [Craveirinha et al., 2013]

Another modification done to the usual Procedural Content Generation methods is that when a feature set is evaluated as with good quality, both the candidate game and its indicators are returned to the designer for further

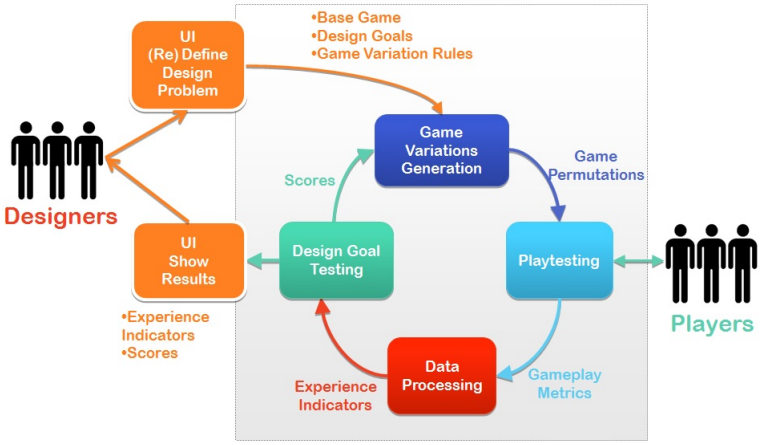


Figure 2.2: The ACPCG algorithm proposed in [Craveirinha et al., 2013]

evaluation. The process is interactive and only semi-automatic. If the resulting candidate game is accepted by the designer, the process is considered concluded; otherwise a new iteration of the whole process starts, where the system may continue to improve the generated games, or where the designer re-thinks the problem and tells the system to optimize a new set of target indicators. This last step makes it so that the process result is always in accordance with the designer's initial idea. [Craveirinha et al., 2013]

2.1.3 Comparison between EDPCG and ACPCG

While there are several Procedural Content Generation methods, not all of them are intended to be used in the same contexts, nor do all have the same degree of experimentation. If we are going to choose a Procedural Content Generation method to use in the project, a comparative analysis is needed. For that reason, this section makes a comparison between two distinct methods, analyzing both the usage and usefulness of the EDPCG and the ACPGC. The game used in both cases the same, the 'Infinite Mario Bros', and their full analysis can be found in [C. Pedersen and Yannakakis] and in [Craveirinha et al., 2013] respectively.

For the EDPCG, the most important aspect of the content is the player experience. The first step in the methodology was to create a representation for the playing level. This was achieved by using a vector with the amount of holes, their sizes and their positions. The vector was later used to construct the actual map simply by creating the map from right to left. To create the player experience model, a survey was made to players after they played groups of two levels. This survey asked them to evaluate which map was better to induce the following affective states: fun, boredom, challenge, frustration, predictability and anxiety. The game also collected gameplay metrics, like jumping, shooting and dying, and its data was used to train a neural network to predict the affective states. Each affective state was linked to a set of metrics, creating a connection between a certain event and the generated emotion on the player. By using these experience models, a level can be optimized to elicit the required emotions on any player.

	EDPCG	**ACPCG**
Methodology Focus	Player Experience	Designer's Intent + Player Experience
Experience Model	Static	Personalizable
End Result Influenced by	Chosen Model Player Experience	Target Indicators Parameter Ranges Player Experience

Table 2.1: Comparison between EDPCG and ACPCG

On the other hand, the ACPCG tries to adapt a game to the concept the designer wants for it. On the 'Infinite Mario Bros' case, the designer must first decide what are the gameplay indicators he wants for the game and create a connection between them and the collected metrics. For instance, the number of tries on each level relates directly to that level difficulty and challenge, while the time elapsed in a level determines the play-speed of the run. For this example, the following parameters were varied: the game speed, the number of cannons, koopas and goombas, the number of holes and the number of item blocks and coins. For each indicator, a mean value and a range must be defined, in order to define a suitable variation of these values. Using these ranges and the fitness value of the current generation, the genetic algorithm creates the parameter values for the next one. This way, the methodology can elicit player experience while maintaining the designer's target gameplay experience.

As seen from the examples, the methodologies have different objectives. The EDPCG is mainly concerned about giving the player a level fit for his gameplay style. If a player has fun collecting coins but does not enjoy hard challenges, a level that promotes coin collection over jumping holes is created. But if a player enjoys the challenge of traversing a difficult map, the generated level will be filled with large and challenging gaps. The ACPCG tries to deliver a good player experience but restricted by the designer's intent. If the designer wants to create a game based on the concept of a challenging speed run, the generated parameters will try to fit the player experience in that concept. Concerning the used models, the EDPCG uses preprocessed models while ACPCG models can be personalized. This means that the result of the EDPCG model can only be influenced by the type of model used and by the player interaction. ACPCG models, however, can be adapted and modified, since each parameter's variation is defined by a function controlled by the designer.

2.2 New Proof of Concept

Regarding the ACPCG, a proof of concept has already been developed in [Craveirinha et al., 2013] to sustain the proposed model. This first proof was done using a platform type game, in particular, a version of the 'Infinite Mario Bros', where a total of eight parameters varied: the game speed, the number of cannons, the number of Goombas, the number of Koopas, the number of holes, the number of item blocks, the number of coins and the number of hills. Having obtained positive results in the former case, this project will now test the methodology in a different genre –a strategy game–, as this another step in the giving credence to the approach utility for a wider range of design problems.

Strategy games are well known and popular games, their gameplay have many variations and usually have a wide range of actions to perform. Since their game play can vary so much, their content must be well balanced, support various play styles and still instill the required emotions on the player. This is why a strategy game is a good genre to test the ACPCG, and it is also why this project uses a strategy game as the prototype for a new proof of concept. The new proof of concept will also feature a game designer working with the ACPCG, adapting the prototype in order to achieve his intended gameplay experience. At a later point, other game designers will be added to the proof of concept, since a single designer cannot be seen as a valid population of users for this type of tests. The objective of creating a new proof of concept is to increase the method's credit on aiding game designers reach their intended game design and the differences between the new and the the previous proof of concept are further detailed in section 3.1.

2.3 The Value and Use of Gameplay Metrics

A game quality depends on the quality of its components. This is why there has been such an increase in the user's opinion importance in the last years, leading to an increase in the quality of the processes behind user testing. In fact, several techniques to verify the quality of both the components and the end result have been developed over the years. These techniques vary in data gathering methods, its analysis and purposes.

Although not considered gameplay metrics, one of the most basic methods is the use of gameplay questionnaires. After playing the game, players are posed a survey where they are asked to answer a certain amount of questions regarding their point of view on several game components. Another possibility is the use of recordings and their analysis but, the data gathered by these methods is subject to personal views and interpretations. *"However, these approaches are limited in that information is often hand-coded (surveys, analysis of audio-visual recording), meaning that getting highly detailed data about user behavior is either incredibly time consuming or downright impossible."*[Drachen and Canossa, 2009a]

By gathering metrics, they can be used to analyze and recreate player experience [Drachen and Canossa, 2009a]. This way, it is possible to understand aspects such as the game's difficulty variation. If there is a certain point in the game where the players need a lot more attempts to succeed than the rest of the game, then the players might experience frustration when playing it. Gameplay

recreation is a deeper form of analysis, usually used to verify whether the players actions stray too far from the designer's intent.

Another way to analyze player behavior using gameplay metrics is to create play-personas. As seen in [Tychsen and Canossa, 2008], it is possible to determine several game play styles by analyzing player behavior. These can be defined by verifying several metrics: the players usual locations, actions, response time and so on. By finding and understanding these play styles, it is possible to adapt the game, rewarding or punishing players using them. The authors also pointed that the collected data can be used to improve gameplay and/or to redirect the design back to the designer's original intent. As seen before, a player belonging to a certain persona might find some aspects of the game lacking or unbalanced, while others might find them fun or exciting since, as stated in [Tychsen and Canossa, 2008], a persona may comprise different play styles. The example given for this event is described based on the game 'Hitman: Blood Money'. One of the possible personas developed for this game is the 'Silent Assassin' and it is detailed as a player that leaves no trace and rises no suspicion. Despite the player choosing a close combat or far away sniper like play style, as long as he maintains these traits, he still fits the 'Silent Assassin' persona.

On the other hand, more objective and detailed data about user gameplay can be gathered through the automated collection of gameplay metrics. As previously described, this data can be used to verify the players behaviors and verify whether the gameplay estimations were correct and if not, help understand what went wrong. *"Where methods such as usability-testing and playability-testing focus on establishing whether a player can interact with a game effectively, and if doing so is fun, gameplay metrics-analysis is targeted at clarifying what the player is actually doing, and sometimes why the behavior indicated arises."*[Drachen and Canossa, 2009a]. Gameplay metrics alone might not suffice to understand the why, and sometimes it requires a combination of other techniques more user-oriented, such as surveys or more specific methods.

Another form of metrics researched and developed is the use of physiological measures, or biometrics [Mirza-Babaei et al., 2013]. This approach uses indicators such as skin-conductance level, facial electromyografy, heart rate and electroencephalografy to verify the user's state while playing the game. Although it tends to have good results, the measurements have to be supervised and registered because they might be caused by a stimulus other than the game itself, sometimes even something that cannot be tracked. One of the biggest downsizes of this method is that it requires complex machinery and people that understand how to operate such machinery. The methodology itself is hard to use as it requires the analysis of very specific data types. One of the conclusions of the study, however, supports that games that use some sort of user testing in their development tend to yield better results, both in content quality and in gameplay experience.

Chapter 3

Methodological Approach

The described project is composed of two intertwined goals. On one hand there is a research agenda we are pursuing, and on the other hand, there is a development schedule that had to be fulfilled in order to advance said research. Therefore, these two steps need to be addressed in a particular order and each has its distinct way of evaluation.

3.1 Objectives

The main goal of this project is to help answer the research questions of if and how ACPCG can aid game designers in their creative design process. To do so, one must study the active use of this method by game designers, which in turn requires a working prototype of an ACPCG application, complete with an interface developed for said designers, and a working videogame prototype connected to the ACPCG application.

To research the ACPCG method in a creative context, an actual use case with a game designer was needed, and that required a prototype connected to the platform. Thus, in the early phases of development, instrumentation of an open-source game was carried out. This required connecting the game to the existing ACPCG server, so that it could provide the supplied game parameters to evolve each new game-prototype and also collect gameplay metrics referent to play sessions. Because this had already been done for the 'Infinite Super Mario Bros' (see chapter 2.2), a new game in a new genre was chosen for this new test, so as to provide diversity to the cases tested.

In a second phase, the ACPCG interface was developed. This phase was crucial because users currently had no way to easily configure the ACPCG, as it required editing complex configuration files. The interface was developed using already existing paper prototype mockups (described in chapter 4.1.1) as models. The objective of this phase was to grant target users an easy way to manage the ACPCG's definitions for their games. This required a good understanding of the existing code, as well as a good comprehension of the current database schema. Also, because the platform uses the Play framework, with which we had no experience, a required investment in learning how to use the technology was needed.

Once the user-interface and prototype were developed, a usability test was

conducted to discover flaws in the designed interfaces. The usability tests were planned so as to validate the interface's adequateness to the task it was meant to fulfill, and to map out a plan for future revisions and additions needed to improve the application. Usability tests were video recorded, and data from videos was analyzed to find issues with the interface; these were complemented with System Usability Scale [Brooke, 1996] questionnaires, as well as with task and concept-based inquiries on complexity of the interface and its workings.

Finally, the actual research agenda was carried out by experimenting with a game designer. He was given the task of taking the game prototype and ACPCG application and using them to create a novel design. To acquire data during the procedure, the ACPCG interface was instrumented with UI metrics and several questionnaires were posed to the designer. Data was analyzed to help test three hypotheses held in this case-study:

- Designers can define game design problems using the ACPCG application and its underlying ontology.
- Games resulting from a design process supported by the ACPCG application are adequate solutions that fulfill game designers' agenda.
- Using the ACPCG application aids designers in finding new design problems and solutions.

3.2 Process

For production of the ACPCG prototype and its interfacing with an existing game, an agile development methodology was adopted. The reason for an agile method is that this project is strongly connected to an ongoing development process and, as such, it was very likely for changes in requirements to repeatedly appear. The method was both iterative and incremental where, for each component, there would be an initial planning about how its elements should be developed. Afterwards, the development of said elements would start and, once finished, evaluated. The evaluation was based on whether the elements had the expected behaviors and, if they did, they would be considered as finished, otherwise they would be fixed. Given the fact that we used the Play framework, the deployment phase is integrated with the development, because every time a file is changed, the framework deploys a new version of the project.

Every time we finished a new component, we asked our "client" to check its functionalities and to report us any wanted changes on them. Our client is the PhD student currently working on and researching the ACPCG methodology. Contact with the "client" was frequent, allowing for design and implementation to be discussed and adjusted dynamically, almost on a real time basis, instead of over long periods of time. This fact is strongly related to our workspace being the same as our "client's".

Besides the informal requests from us to clarify architecture related doubts, we had contact with the client in a weekly basis, in the form planned meetings. The meetings duration ranged from 20 to 30 minutes and consisted on progress updates from us and requirements changes from him. We would often use these meetings to clarify how some elements on the prototypes should behave, or how

they should interact with the ACPCG platform, as he was the one behind its development.

Being in the lead of the ACPCG development, our "client" was the one to supply us with documentation regarding the connection requests for the prototype adaptation, as well as the metrics desired format and the database structure in the web server.

Due to the fact that practically every element in the interfaces was required for it to be functional (except the data filtering in the results screen), we focused on making everything work, before making any special adjustments. The order by which we developed the various interface parts was the same order that the game designers take in their process of creation: design problems, test setup and results. This order made clear what users would be expecting when moving from one interface to the other.

3.3 Scheduling

In this section, we will draw a comparison between the estimated schedule at the end of the first semester and the actual schedule taken to realize all of the activities planned for the second semester. Figure 3.1 represents the Gantt chart created and delivered in the last report, having the estimated times taken for each of the new semester's required activities. When comparing them to the actual time taken and order of activities (in figure 3.2), we can conclude that the estimations were greatly off target.

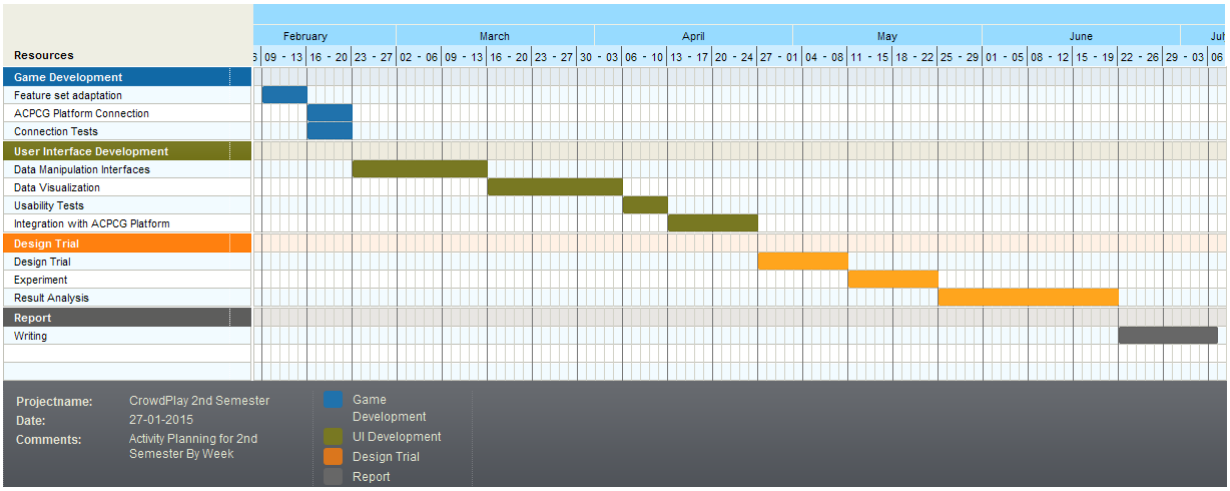


Figure 3.1: Task time estimation realized in the first semester

We were optimistic about all of the required tasks, mostly because of the lack of experience with the used tools, but also because of the work inexperience in a project of this kind, one that requires integration, constant testing and analysis, specially with a development team so small.

Not only did the estimations failed in terms of time, they also failed in terms of task ordering, as the original estimation had a sequence of tasks and the actual order has several parallel tasks being taken care at the same time.

Figure 3.2: Actual time taken for the second semester activities

In the end, due to the delay from the first semester and the failed estimations, not all tasks were finished until this delivery but they will be finished after it.

3.4 Deliverables

At the end of the dissertation, the following elements should be made available:

- The working prototype game developed, compiled and ready to play with all required functionalities.
- The source code for the adapted prototype game
- The web platform ready to be accessed and utilized by the game designers
- The documentation regarding the web platform development activities and choices taken
- The data gathered from the usability tests, its analysis, the list of issues found and proposed corrections.
- The data collected from the design case test and its analysis.

3.5 Plan and Actual

The plan for the project was to instrument the prototype game and, as soon as it was over, start implementing the paper prototypes for the interfaces. Once the development phase was over, the designer's interface usability tests would be conducted through which its prominent issues would be found. After fixing the most urgent problems, the design case test would begin. In the test, a game designer would use the developed interface to adapt our archetype game to his novel game design, at which point the archetype game would be played by a group of players. After the players were done, the game designer would

evaluate the processed experience indicators, verifying whether the game was being played as he expected. Depending on this evaluation, the process of redefining the the experiment parameters, playing the game and its reevaluation would all have to be repeated as a new iteration. In the end, conclusions would be drawn from this test, confirming or not the utility of the developed tools for this case.

In reality, the instrumentation of the prototype took longer than anticipated, leaving less time for the remaining tasks. After the instrumentation, and before starting to implement the interfaces, the available paper prototypes had to be converted into web mockups, as the behaviors in the paper prototype and in the web are identical but not quite the same. The usability tests took place as expected, and its conclusions drawn, but there was no time to fix any urgent issues before beginning the design case test. Instead, the most problematic issues were solved during the design case phase, in the periods of time when the designer was not using the interface. Due to all delays in the development phases, the design case test could not be finished in time for this report, so no conclusions can be reached at this point, other than the designer chose a very uncommon game design.

Chapter 4

Design

In this chapter we describe the design of the ACPCG interface. Although the design process that resulted in the ACPCG interface is not an integral part of this dissertation, an understanding of how it came to be is still important, given that it influenced design and development decisions that were taken during this project (for example, the new terminology). Hence, it will be described in this chapter, as a synthesis of a paper that is pending publishing, written by Craveirinha and Roque [Craveirinha and Roque].

4.1 Design Process

The ACPCG interfaces were created via a set of two Participatory Design(PD) sessions. PD is an approach where the target users take a leading role in the design process; given that the game creators are the interface's target audience/users and that they know their requirements and preferences, it made sense that the design process involved them. This made it possible not only to design an interface from the perspective of the target users, but also to provide insight on how they would interact with the new PCG approach and ideas.

By engaging the game designers in the creation of the prototypes, they were able to interact with and discuss them, leading to improvement in a much faster fashion than user testing. Another advantage of using this approach was that user intentions and expectations became clearer as they advanced in the development, even more while analyzing audio recordings taken from the development sessions.

4.1.1 Results

Besides the resulting interfaces, the two PD sessions resulted in a series of observations that were then used to incorporate new user requirements into the ACPCG application and interface. After careful analysis of the collected contents, the Participatory Design process had two major issues that needed to be addressed.

The first was that game designers actually want to test their game's variations with no target values before they decide what these are going to be. This can be seen as an 'exploration' of the game before the 'optimization' of its pa-

rameters, leading to an unforeseen use case: the generation of game variants that had no target indicators. More than an option, this comes as a need, since participants had difficulties defining the design problems and solutions needed. A possible solution to this situation can be the use of a first exploration and result visualization before designers commit to a design agenda. This way, the designers can better understand the possible design paths available and how well defined they need to be.

The second was the fact that the overall concept behind PCG and its utility remained hard to understand to new designers. There are several aspects in which the current ACPCG is lacking; being the most troublesome its terminology. In an approach where there are so many terms, and where some are mere small variations, it became hard to find suitable and intuitive names for everything. When added to the already complex way of the application's working, this became a serious limitation to new users. Based on this information, several revisions to the UI's metaphor were then made.

Instead of merely having to define each of the PCG approach's varibales, a Test metaphor was inserted into the logic of the application. The metaphor is simple: each generated game candidate is evaluated according to tests. Each test is comprised of an "if-then-else" logic. If a given candidate has an experience metric inside a test's boundaries, then the candidate passes the test and gets a score, if not, it gets either zero score or a penalty. For example: if game candidate has the number of level tries between zero and three it gets 100 points, else, it gets 0. Score attribution, besides making it clearer how the system judges games candidates quality, allows designers to give different weights to different target metrics. The test logic, hopefully, would help users understand the PCG approach, as it mirrors how games work.

