PONTIFICIA UNIVERSIDAD CATOLICA DE CHILE
ESCUELA DE INGENIERIA

THE VALUE OF GUIDANCE WHEN TEACHING SCIENCE USING AN EDUCATIONAL SIMULATOR

PEDRO A. CORI GRONEMEYER

Thesis submitted to the Office of Research and Graduate Studies in partial fulfillment of the requirements for the Degree of Master of Science in Engineering

Advisor:
MIGUEL NUSSBAUM

Santiago de Chile, September 2013

PONTIFICIA UNIVERSIDAD CATOLICA DE CHILE
ESCUELA DE INGENIERIA

THE VALUE OF GUIDANCE WHEN TEACHING SCIENCE USING AN EDUCATIONAL SIMULATOR

PEDRO A. CORI GRONEMEYER

Members of the Committee:

MIGUEL NUSSBAUM

MIGUEL RÍOS

JUAN PABLO GARCÍA-HUIDOBRO

YADRAN ETEROVIC

Thesis submitted to the Office of Research and Graduate Studies in partial fulfillment of the requirements for the Degree of Master of Science in Engineering

Santiago de Chile, September, 2013

To Trix, without whom this project would have taken half the time and been half as fun.

ACKNOWLEDGEMENTS

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

RESUMEN

El potencial pedagógico de los simuladores computacionales ha sido investigado extensivamente a medida que la tecnología ha avanzado y se ha hecho cada vez más accesible. Los simuladores pueden incluir guiones o métodos de guía para incorporar las mecánicas de juego progresivamente. El objetivo de este trabajo es estudiar cómo deben construirse estos componentes guía. Con este fin, definimos un modelo de simulador que agrupa elementos que han probado que mejoran el aprendizaje, incluyendo el componente guía. Experimentalmente comparamos dos simuladores educacionales diseñados para enseñar Respiración Celular (una unidad de biología de la célula) a alumnos de enseñanza media. Ambos simuladores comparten la lógica del software subyacente y sus mecánicas, pero difieren en sus sistemas de guía (i.e. en el guión). En un estudio con 130 alumnos de 2º medio, se obtuvieron resultados de aprendizaje estadísticamente significativos para ambos simuladores. Sin embargo, la diferencia entre la mejoría en el resultado promedio entre los que usaban una u otra versión del programa no fue significativa. Por lo tanto concluimos que el conjunto mínimo de características de un guión es suficiente para lograr un aprendizaje.

Esta investigación recibió apoyo del Centro de Estudios de Políticas y Prácticas en Educación (CEPPE) CIE01-CONICYT.

Palabras clave: guías, guión, simuladores educacionales, progresión de modelo

ABSTRACT

The teaching potential of computer simulators has been thoroughly investigated as technology has advanced and become more readily available. Simulators can include guidance and scripting to progressively incorporate the game's mechanics. The aim of this work is to study how this guiding component should be built. For this purpose we define a simulator model which collects elements that have proven to enhance learning including the guiding component. Experimentally we compare two educational simulators designed to teach Cellular Respiration (a unit in cellular biology) to high school students; both simulators share the underlying software and mechanics but differ in their guidance systems (i.e. script). In a study with 130 10^{th} grade students, statistically significant results were obtained for both simulators; however, the difference between the mean score improvement using either version of the software was not significant. We therefore conclude that a minimal set of script characteristics is sufficient to achieve learning.

Research supported by the Center for Research on Educational Policy and Practice (CEPPE), Grant CIE01- CONICYT

Keywords: guidance, script, educational simulators, educational simulation, model progression

1. INTRODUCTION

1.1. Motivation

Simulators have existed almost as long as there have been computers, and have been attempted to be used for teaching (systematically and keeping a record of it) for at least four decades (Smetana & Bell, 2011). However, when a branch of software simulation specifically designed for educational purposes (instead of scientific experimentation, etc.) divided itself from the main stem, the software's objective changed drastically. Going from representing a model of reality as accurately as possible to helping students learn concepts about a subject. Given the more ambiguous nature of this new objective, the best way of achieving it was much less clear, and gave way to a sort of alchemical process in the design and development of didactic simulators very similar to the one still present in game design (where the objective of entertaining users is also very open) (Cook, 2012). Every developer designed their simulator under the criteria they thought best to solve the problem, without much regard for what had been done before or if it had proven effective. This naturally produced a wide range of simulators that radically differed in their results and contradicted each other on their effectiveness as teaching tools, as can be seen in reviews of older research on pedagogical simulation and the multitude of approaches to their study (de Jong & van Joolingen, 1998; Rutten, van Joolingen, & van der Veen, 2012; Smetana & Bell, 2011).

Since the beginning of the '90s, some efforts have been made to take a more scientific approach towards educational simulator design, determining the importance of certain common elements (Amory et al., 1999; Kuk et al., 2012), the effectiveness of distinct mechanics or pedagogical techniques (Charney et al., 1990; de Jong & van Joolingen, 1998; de Jong et al., 1999), and even proposing some general guidelines or frameworks for their development (Amory et al., 1999; de Jong et al., 1994; Paras, 2005). Nonetheless, the migration towards a systematic development of optimal educational experiences with simulators is far from done. A specific and proved model for their construction has yet to be proposed and validated, and the relevance of the common elements, mechanics, and techniques mentioned above haven't been compared extensively to each other to determine their influence in a potential model.

This study attempts to further the shift from alchemy in the design of educational simulators towards a scientific approach that could guarantee optimal learning experiences through the use of this software.

A fundamental part of this is believed to be the comprehension of the design of effective guidance systems for this type of tool, given the importance it has had in learning outcomes of multiple studies (Kirschner, Sweller, & Clark, 2006; Moreno & Mayer, 2005; Reid, Zhang, & Chen, 2003; Schrader & Bastiaens, 2012). Although these experiments have proven that certain types of guidance systems yield better results in different types of student learning when compared to non-guided experiences, little research has been done to assess their comparative advantages and the appropriate complexity they should have for optimal scripting of the simulators. In this sense, despite having the knowledge that guidance systems are necessary for the development of effective educational simulators, the details of the implementation of these is still somewhat of a gamble left to the designer of each individual software. Therefore it is highly relevant to strive for a model that comprehensively prioritizes these techniques and in which measure they should be included into pedagogical simulations. This research also delves into this matter by comparing guidance system complexities to begin the confection of this model.

In summary, a schema of proven and effective elements for educational simulators needs to be created and refined in order for them to be ever more consistently useful and efficient as teaching or reinforcement tools. This work will try to bring this objective a step closer to the scientifically and systematically optimal and away from the alchemically uncertain.

1.2. Detailed design

The *SimBIOS* educational software was designed to simulate the process of aerobic cellular respiration in animal cells; an integral part of the biology subject in the Chilean high school curriculum.

Given the motivation for making this simulator, the objective was not only that the different chemical reactions involved in cellular respiration were memorized or understood in a theoretical sense, but that the whole process could be experienced holistically. All these reactions and sub-processes only really make sense when viewed as cogs that work in unison in a machine that has a real purpose. Therefore, in the design of this application, a model was created that tried to achieve this pedagogical objective, the holistic experience and understanding of the cellular respiration process, while attaining to a set of standards proposed as the effective way of making guided educational simulators proposed in the paper.

1.2.1. The model

The model devised for the simulation will be described from its most basic components, building towards their interaction in respiration sub-processes, to finish with the whole cycle where the process can be viewed in its totality.

1.2.1.1. Molecules

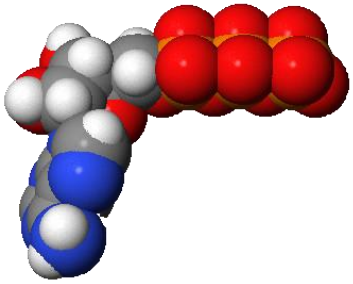
All chemical reactions and sub-processes that comprise cellular respiration involve the interaction between different molecules present in the cytoplasm of the cell or its mitochondria. In this model, the most critical of these molecules are represented by their 3D space-filling molecular structure model (a 2D image of them), and are the only elements in the simulation that can be directly manipulated by the user. He can make them move and collide with each other or with mitochondrial structures by clicking and dragging them around the 2-dimensional representation of the cytoplasm (Fig 1-1).

Fig 1-1 An ATP molecule is dragged by the user towards a Glucose molecule in the cytoplasm

The images for the molecule models were mainly obtained from models rendered for the Wikimedia Commons database, although some had to be modeled independently in 3D Studio Max, and rendered specifically for *SimBIOS*. Scale was relatively kept between molecules, but not exactly (as images came from different sources), but was not kept with respect to other cellular structures within the simulation, as the differences in size would have made direct interaction with the molecules impossible if the other structures were to fit in the screen.

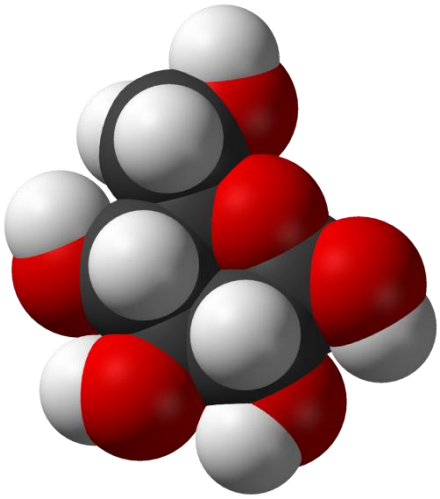
a. Types of molecules

These are the different types of molecules involved in the cellular processes implemented in *SimBIOS* with their respective visual representations used in the program.



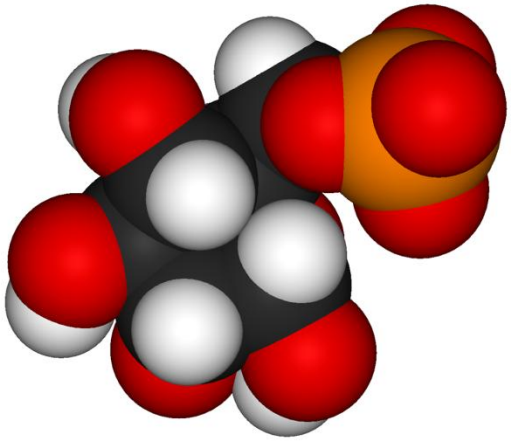
ATP:

It is the main energetic molecule for the cell's processes. It acts as the energy source for thousands of reactions; its production is the main function of the cellular respiration process.



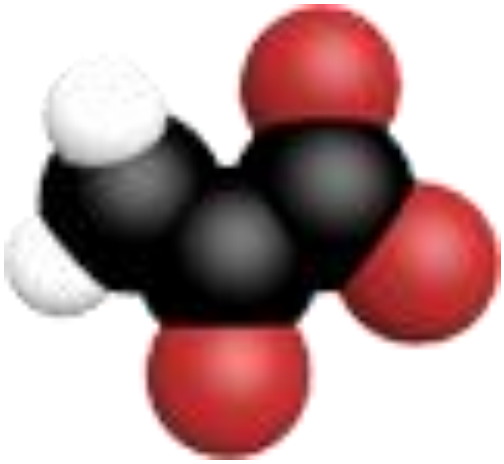
Glucose:

This molecule is the "fuel" for the production of energy in cellular respiration. It is progressively broken down through the different sub-processes which will eventually produce a gradient of protons that enable ATP production. It is the final product of the decomposition of complex carbohydrates, proteins, or fat (i.e. food) before it is used in respiration.



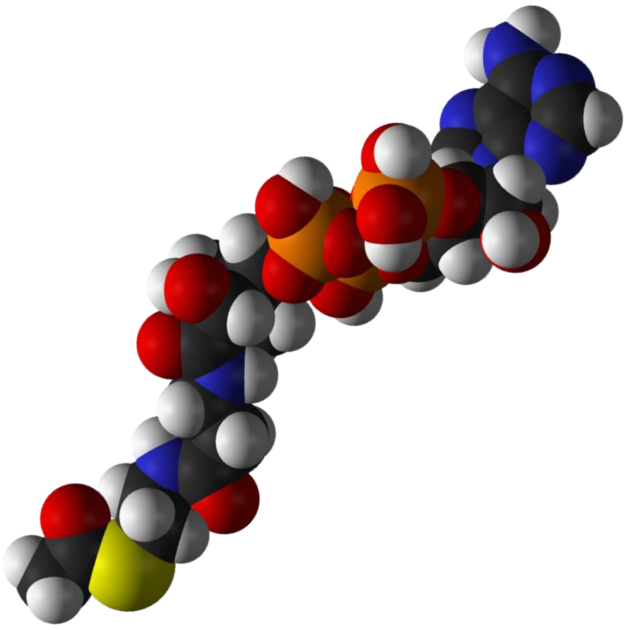
Glucose-6-phosphate:

Glucose-6-phosphate is an intermediate molecule formed during the process of Glycolisis (Section 1.2.2). In the simulation, only an animation is shown of an incomplete Glycolisis to illustrate the necessity of a second ATP molecule to be added to the reaction and complete it.



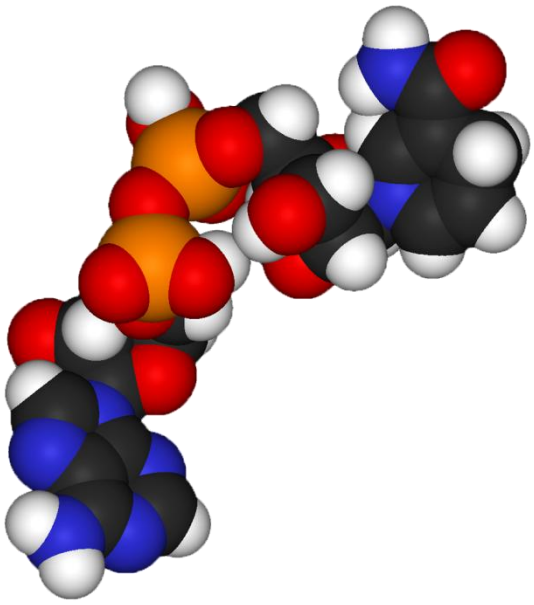
Pyruvate (Pyruvic acid):

Pyruvic acid, along with NADH, and ATP, is a byproduct of the decomposition of Glucose through the sub-process of Glycolisis in the cytoplasm of the cell (Section 1.2.2). Two of these molecules are produced from each Glucose molecule that undergoes Glycolisis.



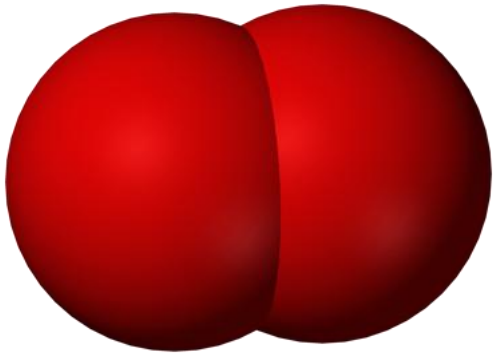
Acetyl-CoA:

When Pyruvate molecules enter the mitochondria, they undergo oxidative decarboxylation by enzymes to be further transformed into one Acetyl-CoA molecule and one NADH molecule each. This is the Glucose-derived molecule which enters the Citric acid cycle (Krebs cycle).