4.1.2 New Terminology

In complex applications, terminology must be as direct and clear as possible, especially if the amount of used terms is high. Results from the PD sessions indicated that this was a major hurdle for the potential users of this application, made worse due to the complexity of the algorithm and how intertwined some concepts are. Thinking of this, we met with the client for a brainstorm session with the objective of replacing some parts of the original terminology with new terms. In the end, we achieved a list of more intuitive expressions, described in table 4.1, that would be easier for users to understand. The adequacy of the new terms was then tested in the Usability test, with specific questions directed at validating this new terminology.

4.1.3 Interfaces

As another result of the PD, the final paper prototypes are listed here; these include revisions done by the client before we developed them. These paper prototypes are the base for the ACPCG platform interface and are part of the development phase of this dissertation. They will be the connection between the users and the actual ACPCG algorithm, so they were made as a clean and easy to use as possible. In this section we describe the most important prototypes, the rest being available in Appendix A.

Old Term	New Term	Description
Project	Experiment	The set of all definitions done with the intent of achieving a game design
Feature	Game Variable	One of the game's registered parameters
Game Quality	Game Variation	The parameter set by which a game variable varies in the experiment
Candidate	Game Permutation	A set of Game Variations values to be used by the Archetype game
Best Candidate	Best Permutation	The Game Permutation with the best overall score for the experiment
Domain	Test Range	Values between which a design goal can be comprised without being penalized
Score	Test score	The score given for a design goal when its indicator is processed
Minimum Evaluations	Minimum Play Sessions	The minimum number of times the prototype game has to be played for an evaluation to be drawn
Optimization	Testing	Represents if the design goal is being used for scoring candidates
Active	Vary	Represents if the game variation is set to vary in the experiment's candidates

Table 4.1: List of terms changed with the new terminology

The design problems interface (Figure 4.1) allows an overview over the artifact qualities and design goals of a project. This interface also allows the user to navigate to the design goals (Figure 4.2) and the artifact qualities (Figure 4.3) interfaces.

Accessing the design goals interface (Figure 4.2) allows further detailing of a single design goal (translatable into target indicator), either on creation or while editing it. As such, the user is able to change its configuration, from the scoring function, used for the goal's test condition, to its domain and formula. This formula is the way that the ACPCG has to process experience indicators, assigning a score to a candidate's design goal depending on the formula's final value and the scoring function.

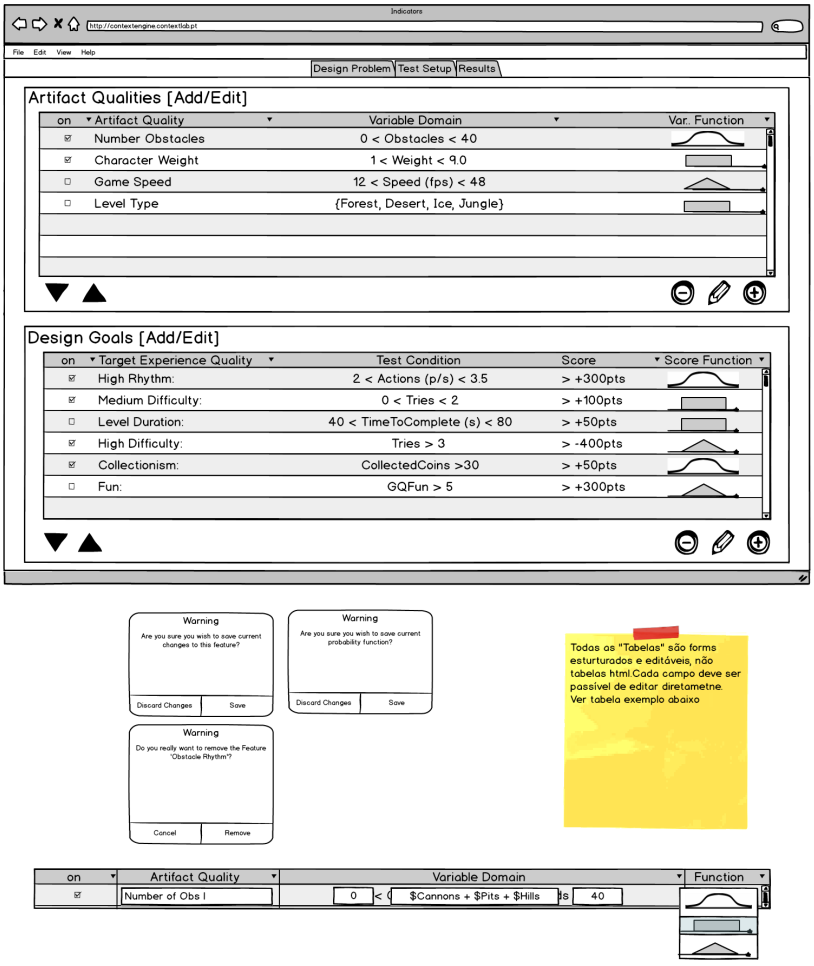


Figure 4.1: Prototype for the interface to manage design problems

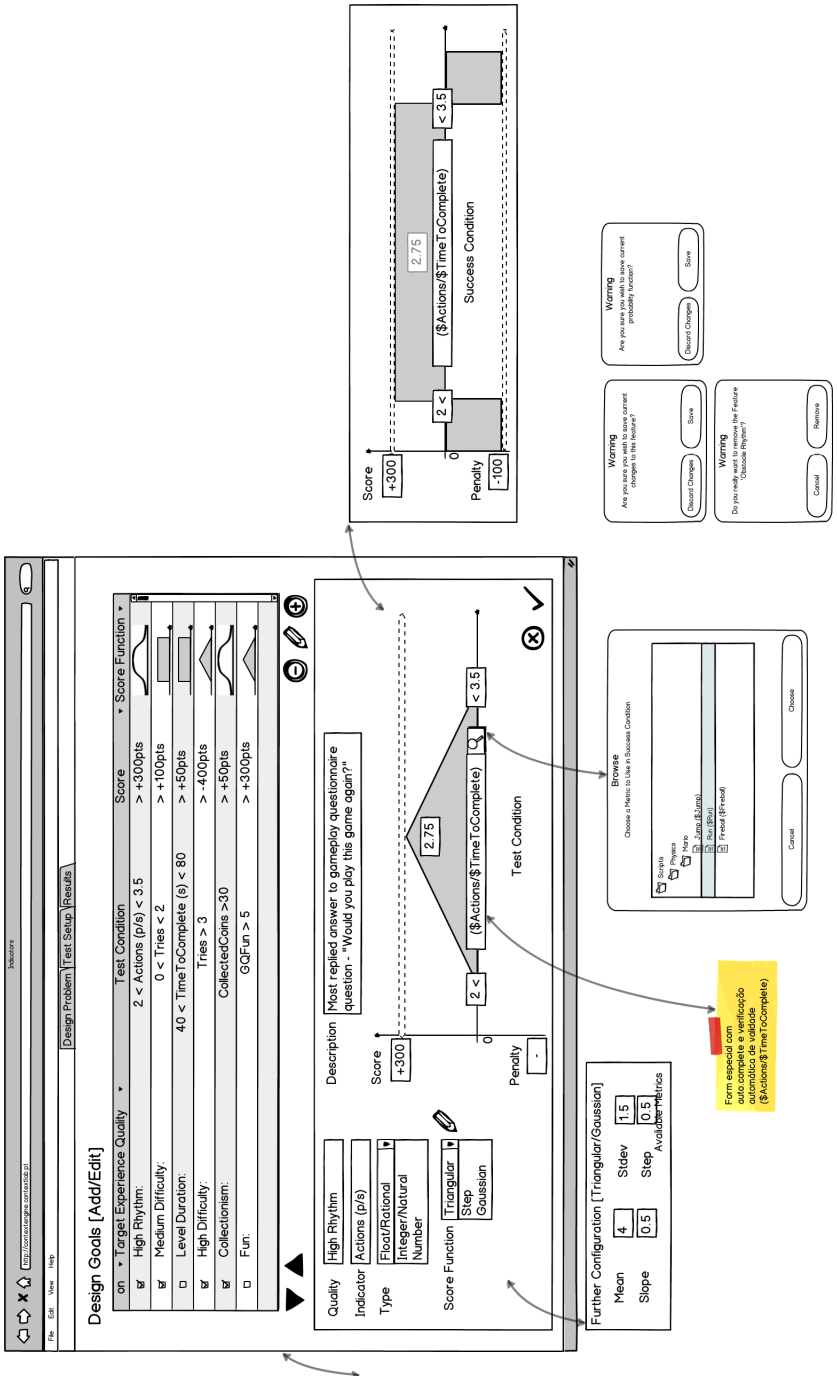


Figure 4.2: Prototype for the interface to manage design goals

The artifact qualities interface (Figure 4.3) is similar to the design goals interface but is used to manipulate artifact qualities, or game parameters variations. Again, the user can define the various parameters, their value ranges and the probability function used to generate them.

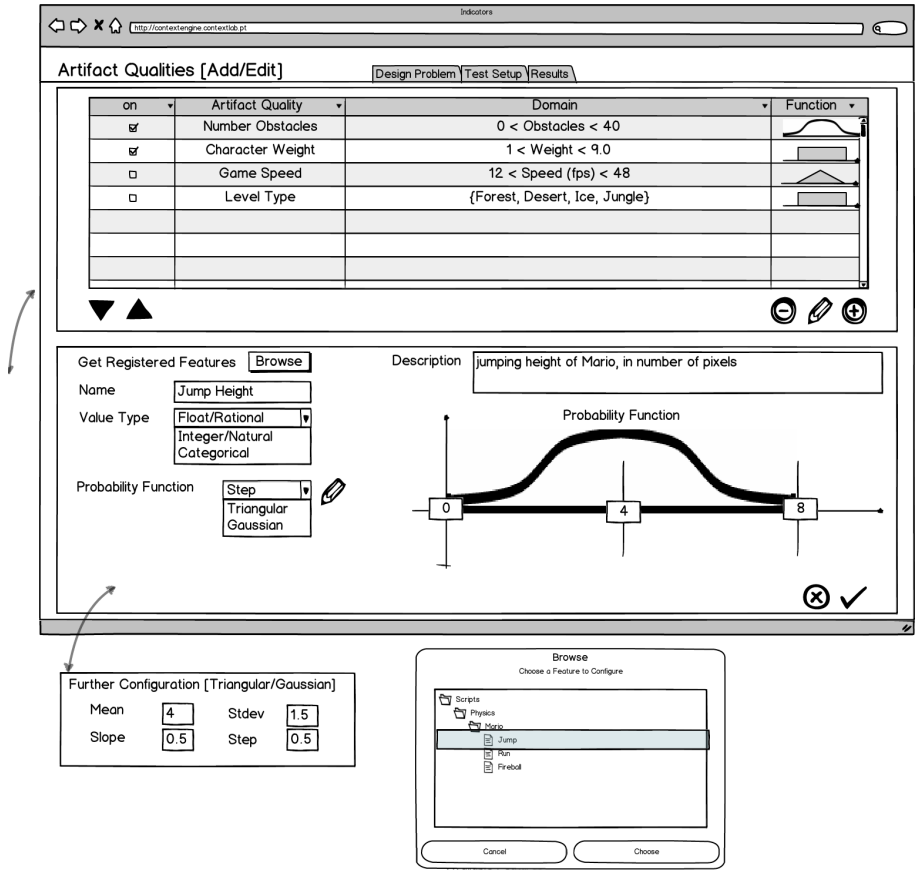


Figure 4.3: Prototype for Artifact Qualities management

The results interface (Figure 4.4) is where the user can analyze and filter the results referent to the gameplay data collected. There are several layouts that can be used, such as having one, two or four analysis tools and using various charts or tables to display the information. The data present in the interface should be dynamic, meaning that it can be filtered by either using the filter data tool on the left or by changing the attributes of the charts if they allow it.

During development, it was verified that the prototypes for one of the elements was missing: the formula management interface. Since this element is of high importance, we had to design it and implement it. Given that it is one of the most complex components of the whole platform, its interface should be as simple as possible while still giving the user as much options and power over them as they need. Notice that, although the used guidelines were very similar, this prototype was developed completely apart from the PD stated above and is not one of its results. Instead it falls under work and responsibility of this dissertation.

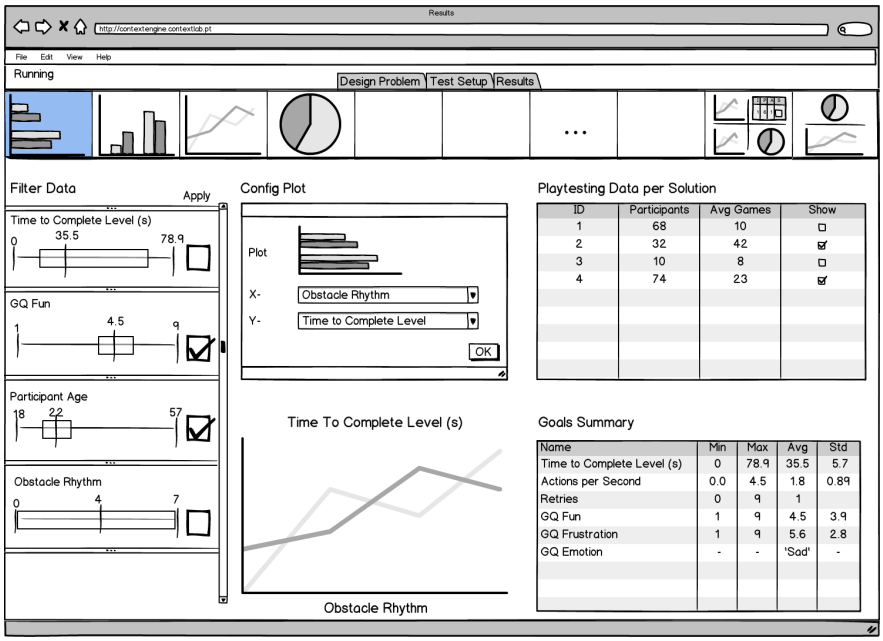


Figure 4.4: Prototype for result analysis visualization

The formula management interface features a list of already existing formulas and the possibilities to: a) choose an existing formula to add; b) create a new formula; c) delete a formula from the list. When creating a formula, a new interface module is opened where the user can input the desired formula name, the function to use, the variable which the formula will target and any number of conditions to be met by it. As stated before, the interface layout and elements were chosen based on the already developed interfaces.

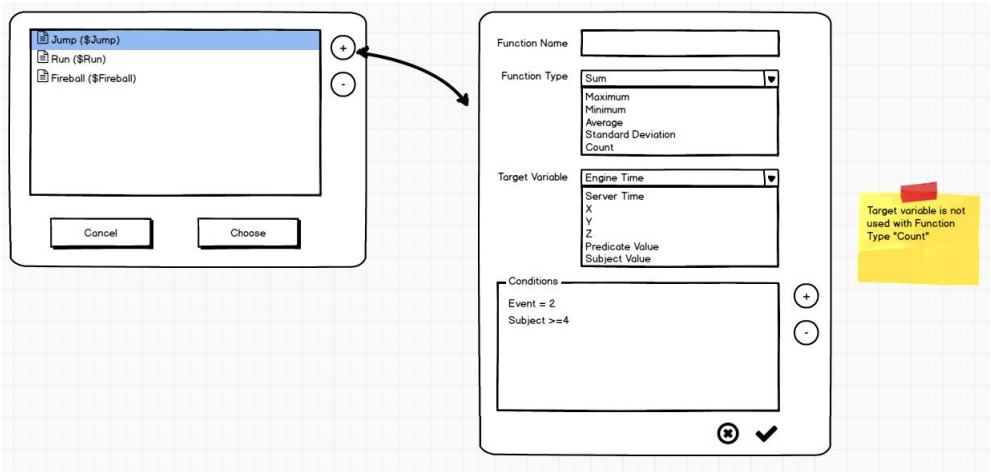


Figure 4.5: Prototype for the formulas management interface

Chapter 5

Architecture

This project is part of an already existing and functional system, meaning that it compromises both the development of some components and the integration of others. This chapter describes the initially existing architecture and the final achieved architecture while providing essential information about each of its modules and components. Notice that some modules might not be described in detail since they were already existing black boxes. However, all of their used functionalities will be listed and explained.

5.1 Overview

As seen in Figure 5.1, the project is divided in three major components. The candidate game, in this case the instrumented DuneLegacy, the web server and ACPCG platform to generate feature sets and to manage the collected information and the developed web interface to access and use the ACPCG platform.

The web server is located at the center of the Architecture. It supplies the ACPCG with parameter data, used to generate feature sets, and collected metrics, needed to process gameplay indicators. The readied feature sets are passed to the game and, after being played, it returns the gathered metrics to the server. After being processed, these metrics are transformed into gameplay indicators and serve as the feature set quality measurement. Through the web interface, the designers can manage the ACPCG's definitions and visualize the current available results. This way, the designer can control the process management so it can be done in real time instead of having to wait for the process to end, enabling a more efficiency and smother optimization.

5.2 Web Server for Requests

The server uses the Play Framework, mainly using Java as programming language and Scala for web development. There is also a mySql database used to store all the ACPCG related information.

Due to the size of the application, the server architecture follows a Model View Controller pattern where the controllers manage the information in the models and pass it to the views for visualization. In the current architecture,

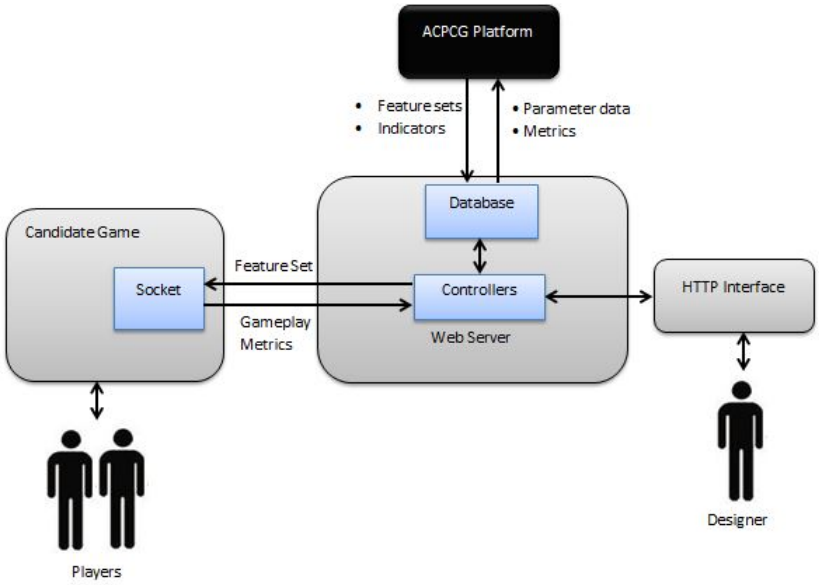


Figure 5.1: Architecture overview

Models are the database entities, Controllers are the actual Play controllers and Views are the web pages built using HTML and Scala.

The Web Server communicates with the ACPCG platform that, as seen in figure 5.1, is a black box in the architecture. Its functions are to process experience indicators based on the collected metrics, and to use a Genetic Algorithm to generate new candidates for the prototypes. The indicated HTTP interface is the developed user interface that designers use to interact with the methodology, defining their experiment's parameters and viewing the results obtained. On the other side, there is the candidate game, which is our prototype game after been given a set of generated features. The main communications with the server are requests for feature sets and the return of gameplay metrics.

5.2.1 Web Server's Data Model

The web server's database is structured as shown in Figure 5.3. For a clearer and broader view of the Entity-Relationship diagram, its full version is available in Appendix C. All data exchange between the interfaces and the Data Model are illustrated in figure 5.2.

The central entity in the diagram is the Project. It uses a single game as its prototype but there can be several projects using the same game simultaneously. This enables the possibility of creating different projects with different goals and parameter variations for the same game. To each project is associated an experiment, each having any amount of Design Goals and Game Qualities.