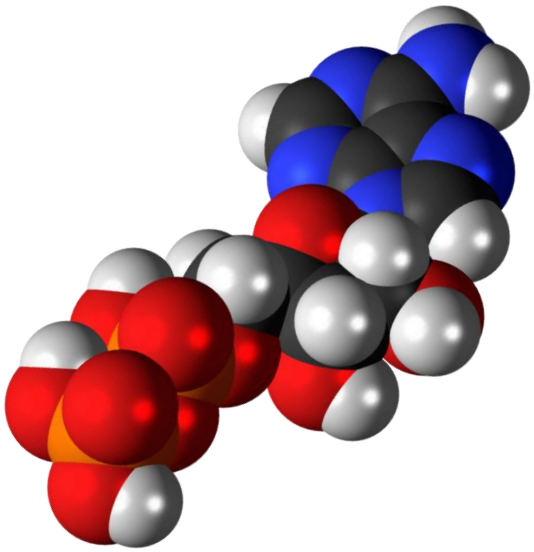
NADH:

When Acetyl-CoA enters the Krebs cycle, as well as in other sub-processes like Glycolisis or the Oxidative decarboxylation of Pyruvate, NADH is produced. This molecule is relatively unstable, and tends to lose two electrons to become NAD^+ and H^+. That is why this is such an important molecule in the cellular respiration process; it is the main electron yielder molecule for Oxidative phosphorylation (Section 1.2.2). Another, less influential electron yielder ($FADH_2$), was left out of the simulation because the complexity it added to the model was deemed disproportional to its less than 10% contribution to ATP generation in respiration.



Oxygen:

Its main function in cellular respiration comes in the Oxidative phosphorylation sub-process (Section 1.2.2), where it is used to extract the electron pairs yielded by NADH molecules to the Electron transport chains and keep electron flow going to maintain proton concentration gradient. It is the electron receiver in this process.



ADP:

This is the molecule which, through the addition of a third phosphate group by the ATP Synthase enzyme during the Oxidative phosphorylation phase of respiration (Section 1.2.2), becomes ATP. The whole process of cellular respiration revolves around transforming ADP into ATP.

1.2.1.2. Structures

All other elements, with which the user can interact indirectly by moving molecules, are considered structures. Specifically, these are the mitochondrion and its most important membrane proteins.

a. The Mitochondrion

Mitochondria are the cellular organelle whose primary role is aerobic respiration (i.e. the production of energy in the form of ATP molecules needing oxygen), and most of the sub-processes that compose it occur inside them.

In the *SimBIOS* simulator, only one mitochondrion can be interacted with, and is shown in the foreground, covering a large portion of the visible area for the user. Other mitochondria can be seen floating in the background, but only one was kept for user interaction for simplicity and focus. Since the simulation is in two dimensions, an abstraction needed to be made for the representation of the mitochondrion. As with most academic texts and diagrams, this comes from a simplification of the electron microscope images taken of mitochondria (Fig 1-2), and a longitudinal cross section of the organelle was chosen for representation. However, it was still modeled and rendered in 3D to clarify that in reality it has depth (Fig 1-3). This cross-section was chosen because it clearly shows both membranes of the mitochondrion, and it allows the most space in the mitochondrial matrix for the user to carry out the processes inside.

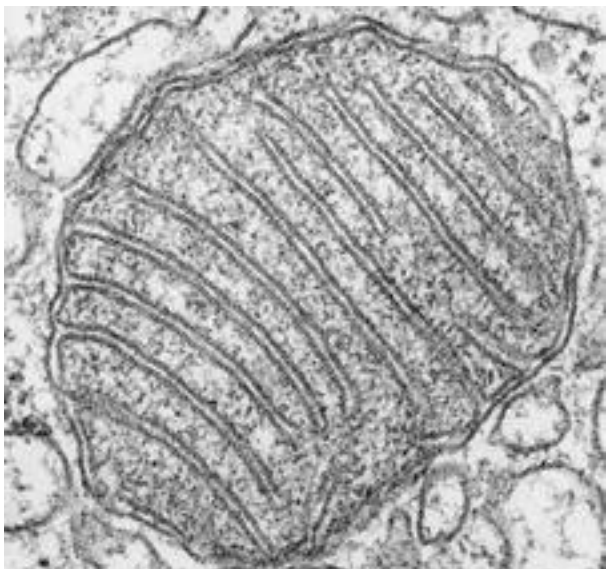


Fig 1-2 Electron microscope image of a mitochondrion

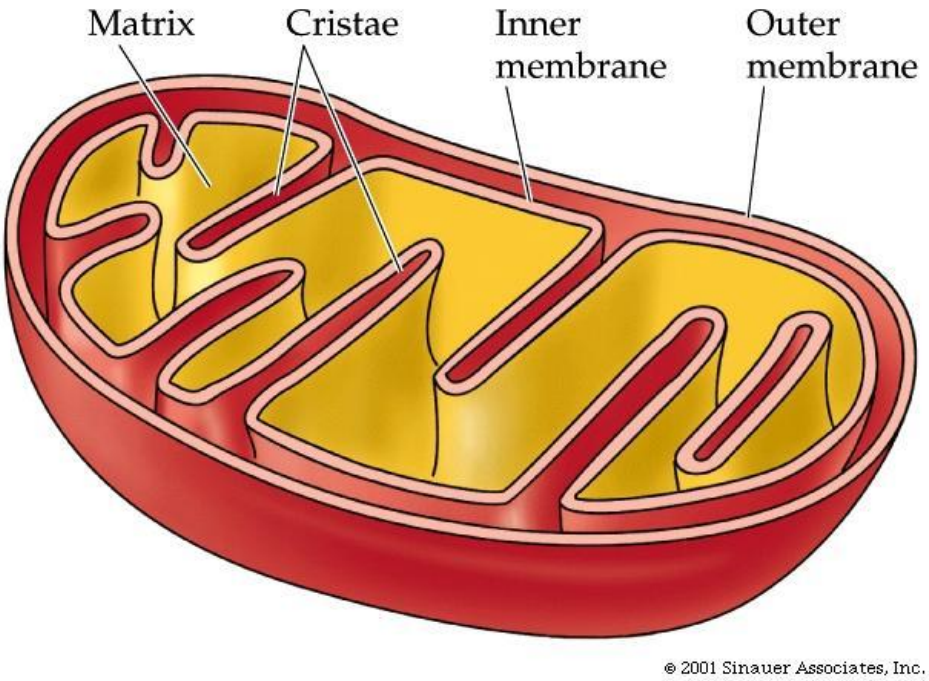


Fig 1-2 Typical textbook cross section representation of a mitochondrion.

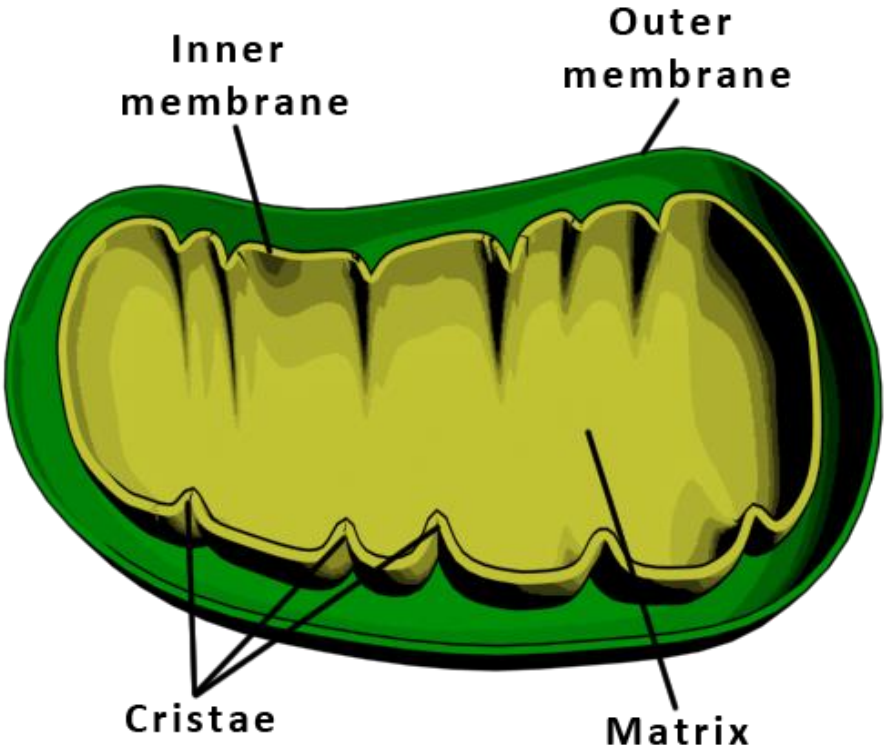


Fig 1-3 Representation of a mitochondrion used in the *SimBIOS* simulator

1.2.1.3. Electron transport chain (cytochrome chain)

One of the protein structures present in the mitochondrial inner membrane are cytochromes, which form chains of four specific types of cytochrome to transport electrons from donor molecules to acceptor molecules through redox reactions (Section 1.2.2).

For the *SimBIOS* simulator, the electron transport chains were modeled roughly based on their spatial configuration since the importance of their functions as a group was considered a higher priority than their molecular anatomy. The transport chains had several animations, including one for when electrons are transported through the cytochromes, for the expulsion of hydrogen protons from the mitochondrial matrix, and for the donation and reception of electrons in the first and last cytochrome of each chain. The cytochromes in the chain were color coded and labeled since it is important to be able to distinguish them, especially the first and last where the donor and acceptor molecules must respectively collide to produce electron transport (Fig 1-4).

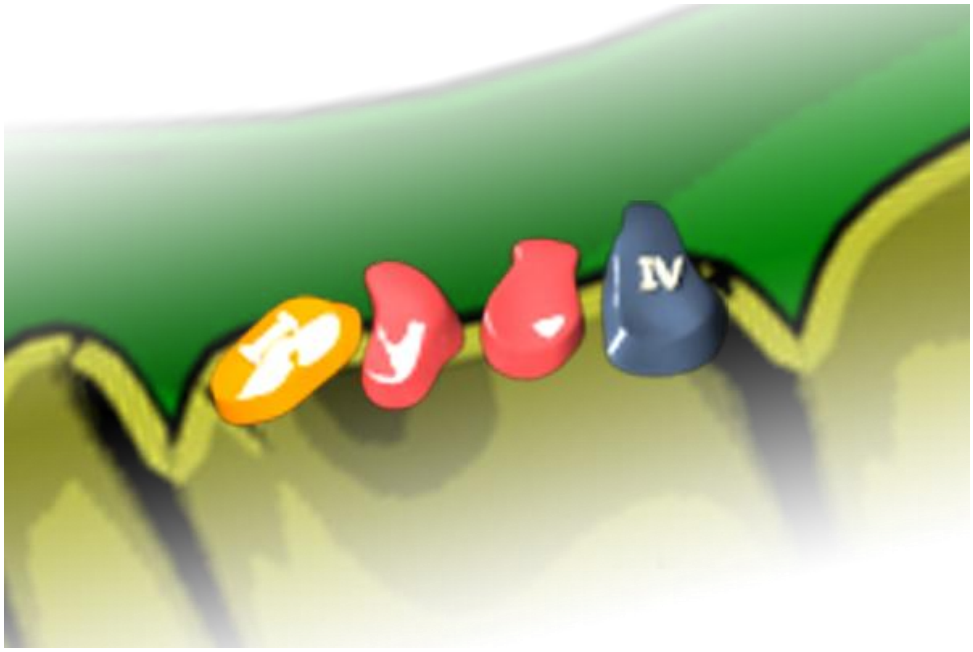


Fig 1-4 The *SimBIOS* representation of an electron transport chain on the mitochondrion's inner membrane

For simplicity and to avoid information overload, only a few electron transport chains are displayed on the inner membrane of the mitochondrion in *SimBIOS*. In reality there are many more, but it would have been impractical to show them all.

1.2.1.4. ATP synthase

This is the other main proteic structure present in the mitochondrion's inner membrane. It uses the electrochemical proton gradient between the intermembrane space and the matrix to transform mechanical energy (protons entering the matrix in favor of the concentration gradient) to transform ADP molecules into ATP (Section 1.2.1.1).

The ATP synthase enzyme was represented, like the electron transport chain, by its rough spatial shape in the *SimBIOS* software (Fig 1-5). It is also animated to show the influx of protons from outside the inner membrane, the addition of ADP molecules to it, and the completion of ATP molecules as a result of the previous two.

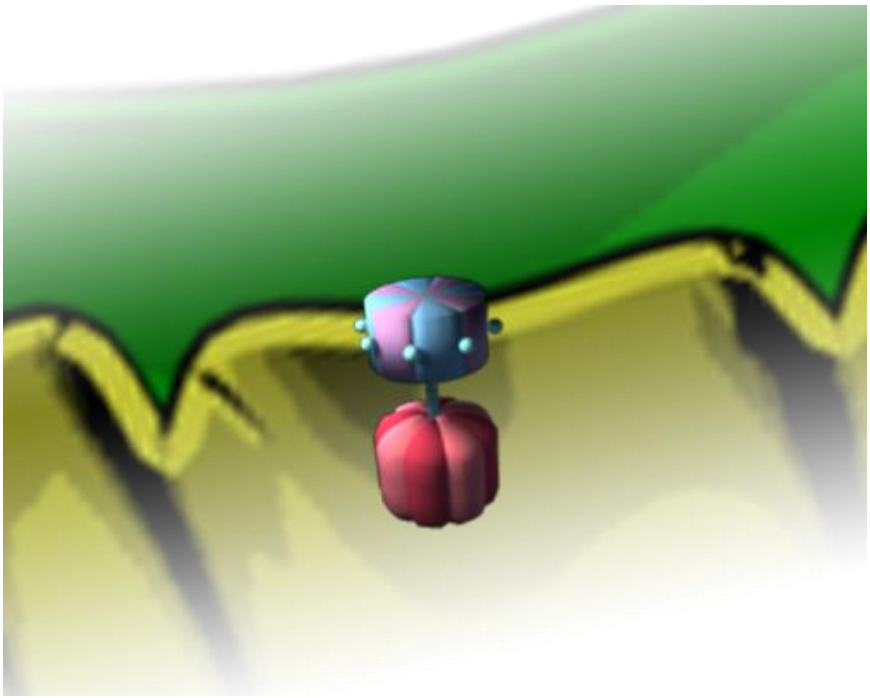


Fig 1-5 The *SimBIOS* representation of an ATP synthase enzyme on the mitochondrion's inner membrane

Like the electron transport chains, only a few instances of the ATP synthase enzyme are displayed in the mitochondrion of the simulation to reflect the presence of many of them in reality, but diminish information overload.

1.2.1.5. Other design elements

Some elements not pertaining directly to the simulation were added to the *SimBIOS* software either for aesthetical effect and immersion of the user, or for a better interface between the user and the simulation and its feedback.

a. Heads-Up Display (HUD)

In videogames, a heads-up display (HUD) is a part of the user's interface with the virtual environment in which information is placed as an overlay to the main visualization. In this way, the player is able to quickly ascertain the state of multiple variables pertaining to the current experience without having to rely on menus or other instruments.