An experiment is also composed of an end condition and a GA (genetic algorithm). The end condition contains the kind of end condition (max number of evaluations, target score or manual), along with the corresponding target values. The GA represents the used genetic algorithm and contains its definitions,

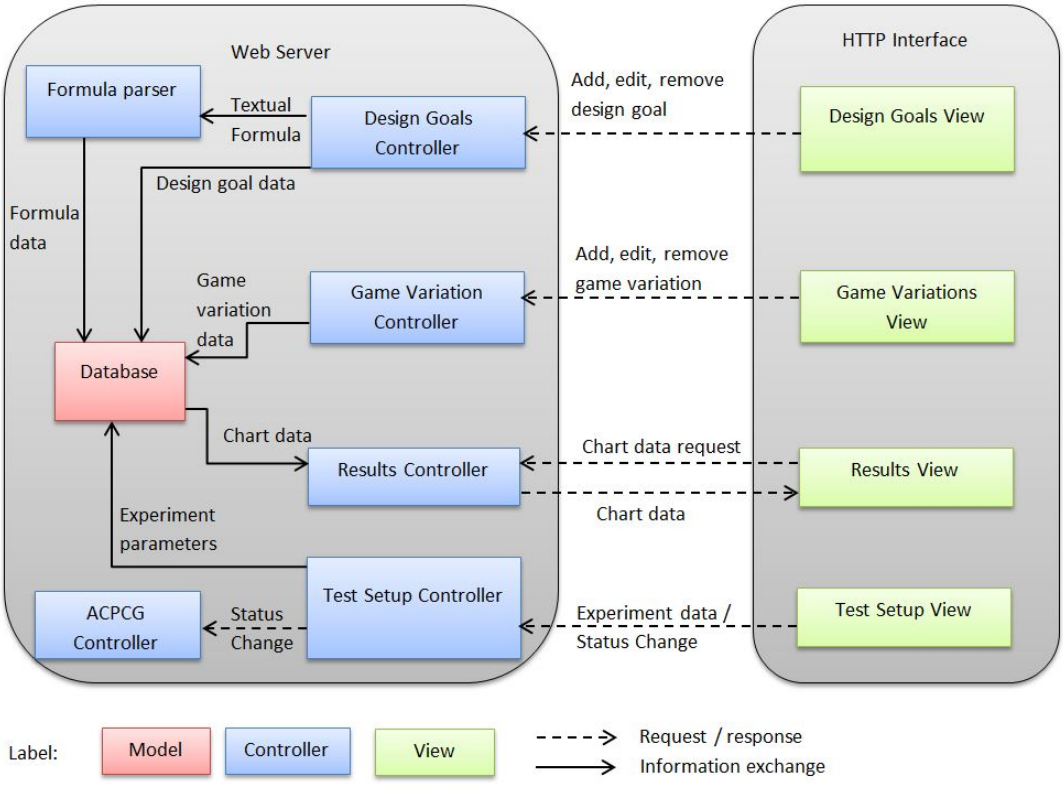


Figure 5.2: The main data exchanges between the interfaces and the data models

such as the crossover and mutation types and selection method.

The Game Quality entity represents the variation details of each feature, which represents a game parameter that can be modified. The variation is regulated by a domain, a step value, a probability function and, depending on the chosen function, a mean value and a standard deviation.

Design goals are the objectives that the designer seeks to achieve, defined in the form a formula and bound to a domain. The formula representation in the database was designed in a joint effort between us and the client, in order to find a dynamic and efficient method that could be used for as many situations as possible. The result was the Formula Block entity, which works as a formula tree, having a first and a second Formula Block, along with an operator, a function and a value. While a formula block that has a value field represents a number, one that has a function represents a user created function, for example, a function for counting the appearances of an event. The various blocks are then united blocks that have a left and right block, along with the operation (sum, difference, multiplication or division) to be applied to them, and so the formula tree is connected. For instance, using the formula *2 + $selects* will result in the following 3 formula blocks:

1. one with null in the first and second blocks, null operator, null function and value 2

2. one with null in the first and second blocks, null operator, null value and the function relative to $selected (count in this case)

3. one with number 1 in the first and with 2 in the second blocks, sum as the operator, null function and null value.

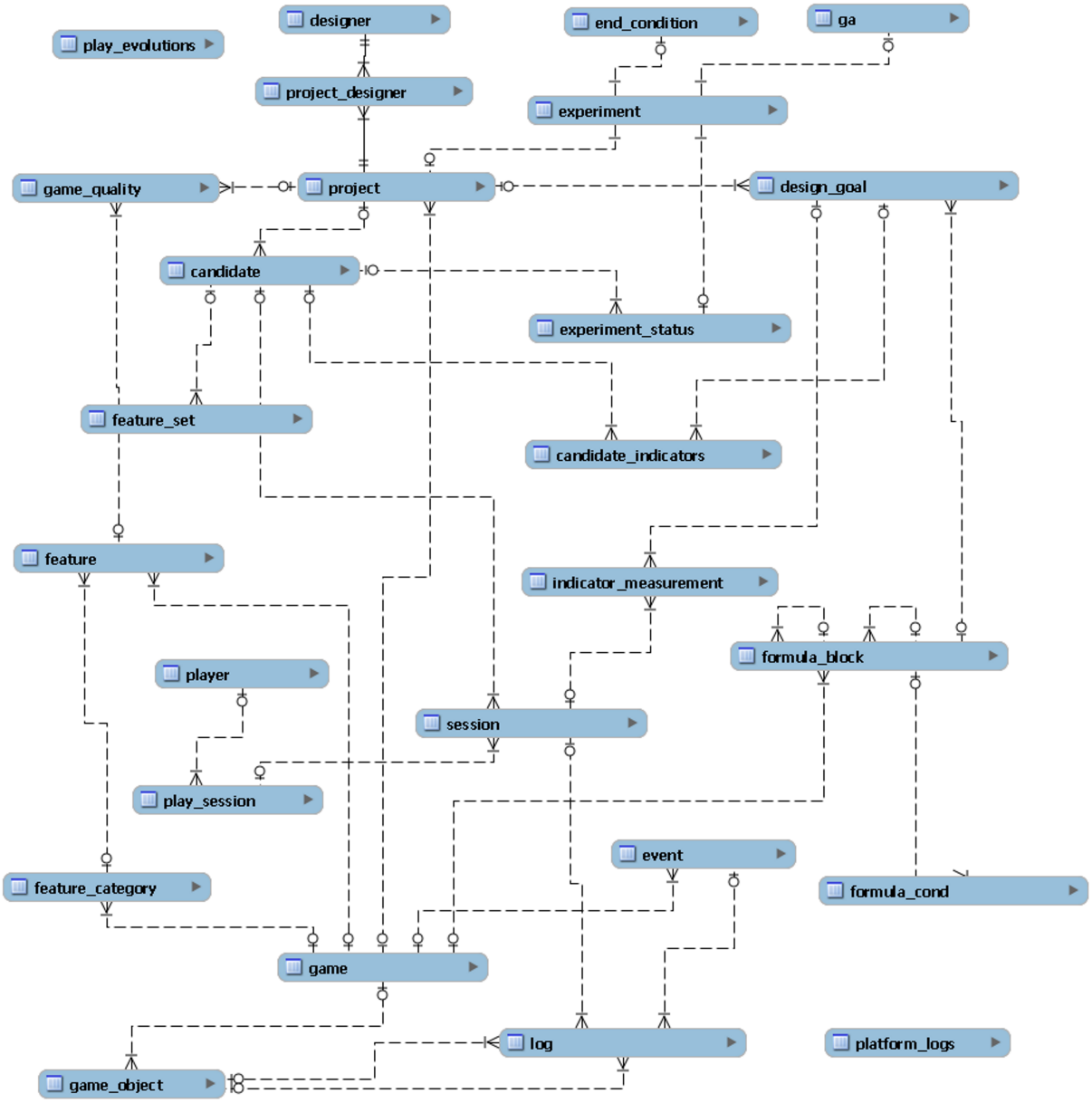


Figure 5.3: Compacted version of ER diagram for the Database. The complete version can be found in Appendix C

In order to transform a formula in the form of a string into an actual for-

mula tree, we needed to parse it and create the corresponding Formula Blocks. When parsing and creating its formula tree, we were faced with the problem of building the tree in accordance with operation priorities. Given that operation priorities and ambiguity are an Infix notation problem, we used the Shunting Yard algorithm[1] to change the formula to an unambiguous Reverse Polish Notation. Not only does this transformation remove any operation priority issues, it also allows a more simple and linear construction of the tree.

When a feature set is generated, it is stored in the Candidate entity and the Feature Set entity stores the current iteration of each candidate.

The Play Session entity contains the existing play sessions and their information. A Play Session is compromised of various Sessions and are associated to a player in the Player entity, which contains the registered player's informations. The Session has its basic informations (start time, end time, ...) as well as the id of the candidate used in said session. While the Play Session is comprised of the full play time, the Session is a block of that played time. For instance, playing a campaign can be seen as a Play Session and each mission of that campaign is a different Session. Each Session also has its corresponding Logs (metrics) that will later be compiled into the used candidate's Indicator Measurements. Indicator Measurements are the scores given to Design Goals for each session and are also compiled into the averages for each candidate and stored in the Candidate Indicators entity.

Each entry in the Game entity corresponds to an archetype game. A game has a set number of registered game objects, events and features. The game objects and events exist for the purpose of registering metrics (On the log entity). Game Objects represent the in-game objects such as a unit type and are used in the Logs subject and predicate fields. Events represent something that happens in the game, such as the player taking an action or a unit being destroyed/defeated.

5.2.2 User Interface for the Designer

The user interfaces were developed according to the previously mentioned MVC architectural pattern.

The interfaces data flow is done in a simple way. After the user uses an address, the Play routing system verifies which controller is supposed to call and what function of that controller is to be executed. The interface related addresses are all directed to the Project Controller and, based on this address, it returns the corresponding page rendering back to the user, as seen in figure 5.4. When creating the interfaces, the need for data structures that could store information and be transfered to the views arose. The solution was the use of Data Transfer Objects (DTOs), classes made specially to hold the desired information and that can be manipulated at will.

Since there were no security requirements, there was not much concern about protecting the system against most kind of attacks. The only protections implemented were the use of an authenticator and query builders. All function calls on all controllers are protected by the Authenticator controller, which verifies if the session has a user logged in and, if not, returns to the login page. The query builder provides an easy way to create queries to the database and, at

[1] https://en.wikipedia.org/wiki/Shunting-yard_algorithm

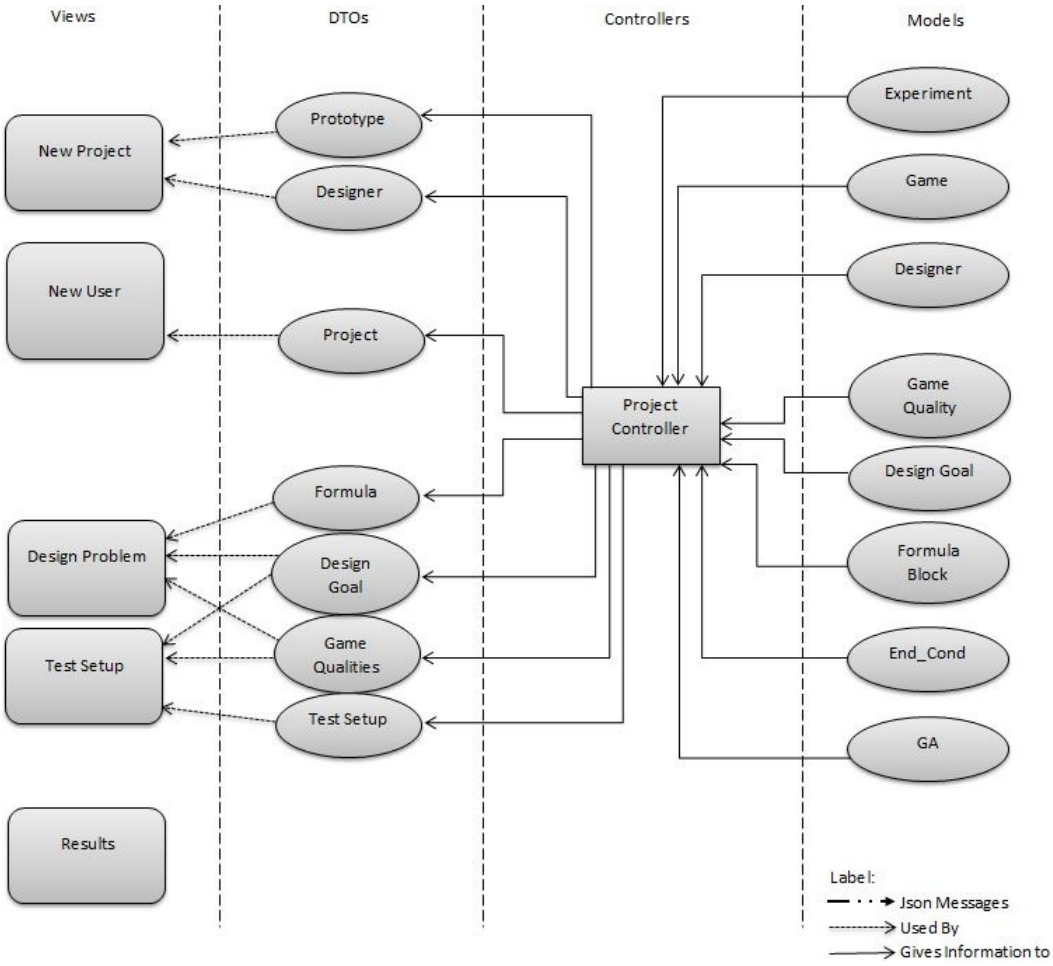


Figure 5.4: Centralization of Project Controller for DTOs creation and Views rendering

the same time, is a way to prevent some forms of SQL injection.

Figure 5.5 displays the MVC elements referent to the interfaces. Since the login interface operations are simple, that view was not included in the diagram. Besides the login, all other 5 interfaces are present, along with their DTO's, their respective controllers and the accessed models.

The New Project view corresponds to the interface for the creation of a new project (figure A.2) and is managed by the new Project Controller. When creating the page, the list of designers is presented to chose which will have access to the new project, this way requiring the DTO with the list of available Designers. Given that the project needs to use one of the registered prototypes, the DTO with the list of currently available prototypes is also sent. When the user creates a new project, the request is sent to the Project Controller, that will then proceed to add the new project to the Project entity and will also create its entry on the Experiment entity.

The New User view uses the New User controller to add new designers into

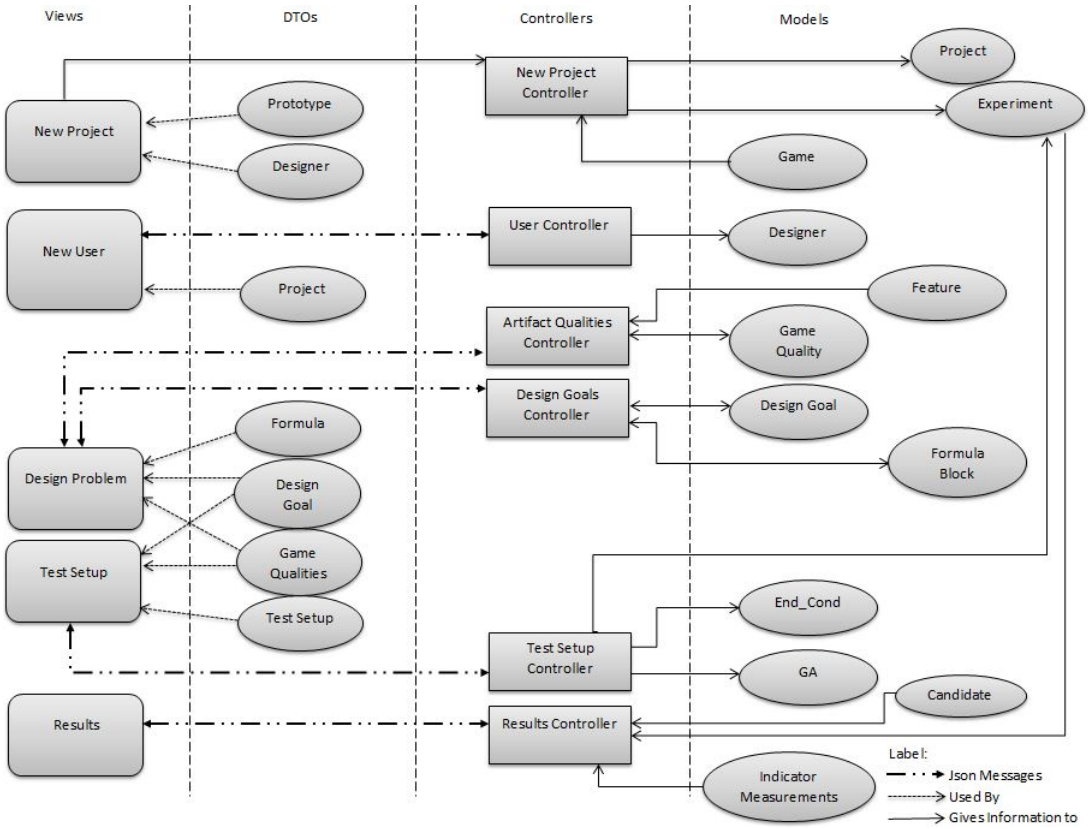


Figure 5.5: The MVC for interface related elements

the system. When creating the user, the option of adding permissions to new projects is present and their information is passed through the project DTO.

The Design Problem view is composed by two different parts of the project: the design goals and the game qualities (previously called artifact qualities). For this reason, instead of using a single controller, its functionalities were split across two different controllers. The Design Goals and Game Qualities DTOs are used to store the information that will fill the tables seen in figure 4.1.

The Artifact Qualities Controller is the responsible for the possible operations in the interface corresponding to figure 4.3, such as adding and editing artifact qualities and making them vary or not. Any of these actions need to take place as soon as the user requests them, so they are all done using AJAX to trade JSON messages. Similarly, when choosing the wanted feature, a Json message is sent to the server and its response contains all the possible features for the project along with their categories. This way, any modification done to them while using the interface will be dynamically updated instead of waiting for the user to refresh the page.

The Design Goals controller offers all the functionalities related with the design goals management as well as formulas management. Just as with the artifact qualities, the design goals management actions are done using JSON messages through AJAX. Although the management of the Design Goals ba-

sic parameters is simple and straightforward, the formulas are a bit harder to manage. When a formula is created, the controller simply creates a Formula Block tree with the corresponding expression. The edition of a formula is not as linear as the addition, because the formula can be altered by adding or removing blocks or by completely rewriting it. Since the formula can be edited in so many ways, the controller clears the tree as much as possible by deleting all Formula Blocks in it except for the custom formulas created by designers. A new tree is then built, just as if it was the first time adding it to the design goal. This simplifies the process and clears unwanted unused Formula Block nodes from the database.

The Test Setup DTO contains the information needed to display the Test Setup view. It contains all the information about the test (number of play sessions, state, ...) as well as the definitions of the currently used genetic algorithm (selection type, crossover type and mutation, ...) and its end condition (type of end, ...). The Test Setup also uses the information from Design Goals and Game Qualities as they are displayed for the user to use as reference. When saving the changes made or when changing the state of the experimentation (start, pause, stop and restart), a JSON message is sent in a AJAX request. Just like in previous cases, this is done to enable real time requesting to the server, without the need to refresh or change pages.

The Results view is a mere display of gathered information from the database. it initially uses no data from the server so it does not need any kind of DTO. When a chart or table is requested, a JSON request is sent via AJAX to the Results controller, which will collect and compile the asked data. Once this is done, the controller sends back the information for visualization. The used visualization methods are fully described in section 6.2.2.

5.3 Instrumenting Dune Legacy for Data Collection

Since Dune Legacy's purpose was redefined as an archetype goal with the goal of gathering data, its structure had to be slightly changed. Given that a full developed game was used, the architectural changes are few, but some affect the game at multiple different points. The only quality requirement for the archetype game was that the process of fetching the feature sets, applying them and that the collection of metrics and their submission were transparent to the user.

The connection to the server was achieved using sockets to send and receive HTTP requests and responses. As C++ does not have a built-in support to manage HTTP requests, all the messages sent to the server were created by us. The standard used was HTTP 1.1, so we had to manage all the required headers, such as the message length and the message type. The creation of all messages was managed by a class created solely for that purpose, the ConnectionManager.

The metrics collection was done creating a class that would manage their creation and their writing to the metrics file. By doing so, whenever a metric had to be collected, we would just call said class and pass the arguments to its respective method. This was the simplest and cleanest way found because the metrics collections are done in tens of places scattered across the game's source

code.