In *SimBIOS*, a simple semitransparent HUD was implemented to be permanently overlaying the simulation. It has four main sections which display different types of data relevant to the activity being carried out at every moment (Fig 1-6).

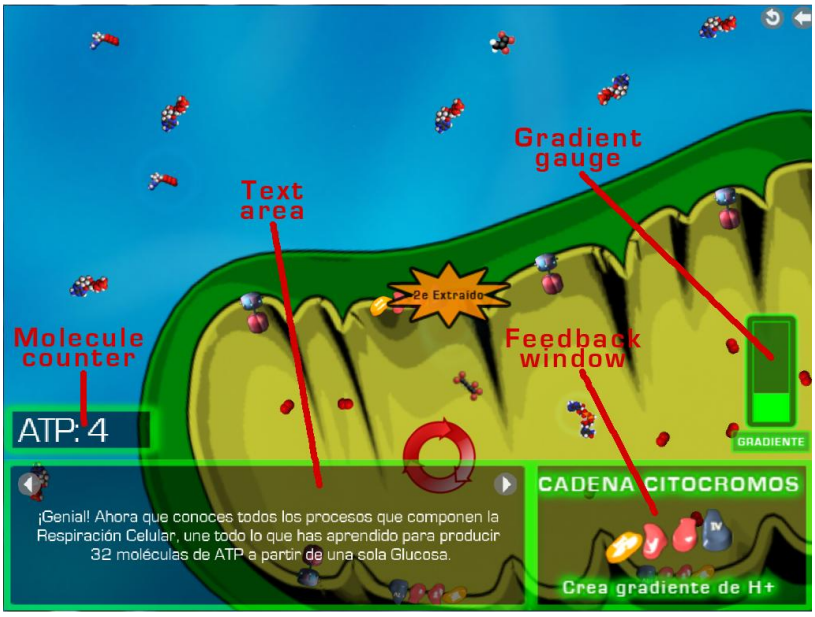


Fig 1-6 Running *SimBIOS* simulation with overlayed HUD

- **Text area:** This is the biggest segment of the HUD, and it is used to display text segments from the script to describe context, current activity information, or objectives. It has two buttons which can be used to fast-forward to the next text segment of the script if it doesn't require some objective to be completed, or to revisit previous text segments.

- **Feedback window:** This square right beside the Text area displays information about the molecule or structure over which the user's cursor is currently hovering. For each molecule and structure, a feedback sprite was made which contains a larger image of the entity, its name, and its main function or characteristic. This was made so the molecules and structures mentioned in the texts and objectives could be identified by the user without previous knowledge of their appearance or if the display window describing it was already closed.
- **Molecule counter:** Most objectives presented by the script during the simulation require the user to produce a certain amount of some type of molecule using the different sub-processes of cellular respiration. These counters show how many of the type of molecule required by the objective are currently in the simulation, therefore giving a constant gauge of progression towards objective completion.
- **Gradient gauge:** From the third level onward, the user must generate an electrochemical concentration gradient of protons within the mitochondrion to produce ATP. To measure if there is enough gradient to produce the reaction, a concentration gradient gauge is present on the HUD for these levels. When there is no gradient (i.e. the amount of protons is the same on both sides of the mitochondrion's inner membrane), the gauge is empty. As gradient is created by the user through the electron transport chains, the gauge is gradually filled until a maximum is reached (there is a limit to the difference in concentrations possible). When gradient decreases again because of ATP synthesis, the gauge gradually empties.

The HUD also includes two extra buttons in its overlay, one of which restarts the current level, and the other returns the user to the main menu.

Also, in the Full version of *SimBIOS*, the script includes an extra type of overlay, called the display window. These are pop-up screens that overlay the simulation and accompany the script's text segments displaying the different molecules and schematics of the processes that are being described (Fig 1-7). These images are not displayed in the Minimal version, and have no equivalent.

Fig 1-7 A display window describing electron transport through a cytochrome chain in the *SimBIOS* Full version

1.2.2. Cellular respiration sub-processes

The main cycle of cellular respiration is subdivided into a series of sub-processes or reactions which progressively break down glucose molecules to finally produce ATP from ADP molecules. Each sub-process' produces output molecules which serve as input molecules for another reaction in the chain. In the *SimBIOS* simulator, these sub-processes must be carried out by the user by dragging the appropriate molecules towards other molecules or mitochondrial structures in the correct order and place. They are progressively included into the simulation according to the script (Section 1.2.4), and can be roughly divided into four distinct sub-processes.

1.2.2.1. Glycolysis

This is the first step in cellular respiration, and all glucose molecules must undergo this reaction to be further processed. It basically consists of a series of reactions between a glucose molecule and 2 ATP molecules to produce an output of 2 pyruvic acid molecules, 4 ATP molecules, and 2 NADH molecules. It is the only sub-process of

aerobic cellular respiration which occurs outside of the mitochondrion (in the cytoplasm).

To generate glycolysis in *SimBIOS*, the user must first drag an ATP molecule, and make it collide with a glucose molecule or viceversa. This triggers an animation which represents the intermediate reactions of glycolisis (including the generation of the glucose-6-phosphate molecule described in Section 1.2.1.1). Then, the user must drag another ATP molecule towards the incomplete reaction to produce the glycolysis reaction and the output molecules (Fig 1-8).

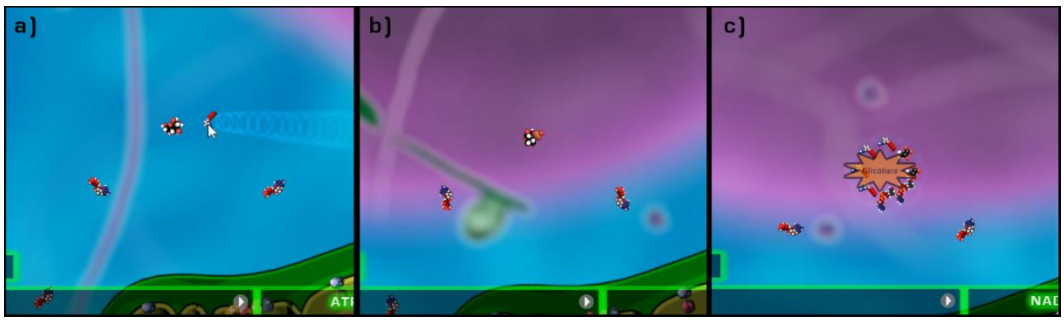


Fig 1-8 The process of glycolysis. a) Dragging the first ATP molecule towards the glucose molecule, b) intermediate reaction, and c) glycolysis reaction and output after the second ATP molecule is added

1.2.2.2. Oxidative decarboxylation

After a glucose molecule undergoes glycolisis, two pyruvate molecules are obtained. These pyruvic acid molecules, upon entering the mitochondria, react with various enzymes in a process called oxidative decarboxylation. After this sub-process of respiration, each molecule of pyruvate is transformed into an Acetyl-CoA molecule and one NADH molecule.

For this process to occur in the *SimBIOS* software, the user must simply drag a pyruvate molecule to the edge of the mitochondrion's outer membrane, which automatically transports it through the membranes and into the mitochondrial matrix where it is processed into Acetyl-CoA and NADH (Fig 1-9).

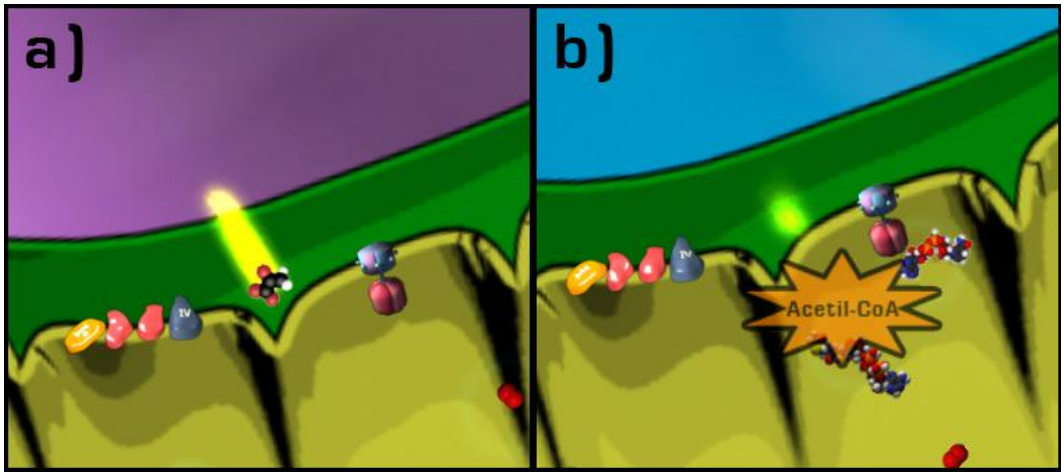


Fig 1-9 Oxidative decarboxylation in *SimBIOS.* a) A pyruvate molecule being transported to the inside of the mitochondrion, and b) the pyruvate molecule breaking up into acetyl-CoA and NADH

1.2.2.3. Citric acid cycle (Krebs cycle)

The citric acid or Krebs cycle is a series of ten consecutive reactions starting with the reaction between acetyl-CoA (product of the previous oxidative carboxylation), oxaloacetate, and water and ending with the production of oxaloacetate which enables it to react with a new Acetyl-CoA molecule indefinitely. During the cycle, 3 molecules of NADH and 1 GTP (energy source like ATP) molecule are produced as well as byproducts like $FADH_2$, carbon dioxide, water, and protons.

In the *SimBIOS* simulation, the Krebs cycle is represented by a spinning icon and an oxaloacetate molecule with which an acetyl-CoA molecule can interact. When the user drags an acetyl-CoA molecule to the cycle, an animation representing the transformations it undergoes is displayed and the steps where NADH or ATP is produced can be seen (Fig 1-10). Although in reality this sub-process of respiration occurs simultaneously many times around the mitochondrial matrix, in *SimBIOS* only one such cycle is displayed to be able to see it more clearly and allow space for the other reactions the user must perform.

1.2.2.4. Electron transport

Although technically a part of the oxidative phosphorylation, the transport of electrons through the cytochrome chains is described as a separate sub-process just to highlight the importance of its two distinct steps. On this first one, NADH molecules produced in glycolisis, oxidative decarboxylation, and the citric acid cycle (as well as other electron

donor molecules like $FADH_2$; see Section 1.2.1.1) collide against the first cytochrome (cytochrome I) of mitochondrion's electron transport chains to break up into NAD^+ and H^+ molecules, and "donate" or "yield" two electrons to the cytochrome. These donated electrons are then transferred sequentially to the others in the chain until they arrive at the fourth cytochrome (cytochrome IV). There, they must be retrieved by an electron "receiver" or "acceptor" molecule such as oxygen. The passage of electrons along the ETC complexes releases energy that enables them to act as pumps that expel protons from inside the mitochondrial matrix towards the intermembrane space, creating a proton concentration gradient which will later be used by ATP Synthase enzymes to generate ATP.

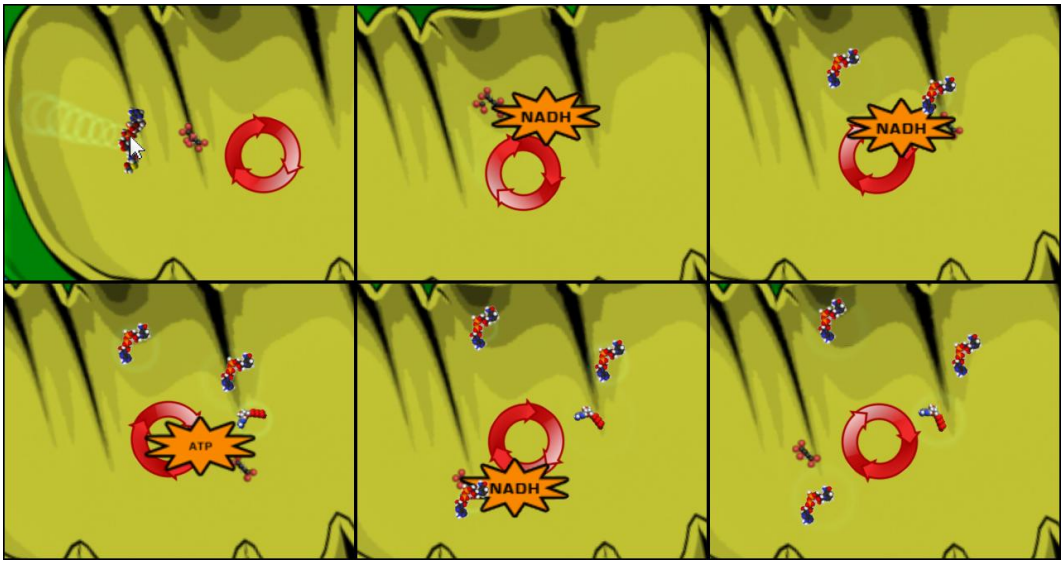


Fig 1-10 Acetyl-CoA undergoing the citric acid cycle, highlighting the steps in which NADH or ATP is produced

Like the other processes of the simulator, for this one, the user must simply drag the correct molecules to the adequate mitochondrial structures. First he must drag a NADH molecule to the first cytochrome complex (complex I). When they collide, an animation is shown that notifies the user that 2 electrons were donated, they are shown "jumping" along the electron transport chain, and a particle emitter shows hydrogen protons going out of the mitochondrial matrix through the ETC. Then, the user must drag an oxygen molecule towards the final cytochrome of the same chain where the electrons were donated (complex IV), and an animation is displayed to notify him that the 2 electrons were retrieved. Then the gradient bar on the HUD rises by the appropriate amount to represent the increase in the proton gradient difference (Fig 1-11).