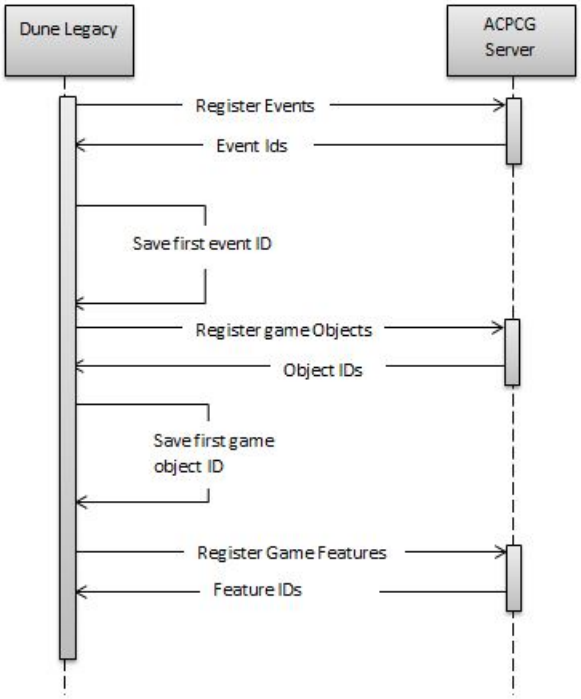


Figure 5.6: sequence diagram for the event, game objects and features register in the ACPCG platform

To use the archetype game with the ACPCG, we needed to register all of the game variables to be generated, along with all events and game objects that were going to be present in the metrics files, into the web server's data model. In this project, we decided to have the prototype registering all of the required information, because the game needed a relation between its objects and their Ids in the database. Figure 5.6 show the sequence of steps done by the game in order to register all needed components. The game starts by register all events and saving the id of the first registered event. This first Id will later be used as an offset between the Ids used by the game and the Ids in the database. For example, a game uses 3 events (a, b and c) and their Ids in the game are 0, 1 and 2. If, when registering them in the database, we get Ids 3, 4 and 5, the offset between them will be 3, meaning that any metrics sent must have the game's event id +3. The reason behind saving only the first Id is that, at the moment, there is no concurrency in the system, therefore the registered ids are in sequential order. The game objects registration happens exactly like the described event registration. When registering features, there is no need to save their ids because they are always passed to the game as an ordered list.

The actual expected sequence for the game is illustrated in figure 5.7. When starting the game, the player inserts a username and it is registered in the web server's models, returning the id of the new or existing user for that username. At this point, the player's play session is registered, bound to the player by his

id. A play session will be composed of one or more sessions, each corresponding to a mission in our archetype game. The game then fetches a Feature set from the web server and registers a session associated to it. While the player plays the game, the game is collecting gameplay metrics and, when the mission is over, the game registers the session's end followed by the metrics file's upload. The cycle of fetching feature sets, registering the session's start and end and uploading the metrics file continues until the user leaves the game, at which point, the game registers the play session end.

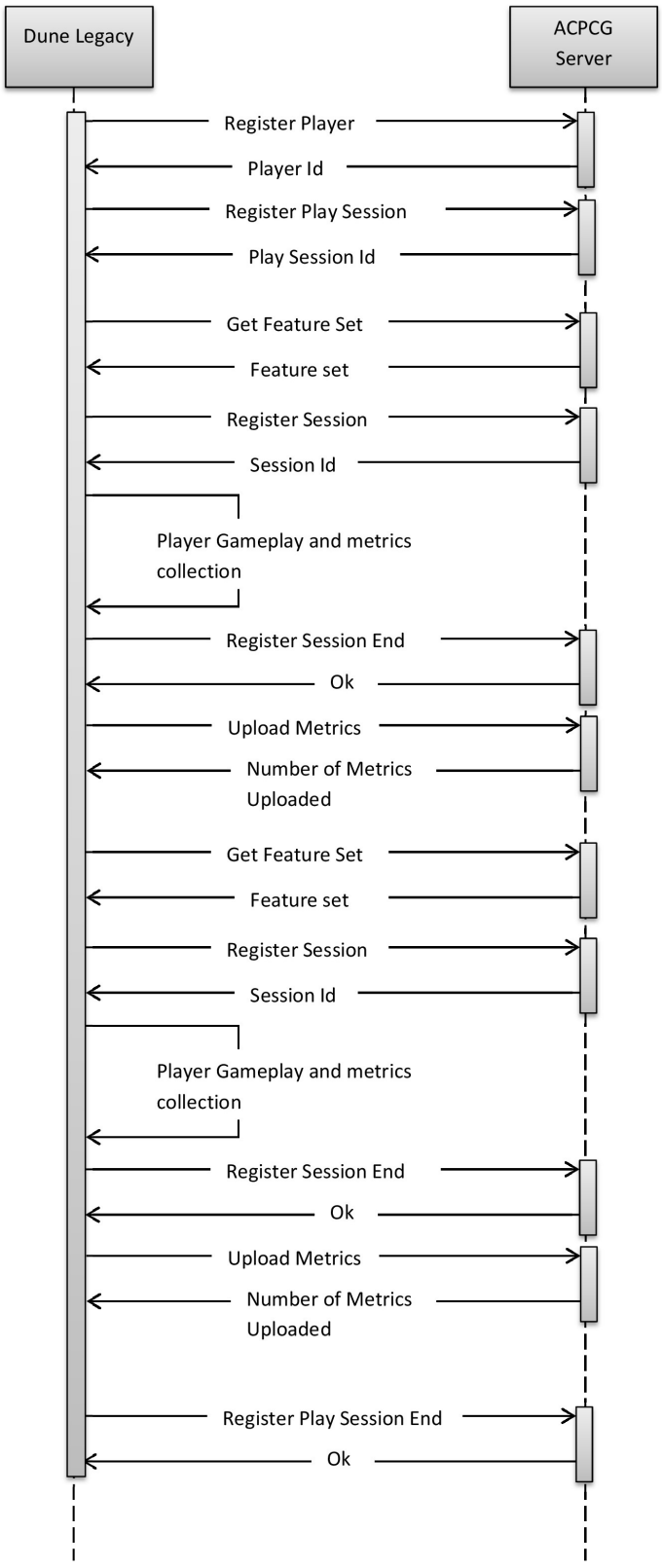


Figure 5.7: sequence diagram for 2 maps played by player

Chapter 6

Development Activities

This chapter describes the development activities done regarding both the Archetype Game and the Web Interface. It contains the used approaches, as well as the used tools to achieve the architecture proposed in chapter 5.

The first phase was the selection of the game to transform into an archetype game, followed by the actual development activities. The development activities were done in the following order:

- Selection and Instrumentation of the Game Archetype:
  - Game Archetype Selection
  - Metrics Collection
  - Feature Set Usage
  - Connection to Server
- Development of the Crowdplay Platform Interface for Designers
  - Login and Register
  - Paging System
  - New Project
  - Design Problem
  - Test Setup
  - Results

The listed interface development activities refer to the creation of all elements necessary for them to work, including the respective Scala/HTML pages, Javascript files and controllers.

Considering that an agile methodology is being used, the fix for any problem found with already developed elements can be scheduled between the current and the next activity or even, depending on the effort needed to fix it, during the current one.

The next sections describe the process behind the development activities as well as the archetype game selection.

6.1 Selection and Instrumentation of the Game Archetype

In order to test the platform and the investigation hypothesis, an archetype game was needed. Although these tests could have been performed with the already developed Infinite Mario Bros archetype game, as a way to increase the range of tests done to the ACPCG, a different game was chosen. Being that the last archetype game belonged to the Platform genre, and that the tests reflect a positive result, the new game should belong to a completely different genre, such as the Strategy genre. The reason behind the Strategy genre is tied to the amount of available data that can be collected about the player's gameplay. Not only that but strategy games are one of the types of games with the most complex mechanics, putting together resource and population management at the very least. If the set system can balance a game with so many different elements, it is likely to be able to produce good results in other games as well. Since the new game would have to suffer changes to its structure, the list of possible games had to only include open-source games, leaving us with the following list.

6.1.1 Candidate Game List

Micropolis[1] is one of the oldest city building games and was based on the original SimCity code. In this game, the player takes the role of Mayor and City Planner in order to build and maintain a city, which is populated by Sims (short for simulated citizens). These Sims work as the player's population so it is the player's job to take care of them and make them prosper. If the player is unable to do so, the citizens will become unhappy, leaving the city and reducing the amount of taxes that can be collected. Lower taxes means less money, leading to less options for the player. Besides having to provide the population with the needed buildings and services (such as fire departments, power stations, ...), the player also has to deal with several incidents such as explosions or malfunctions.

Dune Legacy[2] is a game based in the Real Time Strategy game, Dune II. Dune II was one of the first Real Time Strategy games to be launched, and the first to use the mouse as a mean to control the player's units. This established a new way to play strategy games, leading the way to the development of new games such as Command & Conquer. The game is set in planet Arrakis, also known as Dune for being a planet covered mostly by deserts. Planet Arrakis is the only source of the most important resource of the universe, the melange spice. Arrakis is being contested by 3 houses that are competing for its control: Ordos, Harkonnen and Atreides. The player is set as a commander of one of the houses and is given several missions to expand the house's influence on the planet. The missions objectives vary along the lines of defeating all other present houses or collecting a given amount of spice.

BosWars[3] is a futuristic game in which the sole propose is to beat every enemy unit on the map. In order to accomplish this, the game is based in a strong economy and resource management. The players have two different

[1] http://www.micropolisweb.com/
[2] http://dunelegacy.sourceforge.net/website/
[3] http://www.boswars.org/

Figure 6.1: Micropolis game screenshot

Figure 6.2: Dune Legacy game screenshot

resources at their disposal: Energy and magma. While energy can be produced by power plants, magma needs to be pumped from hot spots spread across the map, meaning that controlling a larger portion of the map will grant the player a better economy to play with.

Although not a Real Time Strategy game, the *opensource* version of Sid Meier's Civilization franchise, Freeciv[4], was also thought of as a candidate and added to the list. As a Turn Based Strategy Game, the players take turns giving orders and assigning actions to their units. Most of the desired components are

[4]https://play.freeciv.org/

Figure 6.3: BosWars game screenshot

still present in the game: resource management, player economy, military power and strategy. FreeCiv is, probably, the most complex game in the list, featuring many different perspectives in a nation management. While other games focus mainly in warfare(BosWars) or resource management(Micropolis), Freeciv has a good balance in all elements, leaving the players with several different ways to be successful and much more diverse strategies. The player is set as the leader of a chosen civilization in the beginning of history and is given a few units, including a settler used to create a city. During the game, the player will be confronted with other civilizations who can be interacted with. The objective of the player is to manage the scientific, military and economic growth of the nation while keeping the citizens happy and managing the diplomacy with other civilization leaders. The ending conditions contain victory by conquest, in which the last standing civilization is the winner; by being the first nation to send a spacecraft to Alpha Centurion, once the civilization technology level allows it; or by being the civilization with highest score at the end of 5000 turns.

6.1.2 Choosing Criteria

In order to choose a Strategy game for this work, there were some development related criteria the games should fulfill.

The code structure and clarity was an important aspect since the available time for learning it was short, resulting in games with a very complex structure being left out. A second aspect was how much of the code could be changed, or rather, what kind of changes could be made? Is it possible to make big

Figure 6.4: FreeCiv game screenshot

game changing modifications or simply slight changes to behaviors? A concern associated with these two criteria was the game's development language, since it can both limit the possible modifications and affect the comprehension of the existing code. The chosen game should also be checked for stability issues that could compromise the data gathering and, consequently, the results of the experimentation. On the other hand, the chosen game should have a good parameter variety and the possibility to gather as much information as possible in the form of metrics. Given these restriction, a number of candidate games were promptly readied and analyzed.

Candidate Game Criteria Evaluation

In order to choose which game to use, a table was created and filled with the characteristics mentioned before: code clarity, modifiable code, stability, possible features and extractable metrics. Although some of the table's fields were not completely filled (metrics and features), it shows the main notions for each game, which is enough to make a supported choice on which game to use. Since the game is part of the strategy genre, metrics such as unit movement and building creation are common to all the games. Following is a description of that table, which can be found in Appendix B.

Freeciv's code was clear and easy to read, having intuitive variables and built-in functions to manipulate the game's lists (units list, cities list, players list, ...), but the main issue was finding the locations of such elements in the game's files, making it a time-consuming task. Considering the HTML5 version of the game, it uses the same engine as the regular version, leaving the same structural problems. This game uses ruleset files, similar to .ini configuration files, to load game parameters, making it easy to use the needed feature sets.

The rulesets are separated through several different files, having files specifically for units, terrain, technologies, government and so on. Documentation for the game comes in the form the Freeciv Wiki webpage which contains information about some module locations, coding style and structure. As for stability issues, the game may crash while creating a new play session and there are some known bugs when automating the units behavior, but is otherwise stable. The following parameters were thought as possible features: unit requirements and prices, technology requirements, map visibility, city growth rate and unit promotion probability. Retrieved gameplay metrics can take form in the amount of units, the amount of resources, the game year (works as turn counter), the amount of resources spent and the number of promoted units.

Micropolis had its interface coded in TCL which, when added to the non-existent experience with the language, made it hard to read and comprehend. Micropolis also had several file types, such as the .cty files, that could not be directly manipulated outside the game or were not intuitive, making it hard to verify their manipulation easiness and, on top of the low intuitive code, only a few possible modifications were seen as easy or fast. Regarding the game's stability, no problems were found in the game's development documentation that could cause serious consequences. The tax rates, the game speed and the initial funds were all seen as possible features. Taken gameplay metrics for Micropolis would be the city population, the citizens happiness, the money in the treasury and the public opinion, among others.

DuneLegacy's code was well structured, hierarchically divided into folders and its variables had intuitive names. Being well structured and divided, it is easy to find elements that need modifications besides having a good support on both the game engine and the project deployment. The game is fully developed in C++ and a CodeBlocks project file was also supplied to ease the compilation process. The game engine is stable and the most worrying stability problem is related to very long periods of gameplay, although this problem could not be replicated while testing the game itself. Possible features are the harvesting speed, how much slower a full harvester moves, units and buildings attributes (such as health and damage resistance), game speed and Starport's prices update delay. Metrics for DuneLegacy can be the time spent in each map, the amount of harvested resources and the amount of built/lost units or buildings.

BosWars was very similar to DuneLegacy in terms of code structure and clarity. It has a clear structure, hierarchic division, seemed easy to parameterize and the source code contains documentation to help development. The game engines are built in C++, but most components are coded in LUA language which, even with no experience, is easy to understand. Modifying unit parameters can be easily done since each unit has its own file where they are stored, but changing engine related aspects (such as game speed and resources production) might need more effort to find. All known bugs in the game do not seem to compromise its stability or performance. Besides modifying the units parameters, the amount of initial units, the amount of starting resources and game speed seem good features to influence the gameplay. Extracted metrics are also similar to DuneLegacy, varying from unit count, resources usage and game time.

Criteria Evaluation Results

Given the above details on each game, Micropolis was the first to be excluded because it did not seem to have the same content quality as the rest, for its lack of intuitive code and due to the lack of experience using TCL. Although Freeciv was a very complete game with varied elements, this also made it a very complex game and difficult to fully manage in the alloted time. The chosen game was then either BosWars or DuneLegacy. Since their elements are so similar it became a matter of preference rather than a technical comparison and so, DuneLegacy was chosen because it offered different styles, other than military dominance and because its elements were all developed in the same language.

Candidate Features

With the Candidate Game chosen, a list of features to be generated by the ACPCG was readied and analyzed. Since the game is compromised of several different units and each has various parameters, all of those parameters were set as an individual features. This means that, for each of the existing 40 units/buildings, a total of 12 parameters can be modified every game. Although buildings do not usually have parameters such as Weapon Reload Time and Damage, these were still counted as they are present for all objects in the game engine, and they might be useful for certain game designs.

Besides the units and the buildings parameters, there were also selected parameters related with the game engine itself, such as the game speed and the mission to play. The fact that the mission to be played is a feature means that the player can no longer choose his preferred house, removing some menus from the player's interaction.

Counting 12 parameters for each of the 40 units/buildings, along with the 11 game engine values, each feature set contains a total of 491 parameters. For reference, the complete list of units and their parameters can be found in Appendix D.

Candidate Metrics

The metrics taken from the game are mostly from interactions from the player with the game. The collected metrics follow the ACPCG's pattern for knowing where, when, who, what and to whom the actions were taken. For this, each metric contains the x and y (z is always 0), the time and engine time, the subject and its value, the event and the predicate and corresponding value. They are taken in several occasions, such as moving a unit, ordering to attack, placing buildings, and so on. Each interaction uses its own event and, when keeping info about a unit, the subject and/or predicate fields refer to the unit type and their values is the unit's in-game object id. The only exceptions to this rule are the Resources and Health whose value represents their respective amount. All of the collected metrics and their descriptions are listed in Appendix D.

Second phase of features and metrics

The initial list of features and metrics were filled with the general requirements a designer could want but, since we did not know what kind of game design

were going to be chosen, it proved to be incomplete.

When realizing the first use case with the actual game designer, he showed the intention of changing the game's playability in an unpredictable way (described in section 7.2). To do this, he required a set of features that were not initially thought of, some of which were not even in the game to begin with and, for these, new game mechanics had to be implemented.

First of all, he required that standing units would lose health points, forcing the player to move them about as much as possible. The units health points would have to be restored in two ways: by moving and by being attacked by the enemy. While the enemy attacking would restore health, the designer also said that the player attacks would have to continue inflicting damage, as in the original game. The designer also asked if there was a way to easily define the same value for a parameter in all units, so we created an override system that contained all unit parameters, allowing the designer to manipulate the player's and the opponent's parameters separately.

Along with the new features, and in order to create a specific indicator, the designer also asked for a new metric: a state collection of all units, that would represent whether the unit was stopped or moving. This new metric is collected every second, along with the existing units health and the player's resource count.

Candidate Game Modifications

To be used along with the ACPCG, to be able to use the feature sets received from it, and to enable the collection of gameplay information, a series of changes must be made to the original game. These changes compromise the creation of a connection to the ACPCG, the communications with it, the needed modifications to change the original variable values, the automatic collection of the information needed to create metrics logs and some minor changes to the game interface flow.

Connection To ACPCG The connection on the ACPCG server side was already developed, meaning that, in order to create a connection and communicate, the archetype game should follow its model.

The first problem encountered was that c++ language (pre c++11) does not have a built in library to create HTTP requests. This led to two options: either use an existing library to manage the requests or create a module that creates and manages the requests.

After exploring the first option, a few different libraries were found such as Qt, Boost and curlpp (libcurl's c++ wrapper). After further analysis and experimentation, problems or limitations were found on all of them. All that was needed was a connection manager but, to use the one of Qt or Boost, their whole libraries and packages had to be added for them to work. On the other hand, curlpp is a library dedicated to create and manage connections and, on top of that, it was also the tool used to test the server's requests in the connection's supplied documentation. When trying to integrate curlpp with the project and setup a simple test, problems related to compatibility and dependencies appeared and so it was also put aside.

Given the tight schedule and the fact that the base connection should be done within 1 or 2 days of work, the second option was chosen. This approach

was thought of as better for 2 reasons: it was not dependent on other libraries, as it only used built in functions; and there was a deep understanding of how it works, given that it was a module created solely for this project. On the other hand, the development process is more complex than using an existing library and could take more time than expected. Since the game was originally bound to Windows operating systems only, the connection was implemented using the c++ built-in library winsock2 to create and manage connections using sockets. The major development issue was that, using this method, all the messages had to be manually created, from the headers to the actual messages, using the HTTP 1.1 specification.

The requests done to the ACPCG server all use JSON messages as a means of communication. These JSON messages must always contain 3 key elements that serve the double propose of security and experiment identification: the archetype name registered in the ACPCG and the login and password associated to a specific experimentation that uses this archetype. Aside from these, the message must contain the specific fields for the given request. For instance, the request used to register a player (figure 6.5) must contain the player's username to be registered, along with the previously mentioned elements. Due to the limitations of using sockets to upload files to the ACPCG and the need for the 3 key elements stated above, the file contents are also sent in a JSON message. Although this approach has not been throughly tested, there has been no signs of unwanted behavior, such as long waiting times or data loss.

For every request done to the ACPCG server, it sends a response back with the appropriate status (Ok, BadRequest, ...), a status message and a any additional fields depending on the done request.

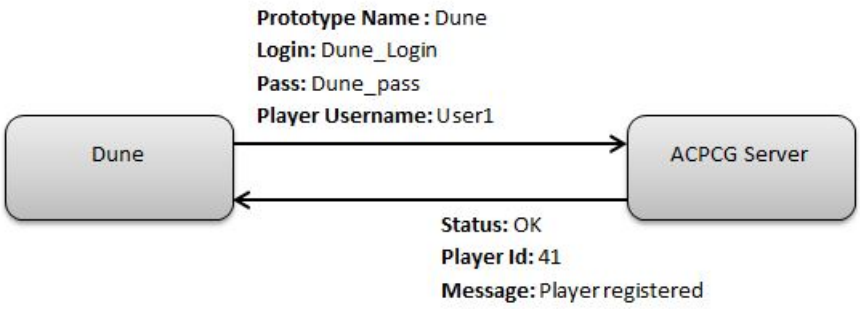


Figure 6.5: Message exchange example for registering a player

Game Flow While modifying the original game, there was a need to not only add new elements to its structure but also to change the players access to certain elements.

To play a campaign game in the original game, the player starts at the main menu and then proceeds to the single player menu, where he can choose the campaign mode. After selecting it, the player has to choose a house to represent and then proceeds to play that house's 9 missions in sequence. At the beginning of each mission there is a briefing of the objectives, followed by the gameplay and finally a debriefing of the players achievements. After the debrief,

and if the player cleared the mission, the game statistics window is shown and the player advances to the next one, otherwise the mission stays the same and returns to its briefing.

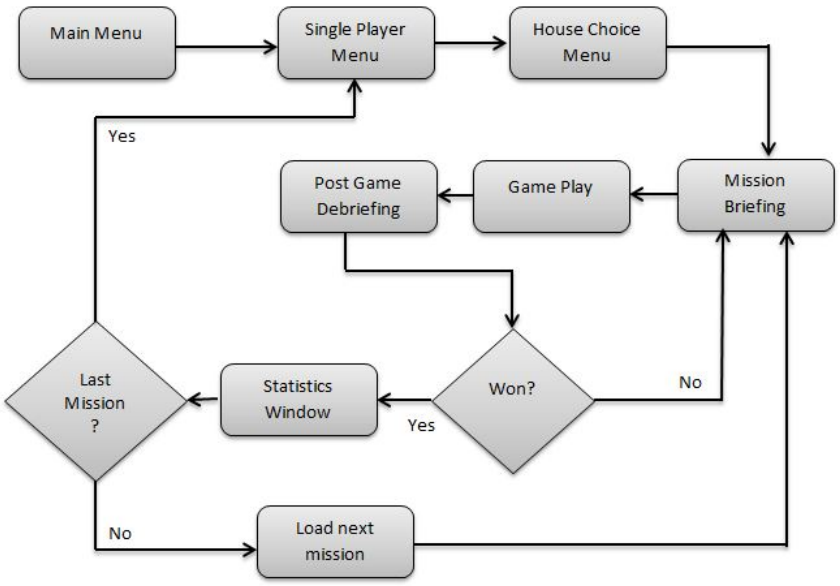


Figure 6.6: The original game sequence to play a campaign

With the new game flow (depicted in figure 6.7), the player also starts at the main menu and still chooses the single player option to go to the respective single player menu. The first change is when the player clicks the campaign button. Instead of going to the house selection menu, the player is taken to a menu where the desired username must be chosen. Using this username, the player is registered in the ACPCG server and a play session associated to that player is also registered. Given that the house and respective mission are both part of the features sent by the ACPCG, the house selection menu is skipped and instead the game asks the server for the next feature set. A session, that is bound to both the feature set and to the current play session, is then created and the game loads the received features into the respective parameters. The mission given by the respective feature is then briefed to the player and its gameplay starts. In the end, the session's end is registered and the player watches the mission debrief. If the player won, the statistics window is displayed and the metrics file is uploaded to the server. If the player lost the statistics window is not displayed but the file is still uploaded. If there are still more feature sets to be tested, the game will ask the server for the next one and repeat the process, otherwise the game will return to the player name choose screen.

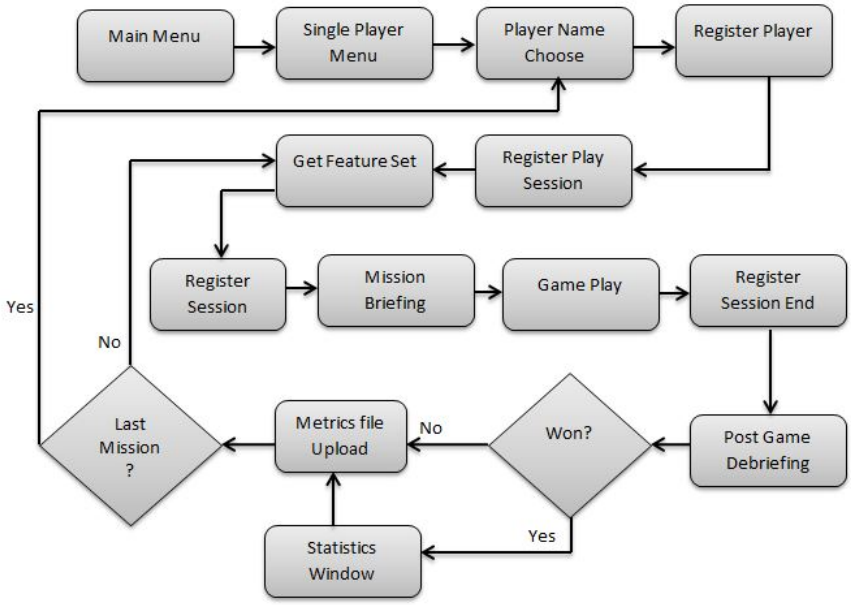


Figure 6.7: The new game sequence to play a campaign

6.2 Development of the Crowdplay Platform Interface for Designers

Once the archetype game development phase was over, the web interface started to be developed.

Given the lack of experience with the used framework, the first task was a simple one: the creation of a login screen. This simple task had the secondary intent of understanding the framework's working process. After this screen was finished, it was expected that there would be enough experience to move on to the development of more complex components. As expected, after finishing the login screen, the main Play functionalities were understood and doing the register page took no effort.

The order by which the rest of the pages were developed is related to the dependencies between their elements. The first was the new project page, which has no dependencies with the other ones and is also straightforward. After that one, the development continued to the design problem page, as it is a requirement for the test setup page.

The design problem page is the page with the most user interactions and, as such, is also the one where the Javascript code is more predominant. Since the page is divided into two parts, the design goals and the artifact qualities, the Javascript was also divided into two files with the design goals functions in one and the artifact qualities in another. This was done so that the elements are dependent only of the corresponding file, giving the future development some freedom for changes.

After finishing the design problem page, the development followed to the test setup page, as its requirements were already fulfilled and it needed little

Javascript effort.

The last page to be developed was the results page. This is due to the fact that it seemed the most complex of all pages, meaning that having the others completed would grant the rest of the time for its conclusion. The page features a series of charts and tables that are built dynamically and their layout can be changed at any time. There should also be the option to filter the charts and table data using filters but, due to the lack of time and the complexity of the task, it was canceled. Although missing this functionality will affect the final quality of the interface, it will still work properly.

6.2.1 Paper Prototypes Adaptation to Web Interfaces

The process of converting a paper prototype into its digital version can be complex. Expected behaviors on the paper prototype may, sometimes, not be possible to completely replicate when developing them. Even those that can be, sometimes take added effort to adapt, requiring a longer time span to achieve.

The game variations interface final aspect (figure 6.8) is very similar to its prototype counterpart. All of its requirements were fully implemented, with the exception of showing the used function in the game variation table and of being able to directly edit the table parameters.

The development of the design goals interface (figure 6.9) was identical to the game variations, making the development of one much faster once the other was finished. The few differences between them are the use of a formula in the design goals, instead of the game variations choice of a game variable; and the score and penalty associated with the scoring function, which were not needed in the probability function.

The current experiment can be defined and started/stopped from the test setup interface (figure 6.10). This interface features all the parameters that can be changed in an experience. The current state shows the state of the experiment: setup if it is not yet running and the user is able to change the experiment's parameters; running if the genetic algorithm is generating candidates and/or waiting for gameplay metrics; and complete if the process is over. The type of experiment determines the generated candidates. In complete type all possible candidates are generated, in sample a number of candidates are randomly generated and in procedural the platform uses the genetic algorithm for the candidates generation. Since the procedural type is the only that uses the genetic algorithm, its parameters (the genetic algorithm details box) is hidden when another type is selected. Also, depending on the chosen end type, the respective box also shows the target score, the number of maximum evaluations or nothing (if manual is selected).

The results interface (figure 6.11) is the one that looks less with the paper prototype due to the missing data filters and the added status box in the bottom part of the screen. The Current Status box was one of the final requirements of the client, and it shows details on the current experiment's statistics, such as the best score achieved, the id of the best candidate and the current cycle for Genetic Algorithm generations. There are a total of three different layouts: 4 charts, 2 charts and 1 chart. To swap layouts, simply click one of the upper right icons symbolizing the respective chart distribution. To create a chart, we have to drag one of the chart symbols on the upper left corner to one of the plus signed boxes, and then choose the chart axis's data. All charts can be sorted by

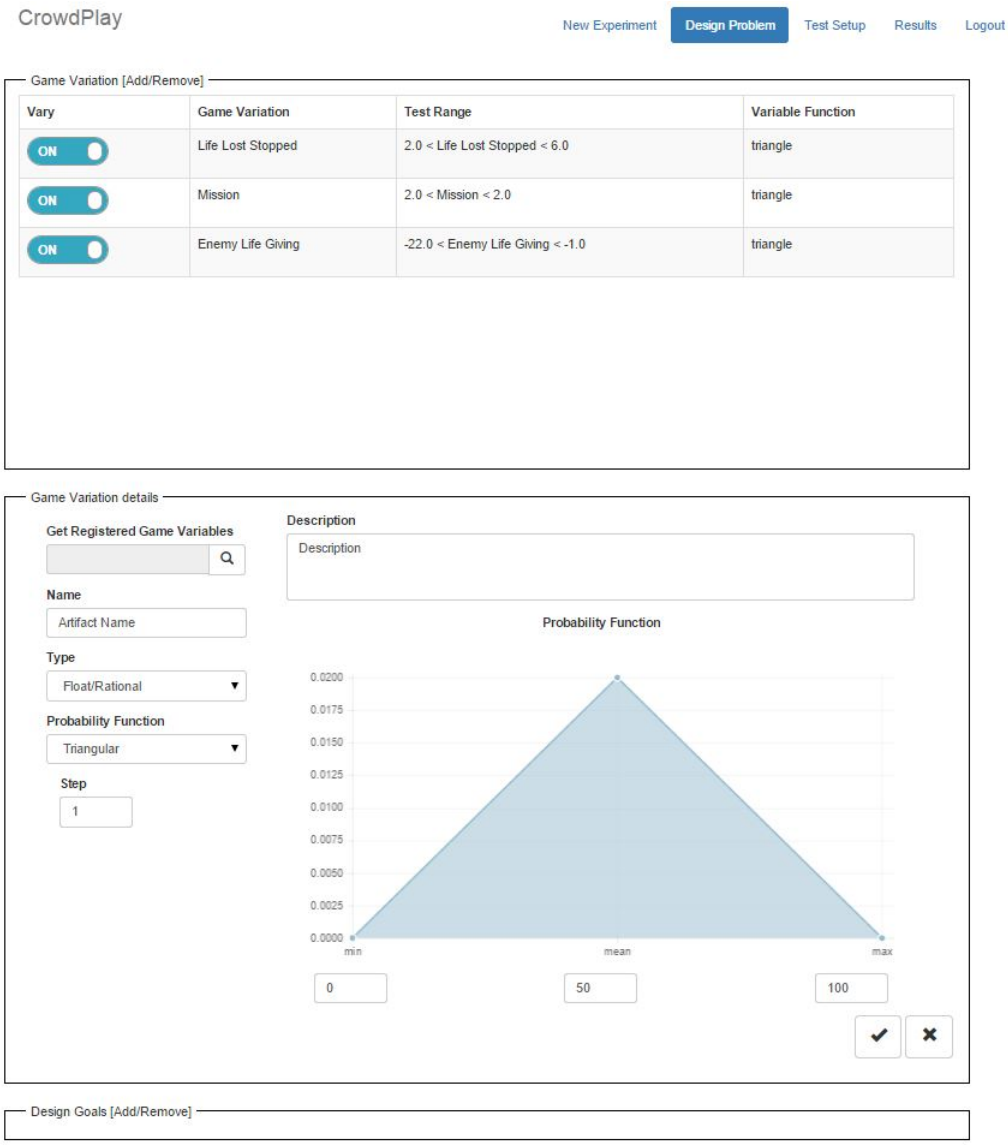


Figure 6.8: Snapshot of the developed game variations interface

dragging them to the desired slot, being the others repositioned to obey that order.

6.2.2 Technologies Used

All of the pages use the following technologies:

- HTML - for the elements creation and page layout;
- Scala - to manipulate the variables passed at the creation of the page, namely the DTOs and their lists;

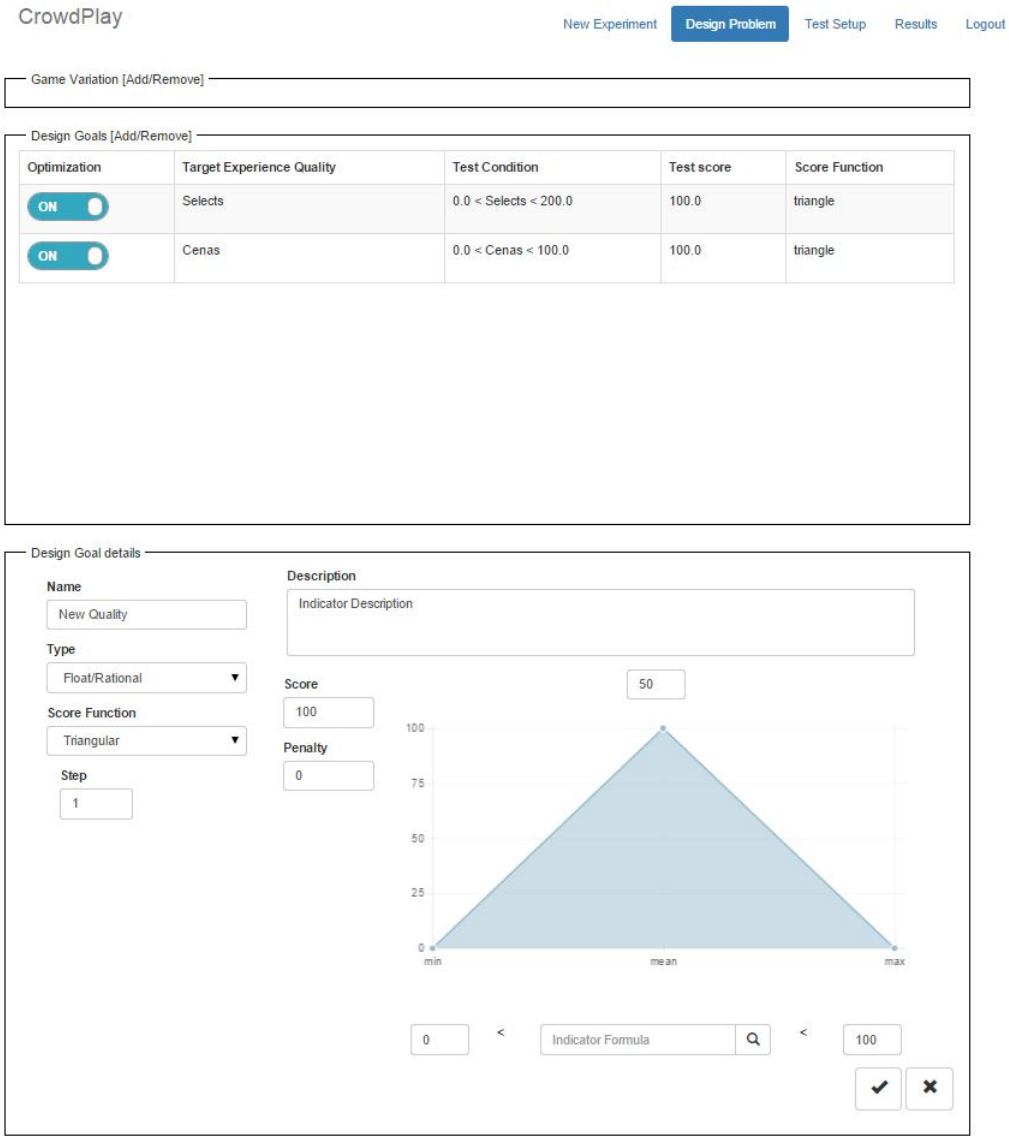


Figure 6.9: Snapshot of the developed design goals interface

- CSS - for styling and element formatting
- Bootstrap - Most of the CSS present was achieved using Bootstrap, as it provides easy to use classes that can achieve a clean layout.
- Javascript and Jquery - used when the wanted behaviors couldn't be achieved by any of the other tools.

In the design problems page, in the adding/editing forms for both design goals and artifact qualities, there was the need to create a visual representation of the selected function. This was achieved by using two different libraries: C3Js

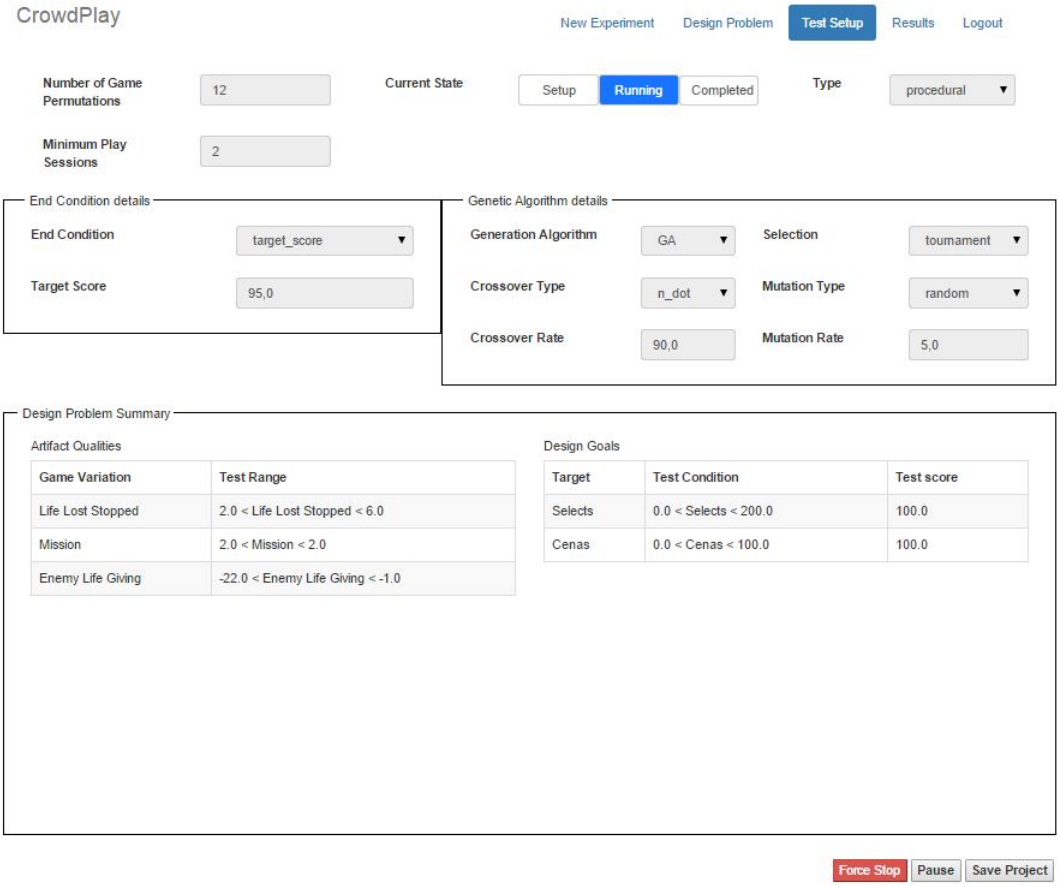


Figure 6.10: Snapshot of the developed test setup interface

and ChartJs. Initially, ChartJS was used to create the triangular and gaussian functions, but it did not give any support to generate a step probability function. The solution was using C3Js to create the step function but, instead of redoing the triangular and gaussian functions, kept the ChartJs as the C3Js did not offer a gaussian approximation.

In the results page, the chart selectors were supposed to be dragged and dropped and, for that purpose, the JqueryUI was used. JqueryUI is an API that offers several methods, between them some for dragging and dropping, leaving the work cut out to deciding what to drag and where to drop. The page layout changes were done using simple Javascript and Jquery to alter the classes of the displayed elements. This page also needed charts but, unlike the design problems page, the D3Js library was used. The reason behind this choice was that, although the learning curve of D3Js is much steeper than the other already used libraries, there was a Javascript library for filtering data, the Crossfilter, which could be used with D3Js. So, in an attempt to use this library to create our filters, we decided to use D3Js as the charts creation library.

Figure 6.11: Snapshot of the developed results interface

6.2.3 Developed Modules for Crowdplay

This section presents a quick list, describing all the functionalities, ordered by development, that were implemented in this project.