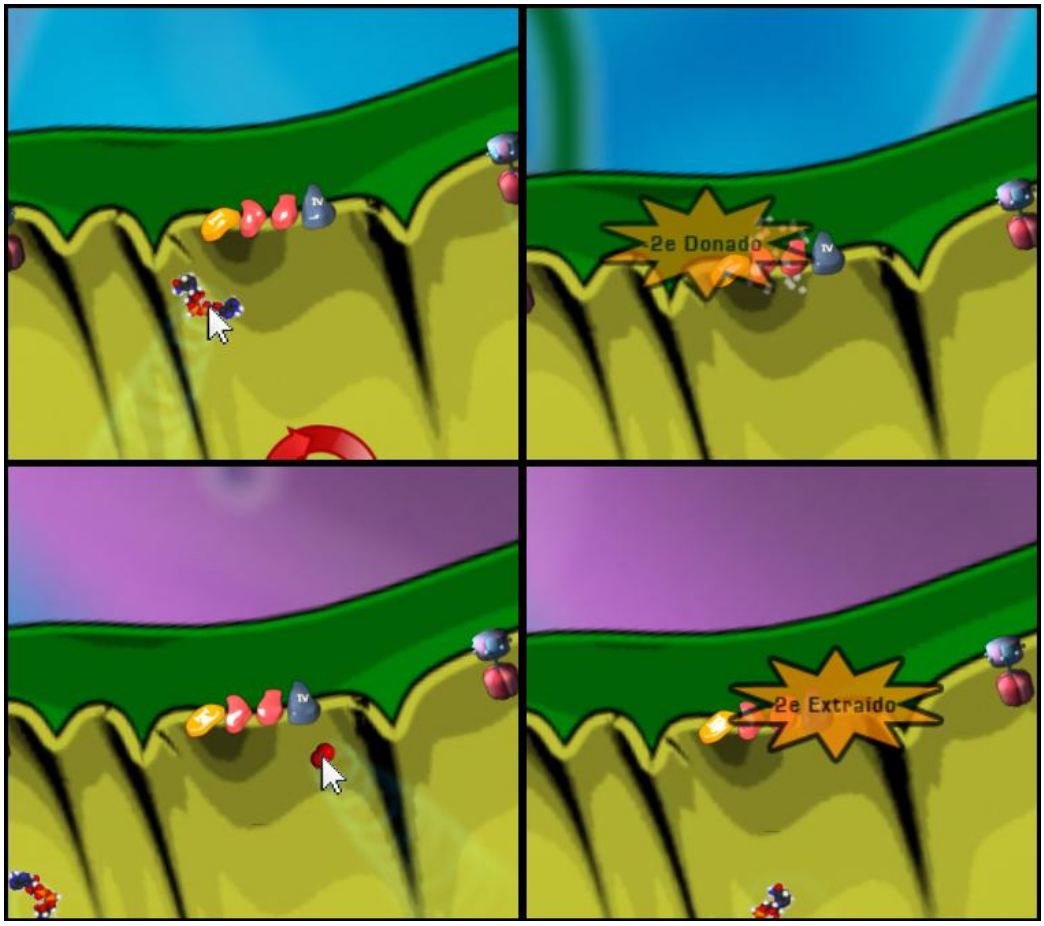


Fig 1-11 Electron transport in ETCs and proton gradient generation in *SimBIOS*

1.2.2.5. Oxidative phosphorylation (ATP synthesis)

The proton gradient between the outside and inside of the mitochondrial matrix produced by electron transport through the cytochrome chains effectively creates a mechanical energy potential. These protons flow back into the mitochondrial matrix through the ATP synthase complexes present in the inner membrane, doing mechanical work, much like a turbine. This work enables a third phosphate group to be coupled to an ADP molecule inside an ATP synthase to produce an ATP molecule, and be released from the mitochondrion. This is the main mechanism through which ADP molecules are transformed into the ATP chemical energy cells need.

In the *SimBIOS* simulation, this last step requires the user to have previously generated a proton gradient between the inside and outside of the mitochondrial membrane by using the ETCs. Then he must drag an ADP molecule (present in the cytoplasm) towards the

mitochondrion, where it is automatically transported towards the nearest ATP synthase enzyme of the inner membrane. If the proton gradient is enough, as soon as ADP molecules are added to the ATP synthase structures, an animation is displayed showing the spinning ATP synthase and the protons going back into the matrix. After a few seconds an ATP molecule is produced, notifying the user (Fig 1-12). If ADP molecules are loaded into an ATP synthase structure, but there isn't enough proton gradient to produce ATP, a warning from that enzyme issues every few seconds to notify the user that more is needed.

Fig 1-12 An ADP molecule being transformed into an ATP molecule by an ATP synthase structure using the proton gradient

1.2.3. Cellular respiration

All the sub-processes described in Section 1.2.2 form the cycle of eukaryotic aerobic cellular respiration; the most important form of energy generation in animal cells. By the end of the experience with the *SimBIOS* simulator, the user must be able to understand and concatenate all these sub-processes, using the output of each as the input for the next to simulate the parallelism of the whole process and ultimately produce ATP from glucose and ADP.

1.2.4. Script

The main comparison in this study was made between different ways in which the activities and concepts trying to be taught by the simulator were presented to the student. That is, in the scripting of the experience; specifically on the impact of complexity and rigidity/flexibility of the script in the learning of concepts.

For this purpose, two versions of the *SimBIOS* simulator were made. One which incorporates the minimal elements of model progression and game-like objectives described in Section 3 of the paper (the Minimal version) and one which incorporates more elements of tutorial type guides, contextual images, and intermediate objectives

which restrict the user's actions, limiting the discovery process and making it more expository (the Full version).

In both cases, the script of the activity is divided into four levels, each one introducing a sub-process of cellular respiration (Section 1.2.2) and then incorporating it with the ones already learned in an end-of-level objective that allows the student to progress to the next level. When the simulator is started, a screen is shown where the user can select a level to begin (Fig 1-13). This is in case he wants to go back to a previous level or wants to skip to a specific level if restarting the experience. The users can go back to this menu at any time, but it is usually recommended that they start on the first level (Glycolysis), and go through the levels by completing the objectives in each one, thus following the planned model progression.

Fig 1-13 SimBIOS main menu where the user can pick the level he wants to start in

The four levels are described in detail below, showing the differences between the Full and Minimal versions in each one.

1.2.4.1. Level 1: Glycolysis

In this level, the student is presented with the context of the activity (presentation of the cell, its cytoplasm and the mitochondrion) and cellular respiration's glycolysis sub-process.

Minimal version	Full version
Welcome text and animation of zooming in from outside the cell towards a mitochondrion.	Welcome text and animation of zooming in from outside the cell towards a mitochondrion
Introduction text to the mitochondrion.	Introduction text to the mitochondrion accompanied by images of mitochondria through electron microscope and its representation in *SimBIOS*.
Introduction texts to cellular respiration and the first sub-process of glycolysis.	• Introduction text to cellular respiration and the first sub-processes of glycolysis. • Each text is accompanied by an image of the molecules
End-of-level objective is given: produce 10 pyruvate molecules with a simplified glycolysis formula as a clue. All actions on molecules are enabled.	• Objective is given: drag a first ATP molecule towards a glucose molecule. No other actions can be performed. • Objective is given: drag a second ATP molecule towards the incomplete glycolysis reaction. No other actions can be performed. • When first glycolysis is achieved, the simulation is reset. • End-of-level objective is given: produce 10 pyruvate molecules with a simplified glycolysis formula as a clue. All actions on molecules are enabled.

1.2.4.2. Level 2: Oxidative decarboxylation and Krebs cycle

This stage teaches the oxidative decarboxylation sub-process, but since the mechanics involved are too simple, it also includes the citric acid or Krebs cycle sub-process. First, a pyruvate molecule provided must be decomposed to acetyl-CoA and NADH, to then use the former in the Krebs cycle. After that, the simulation resets and the student must

produce a number of NADH molecules using all sub-processes up to that point (i.e. glycolysis, oxidative decarboxylation, and citric acid cycle).

Minimal version	Full version
Introduction texts to the sub-processes to be learned.	Introduction text saying that pyruvate will now be used inside the mitochondrion to produce ATP.
Objective is given: produce NADH molecules from pyruvate. Simplified equations for oxidative decarboxylation and Krebs cycle are given as clues.	Objective is given: drag a pyruvate molecule to the mitochondrion. No other actions can be performed.
	• Explanation texts of the process by which pyruvate are decomposed into acetyl-CoA and NADH with images of relevant molecules. • Explanation texts of the Krebs cycle process with schematic images of its steps. • Objective is given: drag an acetyl-CoA molecule to the Krebs cycle
When 4 NADH molecules are produced (1 from oxidative decarboxylation and 3 from the Krebs cycle), the simulation is reset with glucose and ATP molecules.	When the previous objective is completed, the simulation is reset with glucose and ATP molecules.
End-of-level objective is given: produce 20 NADH molecules from glucose. All previous clues are given, and all actions on molecules are enabled.	End-of-level objective is given: produce 20 NADH molecules from glucose. Relevant clues to the sub-processes needed are presented (see Minimal version script), and all actions on molecules are enabled.

1.2.4.3. Level 3: Oxidative phosphorylation

This is the last level in which new sub-processes are introduced. They are the ones that compose oxidative phosphorylation (ATP synthesis): the creation of proton gradient through electron transport, and the production of ATP from ADP on ATP synthase enzymes. They are introduced separately during the level to keep the model progression.

Minimal version	Full version
Introduction texts explaining the necessity of a proton gradient	Introduction texts explaining the necessity of a proton gradient accompanied by images illustrating proton gradient in mitochondria.
• Explanation texts of the functions of ETCs, electron donation by NADH,	• Explanation texts of the functions of ETCs accompanied by images of the

electron transport, and electron reception by O_2. • Objective is given: produce proton gradient using ETCs. NADH and O_2 are provided, as well as a simplified formula for electron transport as a clue. Only actions related to ADP are restricted.	structure. • Explanation text of electron donation by NADH accompanied by a schematic image of the process in *SimBIOS.* • Objective is given: drag a NADH molecule to the first cytochrome complex in an ETC. No other actions can be performed
	• Explanation texts for electron transport and reception are displayed along with schematic images of these reactions in *SimBIOS.* • Objective is given: drag an O_2 molecule to the fourth complex of the same ETC to retrieve the electrons and produce gradient. No other actions can be performed.
• Once a concentration gradient has been established, explanation texts for ATP synthesis and subsequent gradient decrease are displayed. • Objective is given: produce 6 ATP molecules from ADP while maintaining proton gradient. Simplified formulas for ATP synthesis and electron transport are given as clues, and all actions on molecules are enabled.	• Once a concentration gradient has been established, explanation texts for ATP synthesis are displayed along with schematic images of the reaction in ATP synthases. • Objective is given: drag an ADP molecule to an ATP synthase. No other actions can be performed.
	• Explanation texts for the decrease in proton gradient due to ATP synthesis are displayed • Objective is given: produce 6 ATP molecules from the remaining molecules while maintaining proton gradient. Simplified formulas for ATP synthesis and electron transport are given as reminders, and all actions on molecules are enabled.

1.2.4.4. Level 4: Integration

This level is actually the continuation of the previous one, as it is the incorporation of the last two sub-processes of cellular respiration to the cycle as a whole. In it, the student must simulate the complete respiration process from glycolisis to ATP synthesis starting from a single glucose molecule.

Minimal version	Full version
End-of-simulation objective is given: produce 32 ATP molecules from a single glucose molecule. All auxiliary molecules (ADP, O_2, and 2 ATP) are provided and all actions on molecules are enabled.	

When the end-of-simulation objective is achieved, the experience is over. A congratulation message displays, and the program restarts to the main menu screen to be able to redo any level if desired.

1.3. Software engineering

The *SimBIOS* educational computer simulator was written in the C# programming language with the Flat Red Ball engine over the Microsoft XNA engine, using well known software engineering conventions and best practices for current object-oriented programming. Given the platform on which it was developed, the software should be able to run on any computer with an operating system supporting the .NET Framework 2.0, although it has only been successfully tested in Windows Vista and Windows 7.

1.3.1. The FlatRedBall engine

When the project first started, a team of engineering students was contracted to assist with the programming of the *SimBIOS* software, and their first task was to assess the viability of developing it in one of the multiple engines that could suit the needs of the enterprise. The one deemed most promising from early prototypes was the FlatRedBall game engine; a cross-platform set of tools developed independently by Victor Chelaru, and first released in 2005. It was developed as an abstraction of the Microsoft XNA API for DirectX, and distributed as freeware for commercial and non-commercial game development.

1.3.1.1. Pros and cons

The main advantage this engine has, aside from being free to use, is the availability of multiple tools to facilitate the development of 2.5D computer games (i.e. games that simulate 2D graphics through the use of 3D objects), such as an animation editor, sprite editor, level editor, etc. It is also relatively easy to use, well documented, and has a relatively active community giving support for it.

1.3.1.2. Essential functioning

Like most game engines, FlatRedBall provides abstractions in the form of APIs and functions for the display of graphical objects, simulation of physics, and real-time processing of user input, backend logic, and updating of the graphical interface. The first is done through a content pipeline which incorporates files referenced in templates made previously in the sprite, animation, and level editors as needed; and managed through an API by the program implementing the engine. Simple physics like collisions, accelerations, and forces are implemented in the main engine, but all graphics related tasks are passed on to XNA, and through to DirectX. As in XNA, a FlatRedBall program must instance a Game class which will run its two main loops: Update and Draw. These

will process user input and game (or simulator) logic continuously while redrawing all visible graphical objects on the screen at the configured refresh rate per second.

Aside from these functionalities and some extendable classes to work with them, most of the rest of the logic is left to the implementing program, making it quite flexible if a bit "bare bones".

1.3.2. The architecture of *SimBIOS*

The simulator was built hierarchically through class inheritance so as to resemble the semantics of the cellular respiration model as closely as possible while maintaining certain classifications and constraints imposed by the FlatRedBall engine (FRB from here onward)(Fig 1-14). Therefore, all classes and content were divided into 5 categories: Base, Entities, Screens, User interface, and Content.

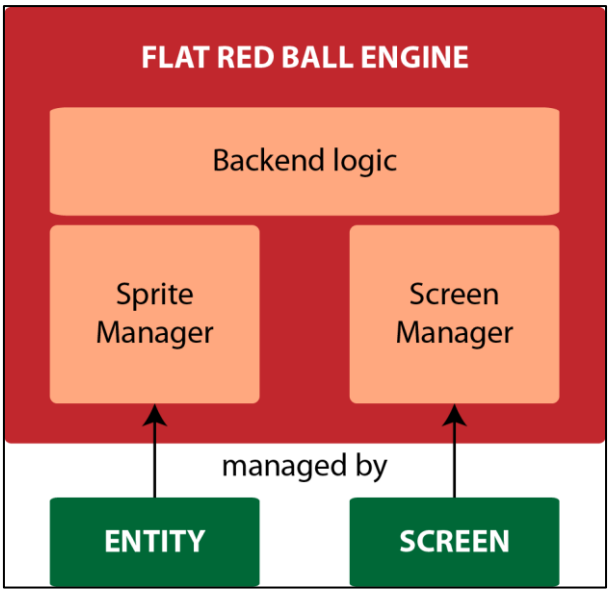


Fig 1-14 FRB engine, its two main content managers and the types of classes they manage

1.3.2.1. Base

In the Base category, all base classes from the engine and their corresponding interfaces are stored. These classes are inherited by most other classes in a FRB application. In this implementation it included the following classes and interfaces:

- **Entity:** This class must be inherited by any other class that wants to be managed by one of the engine's managers. This usually implies the presence of a sprite or a collision polygon, since Entity inherits from the PositionedObject class of the engine; which is handled by managers that place objects on the screen. In this implementation, the Entity class also implements the IFeedbackWindowable interface.
- **FRBExtension:** This was created as a wrapper extension for the normal FRB libraries because of the absence of inner collision detection in the original engine. That is, FRB can detect if two polygons are colliding along their borders or if one polygon is completely within the other. However it cannot state if, when one polygon is within the other, if the one within is colliding along its border with the one containing it (i.e. if the inside polygon is "trying to get out" of the outer polygon). This feature had to be added for the purpose of this simulation.
- **IFeedbackWindowable (interface):** Most objects in the simulator display a feedback window on the lower right corner of the Heads Up Display (HUD) when the cursor is hovered upon them which describes it briefly. This interface was implemented to have a reliable way of assigning, getting, and displaying feedback windows for all Entity derived classes.