A. Instrumentation of the Game Archetype

(a) Development of the module for creating and writing metrics to a file

(b) Metrics extractions added on the needed game actions

(c) Development of the module for receiving and saving the game variations

(d) Replacement, in the game's source code, of the original variables for the saved game variations

(e) Development of the connection module

i. creation of the request and response manager, responsible for building the requests headers and managing the responses status
ii. creation of the needed request methods for:
A. register game objects
B. register events
C. register players
D. register game variations
E. register play session and its end
F. register session and its end
G. get game variations
H. metrics file upload

B. Development of the Crowdplay Platform Interface

(a) Development of the login and register interfaces and respective controllers
(b) Development of the page template used for the remaining interfaces
(c) Development of the Design Goals interface and its controller:
i. creation of the page layout (table, parameters, chart positioning, formulas modal, ...)
ii. creation of the scripts responsible for the page behavior (table selection, details box hide/show, chart visualization, ...)
iii. development of controller method for adding, removing and editing design goals
iv. development of controller method for formula creation
(d) Development of the Game Variations interface and its controller:
i. creation of the page layout (table, parameters, chart positioning, game variables modal, ...)
ii. creation of the scripts responsible for the page behavior (table selection, details box hide/show, chart visualization, ...)
iii. development of controller method for adding, removing and editing game variations
(e) Development of the Test Setup interface and its controller:
i. creation of the page layout
ii. creation of the scripts responsible for the page behavior (GA and end condition details hide/show, state changing, ...)
iii. development of controller methods for starting, stopping, pausing and saving the experiment
(f) Development of the Results interface and its controller:
i. creation of the page layout
ii. creation of the scripts responsible for the page behavior
- selection of chart type or table
- layout change between 4, 2 or 1 chart
- chart sorting

iii. development of controller method for returning chart data
iv. development of the various D3js charts

C. issues fixing on previously developed interfaces

D. addition of results page current status details

E. second phase of archetype game instrumentation

(a) added new mechanics needed by the designer for his game design
(b) added new features requested by the game designer
(c) added new metric for the designer's experience indicators

F. minor interface issues fixing

Due to the lack of time, there were still some interface features pending for implementation, namely, the data filtering on the results page and the correction of the issues found at the usability tests, described in table 7.2.

Chapter 7

Evaluations

After having a functional system with the required qualities, the testing phase began. Given the context of the project and that the platform was fully developed during the course of this dissertation, two kinds of evaluations were needed.

The first is an usability test, to determine whether or not the developed interfaces had issues for the new users and if so, which elements needed to be altered and which would stay the same. The second was a case study, done with a game designer in order to verify if the ACPCG can be used with this particular candidate game. Both tests will allow us to reach our goal of helping getting an answer to the posed research question.

7.1 Usability Test

Usability tests vary in many ways, from the development phase that they are introduced, to the place of testing and the actions required, every aspect related to the test can influence its outcome, and so, this section serves to describe the realized test.

When developing a platform, usability tests are essential, as they can show us where the platform is lacking, be it in its layout or simply in the used terms. For this project's usability test, informal invites were done to students present in the Computer Science Department of the University of Coimbra.

8 subjects accepted, and although the group did not have the ideal distribution, we managed to get individuals with as much varied traits as possible. The ages of the subjects were comprised between 23 and 42 years (average 26.87 and standard deviation 6.31), with 5 being male and 3 female, with occupations ranging from Bachelor students to working professionals. There were a total of 4 individuals with Computer Sciences Engineering backgrounds, 4 with Design, 1 with both and 1 with Educational Sciences; all had experience designing videogames, though some more than others. Game designers usual backgrounds include Design or Computer Science Engineering, hence, the sample being somewhat representative.

The tests were done in an isolated space to avoid interruptions and external influences and took 45-60 minutes each. Since the individuals had little knowledge of the ACPCG, at first they were presented with the ACPCG concepts

and working process, in order to create a base of understanding. They were explained the step by step process, the logic behind the use of design goals along with the usefulness of the algorithm. Given the complexity and specificity of the formulas, the individuals were shown how they are created and a part of their interface. As the part of the interface shown was small and it did not reveal how to reach it, the usability test was not compromised.

7.1.1 Usability Test Walkthrough

After the presentation, the individuals were presented with the following script and asked to follow it while we recorded the individuals and their screens, revealing their actions during the test.

Usability Walkthrough

Please execute these steps one by one. After each step, contact the Experimenter.

I. Register a new Account and login into the "Usability" experiment.

II. Create a new Game Variation

1. Give it the name "Troopers BP"
2. Select the Game Variable referent to Troopers Build Time
3. Choose a 'Step Probability Function'
4. Make the Variation oscillate between 100 and 200
5. Activate 'Troopers Bp" variation

III. Create a new Design Goal

1. Give it the name "Level Time"
2. Change the score function to Triangular
3. Assign it the Score 200
4. Create an Indicator Formula that calculates the time needed to complete a level (this is the maximum value of engine_time)
5. Establish Score boundaries between 20.000 and 30.000
6. Establish mean at 25.000
7. Make it so this is optimized during the experiment

IV. Edit the 'Test Design' Goal

1. Create a new Indicator Formula for it that calculates the double of the number of unit selections in a game (a unit selection is tracked by an event of type 'unit selection')
2. Make it vary between 200 and 300
3. Delete this Design Goal

V. Setup the Experiment to:

1. Have 15 Game Permutations per Cycle
2. Define number of minimum Play Sessions as 5
3. Set the experimentation type as "Sample"
4. Setup the Termination Condition
    i. Set Condition as Target Score
    ii. Define the target score of 95
5. Save the Data.
6. Run the Experiment.
7. Pause it.
8. Stop it.
9. Start it again.

VI. Check Experiment Results

1. Log out
2. Log in with the following account "asd@asd.asd" and password "asd", experiment 'firstdemo'
3. Set Dual Chart View
4. Create Line Chart in the upper section
    i. Plot Sum of Scores in terms of play sessions
5. Create Table in the lower section
    i. Average and Standard deviation of Score, and Average and Standard Deviation of Indicator "Selects" per Generation Cycle

The script is divided in large tasks (numbered from I to VI) containing smaller steps and each individual was asked to do one of the larger at a time and notify us when each was finished. Every time one of the larger tasks was finished, we would mark the time taken, following with a small usability questionnaire about both the realized operations and the terms involved in it. This test was tailor made for the application. After doing this for all the steps in the script, the user would be asked to fill a S.U.S. questionnaire [Brooke, 1996], which is a basic usability questionnaire used to evaluate the global application.

All user's tests were video-recorded (both the screen and facial capture), and this data was compiled and analyzed. The analysis was made in several different scopes and the found issues were ranked based on how problematic they were for the user. The issues ranking can be found further in this report, in table 7.1, and the used questionnaire can be found in Appendix E.

7.1.2 Questionnaire

As stated before, in each usability test, two different questionnaires were posed to the users, each with its own purpose and with different timings.

The tool evaluation questionnaire was done during the test, between every larger task, and its questions were mainly focused in determine the difficulty felt by users while doing them. It also contained questions that would allow us to perceive if the used terminology was confusing or hard to understand, and if

the users understood what repercussions the tasks had in the experiment. Each of the larger tasks was created with the objective of testing a component of the interface and its available functionalities, covering all of the development process in the test.

On the other hand, the S.U.S. questionnaire, filled at the end of the test, was intended to o self-assess the users experience, so it was filled by each of them without supervision. Not only did this method make users feel less pressured about their answers, it also brought some closure to the usability test, allowing them to feel more comfortable while answering. S.U.S. questions relate to usability principles, gathering information about the system's ease of use, enjoyment, complexity, consistency and learnability.

7.1.3 Data Analysis

Based on the goals set for this test, the results were analyzed in three core aspects. In order of importance, these are: the amount of successful task completions, the reported difficulties and the amount of time taken to finish the tasks.

The amount of tasks failed is the most important analysis because, if the users cannot do a task, they will not understand the difficulties associated and we will not know how long they take. For this analysis, we did not operate based on how many actions a user can fail in each task but rather if the user failed any action at all. Optimally, the values for failed tasks would be zero for all tasks, meaning that every user understood how to do all of the tasks. However, and as seen in figure 7.1, this is only true for tasks I - Register and Login, V - Setup the Experiment and VI - Check Experiment Results, being that the other tasks had from 2 to 3 users failing.

In task II - Create a new Game Variation, all of the failures were related to the activation of the game variation. After careful revision, we concluded that one of the users who missed was looking for the option inside the game variation details sub-screen, while the other activated a design goal optimization instead of the variation (since both these elements are mirrored in the design problem screen). Thus, a possible solution is to make the flow between editing the game variation and activating it clearer, by giving users the option of activating game variations and optimizing design goals inside their respective sub-screens.

In task III - Create a new Design Goal, it was verified that every user that failed at least a task was not successful at creating the required formula. The reasons for this problem were various and some were repeated in more than one user:

- 1 user tried to create the formula directly in the text box
- 2 of the users did not notice the button to create a new formula
- 2 of the users thought that the function domain would be placed in the formula
- 2 of the users added unneeded conditions to the formula
- 1 user created a wrong formula
- 1 user forgot to optimize the design goal

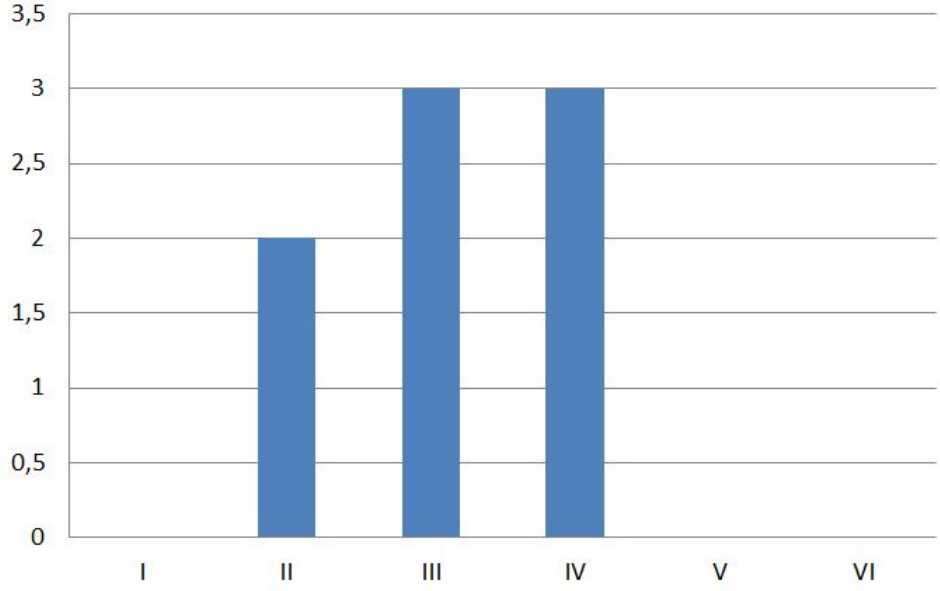


Figure 7.1: The chart shows the amount of sub-tasks failed by the users in each larger task.

Similarly, in task IV - Edit the 'Test Design' Goal - the same problems emerge as in task III - which is to be expected given their similarities. Besides the already described issues, users had trouble applying the double to the formula, trying to create a formula that would count two times the value instead of multiplying a simple formula by two in the text box. These errors highlight that the formula part of the usability test was where users most stumbled, making it a critical component for improvement.

The user reported difficulties were analyzed based on the questionnaire done between larger tasks. These had questions in which the user would have to rate a series of sentences from 1 to 5 but also had questions, about concept and functionalities, that were classified as right or wrong. As an indicator of success, the answers were compiled and then, depending on the question, we counted the users who gave at least a score of 4 or got the question answered right. The threshold of success was set when this count of users was at least 20% (6 users) and the results can be seen in figure 7.2 for score above 4 and figure 7.3 for right answers given.

Usually the threshold of success would be placed at 10% but we raised it a bit to the 20% due to the complexity and specificity of the platform and the inherent terminology. The first question of each task was whether the user found the task easy to complete and, as can be seen, only task I - Register and Login, V - Setup the Experiment and VI - Check Experiment Results were thought of as easy; this means that no problems are perceived in these particular screens.

Noticeably, task III - Create a new Design Goal and IV - Edit the 'Test Design' Goal had no user classifying them with 4 or above in the scale for ease, being the common element between them the use of formulas. Once again this reinforces the problems that task resolution already hinted at. One must,

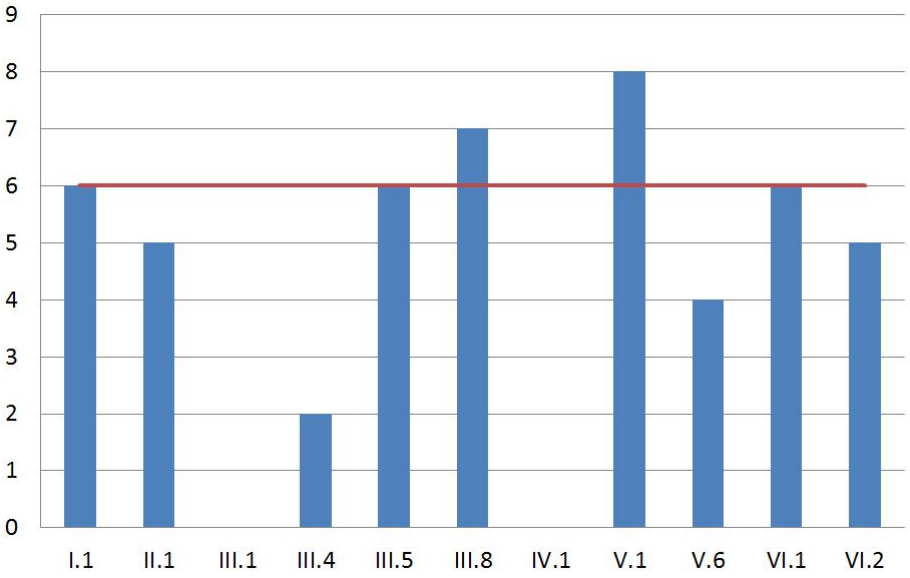


Figure 7.2: The chart shows the results of the questionnaire done during the test, representing the amount of users who gave a score equal or higher than 4 in each question.

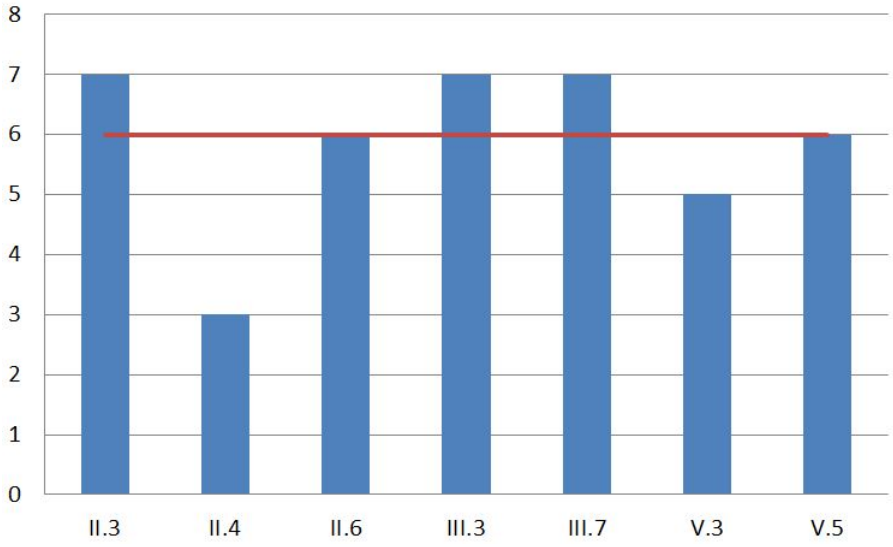


Figure 7.3: The chart shows the results of the questionnaire done during the test, representing the amount of users who answered the questions correctly.

however, take into account the complexity of what was being asked to users: to be able to define and use a formula editor on the level of complexity of an excel formula in a small window of time is no easy feat, even with a brief introduction to its working. Besides overall improvement to the screen, this signals that extra

effort must be placed on the learnability aspect of it. What was found interesting was the fact that, although no user thought about formulas as easy, the majority of them thought of them as a tool capable of creating any Experience Indicator. So, functionality wise, there does not seem to be any problem.

Questions II.3, II.4, II.6, III.7, V.3, and V.5 are questions intended on assessing how well users understood critical terms of the method – Game Variation, Probability Function, Game Variable, Design Goal, Score, Game Permutation and Session respectively. In each, users were required to define, without ambiguity, each of these concepts; if not entirely clear, they would be queried with trick questions to determine whether their understanding was correct or not. This was intended to verify the conceptual adequateness of these new terms, as they had been reported as problematic beforehand. The terms that did not pass the set threshold were Probability Function and Game Permutation. The first is not a critical component needed to use the PCG method; however, clearer presentation of it in the interface can surely be forwarded to users. The second however, needs to be fully understood for proper operation of the method, and thus, a new term should be found that is more readily apprehended by users. Another aspect regarding the terminology is that, when asked if they are confident if they know what a term means, users did not show much confidence (see figure 7.4), even in the terms they correctly answered.

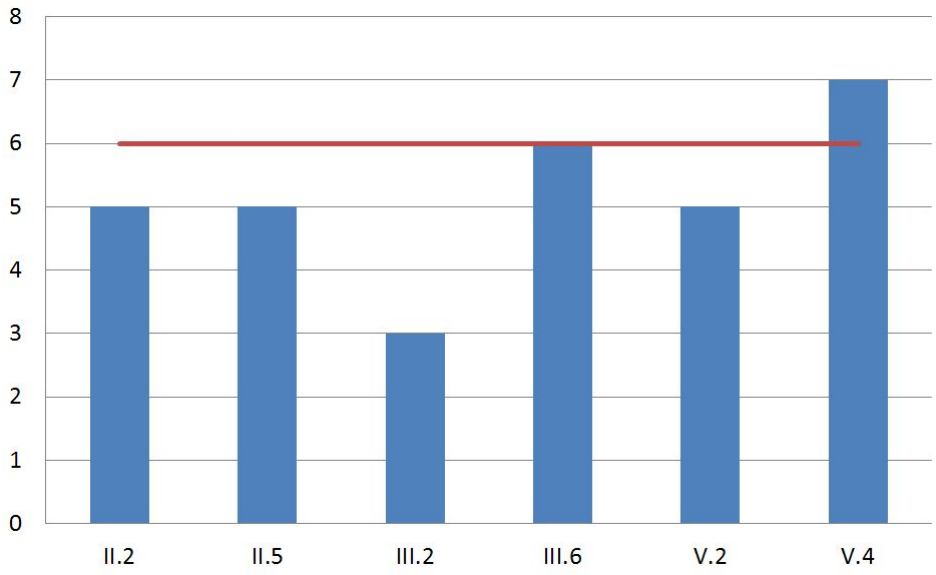


Figure 7.4: The chart shows the amount of users, in a total of 8, who showed confidence in the system's terminology. The line in red shows the success threshold.

Given that the interface is supposed to be an easy to use tool that facilitates the creation process, users should not have to spend long periods of time to finish simple tasks. Before starting the test sessions, we had decided on an acceptable time in which the users were supposed to finish each of the larger tasks. To do this, we followed the created script ourselves, not rushing the completion of each task, and marked the time needed for each task. Taking that time into account,

we set that the time needed to complete each task should not be higher than twice that amount. As before, the success threshold was set at 20% of the users not taking more than the reference time (twice the estimated time) to finish each task. Figure 7.5 shows the number of individuals who took less than the reference time and, while some tasks are right on the threshold, others are still lacking.

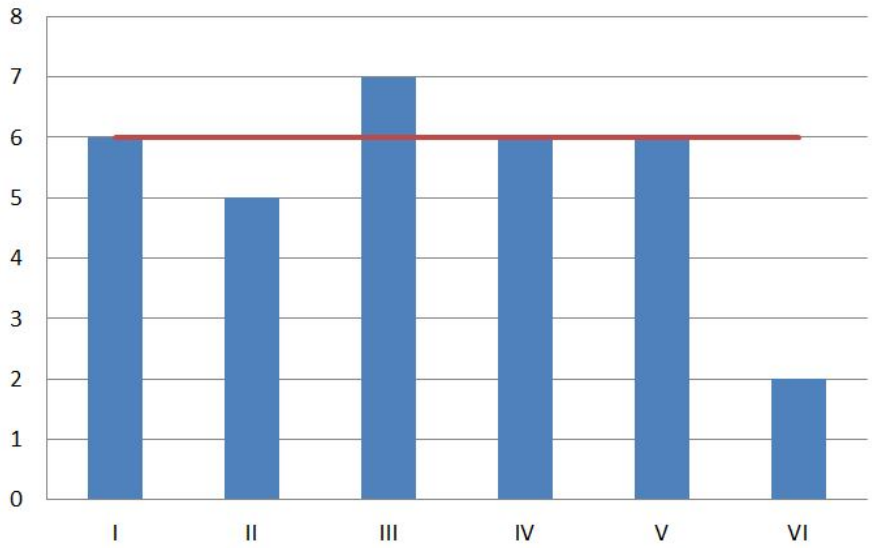


Figure 7.5: The chart shows the amount of users, in a total of 8, who finished each task in less than two times the alloted time for that task. The line in red shows the success threshold.