1.3.2.2. Entities

As its name implies, this category holds all classes that correspond to objects shown on screen (inheriting from the Entity class of Section 1.3.2.1) or classes inherited by these. Here the main semantics of the simulation model can be seen in the syntaxes (Fig 1-15).

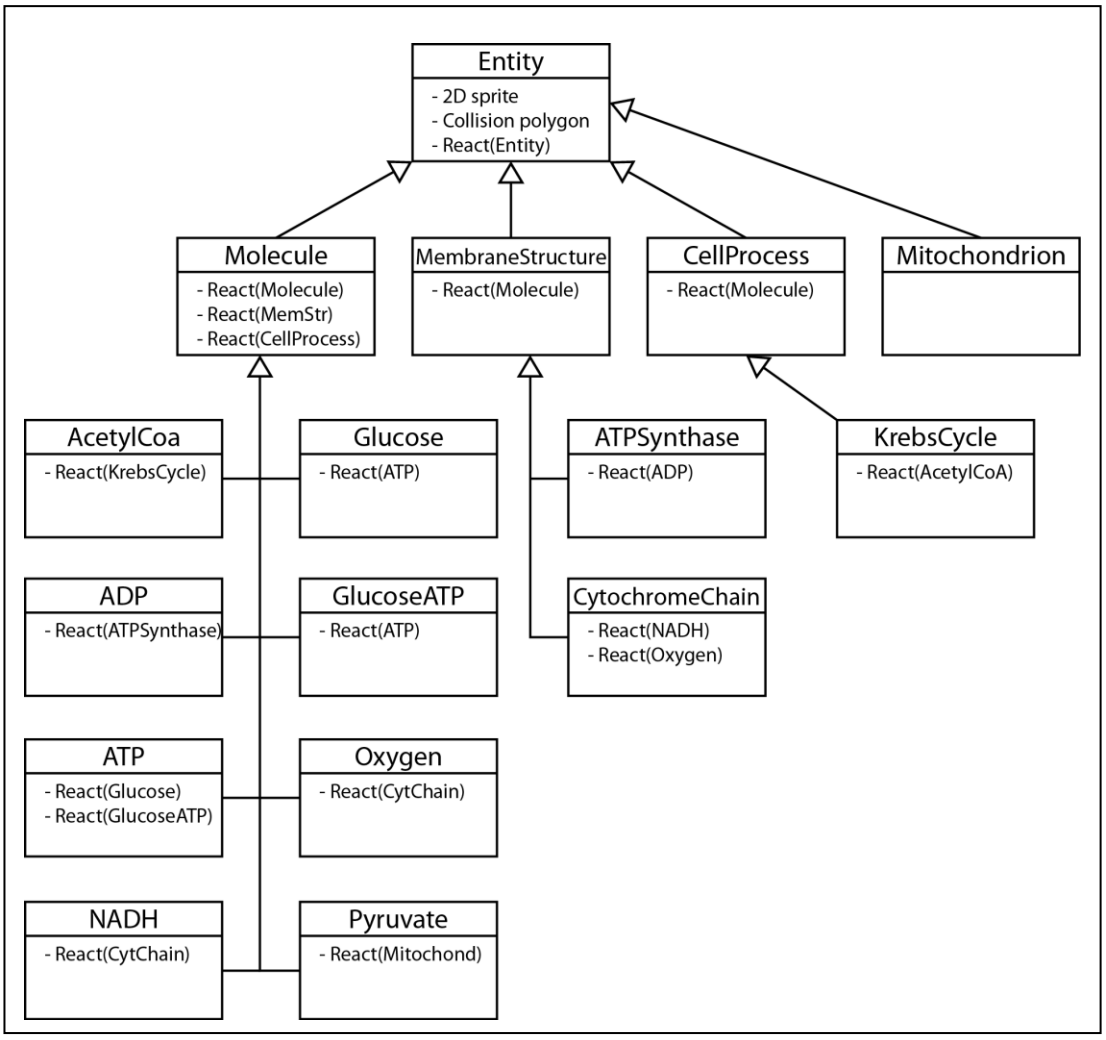


Fig 1-15 The simulation's most important elements and their derivation from the Entity class

- **Molecule:** Class from which all molecules of the simulation inherit. It inherits directly from the Entity class and implements their common functions such as their movement and overrideable methods for reactions with other molecules. Each inheriting Molecule class must provide its own sprite, molecule type (from an enum), and feedback window file.
- **AcetylCoa:** It is a molecule present in cellular respiration. It has two reactions particular to it and a particle emitter for feedback notices on its processes.

- **ADP:** A molecule with no particular methods, only its particular sprite and feedback window.
- **ATP:** A molecule that implements a particular reaction with other molecules; specifically Glucose.
- **Glucose:** A molecule that implements a particular reaction with other molecules; specifically ATP.
- **GlucoseATP:** This intermediate molecule represents an incomplete Glycolysis process. It is particularly animated, and implements a reaction with ATP as well to complete the process.
- **NADH:** A molecule with no particular methods, only its particular sprite and feedback window.
- **Oxygen:** A molecule with no particular methods, only its particular sprite and feedback window.
- **Pyruvate:** This molecule implements a particular reaction depending on its location, and its subsequent transformation into AcetylCoA and NADH molecules.
- **MembraneStructure:** Class from which all mitochondrial membrane structures in the simulation inherit. Like Molecule, it inherits directly from the Entity class, and implements common elements that all of them have such as their animation. Each inheriting mitochondrial structure must provide its own sprite, animations, type (from an enum) and feedback windows.
- **ATPSynthase:** It represents the mitochondrial enzyme that synthesizes ATP; therefore it implements the transformation function between ADP and ATP molecules. It has two animations and a particle emitter for feedback notices.
- **CytochromeChain:** It represents the electron transport chain proteins in the mitochondrial structure. It has particular functions to manage the reception and release of electron charges from NADH and Oxygen molecules, as well as two animations, and 3 particle emitters. Two of the emitters are for feedback notices of electron reception and release, and one is for the simulation of Hydrogen protons exiting the mitochondrial matrix.
- **CellProcess:** Class from which all cellular sub-processes that are represented by a constantly present sprite (or group of sprites) on screen (as opposed to being represented by multiple objects interacting) inherit. It implements no particular methods except to determine if it is collideable or interactable.

- **KrebsCycle:** The only sub-process that was left inheriting from CellProcess in the final version. It implements all the steps of the reaction it represents (the citric acid cycle), including the processing of AcetylCoA molecules, and production of NADH and ATP molecules. It has two animations that must be managed simultaneously on two different sprites.
- **Mitochondrion:** This is one of the most important Entities in the simulation, as it handles most of the molecule's movement. This is done by multiple particular functions that manage the mitochondrion's inner and outer bounds, modifying other Molecules and MembraneStructures' position, reaction status, etc. It has two sprites (one for the inner membrane and one for the outer), an array of MembraneStructures, and it is constantly animated.
- **Background:** This class handles the display and movement of the background sprites. It contains the sprites of the 3 background layers and it has particular functions that move each independently in pseudo-random directions and different speeds to give the illusion of multiple depths; while keeping them inside predetermined bounds.
- **IntroAnimation:** An Entity derived class particularly made to display the first contextualization animation of the simulation where the cell is shown from outside and then zoomed into the cytoplasm to start the first stage. Since this animation was radically different from the other sprites and had so many variables that could be tweaked to fit the script text and events (e.g. time until zoom, zooming speed, amount to zoom, point of the image to zoom to), that it was decided worthwhile to implement on its own.

1.3.2.3. Screens

Screen type classes are a convention that the FRB engine uses to manage game "levels" or "stages". Each represents a managed context in which certain sprites, animations, polygons, etc. are loaded and handled under certain logic until a different Screen is loaded (e.g. change of level, pause screen, main menu). These classes are also the ones which hold all the game's, or in this case the simulator's, rules and backend logic. The FRB recommendation is that all common logic that applies to all levels is programmed into a base Screen class (that directly inherits from Screen), and then all other specific levels are programmed with their particular logic into classes inheriting from that base class (Fig 1-16). This was implemented in *SimBIOS* through the following classes:

- **Screen:** This is a pre-made class bundled with the FRB engine, and not meant to be instantiated; only inherited. It manages the loading and unloading of sprite, animation, and physics managers, be it synchronous or asynchronous. It also implements the methods to clean up after a screen is destroyed (to be replaced by another), changing states (pause, unpause, etc.), and some cross-platform compatibility (e.g. with the Xbox 360 console).
- **Level:** The base class inheriting from Screen where all the main logic and rules of the *SimBIOS* simulator are housed. In the main activity loop, called by the engine continuously, the user's input is processed as well as the collision between all Molecules and structures in the simulation. Depending on these, the appropriate reactions on those objects are called. This class also handles the creation and destruction of Molecules influenced by the different sub-processes, the calls to update the HUD, the movement of the camera according to user cursor position, the calls to update the background animation, the display of feedback windows, and other general management of objects.

  Also, since it was decided that the levels were not going to be hard coded into separate classes due to the necessity of two different scripts (i.e. different level events and logic) for the two versions of *SimBIOS*, this class also handles the loading of scenes from the parametrized XMLs which contain the scripts (detailed in Section 1.2.4) when ordered by the LevelEventManager (Section 1.3.2.7).
- **LoadingScreen:** This Screen is a simple one without much logic in its activity loop aside from showing a background sprite and an animation while Level loads the selected stage of the simulator asynchronously in the background.

1.3.2.4. User interface

In this category were all the classes that correspond to objects which help the user interact with the simulation. It contains the following classes:

- **HUD:** This class displays the Heads-Up Display overlay on all the stages of the simulation (except the main menu). It handles the text shown, the feedback windows for when an object of the simulator is hovered upon with the cursor, the counters of different molecules which indicate

objective completion, and the buttons to skip/rewind texts, restart the current stage, return to main menu, etc.

- **DisplayWindow:** This was used to show pop-up windows with information about molecules or structures of the simulation and schematics overlaying the simulation to aid the text script in the Full version of *SimBIOS*. These were not used in the Minimal version of the simulator.
- **SimulationButton:** This was an extension of the FRB Button class (FlatRedBall.Gui.Button) made to instantiate the different buttons available to the user in the HUD. It doesn't have particular logic different from its parent class, as it was implemented mainly for typification purposes.

1.3.2.5. Content

This section contains all the resources used by the application during simulation.

- **Fonts:** Image based font files that the FRB engine uses to display text characters during simulation.
- **Levels:** XML files made by the FRB level editor which indicate presence and position of the initial sprites of a level and which are parsed by the Level class of the application.
- **Sprites:** Contains the image files of every sprite in the simulation. It also contains animation XML files (made through the FRB animation editor) which parametrize the size, number of frames, animation speed, etc. in the spritesheets of animated sprites. The polygon XML files (made through the FRB polygon editor) for the sprites that need more detailed collision boxes are also included. Finally, the particle emitter XML files (made through the FRB particle editor) are contained for the sprites that have them.

1.3.2.6. Scripts

This folder of the Content section contains the XML files where the events, actions, and steps of each level of the simulator are described. A customized XML template was made to divide the script: contextualizing texts, feedback, objectives, etc. into ScriptedEvent instances when parsed by the LevelEventManager (Section 1.3.2.7). By doing this, the parametrization of the script was externalized from the logic of the main simulator, making it much easier to make two distinct versions of it without modifying its code. The only difference between the Full and the Minimal versions of *SimBIOS* are their XML files for the levels.

Each level consists of two XML files. One is divided into events, each of which must be triggered by user actions, in-simulation timers, the production or destruction of molecules, etc. to proceed to the next, until the end of the level. The other holds the text strings linked to each of the ScriptedEvents. These are the texts that are shown in the HUD to give context to the activities, feedback, or describe objectives for the current event of the script.

1.3.2.7. Other classes

Since the FRB standard was not implemented to the letter in the *SimBIOS* application due to the intent of making two different versions of the software from the same code, and therefore trying to hardcode the least amount of script (i.e. level or stage) related code possible, some extra classes for the management of scripts and levels were required (Fig 1-16). Here is a description of their functionality, along with other necessary basic classes to complete the program.

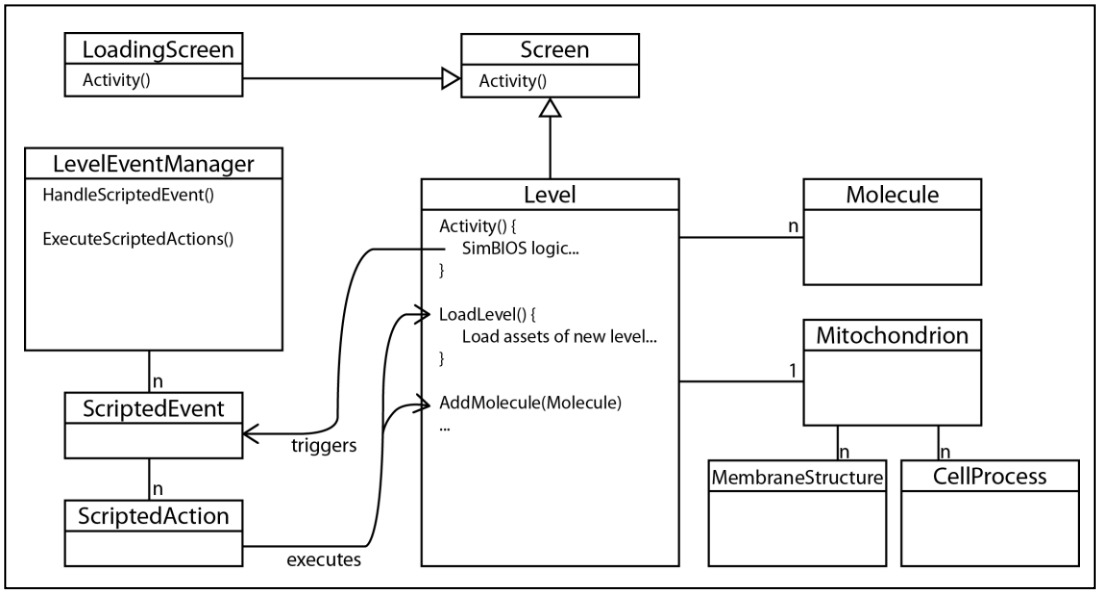


Fig 1-16 The Level class and its most relevant interactions with other classes and the LevelEventManager

- **ScreenManager:** This is FRB's main manager for changing screens (i.e. levels). However, since this type of level management was essentially bypassed by the LevelEventManager to load and unload each stage on the same Screen object, it is only used to show the LoadingScreen instance while Level asynchronously loads the correct stage onto the different managers from the scripted XMLs. According to FRB documentation, this is also a valid form of Screen management for certain cases.
- **LevelEventManager:** This singleton pattern class orchestrates the script of the simulator for both versions. When a level is selected, it parses the XML file corresponding to that level, and creates an array of ScriptedEvent objects to which it subscribes. As these events are triggered (by user actions or timers), this class handles the appropriate response and passes on to the next step in the script for that level, changing the corresponding texts, images, loading scenes, destroying objects, etc. When the end of the script is reached (a specific ScriptedEvent), it signals the Level class to load the next stage.
- **ScriptedEvent:** Instances of this class represent steps in the script of a level or stage of the simulator. They correspond to a type of in-simulator event (e.g. collision of two molecules or structures, the production of a molecule, click of a button, timer elapsed, etc.). When any event is fired in the simulation, the LevelEventManager checks if the current ScriptedEvent corresponds to that type of event, and proceeds to the next step in the script. Each ScriptedEvent instance is tied to a segment of text in the script and a series of ScriptedAction instances to be performed when the ScriptedEvent is handled.
- **ScriptedAction:** Instances of this class represent actions to be carried out by the LevelEventManager when a ScriptedEvent is handled. They consist in producing or destroying molecules, loading new scenes, showing or hiding feedback windows, changing levels, and enabling or disabling certain allowed user actions on the simulator. A ScriptedAction instance is directly linked to one ScriptedEvent, but a ScriptedEvent instance can have many ScriptedAction instances to be executed when being handled.
- **Polygonizer:** A static class implemented to detect the borders of a non-transparent body inside a sprite, and transform it into an optimized FRB polygon type. This is used to obtain the bounding polygon of irregular sprite shapes which need more accurate collision detection than a square or

circle around the sprite. Specifically it is used to determine the contours of the two sprites of the Mitochondrion to establish the bounding frontiers of its inner and outer membrane.