As can be seen, the two tasks in which the success threshold was not achieved are task II - Create a new Game Variation and IV - Edit the 'Test Design' Goal. In both cases, the main cause for this time difference was that the interface was not as intuitive as we first thought it would be. In task two, users tried to write several times in the disabled text box instead of using the search function located besides it. When searching the parameters, several users made remarks about how the categories should be in alphabetic order and not in the given order (in this case buildings followed by units and game engine parameters). This seems like a critical component to change in all list objects of the UI. The last sub task, activating the game variation, was also a point where most users lost a considerable amount of time, as they were looking for the option in the game variation form and not in the table. The main issue in task IV - Edit the 'Test Design' Goal was choosing the X and Ys data for the charts. While the first chart asked was simple, as the default form values were the ones asked, the second required a bit more effort to accomplish. This signals a problem in the way these are presented to the user, with an excess of options that are grayed out in the dropdown lists; a better solution would be to present only the options available at a given time to users.

To better understand the problems found by the users, table 7.1 was readied listing user reported and observed problems during the usability test, along with

Issue	Count	Imp.	Comp.	Description
Alphabetic order	6	3	2	Features are not in alphabetic order
Activate/Optimize	6	3	2	Activate/optimize not in form
Add Formula	9	5	5	Problems related to adding new formula
Axis Label	2	3	1	Lack of axis information in charts
Back from Formula	2	2	1	Formula back button not intuitive
Button Setup	2	2	2	Test setup States are confused with buttons
Chart Buttons	7	3	2	Layouts and charts selection is confusing
Chart Plus	4	2	1	User tried to click the + sign to add chart
Dynamic Boxes	2	3	2	Some boxes sizes should scale dynamically
Edit Delete Chart	4	4	2	Missing edit and delete chart options
Filter	2	3	4	Filter is not implemented
Formula Flow	2	5	5	The steps needed to manage formulas is not clear
Function Selection	2	2	1	Select the new formula in the list on creation
help	4	2	3	On hover help is not intuitive or easy to use
Incomplete Fields	2	4	1	Lack of feedback on incomplete fields
Login Mail	2	1	1	The use of Login instead of email is confusing
Login Register Data	2	1	1	Register data should be saved for login
Login Register Flow	2	4	2	Login and Register pages are too similar
Mean Label	2	2	1	Mean has no label
Min / Max	17	5	1	Domain values verification is troublesome
Small Formula Buttons	2	3	2	Buttons on formulas are not big enough
State	4	3	3	Test setup State is not visible enough
Y disabled items	7	2	1	First Y select disabled items should not appear
Y disabled selects	7	2	1	No options on second Y select confused users

Table 7.1: List of issues reported repeatedly (count greater than 1) by the users, their importance (imp.) and fixing complexity(comp.). The importance ranges from 1 to 5, where 1 is of little importance and 5 is a serious issue that highly confuses the users. The fixing complexity also ranges from 1 to 5, being 1 easy and fast to fix and 5 needs new planning and/or structuring. This data was created through video-analysis of users' tests.

the number of times they occurred and how important and complex it is to fix them.

This table will be the base to choose what issues will be solved first, having in consideration their occurrences, their importance and how difficult they are to fix. While the fixing complexity rating was given based solely on the work developed in this dissertation, the ratings given to the importance of each issue was discussed with the client because it was a decision too important to be taken alone.

7.1.4 Issues Found

After analyzing the previous data, it was time to take the final conclusions about the interface usage by the users.

The system component with which the users struggled the most is also the one with most issues: the formulas editor in the design goals. This is by far the most pressing problem in terms of usability as users feel that the formulas

creation is too complex and requires too much effort to comprehend. In order to fix the formula problems, a new approach is advised, although no solutions have been thought of yet, it should present the users with a more automatized way of creating formulas.

A problem in which all users stumbled upon was the verification of the minimum, maximum and mean values in functions. The warning that appeared when the individuals were trying to set the functions domain was troublesome and made the users feel stressed. Since this problem was so consistent (17 reported cases) and its fixing was so simple, it is already solved and now works properly.

In general, the found issues can be directly solved. The only reported instance where a different solution is needed is the formula related problems. Due to the complexity of their elements, these problems require a better and more simple way of interaction with the users, which must be studied and analyzed before committing to a solution.

7.1.5 Proposed Corrections

Thanks to the list of detected issues previously compiled, we can now create a list of proposed corrections for those issues. Some of the most common and easy to solve issues have already been solved and will not be present in the list. These issues and the performed corrections are:

- Missing label on results charts - Labels for all charts in the results page have been added
- Charts Filter not doing anything - The result's page section for filtering data was removed until filters are functional
- Lack of feedback on incomplete fields - incomplete fields now have a red border along with the message
- Verification of min and max values - removed the verification on the client side, leaving only on server side. The request is done via Ajax so the verification is transparent to the user

The rest of the issues have their correction propositions on table 7.2. However, there is no proposition for the correction of the formula related problems (the only problems with complexity level 5), as these are issues that should be revised with the client first.

7.2 Game Design Case Study

After the usability tests were over, we could finally start realizing a test with an actual game designer, with the intent of gathering data that would help answer the project's research goal. Although a bulk of tests is needed to draw a conclusion about this case, and the time available for this phase was shorter than expected, we decided to do as much as possible before the dissertation delivery, to help the client with the system evaluation. At this point, a single test case is currently underway, being that the design process is not over and there are no conclusive results to analyze.

Issue	Solution
Features not in alphabetic order	Order feature categories in client side to reduce performance impact in server
Activate/Optimize button not in form	Add the choice to activate/optimize in the creation form
Back button in formula not intuitive	Change the button's icon to be more intuitive
Setup states confused with buttons	Change buttons for a bar or a single box with the current state
Layouts and chart selection confused	Change page layout and new icons to distinguish the 2 functionalities
Plus sign on chart divisions	Change the plus sign for a symbol representing "Drop here"
Static boxes make "dead space" in the interface	Make the possible boxes size change dynamically based on their content
Charts cannot be deleted or edited	Add the functionality to remove the chart or edit its parameters
Newly created functions are not directly selected in list	When adding a new function to the list, make it selected
Help messages are hard to perceive and use	Make new system for help using question marks and hand made text boxes
Login Register Data	Pass the registered email to login screen
Login Register similarity	Change the two pages designs to distinguish them
Mean has no label	Make the mean more intuitive or place a label near the value
Small Formula Buttons	Make the buttons in formula related interfaces bigger
Experiment state not visible enough	Place the experiment's current state and the buttons that change it closer
Y disabled items	Hide the disabled items to avoid confusing the users
Y disabled selects	Hide the second Y select box when there is no option available and remove the NONE option

Table 7.2: List of proposed solutions for most of the found interface issues

However, this section describes the test so far, how the rest of the process is expected to continue and how the results will be analyzed.

7.2.1 Completed Work

In order to gather as much useful data as possible, we had to select a user for the test who was related to game design and knew the process behind conventional game designing. After finding such user, and to simulate as much as possible that the prototype was his making, we presented him with the archetype game, explaining the details behind the game's mechanics and what modifications had been made. This explanation was supplemented with remarks regarding any doubt the user had.

In our first meeting with the user, he came up with a game design, in which player units would gain health when being attacked or when moving and would

lose health while not moving. According to the designer, the player's game style would "resemble that of a parasite", and the games should last about 5 minutes. This design and its verification is something that could not be achieved with the current archetype so, in order to simulate the designer's development effort and to enable the collection of the extra required metrics, we had to make a second development phase on the archetype. This second phase was faster and easier because, in the first phase, we had left dynamic functions to take care of the registrations and implementations, and due to our deeper knowledge of the game's architecture.

After the modifications were ready, we scheduled a second meeting with the designer, video recorded for later analysis, which started with a status update on the finished requirements. When we finished the status update, we asked the designer some questions related with the intended game design (available in Appendix F) so that he could express:

- What design he was trying to achieve
- If during the creative process he had thought of other designs
- Why he chose that design in particular
- If the design was adapted to the method used or if it was originally as he wanted it

Once the questionnaire was over, the designer proceeded to use the platform to input the wanted game variations and desired design goals, as well as to specify the test setup variables, leaving candidates ready to be tested. The designer was then posed with a new set of questions, this time trying to answer some platform usage related questions:

- If the wanted design had been successfully translated to the system
- How the result would be like
- If the platform usage influenced the design intent in any way and how
- If the usage of the platform made him want to try new design intents

Lastly, the designer was asked fill a Creative Support Index (CSI), containing two questions to evaluate the system for each of the following dimensions:

- Collaboration
- Enjoyment
- Exploration
- Expressiveness
- Immersion
- Results worth effort

Tool Evaluation Questionnaire Select a number between 1 and 10	1- Highly Disagree 10 - Highly Agree
1. The system or tool allowed other people to work with me easily.	1-2-3-4-5-6-7-8-9-10
2. It was really easy to share ideas and designs with other people inside this system or tool.	1-2-3-4-5-6-7-8-9-10
3. I would be happy to use this system or tool on a regular basis.	1-2-3-4-5-6-7-8-9-10
4. I enjoyed using the system or tool.	1-2-3-4-5-6-7-8-9-10
5. It was easy for me to explore many different ideas, options, designs, or outcomes, using this system or tool.	1-2-3-4-5-6-7-8-9-10
6. The system or tool was helpful in allowing me to track different ideas, outcomes, or possibilities.	1-2-3-4-5-6-7-8-9-10
7. I was able to be very creative while doing the activity inside this system or tool.	1-2-3-4-5-6-7-8-9-10
8. The system or tool allowed me to be very expressive.	1-2-3-4-5-6-7-8-9-10
9. My attention was fully tuned to the activity, and I forgot about the system or tool that I was using.	1-2-3-4-5-6-7-8-9-10
10. I became so absorbed in the activity that I forgot about the system or tool that I was using.	1-2-3-4-5-6-7-8-9-10
11. I was satisfied with what I got out of the system or tool.	1-2-3-4-5-6-7-8-9-10
12. What I was able to produce was worth the effort I had to exert to produce it.	1-2-3-4-5-6-7-8-9-10

Table 7.3: The answers given by the game designer on the first CSI questionnaire

Notice that the evaluation based on the CSI is still incomplete, as the designer will have to fill a new one every time he uses the platform and the results are dependent on all filled CSIs. Nevertheless, the answers of the designer's CSI are listed in table 7.3, grouped by the above dimensions. Notice that, although the questions were posed in a random order, the designer's answers are similar in the questions of most of the referred dimensions. Although not conclusive, the results so far show great potential and are very positive.

This was the last finished step in the process, leaving the players able to play the archetype game to gather results as to continue the ACPCG's iterative process.

7.2.2 Next Phases

Having the first iteration of the test ready and the candidates generated, the next step will be putting players playing and collect their metrics. Once enough sessions have been played, the designer will have to use the platform again to verify the extracted indicators and analyze their meaning. If he sees fit, he will redefine the design problem, allowing for better results and the players will have to play new candidates. This cycle will continue until the designer is satisfied with the best produced candidate and, in every cycle, he will fill a CSI based on the new interaction with the platform.

When the final candidate is reached, the designer's role will be over and his game finalized, at which point, we will only be missing the analysis of all collected elements from the game design case. They will be compiled and verified

against the three hypotheses placed in chapter 3.

A result on whether designers can define game design problems using the platform will be drawn based on the game designer's expectations, on the final game and on the platform logs containing his actions. To verify whether or not the solution resulting of the design process fulfills the designer's agenda, we will analyze his satisfaction with the system, through the filled questionnaires, interviews and his design process. In order to check if the platform helped the designer finding new design problems and solutions, we will use the recorded platform logs, along with the saved recordings, to compare the initial and final intent and design. Using the platform logs and the recordings, we will also be able to identify the problems encountered while using the system, pointing us to the main struggles and issues during the design process.

Chapter 8

Further Work

Most of the planned activities for this project were carried out and finished but there are still issues pending.

First of all, the pending use case needs to be finished. Without it, no conclusion can be reached on whether this approach can be used for the case at hand or if the developed platform aids designer's in their creative process.

When that is done, there is a list of issues detected from usability tests, some more urgent than others, that need fixing, after which, a new usability test should be realized.

Finally, new tests with game designers are needed, as a single test is not enough to verify the usefulness of the developed tools and to answer the research question posed for this project.

Glossary

Archetype game: a prototype game without the parameters that are to be generated. Its game design is still not finalized.

ACPCG: Author-Centric Approach to Procedural Content Generation. A PCG methodology that is more focused on the designer's intended gameplay experience. See chapter 2.1.2.

Artifact Quality: same as game variation.

Best Candidate: The candidate with the best overall score in the experiment at a given time.

Candidate: The set of values for all game variables that will be used by the archetype game.

Candidate game: a version of the archetype game that uses a set of parameters generated by the genetic algorithm.

Design Goal: An objective to be achieved by designer in the archetype game. Defined by a formula and bound to a domain.

Design Problem: The aggregation of all design problems and game variations of a project. Represents the restrictions and tests applied to the experiment.

EDPCG: Experience Driven Procedural Content Generation. A PCG methodology that gives special focus and tries to optimize the player experience. See chapter 2.1.1.

Experience Indicator: A measurement of a player's experience aspect that can be extracted from gameplay metrics.

Experiment: The set of all definitions done with the intent of achieving a game design.

Feature: same as game variable.

Feature set: same as Candidate.

GA: See Genetic Algorithms.

Gameplay session: Interaction between a candidate game and the player playing it.

Game Variable: A parameter in a game that will be given a value in each session.

Game Variation: The set of parameters by which a game variable can be varied for candidates.

Genetic algorithms: Nature inspired, sub-optimal search algorithm. Employed as a Feature Selection method.

Gameplay Metrics: Raw, unprocessed data collected from a gameplay session.

PCG: Procedural Content Generation. Process of automatically creating content using algorithms. See chapter 2.1

Play-Persona: *"larger-order patterns that can be defined when a player uses one or more play-styles consistently throughout the game play session."* [Tychsen and Canossa, 2008]

Play Session: The time lapse associated with consecutive sessions by the same player.

Project: redefined to experiment.

Session: the time lapse associated to the player playing the game using a candidate.

Bibliography

Anders Drachen and Alessandro Canossa. Towards gameplay analysis via gameplay metrics. In *Proceedings of the 13th International MindTrek Conference: Everyday Life in the Ubiquitous Era*, MindTrek '09, pages 202–209, 2009a.

Raphaël Marczak, Jasper van Vught, Gareth Schott, and Lennart E. Nacke. Feedback-based gameplay metrics: Measuring player experience via automatic visual analysis. In *Proceedings of The 8th Australasian Conference on Interactive Entertainment: Playing the System*, IE '12, pages 6:1–6:10, 2012.

Pejman Mirza-Babaei, Lennart E. Nacke, John Gregory, Nick Collins, and Geraldine Fitzpatrick. How does it play better? exploring user testing and biometric storyboards in games user research. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, pages 1499–1508, 2013.

Anders Drachen and Alessandro Canossa. Analyzing user behavior via gameplay metrics. *Proceedings of the 2009 Conference on Future Play on @ GDC Canada*, pages 19–20, 2009b.

Anders Tychsen and Alessandro Canossa. Defining personas in games using metrics. In *Proceedings of the 2008 Conference on Future Play: Research, Play, Share*, Future Play '08, pages 73–80, 2008.

Georgios N. Yannakakis and Julian Togelius. Experience-driven procedural content generation. *IEEE Transactions on Affective Computing*, pages 147–161, 2011.

Rui Craveirinha, Lucas Santos, and Licínio Roque. An author-centric approach to procedural content generation. *Advances in Computer Entertainment*, pages 14–28, 2013.

Julian Togelius, Emil Kastbjerg, David Schedl, and Georgios N. Yannakakis. What is procedural content generation? mario on the borderline, 2011a.

Julian Togelius, Georgios N. Yannakakis, Kenneth O. Stanley, and Cameron Browne. Search-based procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games Issue 3*, pages 172 – 186, 2011b.

Gillian Smith, Jim Whitehead, and Michael Mateas. Tanagra: A mixed-initiative level design tool. In *Proceedings of the Fifth International Conference on the Foundations of Digital Games*, FDG '10, pages 209–216, 2010.

J. Togelius C. Pedersen and G. N. Yannakakis. Modeling player experience for content creation. *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 14.

John Brooke. Sus-a quick and dirty usability scale. *Usability evaluation in industry*, 189:4–7, 1996.

Rui Craveirinha and Licínio Roque. Studying an author-oriented approach to procedural content generation through participatory design. Pending Publishing.

Appendix A

Remaining Interfaces Prototypes

This Appendix refers to the interfaces elements not included in the main body of the document, namely the rest of the prototypes.

A.1 Prototypes

The new user interface (Figure A.1) is used to create a new user on the platform. The user must choose a username and a password and should also check the projects that needs access to. Access to some projects might be limited or restricted.

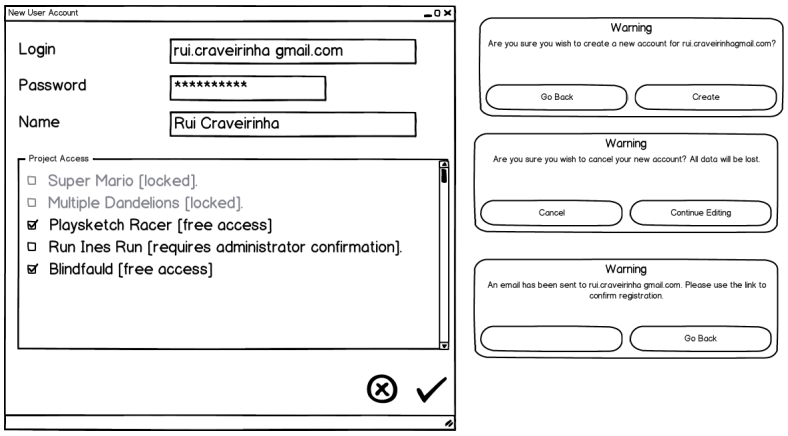


Figure A.1: Prototype for the new user menu interface

With the new project interface (Figure A.2), users can create new projects based on existing prototypes and manage its permissions for other users. By creating various projects that use the same prototype, it becomes possible to test several different game variations.

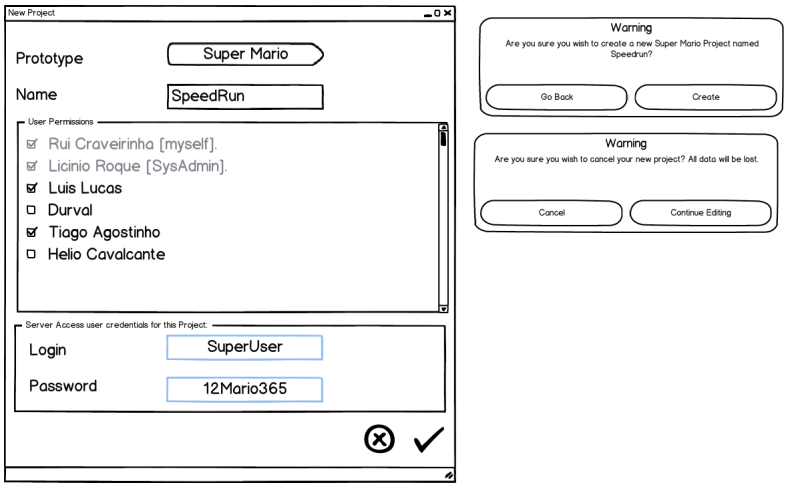


Figure A.2: Prototype for the new project menu interface

Appendix B

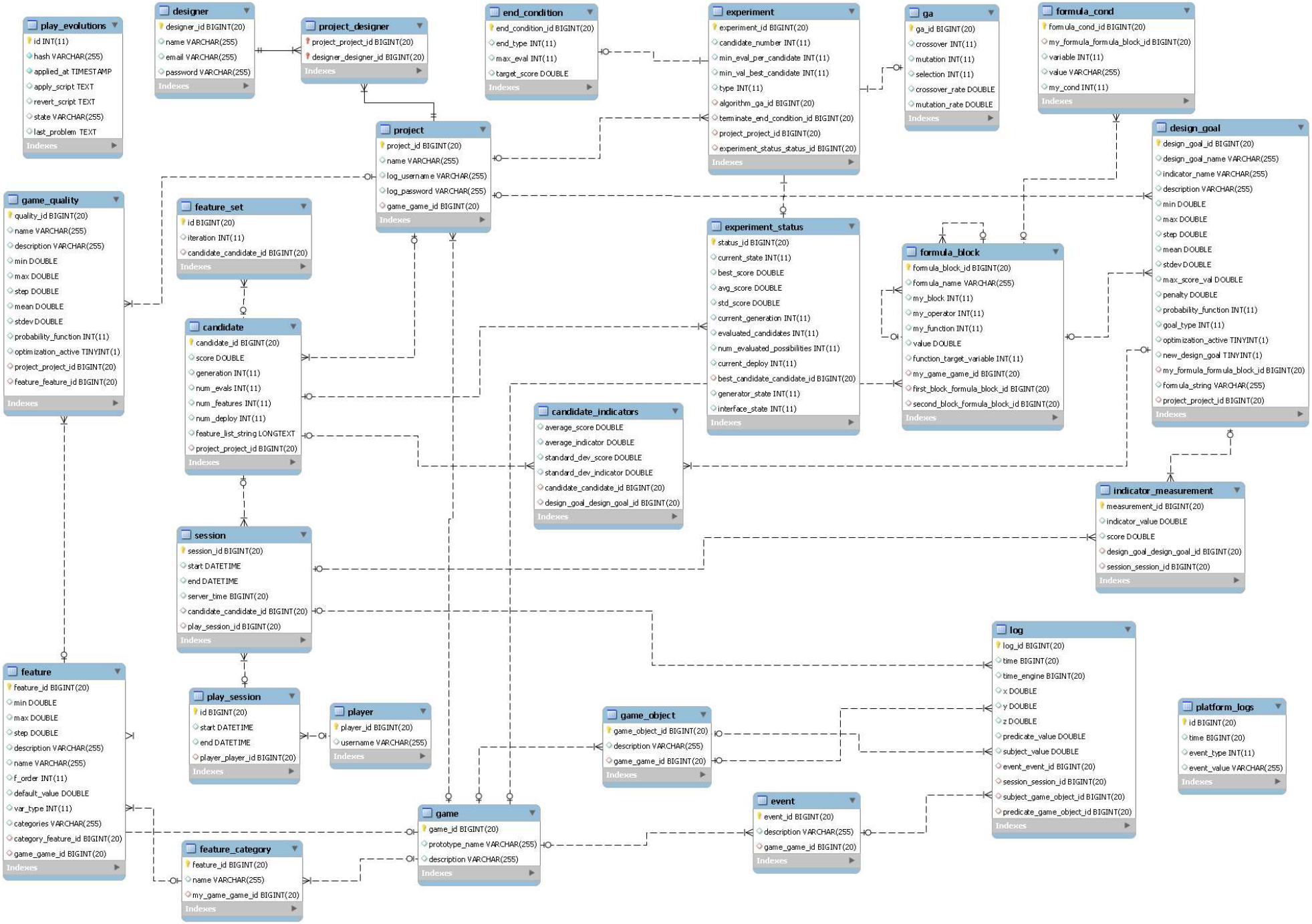
Candidate Attributes table

Jogos	Code Clarity and Readability	Código Modificável
FreeCiv	Easy and clear reading Variables well and intuitively named Built-in functions to manipulate game lists (cities, builtings, ...) The location of certain game modules is unknown Html version uses same server	Most of the elements can be changed by rulesets (Textfiles) units, terrain, technologies, nations, governments Game engine modifications are hard to place Site contains partial code documentation
MicropolisCore	TCL inteface and no experience with it Game engines developed in python e em C++ Several unique file types	Code is not intuitive making it hard to modify Few elements are easy to modify
DuneLegacy	Code well divided by directories Variables have intuitive names Some variables might be hard to find	Code division is easy to change Inheritance is used to create some elements, which might simplify modifications Documentation is available at Dune Legacy's site
BosWars	Code well divided by directories Programming uses LUA (easy to understand) Reamining engines are built in C++	Each unit parameter can be found in that unit's file Source code contains documentation Game engine modifications must be searched in code

Jogos	Estabilidade	Features	Métricas
FreeCiv	Possible crashes during game creation Stable during actual game Unit automation bugs Some components require specific software versions HTML5 version has some stability issues	Units requirements Technologies requirements Visibility City growth Game speed (years per turn) Unit combat promotion probability Diseases factors	Unit count Resources count Year Resources spent by unit Number of combat promoted units
MicropolisCore	No documented stability problems	Game speed Taxes Buildings cost	Sims Happiness Starting funds Taxes rates
DuneLegacy	Stable Engine Some problems reported in long game sessions but none found so far Some problems about not Windows operative systems	harvest speed harvester slowdown amount when full obejct characteristics (life, resistences, damage, ...) Game speed Staport Arrival Time and price update	Units/buildings lost Units/buildings built gathered resources Game time
BosWars	Existing bugs, but none compromises game stability	Unit parameters (Life, damage, size, armor,...) Starting Resources Game speed	Number of units Resource count Units destroyed

Appendix C

Entity-Relationship Diagram


play_evolutions
id INT(11)
hash VARCHAR(255)
applied_at TIMESTAMP
apply_script TEXT
revert_script TEXT
state VARCHAR(255)
last_problem TEXT
Indexes
designer
designer_id BIGINT(20)
name VARCHAR(255)
email VARCHAR(255)
password VARCHAR(255)
Indexes
project_designer
project_project_id BIGINT(20)
designer_designer_id BIGINT(20)
Indexes
end_condition
end_condition_id BIGINT(20)
end_type INT(11)
max_eval INT(11)
target_score DOUBLE
Indexes
experiment
experiment_id BIGINT(20)
candidate_number INT(11)
min_eval_per_candidate INT(11)
min_val_best_candidate INT(11)
type INT(11)
algorithm_ga_id BIGINT(20)
terminate_end_condition_id BIGINT(20)
project_project_id BIGINT(20)
experiment_status_status_id BIGINT(20)
Indexes
ga
ga_id BIGINT(20)
crossover INT(11)
mutation INT(11)
selection INT(11)
crossover_rate DOUBLE
mutation_rate DOUBLE
Indexes
formula_cond
formula_cond_id BIGINT(20)
my_formula_formula_block_id BIGINT(20)
variable INT(11)
value VARCHAR(255)
my_cond INT(11)
Indexes
project
project_id BIGINT(20)
name VARCHAR(255)
log_username VARCHAR(255)
log_password VARCHAR(255)
game_game_id BIGINT(20)
Indexes
design_goal
design_goal_id BIGINT(20)
design_goal_name VARCHAR(255)
indicator_name VARCHAR(255)
description VARCHAR(255)
min DOUBLE
max DOUBLE
step DOUBLE
mean DOUBLE
stdev DOUBLE
max_score_val DOUBLE
penalty DOUBLE
probability_function INT(11)
goal_type INT(11)
optimization_active TINYINT(1)
new_design_goal TINYINT(1)
my_formula_formula_block_id BIGINT(20)
formula_string VARCHAR(255)
project_project_id BIGINT(20)
Indexes
game_quality
quality_id BIGINT(20)
name VARCHAR(255)
description VARCHAR(255)
min DOUBLE
max DOUBLE
step DOUBLE
mean DOUBLE
stdev DOUBLE
probability_function INT(11)
optimization_active TINYINT(1)
project_project_id BIGINT(20)
feature_feature_id BIGINT(20)
Indexes
feature_set
id BIGINT(20)
iteration INT(11)
candidate_candidate_id BIGINT(20)
Indexes
experiment_status
status_id BIGINT(20)
current_state INT(11)
best_score DOUBLE
avg_score DOUBLE
std_score DOUBLE
current_generation INT(11)
evaluated_candidates INT(11)
num_evaluated_possibilities INT(11)
current_deploy INT(11)
best_candidate_candidate_id BIGINT(20)
generator_state INT(11)
interface_state INT(11)
Indexes
formula_block
formula_block_id BIGINT(20)
formula_name VARCHAR(255)
my_block INT(11)
my_operator INT(11)
my_function INT(11)
value DOUBLE
function_target_variable INT(11)
my_game_game_id BIGINT(20)
first_block_formula_block_id BIGINT(20)
second_block_formula_block_id BIGINT(20)
Indexes
candidate
candidate_id BIGINT(20)
score DOUBLE
generation INT(11)
num_evals INT(11)
num_features INT(11)
num_deploy INT(11)
feature_list_string LONGTEXT
project_project_id BIGINT(20)
Indexes
candidate_indicators
average_score DOUBLE
average_indicator DOUBLE
standard_dev_score DOUBLE
standard_dev_indicator DOUBLE
candidate_candidate_id BIGINT(20)
design_goal_design_goal_id BIGINT(20)
Indexes
indicator_measurement
measurement_id BIGINT(20)
indicator_value DOUBLE
score DOUBLE
design_goal_design_goal_id BIGINT(20)
session_session_id BIGINT(20)
Indexes
session
session_id BIGINT(20)
start DATETIME
end DATETIME
server_time BIGINT(20)
candidate_candidate_id BIGINT(20)
play_session_id BIGINT(20)
Indexes
log
log_id BIGINT(20)
time BIGINT(20)
time_engine BIGINT(20)
x DOUBLE
y DOUBLE
z DOUBLE
predicate_value DOUBLE
subject_value DOUBLE
event_event_id BIGINT(20)
session_session_id BIGINT(20)
subject_game_object_id BIGINT(20)
predicate_game_object_id BIGINT(20)
Indexes
feature
feature_id BIGINT(20)
min DOUBLE
max DOUBLE
step DOUBLE
description VARCHAR(255)
name VARCHAR(255)
f_order INT(11)
default_value DOUBLE
var_type INT(11)
categories VARCHAR(255)
category_feature_id BIGINT(20)
game_game_id BIGINT(20)
Indexes
play_session
id BIGINT(20)
start DATETIME
end DATETIME
player_player_id BIGINT(20)
Indexes
player
player_id BIGINT(20)
username VARCHAR(255)
Indexes
game_object
game_object_id BIGINT(20)
description VARCHAR(255)
game_game_id BIGINT(20)
Indexes
platform_logs
id BIGINT(20)
time BIGINT(20)
event_type INT(11)
event_value VARCHAR(255)
Indexes
game
game_id BIGINT(20)
prototype_name VARCHAR(255)
description VARCHAR(255)
Indexes
event
event_id BIGINT(20)
description VARCHAR(255)
game_game_id BIGINT(20)
Indexes
feature_category
feature_id BIGINT(20)
name VARCHAR(255)
my_game_game_id BIGINT(20)
Indexes

Appendix D

List of Archetype Game's Features and Metrics

The following are the lists of both the existing units and the parameters that can vary for each unit.

List of buildings:
- Barracks
- Construction Yard
- Gun Turret
- Heavy Factory
- High Tech Factory
- IX
- Light Factory
- Palace
- Radar
- Refinery
- Repair Yard
- Rocket Turret
- Silo
- Slab Size 1
- Slab Size 4
- StarPort
- Wall
- Wind Trap
- WOR

List of Units:
- CarryAll
- Devastator
- Deviator
- Frigate
- Harvester
- Soldier
- Launcher
- Mobile Construction Vehicle
- Ornithopter
- Quad
- Saboteur
- Sandworm
- SiegeTank
- SonicTank
- Tank
- Trike
- RaiderTrike
- Trooper
- Special
- Infantry
- Troopers

The list of parameters is as follows:

- Hitpoints
- Price
- Power
- View Range
- Capacity
- Weapon Damage
- Weapon Range
- Weapon Reload Time
- Max Speed
- Turn Speed
- Build Time
- Probability to Spawn Infantry at Death

Game engine parameters:

- Mission - The mission to be played. There are 9 different missions for each of the 6 existing houses, allowing to choose between a total of 54 different missions, each with its own layout and victory conditions.
- Game Speed - The speed at which the game flows. This can vary from 8 to 32, being 16 the normal game speed, 8 very fast and 32 very slow.
- Harvest Speed - Determines how fast a Harvester can take spice from a tile.

- Max Spice Carried - The maximum amount of spice a harvester can carry before it has to return and deposit it.
- StarPort Arrival Time - The amount of time the requested units take to be delivered.
- Minimum Spice on Tile - The minimum amount of spice a normal spice tile will have when starting a map.
- Maximum Spice on Tile - The maximum amount of spice a normal spice tile will have when starting a map.
- Minimum Spice on Thick Spice Tile - The minimum amount of spice a thick spice tile will have when starting a map.
- Maximum Spice on Thick Spice Tile - The maximum amount of spice a thick spice tile will have when starting a map.
- Bad Damage Ratio - The percentage of health from which the unit is considered to be badly damaged and, consequently, starts to be applied the damaged speed multiplier.
- Damaged Speed Multiplier - The value by which the badly damaged unit's speed will be multiplied.

List of collected Metrics:

- Select unit - This metric is collected when the player selects any number of units except those that are not his. The select unit info is kept in the subject fields. If the player selects multiple units at the same time, a metric of this type will be created for each of them.
- Select enemy unit - Collected when the player selects an enemy unit, this metric keeps record of the selected unit in the subject fields.
- Move unit - When the player moves one or more units, a metric of this type is created for each of those units. The unit information is kept in the subject part of the metric and the x and y fields keep the new position coordinates.
- Move unit to ally - similar to the move unit metric but is collected when the player orders a unit to move to another friendly unit. Also, that friendly unit's information is stored in the predicate fields of the metric and its position is kept as the metrics x and y.
- Attack - Same as the move unit to ally but the friendly unit information in the predicate is changed with the attacked unit's information. The saved coordinates are the position of the attacked unit.
- Start production - Whenever the player adds an item to a production queue, the game records a metric with the producing building as in the subject fields and the produced item as the predicate. There is no predicate value because the produced item still has no in game id.

- End production - When an item production is finished, the game saves a metric containing the producing building's information in the subject fields and the finished item in the predicate.
- Pause production - Collected when the player pauses an item in a building's production queue. The subject fields contain the building's information and the predicate is referent to the paused item.
- Resume production - If a player resumes an item production in a building's queue, the game will save a metric containing the information about that building in the subject fields and the resumed item in the predicate.
- Place building - While its not needed when building units, after a structure is built, the player must choose where to place it. When the player does, a metric will be saved containing the position where it was placed, along with its information (including its newly acquired object id) in the subject fields.
- Cancel production - If the player cancels an item in a building's queue, the respective metric will contain the building in which the production was canceled (in the subject fields) and the item as the predicate.
- Missing resources for production - After placing an item in a building production queue, its production can be halted due to the lack of resources. In that case, this metric will be saved using the producing building's information in the subject fields and the halted item in the predicate.
- Upgrade building - When the player upgrades a building this metric is saved using the upgraded building's information in the subject fields. The predicate and its value are filled with an UpgradeId and the level to which it is being upgraded respectively.
- Finished upgrading - When a building's upgrade is finished, this metric records the information about the building in the subject fields, the UpgradeId in the predicate and the reached level in the predicate value.
- Missing resources for upgrade - Just like there is a metric for the halted production due to the lack of resources, there is one for the halt of a building upgrade. This takes the building information as the subject fields, uses the UpgradeId to identify the predicate and uses the level to which the building was being upgraded to as the predicate value.
- Destruction - Taken when the a player unit or building is destroyed. The subject fields contain the destroyed unit information while the predicate fields contain the information about the unit that destroyed it.
- Enemy destruction - same as above but for enemy units instead of the player's.
- Start Repairing - Metric taken when a building is repaired. It takes the information about the building that is being repaired, uses it for the subject fields and then places a RepairId and the building's health percentage in the predicate and predicate value respectively.

- Finished repairing - Same as above but, since it is collected when the building is fully repaired, the predicate fields are not needed.
- Spice collect - When a harvester starts collecting spice, the game takes this metric, containing the harvester's information as the subject and the quantity of spice as the predicate.
- Spice deposit - When a harvester finishes depositing the collected spice, a metric containing just the harvester's information in the subject fields is saved.
- Resources & Health Verification - There are two verifications that are being made in the game every second. The first one is the player resources, translated directly in the amount of credits the player has and saved as a metric with that amount serving as value for a Spice subject. The second verification is the health value of each of the player's units and buildings. This means that, for every unit or building the player has, a metric will be created with that unit or building's information as the subject and their current health as value for a Health predicate.

Appendix E

Usability Tests

Tool Evaluation Questionnaire	Select a number between 1 and 5 1- Highly Disagree 5 - Highly Agree	Observations
Step I. Time Taken: _____ (Reference time = 1 min)		
I.1. I found this task easy to accomplish.	1-2-3-4-5	
Step II. Time Taken: _____ (Reference time = 1 min 30s)		
II.1. I found this task easy to accomplish.	1-2-3-4-5	
II.2. I understood what the term Game Variation refers to.	1-2-3-4-5	
II.3. What is a Game Variation?		
II.4. What changes when we select a triangular function instead of a step function?		
II.5. I understood what the term Game Variable refers to.	1-2-3-4-5	
II.6. What is a Game Variable?		
Step III. Time Taken: _____ (Reference time = 4 mins)		
III.1. I found this task easy to accomplish.	1-2-3-4-5	
III.2 I understood what the term Design Goal refers to.	1-2-3-4-5	
III.3. What is a Design Goal?		
III.4. I understood how Formulas work.	1-2-3-4-5	
III.5. I could use the Formula Editor to create every Experience Indicator I would like to extract from a game of mine.	1-2-3-4-5	
III.6. I understood what the term Score refers to.	1-2-3-4-5	
III.7. What does Score term refer to?		
III.8. I find the score attribution useful.	1-2-3-4-5	

Step IV. Time Taken: _____ (Reference time = 3 mins)		
IV.1. I found this task easy to accomplish.	1-2-3-4-5	
Step V. Time Taken: _____ (Reference time = 1 min 30s)		
V.1. I found this task easy to accomplish.	1-2-3-4-5	
V.2. I understood what the term Game Permutation refers to.	1-2-3-4-5	
V.3. What is a Game Permutation?		
V.4. I understood what the term Session refers to.	1-2-3-4-5	
V.5. What is a Session?		
V.6. I understood the differences in the types of test that are available.	1-2-3-4-5	
Step VI. Time Taken: _____ (Reference time = 2 mins)		
VI.1. I found this task easy to accomplish.	1-2-3-4-5	
VI.2 I found the available chart and table tools to be enough to fully explore the results from experimenting with a game.	1-2-3-4-5	

System Usability Scale

	Strongly disagree				Strongly agree
1. I think that I would like to use this system frequently	1	2	3	4	5
2. I found the system unnecessarily complex	1	2	3	4	5
3. I thought the system was easy to use	1	2	3	4	5
4. I think that I would need the support of a technical person to be able to use this system	1	2	3	4	5
5. I found the various functions in this system were well integrated	1	2	3	4	5
6. I thought there was too much inconsistency in this system	1	2	3	4	5
7. I would imagine that most people would learn to use this system very quickly	1	2	3	4	5
8. I found the system very cumbersome to use	1	2	3	4	5
9. I felt very confident using the system	1	2	3	4	5
10. I needed to learn a lot of things before I could get going with this system	1	2	3	4	5

Appendix F

Game Design Test

This appendix refers to the questionnaires posed to the game designer at the beginning and at the end of his second designing session.

Beginning:

- Describe the design you intend to develop for Dune with this application's support.

- What experience qualities do you intend to reach with it? How do you wish that the player plays the game?

- Do you have any idea of how to translate that design to a set of Game Variations and Design Goals? If so, enumerate them and tell us what values you are thinking about giving them.

- How did you get those values?

- Do you wish to do any kind of free experimentation, without setting design goals? Why?

- Do you consider that the design problem that you just translated is exactly the design problem that you wish to see solved?

(Had to change it completely to fit) 1-2-3-4-5-6-7-8-9 (Remained intact)

- If there were any changes, explain why and with what purpose

- When thinking about a design based in the offered ontology, did you get any more design ideas? If so, which, and why did you choose this one?

End:

- Do you consider that you managed to translate your design through the application?

(I had lots of difficulties) 1-2-3-4-5-6-7-8-9 (I did it perfectly)

- In this moment, can you imagine how the final prototype will be?

- What values do you think it will have for the variables you selected?

- While using the application, did you change your initial plan for the game? If so, how?

- While using the application, did you get any ideas for alternative design problems that you would like to solve?

- How interesting do you think the final game will be?

(uninteresting) 1-2-3-4-5-6-7-8-9 (very interesting)

- Why? In your opinion, what aspects will valorize it?

- Do you consider that the experience you seek to reach is totally in line with the set of Design Goals that you set? Why?

(Totally out of line) 1-2-3-4-5-6-7-8-9 (perfectly in line)

- If you had to create this game without using this application, how would you do it?
- Having the choice between using this application and doing the development process without it, what would you choose and why?