- **PolygonNode:** A construct used in the construction and optimization of FRB polygons found by the Polygonizer class.
- **Game:** A mandatory instance of this class must be started in the main loop of the program to initiate an instance of the FRB engine. It handles the engine's initialization and calls its main loops (Update and Draw) as well as setting the screens resolution, refresh rate, or other programmed variables.
- **Program:** Contains the main loop of the application, which initiates the instance of the Game class to start the FRB application.

1.4. Experimental experience

The experience with the *SimBIOS* software consisted of two sessions, one of which was destined to the pre-experience test (45 minutes) and the use of the simulator (45 minutes), and the other to the post-experience test.

For the use of the software, two groups of 10^{th} grade Chilean private school students were formed and assigned to each of the *SimBIOS* versions. Since four classes were part of the experiment (for a total of 148 male students), each group consisted of two of these 36 or 37 alumni classes. In other words, there were 2 classes of 36-37 students using the Minimal version of *SimBIOS*, and 2 classes of 36-37 students using the Full version. The experience was conducted separately with each class in the school's computer labs, where each user had an individually assigned computer with the corresponding version of *SimBIOS* installed.

At the start of the experience, students were told to start form the first level of the simulator (Glycolysis, see Section 1.2.4.1) and try to progress through the activities presented without outside help. They were encouraged to attempt to figure out the problems and objectives by themselves, but interaction with other students or asking the teachers or researchers for help was not forbidden in case they got stuck.

Although the time taken to complete the four levels of the simulator varied slightly from student to student, there was an approximately 30 minute interval in which they were completely focused on their tasks, with minimal interaction between each other. As the first users completed the simulation (usually the ones who grasped the concepts quickest), there was a natural tendency for them to try to help their classmates who were

struggling with some part of it. However, there was also a segment of the students who managed to get through the objectives by random trial and error (i.e. dragging all molecules against all other molecules and structures until the correct ones reacted). Therefore, when they had finished for the first time, they were encouraged to try to do the simulation again with as few moves as possible, trying to understand the underlying processes instead of just the mechanics. This generally gave the students focus for another, more conscientious run of the activities.

After the post-test in the second session, a survey of the experience was administered, registering students' perceptions, motivation, and general opinions about the software and the experiment. According to their answers, it seems that it was generally a motivating experience where they felt they learned in a fun and intuitive way. A majority of them felt they would benefit from more didactical software like *SimBIOS* if incorporated into the regular curriculum.

The results of the pre-tests and post-tests, and the improvement in scores is discussed in Section 2.6.

1.5. General conclusions

Given the significant improvement in test scores by students using the *SimBIOS* software, it is clear that effective educational simulators can be designed through a methodical approach. In fact, by proving that software based on a specific model compiled from research and experience yields good results, the "gamble" of producing didactic software by the instincts of the developer is greatly diminished. In other words, the alchemical process by which educational simulators (and games) are made-- in which the design choices are based primarily on what the creator thinks will work-- can begin to advance towards a more scientific approach that guarantees certain results.

However, the progress made here through the proposed model and guidance, and the subsequent simulator based on them only scratches the surface of a very complex subject. The proposed model has yet to be proven in other domains of science, and the experiment to be extrapolated for software dedicated to longer experiences, older and younger students, etc. Also, the individual characteristics of the model can still be refined by further experimentation, similar to what was done in this research with script complexity (which is only a specific part of the whole design).

Specifically for *SimBIOS*, although it achieved its goal of teaching the concepts of cellular respiration while maintaining a strong motivation and engagement of the students, some improvements could be made to achieve an even better performance. For

example, a limitation could be introduced to the number of moves a user can make during a level. This could reduce the amount of trial and error used by some students, and allow for different difficulty levels to motivate students who found the experience too easy. Another improvement (or design consideration for other software made along these lines) would probably be the porting of the software to a browser based application. This would greatly increase its compatibility with almost any current hardware and operating system, thus making its distribution and widespread use much easier.

1.5.1. Limitations of this work

One great drawback in the implementation of the *SimBIOS* simulator was the use of the FRB engine. Since at first this project consisted principally of a game, it seemed an ideal choice. However, after two years of programming with it, its shortcomings became evident. Since it is developed by a very small team (principally just one person in the main engine), the software has quite a few bugs, and they are not able to fix them fast enough. The project is not open-source, so there is no possibility to try to fix or debug encountered errors inside the engine's processes or to extend its functionalities. There are also some fairly important performance issues, especially with collisions between objects, which made it hard to develop for low-end computers with only integrated graphic cards (i.e. typical school computer lab PCs). In the end, this caused unexpected errors like the random crash of the program within FlatRedBall's main loops. Without the option of debugging the problem or getting reliable support from the developers (given the stochasticity of the issue), the only choice was to leave the bug and hope it didn't occur during experimentation with the software. Some features had to be taken out of the simulator because of performance issues as well. Since some of the engine's functions weren't optimized, their use slowed the program too much in low-end computers for correct usability.

During the experiment, this did eventually cause some unexpected crashes while using the simulator. However, since it was programmed so that any state in the simulation was easily regainable through the main menu, much progress was never lost. Crashes and bugs did not affect the attitude of the students toward the experience according to the post-experience survey, but it is still a problem to fix in future iterations. This could probably be done with the port to a browser-based application.

2. THE VALUE OF GUIDANCE WHEN TEACHING SCIENCE USING AN EDUCATIONAL SIMULATOR

2.1. Introduction

Simulators or computer simulations have been used since the beginning of modern computing, and in education for the past forty years (Smetana & Bell, 2011). A simulator mainly consists of interactive software that models a natural or artificial system and emulates its processes (Blake and Scanlon 2007; de Jong and van Joolingen 1998). It allows users to experiment with phenomena that are either too difficult or too expensive to test in the physical world, manipulate the different variables of their processes, and observe what happens when they are modified (Efe, H. A. and Efe, R. 2011).

The teaching potential of this medium has been thoroughly investigated as technology has advanced and become more readily available (Rutten et al. 2012; Smetana and Bell 2011; Vogel et al. 2006). This has created a branch of educational simulation software which has added a plethora of new elements to conventional simulators. Initially, the software used for this purpose was very similar to its scientific counterpart, providing only the relevant processes and variable manipulation. Although this could be used to reinforce learning acquired through other teaching methods or as a tool for exploratory or discovery learning (de Jong 1991; Reigeluth and Schwartz 1989), it was generally established that this was not enough. Students could easily become lost, confused, and generally unable to gain maximum knowledge from the educational simulator, especially when they were expected to learn material that was either partially or completely new to them using only the simulator (or any other type of discovery learning technique). The need for guidance methods in discovery learning and simulators therefore emerged (de Jong and van Joolingen 1998; Honomichl and Chen 2012; Lee 1999).

In terms of guidance and scripting, much can be taken from the experience of videogames, educational or otherwise. In their most primitive form, computer games offer the player guidance in the form of increasingly challenging and complex objectives which can be solved by progressively incorporating the game's mechanics (Fullerton et al. 2004). In the more elaborate game guides (or tutorials), the player is taken through a series of short steps in which the objectives are very concise and the user's actions are limited to the relevant subset of the total mechanics required to complete each objective (i.e. model progression). This can be accompanied by explanatory images, text,

animations, pedagogical agents, and other ways of focusing the player's attention on learning the new mechanics introduced by the game (Andersen et al., 2012).

These guidance and scripting techniques have been successfully implemented in educational games and simulators. It has also been proven repeatedly that students learning or reinforcing learned concepts using guided computer simulators perform better in evaluations than those exposed to purely exploratory simulators or unguided experiences (Kirschner et al. 2006; Moreno and Mayer 2005; Reid et al. 2003; Schrader and Bastiaens 2012). However, little research has been done into how these guiding systems should be built or which are better for different subjects or types of experience. Moreover, there is no comprehensive categorization for these systems. Some broad aspects of guides, such as the times at which feedback is given to the user, the effects involving pedagogical agents, and the need for an instructional overlay have been categorized (Alessi 2000; Kuk et al. 2012; Moreno and Mayer 2005; Osman et al. 2012; Reid et al. 2003). Furthermore, the elements that are useful for developing guides (e.g. text, images, objectives, feedback, etc.) have been discussed (van Joolingen and de Jong 2003), but they have seldom been compared to each other (de Jong et al. 1999), and their correct use has not been determined. There has also been no attempt made to find the optimal complexity of simulator guides for achieving the best learning results.

The aim of this research is to define a model which collects elements that have proven to enhance learning and would benefit educational simulators that incorporate them. Another goal is to define a model for the guidance systems of these simulators based on a review of the literature. Studies have been done to determine the effects of single elements of guidance systems within educational simulators, but not on the optimal amount or complexity of such elements. In this context, we analyze which elements are effective in simulator guides, and how complex or heavily-present they should be.

In order to do so, we compare two educational simulators designed to teach Cellular Respiration (a unit in cellular biology) to high school students. Both simulators have the same underlying software and mechanics and are based on the same proposed model. However, the two differ in their guidance systems (i.e. script). The first guides the user by giving them only background text, general objectives and feedback on their actions. The users are therefore left to discover for themselves the specific actions required to meet the objectives. The second simulator incorporates all of the features of the first, but also includes more complex tutorial elements such as explanatory images and texts for each mechanism, and step-by-step instructions with feedback that the user must follow sequentially in order to meet the overall objectives. Our aim in comparing the process of learning about cellular respiration concepts between these two test groups is to

determine the degree of complexity and amount of guidance needed in educational simulators.

The rest of the paper is divided as follows. In Section 2, a model is proposed based on a collection of characteristics gathered from previous research into the construction of effective pedagogical computer simulators, and a description of how each of these characteristics was implemented in the experimental simulator for this study. Section 3 describes a set of elements that have proved effective in guidance systems for educational simulators. The complexity and quantity of the aforementioned elements will later be compared in the two versions of the simulator. Section 4 describes the tools used in the experiment, namely the two versions of the *SimBIOS* educational simulator and the differences in their guidance systems. Section 5 explains the experimental design used to test for the correct amount and type of guidance required in simulators. Section 6 reports the results of the tests and their significance in determining the differences between the two simulator guides. Finally, Section 7 discusses the implications of these results and the possibilities for future research.

2.2. Gaming language, fidelity, and transparency in educational simulators

A model was devised to collect and identify the components of an effective educational simulator. This was done by studying various simulators that have proven successful in teaching or reinforcing subjects and reviewing the literature on individual elements that improve learning in educational simulators and games. An educational simulator (*SimBIOS)* was then developed following this model. Here, each element is described together with how it was implemented in the *SimBIOS* software.

2.2.1. Gaming language:

In a time when children and teenagers are in constant contact with videogames and other interactive media for entertainment, gaming language has been found to be a very effective tool to communicate and engage with students (Amory et al. 1999; Annetta et al. 2009; Dickey 2005; Garris et al. 2002).

This does not necessarily mean that an educational simulator has to be a game; rather that gaming elements such as vividly colored and attractive graphics, animations, intuitive and interactive interface, interactive virtual realities, competition, exploration, story, etc., enhance students' experience of the software. Since this language is native to them, they can quickly understand activities presented as a game, and will generally have a more positive attitude and be more motivated towards using the software. Evidence of this is available in extensive reviews of the literature evaluating motivation and other pedagogical factors in educational games (Amory 2010; Dondlinger 2007; O'Neil et al. 2005; Paras 2005). In turn, this motivation facilitates the student's immersion into the activity, allowing them to remain more concentrated on the subject for a longer period of time (Paras 2005).

SimBIOS was designed to incorporate several gaming elements. It has a graphical, two-dimensional virtual environment depicting the interior of a cell in a style similar to modern real-time strategy or simulation games (Fig 2-1). The different structures, organelles, and molecules with which the user can interact are all animated and can be directly manipulated using the mouse.

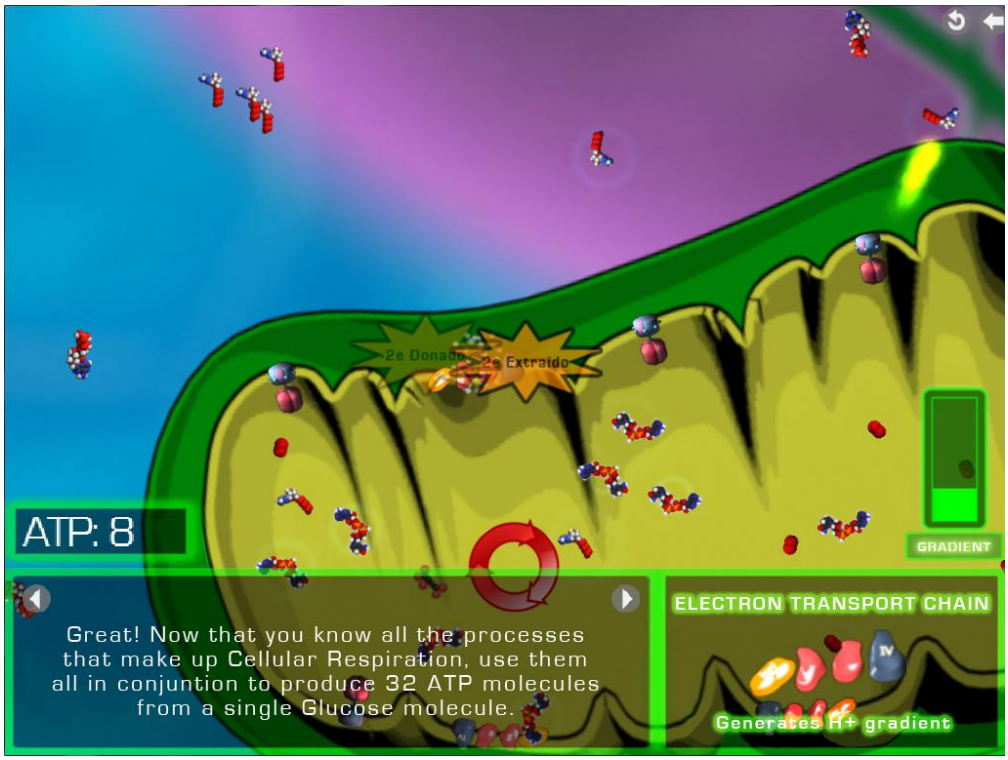


Fig 2-1 Cellular respiration simulation in SimBIOS

2.2.2. Fidelity: Abstraction of a complex model

All simulations are abstractions or representations of real processes, and will thus always be susceptible to simplifications while attempting to emulate real complexity (Reigeluch and Schwartz 1989). However, in the simulator's early stages, there was a tendency to build simulation software that was "as close to reality as possible" so as to get the most reliable results by factoring in as many variables as possible (Alessi 1988). This has been repeatedly proven not to be a good approach, since students could be overwhelmed by the complexity. Also, they might first need to learn things that are outside the scope of the target concept in order to interact with the tool. This can lead them to become frustrated or bored without ever interacting with the intended target concepts (Feinstein and Cannon 2001).

From this evidence, it can be surmised that in order to be more effective the software must have a clear educational objective and that the model to be simulated must be oriented towards that objective. Given that this model will always be an abstraction of

reality, seeking to meet this objective is more important that maintaining absolute fidelity to the processes themselves. This is true both in the semantic and syntactic sense, , i.e. while the model must be an abstraction of reality in terms of the concepts it teaches, the interaction with the user might also need to be abstracted to suit the pedagogical needs of the software. This could even mean that the interface has to be extremely simplified. This allows immediate usage of the simulation with minimal training, and the conjunction of these abstractions results in simplicity in the interaction between user and software.

The aim of the *SimBIOS* educational software was to teach the processes of animal cellular respiration as a whole. The focus of the designed model was therefore the direct interaction of the user with the steps (or sub-processes) involved in respiration. It was also important for the user to understand the way in which these steps connect to form a cycle of parallel sub-processes, where the output of each sub-process provides the input for the following. In reality, cellular respiration involves fairly complex chemical reactions and millions of molecules and intracellular structures. However, in this case only the most relevant steps to understand the overall process were included, with varying degrees of simplification. For example, the Krebs cycle in mitochondria involves 10 distinct and intricate steps, but in *SimBIOS* it was simplified to a process where the user inputs the correct molecules, and the output is given after a brief animation of the 10 stages (Fig 2-2).

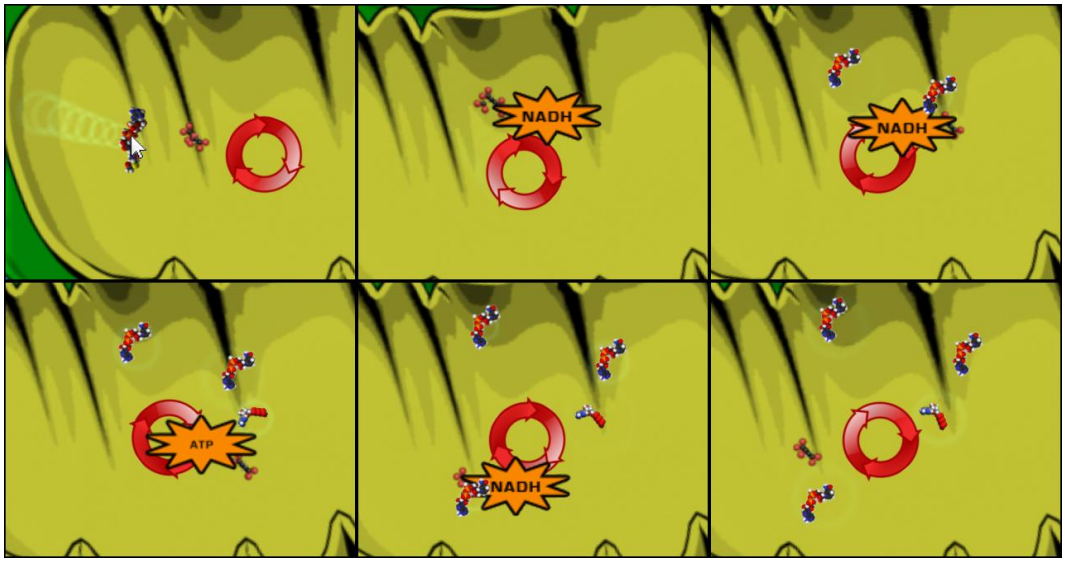


Fig 2-2 *SimBIOS* animation of the Krebs cycle highlighting the steps that produce important output

The interface is also simplified. In reality, the chemical processes of cellular respiration happen because of the random clash between millions of molecules. In the simulator, however, the user directly manipulates a few molecules to join them together and produce these effects. Although this strays from an "accurate" representation of reality, it allows the students to interact much more directly with the sub-processes and their relevance in the overall cycle. The abstraction of the model enabled the design of a simpler and more intuitive interface.

2.2.3. Transparency: Direct interaction with the processes

In general, the more direct a relationship the user can have with the process being learned, the better. Studies in scientific discovery learning over the last few decades have found that it is very difficult for students to formulate theoretical hypotheses based on observed data (i.e. the influence of variables on a system). Students also tend to maintain false hypotheses even in the face of contradicting evidence (de Jong and van Joolingen 1998; Swaak et al. 1998; van Joolingen and de Jong 1997). Therefore, if the variables or interfaces that the student manipulates only indirectly affect the outcomes of the different processes or elements, any hypothesis of causality between the two is probably incorrect. This proposed model deviates from scientific discovery learning to a more expository learning experience in the discovery/expository spectrum (Alessi 2000). Here, the pedagogical objective is to teach a specific process or concept (i.e. conceptual knowledge) rather than for the student to infer an underlying model by observing changes in abstract variables (de Jong et al. 1994; van Joolingen et al. 2005). Therefore, it is deemed that the ambiguity added by variables and different layers of abstraction in the processes (van Joolingen and de Jong, 1997) is detrimental to understanding these processes. For this reason we propose that the variables used in the simulation should be chosen in such a way as to make their effect on the processes as transparent as possible. The decision regarding the level of abstraction of these variables, however, has to be weighed against the possibility of adding complexity to the abstraction model described in Section 2.2. Consequently, interaction with the processes should be as direct as possible, while striving to minimize the complexity of the abstraction model.

In the *SimBIOS* educational software, the user directly interacts with the molecules and mitochondrial structures that carry out the multiple chemical reactions composing cellular respiration. This is done by clicking and dragging the molecules with the mouse. This method was preferred over other options (e.g. manipulating the amount of the different compounds present in the cell) because it allows the user to directly see which molecules must be joined in each process, the quantities required, and the exact output of each process. All this information, which was an important part of the simulator's

pedagogical objective, would have been left to the students' deduction if more indirect interactions had been implemented In *SimBIOS* this information is observed directly.

2.3. Guidance systems

The most important characteristics of optimal educational simulators are the guidance systems. As stated in the introduction, the presence of an incorporated guide (i.e. a script) in educational experiences with simulators has repeatedly yielded better results than purely exploratory simulations (Kirschner et al. 2006; Moreno and Mayer 2005; Reid et al. 2003; Schrader and Bastiaens 2012). These guidance systems are based primarily on the concepts of model progression and assignments (or objectives), and on the feedback given to the user as described below.

The primary objective in the comparative experiment with the *SimBIOS* educational simulator was to find the optimal level of complexity and flexibility for these guides.

2.3.1. Model progression and distinction of sub-processes:

An important concept incorporated into guided educational simulators is the progressive incorporation of new knowledge, increasing levels of difficulty, and understanding of mechanics during the experience with the software. This concept had already been incorporated into videogames much earlier. This basically consists of a cycle in which the user is presented with a new mechanic or concept and is made to practice it in isolation from the rest of the system. This new element is then incorporated into the part of the system that the user has already mastered, and is further used in conjunction with the other mechanics until a new one is introduced (Swaak et al. 2004; White and Frederiksen 1990). A number of studies have shown that similar concepts of model progression in educational simulators lead to higher student performance (Alessi 1995; Rieber 2005; Rieber and Parmley 1995; Swaak et al. 2004). This has also been stated as being good practice in the making of interactive, educational material (Plass 2009).

This technique allows students to progressively achieve the pedagogical objective without becoming overwhelmed by large amounts of unknown data or complex processes. It builds upon the knowledge acquired until a certain point in order to effectively teach the following segment. Also, since a new element is only included once the previous elements have been mastered, the risk of an element going unnoticed due to an information overload is lowered. In other words, it allows users to focus on the relevant parts of a process because it generates a context within a more manageable chunk of what might be a very complex process. If students are not able to distinguish

between and understand the different parts of a system, it will be more difficult for them to understand it effectively as a whole.

SimBIOS implements this concept as a game; using levels or stages of increasing complexity and difficulty. In each level, a new sub-process of cellular respiration is introduced and practiced on its own. The remaining necessary elements (inputs) and conditions are provided by the software. The simulation, including all the elements and sub-processes encountered up to that point, is then restarted and students must complete an objective that involves the new mechanic. However, this must be done in conjunction with the rest of the previously learned concepts (i.e. they must acquire the necessary elements and conditions through other sub-processes). In doing so, students obtain a clear understanding of each element (sub-process) that makes up respiration in order to better comprehend the entire process later on.

2.3.2. Presence of context, progressive objectives, and feedback:

The guidance mechanism proposed here must first provide context or a scenario in which the activities can be carried out, as well as the learning objectives (Choi 1997; Rieber 2002). This means at least describing what is going to be simulated in the software and the role the user will play in each simulation or activity.

Furthermore, the script must provide a sequence of game-like objectives consisting of assignments similar to those proposed by de Jong et al. (1996), but subject to the model progression described in Section 3.1. It is often difficult for students to self-regulate their learning process and assess their progress without the presence of assignments. This is especially the case if there is little prior knowledge of the subjects involved (Charney et al. 1990; Swaak et al. 1996). Given this tendency, the objectives that the user must complete have to be clearly defined and present at every stage of the activity. These should focus students on trying to overcome obstacles or solve problems or puzzles in order to achieve those objectives and progress through the activity. In accordance with the model progression (Section 3.1), the script must provide incremental objectives and challenges. In other words, the concepts and new mechanics must be gradually included as students progress through the activities or assignments. They must also be integrated and used in conjunction with those previously mastered to achieve each new objective before another concept or mechanic is incorporated into the simulation. Finally, users need to be given feedback about their actions inside the simulations so they can understand the consequences of those actions in the virtual environment as well as their implications for the current objective.

In the *SimBIOS* simulator, context and instructions for each activity were given using a Heads-Up Display (HUD) type interface in the form of text, images, and animations. They described the setting and what was going to be simulated in each level. Through this same means and using on-screen counters, the current objective or assignment that students were expected to complete was always visible, along with a display of their progress. The amount of steps required to complete these assignments, or the number of tasks previous to them varied between the two versions of the software. However the final objective of each level was generally the formation of a certain amount of molecules that required an increasingly complex series of chemical reactions to be chained together. In this way, each level incrementally added new concepts and encouraged students to incorporate them into their existing knowledge in order to progress.

2.4. Tools used

For the purpose of this experiment, a simulator (*SimBIOS*) was developed to teach the processes of cellular respiration, a common topic in high school biology. Two variants of the principle software were then developed: one with a simplistic guidance method and script, and another with more elaborate (or rigid) guidance. The latter includes more of the elements usually employed in making tutorial guides or scripts.

2.4.1. The two versions of *SimBIOS*

Both versions of the *SimBIOS* simulator were based on the same back-end software (and the model described in Section 2) and look and behave exactly the same. However, the program was designed to be flexible in scripting its internal guidance system, objectives, permitted actions, and stages. In terms of the core simulator, the two versions (the Minimal version and the Full version) were distinguished by scripting a different guidance system for each.

2.4.1.1. The Minimal version:

The guidance systems in the Minimal version of *SimBIOS* was designed to embody the characteristics described in Section 3 in their most basic form. Contextualization was subtle, the model progression takes bigger and broader steps, and the objectives are more general. Completing these objectives required some exploration and experimentation on behalf of the students. The inclusion of these very “general” guiding mechanisms made this the less restrictive or less “heavily guided” version of the software. Its main guiding characteristics were as follows:

a) Context of the simulation and its assignments and objectives is given through short texts on the HUD.
b) The simulation is divided into four levels in which general objectives are given, along with the necessary chemical formulas to achieve them. Students must explore the different actions available to them as well as decide which of the multiple molecules and structures are needed to reach their objective. For example, the script may give the objective "Generate glycolysis to produce 10 pyruvate molecules," and offer the simplified chemical equation for joining glucose and ATP molecules to produce pyruvate molecules. In this case students would have to figure out which molecules the equation is referring to, drag them together in a certain location, etc., and then repeat the process a certain amount of times to generate the specified amount of pyruvate.
c) In each of the four levels, a new sub-process of cellular respiration is introduced and practiced individually, and then repeated in conjunction with all the previous processes learned. By the end of the fourth level, all the sub-processes are practiced simultaneously, thus reproducing the entire respiration cycle.

2.4.1.2. The Full version:

The guide in the Full version of *SimBIOS* includes all the elements of the Minimal version. However, it also adds other components commonly found in guided software activities, games, and tutorials. The context is presented in several complementary ways such as through images, diagrams and text, and the model progression is implemented in much smaller increments. The objectives also detail more restricted goals. These require almost no exploration or experimentation in their first appearance in the script. Since the Full version includes all of the elements of the Minimal version, the following characteristics only represent the extra features of the Full script. This version of the software is considered to be more "heavily guided", or having the more restrictive guide of the two.

a) Context and activity texts are accompanied by images and diagrams of the different molecules, structures, and processes which are being mentioned.
b) In each level, students are taken through the newly introduced sub-process step-by-step; indicating through text, images, and schematics, the exact actions they have to carry out in order to achieve the desired reaction. In each small step, users' actions are restricted so that they can only perform those that help complete the current objective and they cannot advance to

the next level until it is completed. Once students have completed this "tutorial phase" of the level, the general objective is presented in the same way as in the Minimal version, and they must repeat the mechanics by themselves in order to complete the level's final objective.

c) The subdivision of levels and concepts is the same as in the Minimal version with the exception of the step-by-step explanation of the new mechanics in each level. This effectively limits the user's exploratory search for strategies and solutions to a minimum.

2.5. Experimental design

The quasi-experiment consisted of two groups of students who used the *SimBIOS* simulation software, each in a version that only differed in its implementation of the script. The learning of concepts through this activity was measured using a pre- and post- multiple-choice test.

2.5.1. Experimental groups

A study was performed in a private boys school in Santiago, Chile with 10^{th} grade students (approximately 16 years old). Students were divided into two groups depending on their class schedule (making this a quasi-experimental assignment), each one participating in the experiment using a different version of *SimBIOS* (FULL or MIN). There was an initial total of 148 students. However, only the students that performed the pre-test, the experiment itself, and the post-test were considered in the study. Therefore, the final group sizes were 72 students for the Minimal version and 58 students for the Full version. In general, the students had very limited prior knowledge of the subject, based only on what they knew from a review in biology class the previous year.

2.5.2. Evaluation

The whole experiment consisted of two sessions over two separate days. The first session, lasting approximately 1.5 hours was divided into two 45-minute segments, the first of which was used to administer a pre-test as a diagnostic of their previous knowledge on cellular respiration. The second segment was spent using the corresponding version of the *SimBIOS* simulator. In the second session, two days later, the same test was administered again to compare the knowledge of cell respiration gained through the use of the software.

The instrument used in the experiment as a pre- and post-test was a 34 question, multiple-choice exam administered digitally using a website. The questions all focused on cellular respiration, its products and effects, and the cellular anatomy involved. They ranged from simple facts that had to be memorized in order to be answered correctly, to the application of the concepts relating to cellular respiration in order to deduce their effects or relationships with other, more general subjects of biology. The instrument was devised by the researcher through the analysis of a segment of the local government's biology educational curriculum pertaining to cellular processes, and a review of 663 multiple-choice biology questions taken from the PSU exam (a standardized school-leaving exam in Chile). It was validated by secondary school biology teachers, and its reliability (using Cronbach's Alpha) was 0.82.

2.6. Results

After answers from the pre- and post-test were gathered, the difference between the number of correct answers before and after the use of the *SimBIOS* software for one 45 minute session was calculated to assess the improvement of each student in the aforementioned subjects.

Both groups saw a statistically significant improvement in their test scores before and after their experience with the *SimBIOS* simulator ($p < 1.154e-21$ for the MIN group and $p < 3.969e-13$ for the FULL group; and a Cohen's $d = 1.642$ and 1.342 for the MIN and FULL groups respectively). The Minimal version group (MIN) rose from a mean of 21.8% correct answers to 46.5%, and the Full version group (FULL) from 31.8% correct answers to 52.2%. This proves that both versions of the simulator, used for one 45 minute session, significantly improve the student's knowledge of the concepts relating to cellular respiration.

However, the difference between the mean score improvement using either version of the software was not significant enough to reject the null hypothesis. An independent sample T-test for equality of means rendered a 0.125 significance, and a one factor ANOVA test delivered a significance factor of 0.121 (Table 1), giving the null hypothesis ample favor in a 95% confidence interval.

ANOVA

Improvement

	Sum of Squares	df	Mean Square	F	Sig.
Between Groups	70.899	1	70.899	2.431	.121
Within Groups	3733.224	128	29.166		
Total	3804.123	129			

Table 1 One-way ANOVA test for the mean improvement of experimental groups

2.7. Conclusions

This research shows us that a guided pseudo-exploratory experience with an educational simulator proves to be an effective learning tool. According to the first research goal, a model was proposed to collect the elements necessary in creating an effective educational simulator (Section 2) as well as the necessary guidance systems it should incorporate (Section 3). This model was certainly validated by the significant improvement of the students' knowledge of the concepts that constituted the pedagogical objective in this experiment. It can be affirmed then, that didactic simulators developed following this model will produce effective pedagogical tools, at least in the domain of cellular biology processes, but almost certainly not limited to it.

Given the lack of a significant difference between the improvement in test scores with the two *SimBIOS* versions, the answer to the question of how "heavily guided" an educational simulator has to be in order to better achieve its pedagogical goals is fairly straightforward. The null hypothesis of the experiment could be translated into the asseveration that "the minimum guidance systems and scripting requirements for an educational simulator to be effective (described in Section 3)—and in their most basic form—should be sufficient to obtain the optimal performance from the didactic experience." No evidence could be found to suggest that the addition of extra guidance elements and the development of a much more complex script or "tutorialized" objectives and model progression have a significant impact on learning the concepts imparted by the software. We therefore conclude that the minimal set of script characteristics is, in fact, sufficient for this purpose. This conclusion takes into consideration that the abstraction of complexity in the simulator's model (Section 2.2) is the essential factor enabling the construction of the minimal set of characteristics required for a successful script. It is also worth noting that the results demonstrate that students do not need complete guidance in the activities, and that a simple focusing of the exploration efforts using model progression and game-like objectives is enough for them to understand the concepts behind their actions.

2.7.1. Future research

It would be valuable to expand this research into other fields of education in order to validate the proposed model for the effective construction of guiding systems in other domains. Although other forms of evaluation such as the use of concept maps and other more constructivist approaches are available (Liu & Lee, 2013), we leave as future work how to integrate these into guidance systems

BIBLIOGRAPHY

Alessi, S. M. (1995). Dynamic vs. static fidelity in a procedural simulation. *Annual Meeting of the American Educational Research Association*, San Francisco, CA.

Alessi, S. M. (2000). Designing educational support in system-dynamics-based interactive learning environments. *Simulation & Gaming*, *31*(2), 178-196.

Alessi, S. M. (1988). Fidelity in the design of instructional simulations. *Journal of Computer-Based Instruction*, *15*(2), 40-47.

Amory, A., Naicker, K., Vincent, J., & Adams, C. (1999). The use of computer games as an educational tool: identification of appropriate game types and game elements. *British Journal of Educational Technology*, *30*(4), 311-321.

Amory, A. (2010). Learning to play games or playing games to learn? A health education case study with Soweto teenagers. *Australasian Journal of Educational Technology*, *26*, 810-829.

Andersen, E., O'Rourke, E., Liu, Y., Snider, R., Lowdermilk, J., Truong, D., et al. (2012). The impact of tutorials on games of varying complexity. *Human Factors in Computing Systems*, *1*, 59-68.

Annetta, L. A., Minogue, J., Holmes, S. Y., & Cheng, M. (2009). Investigating the impact of video games on high school students' engagement and learning about genetics. *Computers & Education*, *53*(1), 74-85.

Blake, C., & Scanlon, E. (2007). Reconsidering simulations in science education at a distance: features of effective use. *Journal of Computer Assisted Learning*, *23*(6), 491-502.

Charney, D., Reder, L., & Kusbit, G. W. (1990). Goal setting and procedure selection in acquiring computer skills: a comparison of tutorials, problem solving, and learner exploration. *Cognition and Instruction*, *7*, 323-342.

Choi, W. (1997). Designing effective scenarios for computer-based instructional simulation: classification of essential features. *Educational Technology*, *37*(5), 13-21.

Cook, D. (2012). The Chemistry Of Game Design. *Gamasutra - The Art & Business of Making Games*. Retrieved June 7, 2013, from http://www.gamasutra.com/view/feature/1524/the_chemistry_of_game_design.php

de Jong, T. (1991). Learning and instruction with computer simulations. *Education and Computing*, *6*(3-4), 217-229.

de Jong, T., Härtel, H., Swaak. J., & van Joolingen, W. (1996). Support for simulation-based learning: the effects of assignments in learning about transmission lines. In A. Díaz de Ilarazza Sánchez & I. Fernández de Castro (Eds.), *Computer aided learning and instruction in science and engineering* (pp. 9-27). Berlin, Germany: Springer-Verlag.

de Jong, T., Martin, E., Zamarro, J., Esquembre, F., Swaak, J., & van Joolingen, W. (1999). The integration of computer simulation and learning support: an example from the physics domain of collisions. *Journal of Research in Science Teaching*, *36*(5), 597-615.

de Jong, T., & van Joolingen, W. (1998). Scientific discovery learning with computer simulations of conceptual domains. *Review of Educational Research*, *68*(2), 179-201.

de Jong, T., van Joolingen, W., Scott, D., de Hoog, R., Lapied, L., & Valent, R. (1994). SMISLE: System for multimedia integrated simulation learning environments. In T. de Jong & L. Sarti (Eds.), *Design and production of multimedia and simulation-based learning material* (pp. 133-165). Dordrecht, Netherlands: Springer.

Dickey, M. (2005). Engaging by design: how engagement strategies in popular computer and video games can inform instructional design. *Educational Technology Research and Development*, *53*(2), 67-83.

Dondlinger, M. J. (2007). Educational video game design: a review of the literature. *Journal of Applied Educational Technology*, *4*(1), 21-31.

Efe, H. A., & Efe, R. (2011). Evaluating the effect of computer simulations on secondary biology instruction: an application of Bloom's taxonomy. *Scientific Research and Essays*, *6*(10), 2137-2146.

Feinstein, A. H., & Cannon, H. M. (2001). Fidelity, verifiability, and validity of simulation: constructs for evaluation. *Developments in Business Simulation and Experiential Learning*, *28*, 57-62.

Fullerton, T., Swain, C., & Hoffman, S. (2004). *Game design workshop: designing, prototyping, and playtesting games*. San Francisco, California: CMP.

Garris, R., Ahlers, R., & Driskell, J. (2002). Games, motivation, and learning: a research and practice model. *Simulation & Gaming*, *33*(4), 441-467.

Honomichl, R., & Chen, Z. (2012). The role of guidance in children's discovery learning. *WIREs Cogn Sci*, *3*(6), 615-622.

Kirschner, P., Sweller, J., & Clark, R. (2006). Why minimal guidance during instruction does not work: an analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching. *Educational Psychologist*, *41*(2), 75-86.

Kuk, K., Milentijević, I., Rančić, D., & Spalević. (2012). Pedagogical agent in multimedia interactive modules for learning - MIMLE. *Expert Systems with Applications*, *39*(9), 8051-8058.

Lee, J. (1999). Effectiveness of computer-based instructional simulation: a meta analysis. *International Journal of Instructional Media*, *26*(1), 71.

Liu S. H. & Lee G. G. (2013, in press). Using a concept map knowledge management system to enhance the learning of biology. *Computers & Education*.

Moreno, R., & Mayer, R. (2005). Role of guidance, reflection, and interactivity in an agent-based multimedia game. *Journal of Educational Psychology*, *97*(1), 117-128.

O'Neil, H., Wainess, R., & Baker, E. (2005). Classification of learning outcomes: evidence from the computer games literature. *The Curriculum Journal*, *16*(4), 455-474.

Osman, K., Rahmat, R. A., & Tien, L. T. (2012). Interactive multimedia module with pedagogical agent in science and technology learning: application in electrochemistry. In I. J. Rudas, A. Zaharim, K. Sopian, & J. Strouhal (Eds.), *Recent researches in engineering education and software engineering: proceedings of the 11th WSEAS International conference on software engineering, parallel and distributed systems (SEPADS '12) [also] Proceedings of the 9th WSEAS International conference* (pp. 169-175). Cambridge, UK: World Scientific and Engineering Academy and Society and Society Press.

Paras, B. (2005, June). Game, motivation, and effective learning: an integrated model for educational game design. Resource document. *Digital Games Research Conference 2005, Changing Views: Worlds in Play*. http://summit.sfu.ca/item/281. Accessed 25 April 2013.

Plass, J. L., Homer, B. D., & Hayward, E. O. (2009). Design factors for educationally effective animations and simulations. *Journal of Computing in Higher Education, 21*(1), 31-61.

Reid, D., Zhang, J., & Chen, Q. (2003). Supporting scientific discovery learning in a simulation environment. *Journal of Computer Assisted Learning, 19*, 9-20.

Reigeluch, C. M., & Schwartz, E. (1989). An instructional theory for the design of computer-based simulations. *Journal of Computer-Based Instruction, 16*(1), 1-10.

Rieber, L. P. (2002, July). Supporting discovery-based learning within simulations. Resource document. *Proceedings of the International Workshop on Dynamic Visualizations and Learning., Tübingen: Knowledge Media Research Center.* http://www.iwm-kmrc.de/workshops/visualization/proceedings.htm. Accessed 25 April 2013.

Rieber, L. P. (2005). Multimedia learning in games, simulations, and microworlds. In R. Mayer (Ed.), *The Cambridge handbook of multimedia learning* (pp. 549-567). New York: Cambridge University Press.

Rieber, L. P., & Parmley, M. W. (1995). To teach or not to teach? Comparing the use of computer-based simulations in deductive versus inductive approaches to learning with adults in science. *Journal of Educational Computing Research*, *13*(4), 359-374.

Rutten, N., van Joolingen, W., & van der Veen, J. (2012). The learning effects of computer simulations in science education. *Computers & Education*, *58*(1), 136-153.

Schrader, C., & Bastiaens, T. (2012). Learning in educational computer games for novices: the impact of support provision types on virtual presence, cognitive load, and learning outcomes. Resource document. *The International Review of Research in Open and Distance Learning*, *13*(5). http://www.irrodl.org/index.php/irrodl/article/view/1166/2249. Accessed 25 April 2013.

Smetana, L., & Bell, R. (2011). Computer simulations to support science instruction and learning: a critical review of the literature. *International Journal of Science Education*, *34*(9), 1337-1370.

Swaak, J., van Joolingen, W., & de Jong, T. (1996). *Support for simulation based learning: the effects of model progression and assignments on learning about oscillatory motion*. Enschede, Netherlands: Faculty of Educational Science and Technology, Centre for Applied Research on Education.

Swaak, J., van Joolingen, W., & de Jong, T. (1998). Supporting simulation-based learning: the effects of model progression and assignments on definitional and intuitive knowledge. *Learning and Instruction*, *8*(3), 235-253.

Swaak, J., de Jong, T., & van Joolingen, W. R. (2004). The effects of discovery learning and expository instruction on the acquisition of definitional and intuitive knowledge. *Journal of Computer Assisted Learning*, 20(4), 225–234.

van Joolingen, W., & de Jong, T. (1997). An extended dual search space model of scientific discovery learning. *Instructional Science*, *25*(5), 307-346.

van Joolingen, W., & de Jong, T. (2003). SimQuest, authoring educational simulations. In T. Murray, S. Blessing, & S. Ainsworth (Eds.), *Authoring Tools for Advanced Technology Learning Environments* (pp. 1-31). Dordrecht: Kluwer.

van Joolingen, W., de Jong, T., Savelsbergh, A. W., & Manlove, S. (2005). Research and development of an online learning environment for collaborative scientific discovery learning. *Computers in Human Behavior*, *21*(4), 671-688.

Vogel, J., Vogel, D., Cannon-Bowers, J., Bowers, C., Muse, K., & Wright, M. (2006). Computer gaming and interactive simulations for learning: a meta-analysis. *Journal of Educational Computing Research*, *34*(3), 229-243.

White, B. Y., & Frederiksen, J. R. (1990). Causal model progressions as a foundation for intelligent learning environments. *Artificial Intelligence*, *42*(1), 99-157.

3. APPENDIX

3.1. E-mail of acknowledgement of submission

International Journal of Science Education - Manuscript ID TSED-2013-0370-A

EDITOR_IJSE@hotmail.co.uk <EDITOR_IJSE@hotmail.co.uk> 7 de agosto de 2013 10:54
Para: pacori@uc.cl

07-Aug-2013

Dear Mr. Cori:

Your manuscript entitled "The value of guidance when teaching science using an educational simulator" has been successfully submitted online and is presently being given full consideration for publication in the International Journal of Science Education.

Your manuscript ID is TSED-2013-0370-A.

Please mention the above manuscript ID in all future correspondence or when calling the office for questions. If there are any changes in your street address or e-mail address, please log in to ScholarOne Manuscripts at http://mc.manuscriptcentral.com/ijse and edit your user information as appropriate.

You can also view the status of your manuscript at any time by checking your Author Center after logging in to http://mc.manuscriptcentral.com/ijse .

Thank you for submitting your manuscript to the International Journal of Science Education.

Sincerely,
International Journal of Science Education Editorial Office